IoT communications with blockchain and multi-chain : a case study in the automotive industry

Luc Gerrits

Communications IoT avec Blockchain et Multi-Chain : Une Étude de Cas dans l'Industrie Automobile

Luc GERRITS

Laboratoire d'Electronique, Antennes et Telecommunications (LEAT)
UMR7248 UniCA CNRS

Présentée en vue de l'obtention
du grade de docteur en Électronique
d'Université Côte d'Azur
Dirigée par :
Prof. François VERDIER, Université Côte d'Azur
Soutenue le :
17 Juin 2024

Devant le jury, composé de :
Prof. Jérôme Lacan, ISAE-SUPAERO Toulouse
Prof. Pierre Boulet, Université de Lille
Dr. Pascal Benoit, LIRMM Université Montpellier 2
Dr. Christine Hennebert, CEA-Leti Grenoble
Dr. Patricia Guitton-Ouhamou, Renault
Dr. Xing Liu, Kwantlen Polytechnic University Vancouver
Dr. Massimo Vecchio, FBK CREATE-NET, Italy

Communications IoT avec Blockchain et Multi-Chain : Une Étude de Cas dans l'Industrie Automobile

---

IoT Communications with Blockchain and Multi-Chain: A Case Study in the Automotive Industry

Luc GERRITS

Jury :

Président du jury
Prof. Cyrille Bertelle, Normandie Université
Rapporteurs
Prof. Jérôme Lacan, ISAE-SUPAERO Toulouse
Prof. Pierre Boulet, Université de Lille
Examinateurs
Dr. Pascal Benoit, LIRMM Université Montpellier 2
Dr. Christine Hennebert, CEA-Leti Grenoble
Dr. Patricia Guitton-Ouhamou, Renault
Directeur de thèse
Prof. François VERDIER, Université Côte d'Azur
Membres invités
Dr. Xing Liu, Kwantlen Polytechnic University Vancouver
Dr. Massimo Vecchio, FBK CREATE-NET, Italy

Acknowledgements

I would like to express my sincere gratitude to the individuals who have contributed to the completion of my PhD thesis.

First and foremost, I extend my deepest appreciation to my PhD supervisor, François Verdier, for his unwavering support and guidance throughout the entirety of my doctoral journey spanning over five years. His mentorship, from the transition of a student to a colleague, has been invaluable in providing expertise and dedication, steering me towards excellence. You introduced me to blockchain technology back in 2018, which ignited my passion for distributed systems from the unique perspective of an embedded-system/electronics specialist.

Furthermore, I am deeply indebted to my family. My parents have been a constant source of support, always believing in me and encouraging me. I am grateful to my brothers, whose camaraderie and laughter have provided comfort during challenging times.

I extend my heartfelt thanks to the members of my thesis defense jury: Prof. Jérôme Lacan, Prof. Pierre Boulet, Prof. Pascal Benoit, Dr. Christine Hennebert, Prof. Cyrille Bertelle, Dr. Patricia Guitton-Ouhamou, Dr. Massimo Vecchio and Dr. Xing Liu for their invaluable insights and constructive feedback.

My journey as a researcher would not have been as enriching without the collaboration of the blockchain EDGE team. I extend my appreciation to Roland Kromes, Cyril Naves, Edouard Kilimou, Thomas Mabrut, Manon Arnaudo, and François Verdier for their contributions and support.

Special mention goes to Roland Kromes, whose guidance and friendship have been instrumental since my internship at the LEAT lab in 2018. Marta Ballatore, I am grateful for your innovative perspectives and collaborative efforts, under the supervision of Lise Arena and François Verdier. Our friendship, forged during the SIM project, has continued through various professional activities, and I wish you all the best for future success.

Luca Santamaria, Lyes Khacef, Marino Rasamuel, and Edgar Lemaire, along with Roland, have not only been colleagues but have also become good friends. I'll always look forward to sharing a drink with you all and wish you nothing but the best in your personal and professional projects.

I also extend my thanks to the numerous lab researchers and friends who have surrounded me in my academic experience with engaging discussions, laughter, and friendly chess matches during coffee breaks. Your camaraderie has made this journey even more memorable: Laurent Rodrigez, Flora Zidane, Walid Chekar, Julian Roqui, Lionel Tombakdjian, Jonathan Courtois, Francesco Positano, Dalia Hareb, Yacine Khacef, Thomas Louis, Maymouna Dabbous, Yassine Chouchane, Imourane Abdoulaye, Zhuoer Li, Artem Muliukov, and Ruochen Ding.

Lastly, I extend my gratitude to all the teachers, from my formative years in school to my professors at the university, for imparting invaluable knowledge and shaping my academic journey.

Thank you to everyone, and my apologies to those not mentioned explicitly, your support and presence made my journey truly memorable.

Résumé

L'Internet des Objets (IoT) a révolutionné la manière dont les dispositifs interagissent et communiquent entre eux. Avec l'augmentation du nombre de dispositifs IoT, il devient nécessaire de développer des protocoles de communication sûrs et fiables. La technologie blockchain s'est imposée comme une solution prometteuse pour assurer une communication sécurisée et fiable dans les réseaux d'IoT. La blockchain est une technologie de registre distribué (DLT) qui vise à établir la confiance entre les individus à l'aide d'un réseau décentralisé d'ordinateurs. L'IoT peut exploiter cette technologie pour stocker des informations, des identités et également une logique personnalisée dans la blockchain, agissant ainsi comme une ressource d'information fiable régie par un algorithme de consensus. La mise en œuvre des algorithmes de consensus sur une blockchain présente plusieurs défis et des compromis (évolutivité, décentralisation, sécurité, etc.). L'utilisation de multiples blockchains (également connue sous le nom de technologie multi-chaîne), peut améliorer davantage l'évolutivité et l'interopérabilité des réseaux d'information, et des échanges de données IoT en ce basant sur la blockchain. L'industrie automobile est l'un des domaines clés où les technologies IoT sont rapidement adoptées. Avec l'avènement des véhicules connectés et autonomes, le besoin de communication sécurisée et fiable entre les véhicules, les infrastructures et d'autres dispositifs devient de plus en plus critique. Cette thèse vise à étudier l'utilisation de la blockchain et des technologies multi-chaînes pour les communications IoT dans l'industrie automobile.

L'étude de cas se concentrera sur un cas d'utilisation spécifique dans l'industrie automobile, tel que l'accident de la route. La recherche analysera les performances, la sécurité et l'évolutivité des protocoles de communication blockchain et multi-chaîne pour les IoT connectés dans ce contexte. L'étude examinera également la faisabilité de la mise en œuvre de ces technologies dans des scénarios réels et identifiera les défis et limitations potentiels.

L'étude fournira des informations sur les avantages et les limitations de la blockchain et des technologies multi-chaînes pour les communications IoT et guidera les recherches futures dans ce domaine. En fin de compte, cette thèse vise à faciliter l'adoption de protocoles de communication basé sur blockchain pour des IoTs sécurisés et fiables dans l'industrie automobile et au-delà.

Mots-clés : blockchain, IoT, BIoT, interoperabilité

Abstract

The Internet of Things (IoT) has revolutionized the way devices interact and communicate with each other. As the number of IoT devices increases, there is a growing need for secure and reliable communication protocols. Blockchain technology has emerged as a promising solution to ensure secure and trustworthy communication in IoT networks. Blockchain is a distributed ledger technology (DLT) that aims to establish trust between individuals using a decentralized network of computers. IoT can leverage this technology to store information, identities and also custom logic in the blockchain, thus acting as a trusted resource of information governed by a consensus algorithm. The implementation of a consensus algorithm on blockchain comes with multiple challenges and a trade-off (scalability, decentralization, security, etc.). The use of multiple blockchains (aka multi-chain technology) can further enhance the scalability and interoperability of blockchain-based IoT networks. The automotive industry is one of the key areas where IoT technologies are being adopted rapidly. With the advent of connected and autonomous vehicles, the need for secure and reliable communication between vehicles, infrastructure, and other devices is becoming increasingly critical. This thesis aims to investigate the use of blockchain and multi-chain technologies for IoT communications in the automotive industry.

The case study will focus on a specific use case in the automotive industry, such as the road accident. The research will analyze the performance, security, and scalability of blockchain and multi-chain-based IoT communication protocols in this context. The study will also examine the feasibility of implementing these technologies in real-world scenarios and identify any challenges and limitations.

The study will provide insights into the benefits and limitations of blockchain and multi-chain technologies for IoT communications and guide future research in this area. Ultimately, this thesis aims to facilitate the adoption of blockchain-based communication protocols for secure and reliable IoTs in the automotive industry and beyond.

Keywords : blockchain, IoT, BIoT, interoperability

Author's Publication List

Peer-Reviewed Journal Paper

[J1] M. Ballatore, **L. Gerrits**, R. Kromes, L. Arena and F. Verdier, "Toward the Conception of a Multichain to Meet Users' Future Needs: A Design Science Research Approach to Digital Servitization in the Automotive Industry," in *IEEE Transactions on Engineering Management*, 2023

Peer-Reviewed International Conference Paper

[C1] **L. Gerrits**, T. Mabrut, F. Verdier. "Communication Vehicle Accident Data to Blockchain for Secure and Reliable Record Keeping Using Android Automotive Application". BCCA 2023 BCCA 2023 : *The Fifth International Conference on Blockchain Computing and Applications*, Jun 2023, Koweit City, Kuwait. pp.6.

[C2] M. Arnaudo, **L. Gerrits**, I. Grishkov, R. Kromes, and F. Verdier. 2023. Blockchains Accesses for Low-Power Embedded Devices using LoRaWAN. In Proceedings of the *12th International Conference on the Internet of Things (IoT 2022)*. Association for Computing Machinery, New York, NY, USA, 119–126.

[C3] **L. Gerrits**, C. Naves Samuel, R. Kromes, F. Verdier, S. Glock, and P. Guitton-Ouhamou. 2021. Experimental Scalability Study of Consortium Blockchains with BFT Consensus for IoT Automotive Use Case. In Proceedings of the 19th *ACM Conference on Embedded Networked Sensor Systems* (SenSys '21). Association for Computing Machinery, New York, NY, USA, 492–498.

[C4] **L. Gerrits**, E. Kilimou, R. Kromes, L. Faure and F. Verdier, "A Blockchain cloud architecture deployment for an industrial IoT use case," *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, 2021, pp. 1-6.

[C5] **L. Gerrits**, R. Kromes and F. Verdier, "A True Decentralized Implementation Based on IoT and Blockchain: a Vehicle Accident Use Case," *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, Barcelona, Spain, 2020, pp. 1-6.

[C6] R. Kromes, **L. Gerrits** and F. Verdier, "Adaptation of an embedded architecture to run Hyperledger Sawtooth Application," *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, Vancouver, BC, Canada, 2019, pp. 0409-0415.

Peer-Reviewed National Conference Paper

[N1] **L. Gerrits**, F. Verdier. ”Using Blockchain and Android Automotive Application for Secure and Reliable Vehicle Accident Data Management”. *17ème colloque du GDR SoC2 2023*, Jun 2022, Lyon, France.

[N2] **L. Gerrits**, C. Naves, F. Verdier. ”IoT-Blockchain Based Ecosystem using Hyperledger Sawtooth”. *16ème Colloque National du GDR SOC2*, Jun 2022, Strasbourg, France.

[N3] **L. Gerrits**, K. Tetouhe, F. Verdier. ”Hyperledger Sawtooth Blockchain for IoT-Blockchain Based Ecosystem”. *15ème Colloque National du GDR SOC2*, Jun 2021, in remote, France.

Contents

1 Introduction

1.1 Introduction

The Internet of Things (IoT) represents a significant shift in the digital-physical landscape, contributing to an era where interconnected devices are present in every aspect of our lives. This network of devices, ranging from the most simple household items (mobile phone, smart thermostat, smart home…) to complex industrial equipment (autonomous vehicles, city air/water sensors, warehouse automation…), has transformed the way we interact with the physical world. However, as IoT presence and applications expand, it brings forward multiple critical challenges, primarily centered around the efficiency, data management and security of these networks. Addressing these challenges is not only essential but it is also a key to unlock the full potential of IoT.

Blockchain technology, famous for its robust security and decentralized architecture, emerges as a potential solution to these challenges. It offers a paradigm shift from traditional centralized systems, providing a framework for secure, transparent, and efficient communication mechanism between devices.
The decentralization aspect of blockchain, holds significant promise for IoT networks, as it eliminates single points of failure, thus enhancing overall network resilience.
Additionally, automation of processes provided by blockchain (i.e. smart contracts) enhances the efficiency and functionality of IoT systems, allowing complex and secure interactions between devices. These smart contracts can execute pre-defined conditions autonomously, reducing the need for human intervention and potentially decreasing the likelihood of errors. This automation, combined with blockchain's inherent traceability, by its immutability, ensures a high degree of reliability and accountability in IoT networks.

This thesis explores the integration of blockchain technology in IoT communications. The industrial automotive context of this thesis provides a central thread that allows us to question the current state of blockchain and its application in the IoT domain. The automotive sector comprises of many heterogeneous IoT devices (roadside systems, vehicles…) which requires secure and reliable communication protocols.

Our journey will explore the limits of blockchain technology targeting IoT use cases, the implementation of a vehicular use case using blockchain, and its practical integration within IoTs. Although the findings of the thesis revolve around a specific use case, the scope of the research encompasses a wider range of IoT applications, giving insight into the adaptability of blockchain technology in potentially different contexts.

In essence, this thesis is an exploration of the confluence of two revolutionary fields: IoT and blockchain. It aims to bring to light the complexities of their integration and pave the way for the development of secure, efficient and scalable IoT networks, leveraging the transformative

potential of blockchain technology.

1.2 Thesis Context

This thesis is part of the development of smart cities, focusing on the specific case of vehicles. In the near future, cars will undergo a major shift in their interfacing and inter-communication with the outside world. The car is an IoT that increasingly communicates with numerous services that require interoperability.

1.2.1 Smart cities: Towards Smart Mobility

In a likely future world, smart cities would be a reality: It will enhance many aspects of life by creating a smart economy, a sustainable governance, a smart housing, a sustainable environment, and a smart mobility[1]. A smart city provides an intelligent way to manage components (Figure 1.2.1) such as health, energy, transport, homes and buildings [1]. These cities components primarily generate data from wireless sensor networks, and these sensors are deployed in multiple applications such as health monitoring, smart industry, smart home, water monitoring and environment monitoring. Existing wireless infrastructure (3G, 5G, LoRaWAN, Wi-Fi, ...) are used by IoT devices to communicate there data.

Advances in wireless communication (5G, 6G, etc.) has an effect to ease information exchange capabilities, allowing the deployment of a greater number of objects [3] [4]. Additionally, smart cities need to operate in a secure way to avoid potential breaches and ensure the privacy and safety of their inhabitants. The IoT serves as a backbone for the creation of smart cities; interconnected devices ranging from traffic sensors to personal wearables all collecting, sending, and processing data in real time. This web of connectivity, despite its numerous advantages, simultaneously opens the door to a vast array of inefficient data management and security vulnerabilities [5].

Cybersecurity in smart cities cannot be neglected: it must be constructed into the fabric of the technological infrastructure from the start (*security by design*). The complexity of these environments, with countless devices interacting with each other, necessitates a secure, transparent, and efficient way to manage not just data, but also the data exchanges (*transactions*) and communications between these devices. Herein lies the potential of blockchain technology.

Blockchain's inherent characteristics – decentralization, transparency, immutability, and cryptographic security – offer a framework that could revolutionize smart city initiatives [2]. Decentralization ensures that there is no single point of failure that could compromise the entire system. The transparency of blockchain provides a verifiable and auditable history of all transactions, contributing to an environment of trust among devices and users. Immutability protects past data transactions from being altered or deleted, which is crucial in maintaining the integrity of the data collected by smart city sensors and devices. Finally, the cryptographic security of blockchain safeguards against unauthorized access and tampering.

Moreover, smart contracts on blockchain platforms have the potential to automate and streamline processes and services, reducing the likelihood of human error and increasing efficiency. It can provide governance automation: smart contracts can automatically execute a process when certain conditions are met. An example could be releasing payment once a public project is completed and verified, thus reducing corruption and increasing accountability.

[1] The term "smart" here generally refers to the use of technology to enhance efficiency and quality of life.

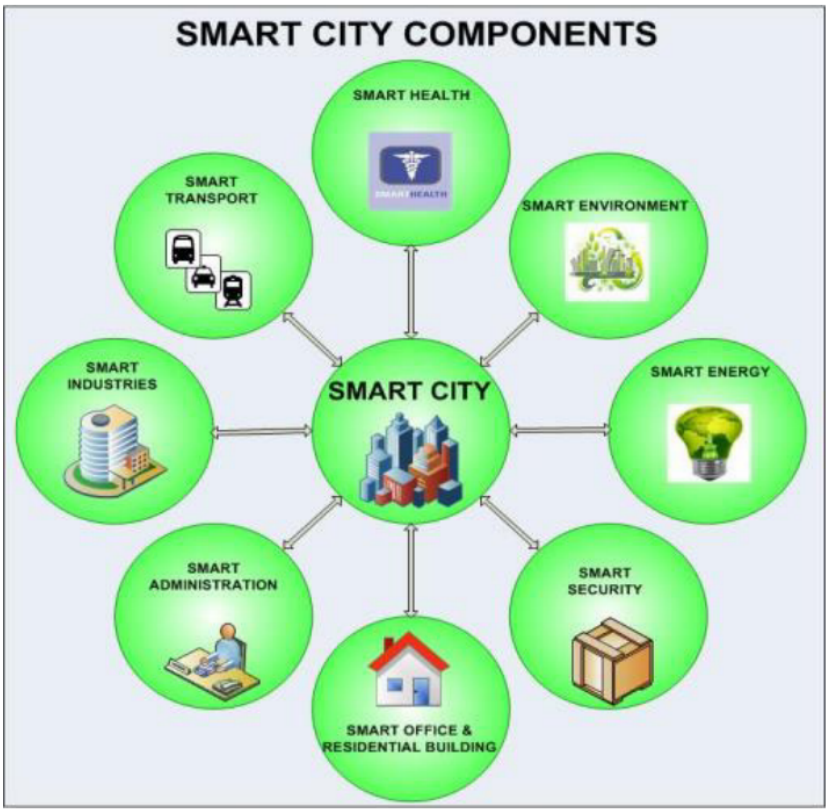


Figure 1.2.1: Smart city components [1]. Treiblmaier et al. propose the definition: "A smart city is a geographical area with a high population density that uses information and communication technologies (ICT) to connect and monitor critical infrastructural components and services with the goal of improving the efficiency and the environmental, economic and social sustainability of its operations as well as the quality of life for its citizens" [2].

These blockchain-enhanced processes are particularly relevant when discussing smart mobility. Smart mobility involves the integration of technology to manage a city's transportation system, ensuring that it is efficient, eco-friendly design, and capable of adapting to the needs of its users in real-time. By utilizing blockchain technology, smart mobility solutions can become more secure, more resilient to cyber threats, provide transparent and immutable records of transactions and interactions within the system.

For example, a blockchain-based smart mobility system could securely process payments for transportation services, from bus fares, vehicle tolls and parking fees, without the risk of fraud or breaches in payment systems [6] [7][8][9]. In addition, the data generated by transportation networks (like traffic patterns, usage rates, and vehicle diagnostics) could be more securely stored with blockchain, giving city planners access to reliable data to create better decisions to improve traffic flow and reduce congestion.

1.2.2 Focus on the Automotive Industry and the Android-Automotive Trending Integration

The automobile sector is currently undergoing a paradigm shift in the areas of in-vehicle infotainment (IVI) and connectivity. This evolution is largely influenced by the growing popularity of powerful computing devices in cars at a relatively low cost, enabling the user experience to be

enhanced, additional connected functions to be incorporated and advanced driving mechanisms to be improved.
The concept of Software-Defined Vehicles (SDV) emerges as a consequence of the gradual evolution of cars from *"mechanical-electrical interfaces into sophisticated, upgradable mobile electronic platforms with the capability of ongoing enhancements"* [10].

The use case this thesis is gravitating around, is part of this automobile evolution, where the car is no longer seen as a simple standalone mechanical vehicle, but as a complex connected object with the ability to communicate with its environment. These modern vehicles are supported by various operating systems (OS), predominantly QNX, Android, and other Linux or Windows based. In addition, new vehicles have the ability to be software-upgraded, possibly creating more value over time. A vehicle value can increase by creating links between Original Equipment Manufacturers (OEM) and automotive/internet software companies. These new vehicles also improve user experience with additional applications to the IVI platform [10].

Currently, numerous vehicle manufacturers are either preparing or have already begun to transition to the Android Automotive operating system (a Google-developed open-source platform) as an alternative to their in-house built IVI systems. According to a report on the automotive software and electronics market, it is anticipated that the software sector will experience an annual growth of 9.1% until 2030 [11]. The IVI systems in these advanced vehicles are generally built using System-On-Chips (SoC) and have a hardware design similar to that of single-board computers.

It is projected that the global IoT market will expand to over 43 billion connected devices, with a potential economic impact of $11 trillion by 2025 [12][13]. The automotive sector is a significant contributor to this trend. The rapid increase in IoT devices introduces a spectrum of ever-evolving challenges (such as centralization, privacy, scalability, and security). In the realm of automotive IoT, these challenges pose substantial risks, including serious injuries or even fatalities. Recent research has shown a growing interest in blockchain technology as a potential solution for various IoT challenges, including related to vehicular IoT applications [14][7][8][9].

This thesis explores integration of blockchain for constrained IoT devices (Chapter 4), however in the context of this new technological shift in automotive industry by using Android Automotive OS (AAOS) we also focused on this specific combination of AAOS and blockchain (Section 4.6, page 139).

1.2.3 The Accident Use Case: Multiple Interconnected Services

The SIM automotive use case is about studying the accident of a connected vehicle, riding in a smart city environment, interacting with one or more services (OEM, insurance, car mechanic, etc.) [15]. These services are assumed to be using blockchain technology as a foundation to allow the seamless exchange of information, creating a cloud of interconnected blockchain-based services. As we saw earlier in the state of the art, blockchain can be used in a private/consortium or public configuration. In 2017, the interconnection of multiple blockchains was a new concept with limited implemented solutions. Today, in 2023, multi-chain and interoperability protocols exists and have a stable version enough to be a candidate to production implementations, thus also potentially industrial usage.

The SIM project identified use case is part of the vehicle "Smart Service Book" in which the vehicle life-cycle is digitized and realized using smart contracts. The collection of information stored using blockchain can thus use the potential advantages of smart contracts to automate and create a more seamless experience for all parties involved with the car. This project was introduced in 2018 [16], in parallel with the work of Brousmiche et al. [9] who two month prior proposed a related work on the automation of data processing around the vehicle with a "Vehicles Data and Processes Ledger framework". The SIM project takes the work of Brousmiche et al. a step further by introducing the use case in different disciplines and regrouping them in a common multi-disciplinary project. This multi-disciplinary has allowed to collaborate and work on practical challenges that blockchain posed to the use case [17][18][19][6].

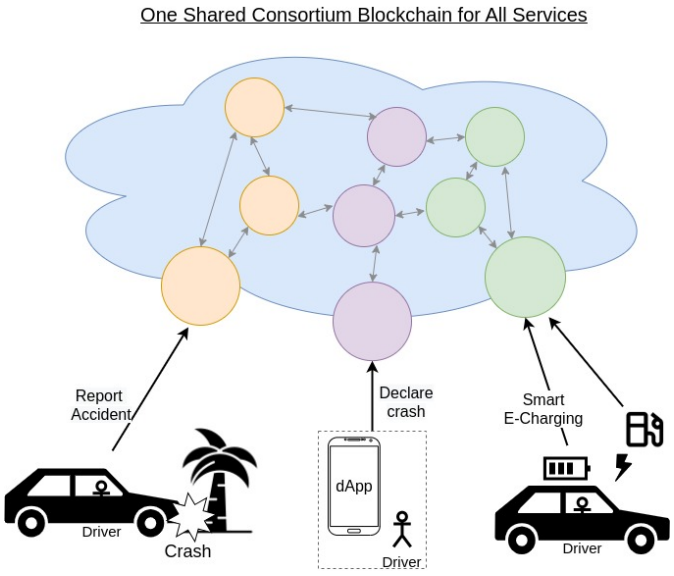


Figure 1.2.2: Smart IoT for Mobility ecosystem architecture using a consortium blockchain configuration

In Figure 1.2.2 and Figure 1.2.3 we propose the two possible SIM use case ecosystem architecture overview. First a single blockchain (i.e. solo-chain) can be used to create an interoperable system (from the OEM, driver, etc.) point-of-view. In this situation, a private or consortium blockchain is typically employed. Thus, the solo-chain ledger governance is managed by multiple parties and can be configured in various ways (blockchain technology/platform-depend). We will later explore this consortium configuration, where all services are hypothetically interacting with one blockchain. This first implementation of the consortium configuration provide ground work

for the use case feasibility using blockchain and performance study for a matching real-world scenarios.

The second configuration in Figure 1.2.3 provide an interoperable ecosystem of blockchains that is much more complex as the first method, due to multiple solo-chains being interconnected through a common blockchain. In the figure example, three services (OEM, insurance and electric grid) have each their own domain-specific blockchains. Cross-communication between ”parallelized” networks are possible by using a intermediary blockchain (interoperability). We address this multi-blockchain ecosystem configuration as the last part of this thesis.

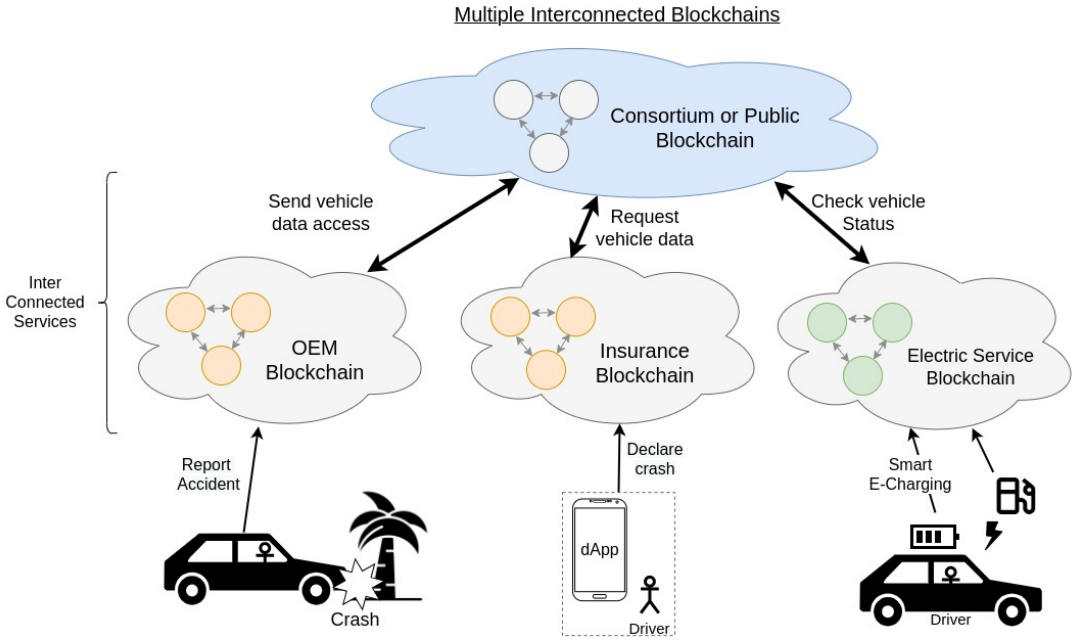


Figure 1.2.3: Smart IoT for Mobility ecosystem architecture using a multi-chain configuration

1.3 Research Gap and Underlying Assumptions

The state of art from next chapter 2, extensively describes current approaches and findings in the intersection of blockchain and IoT. Based on this state of art we will discover a lack of research at the some intersection points of blockchain technology and IoT.

Blockchain, introduces a relatively novel philosophy in information communication and storage, providing a secure, decentralized approach to data management. As suggested in literature, in the context of IoT, where billions of devices continuously generate and exchange data, the application of blockchain can revolutionize the way these networks operate [20]. However, despite its potential, there is a noticeable gap in comprehensive research that specifically addresses the practical implementation and integration of blockchain within IoT ecosystems. BIoT has been studied since 2018, with numerous proposals and even new businesses but blockchain still lacks to answer the requirements of IoT use cases. However, with the arrival of novel blockchain paradigms (i.e. multi-chain), BIoT has the possibility to be adopted in the current years because of matching performances, adaptation to legislation, and interoperability with other systems. This thesis is about filling the gaps in the complex BIoT realm.

Our research is grounded in several key assumptions regarding the integration of blockchain into the IoT domain. Firstly, we presume that blockchain significantly improves the security of IoT data communication. This includes the assumption that blockchain, along with multi-chain ecosystems, is capable of addressing a number of vulnerabilities inherent to IoT networks. The privacy of IoT data is explicitly left out of this thesis due to the fact that it goes beyond the subject.

Furthermore, we assume that the successful integration and implementation of blockchain within IoT ecosystems depends on the collaborative efforts of various stakeholders. This group encompasses IoT device manufacturers, service providers, and users, whose cooperation is essential for achieving the desired outcomes.

Another foundational assumption is that the deployment of blockchain technology in IoT realm adheres to existing legal, regulatory, and ethical standards. This is particularly crucial with respect to data privacy and confidentiality, ensuring that such implementations are compliant and respectful of these considerations.

Lastly, we posit that the integration of blockchain technology into IoT networks – especially within smart mobility and the automotive sector – is economically viable. This viability extends to multi-chain ecosystems, where the assumption is that they encourage new economic models and opportunities for data monetization, contributing to the overall economic feasibility of such ventures.

1.4 Research Contributions and Objectives

This thesis outlines specific research objectives and contributions aimed at exploring the integration of blockchain technology in IoT, with a focus on the automotive industry. The objectives span from analyzing the performance of blockchain platforms in automotive settings to developing blockchain client software for IoT devices, and investigating multi-chain solutions for IoT. Each objective is structured to progressively build on the knowledge and findings from the previous, providing a structured approach to understanding the application and potential challenges of blockchain technology in the IoT domain.

Performance Analysis of Blockchain Platforms in Automotive Context

- *Objective*: To assess the suitability of private/consortium blockchain platforms for automotive applications.
- *Method*: Conducting a thorough performance analysis focusing on key metrics like speed, reliability, and scalability of various consensus algorithms.
- *Expected Outcome*: Determining the most efficient blockchain platform for automotive scenarios, providing valuable insights for the application of blockchain in the automotive industry.

Transitioning from understanding the best blockchain platforms for automotive applications, the research then explores the subject of implementing blockchain within the IoT landscape.

Exploration of Blockchain Client Applications on Development Boards

- *Objective*: To explore and evaluate the implementation of multiple blockchain client applications on relatively powerful development boards.
- *Method*: Testing and assessing various blockchain clients on development boards, focusing on their performance, capabilities, and limitations.
- *Expected Outcome*: Gaining a comprehensive understanding of the feasibility and efficiency of blockchain clients in more capable IoT environments, setting the stage for further optimization.

Following the exploration and understanding gained from the initial phase, the research then focuses on developing an optimized blockchain client application for more constrained IoT devices.

Development of an Optimized Blockchain Client for Constrained IoT Devices

- *Objective*: To develop an optimized blockchain client application with a generic protocol for IoT devices with limited connectivity.
- *Method*: Designing and implementing a streamlined, adaptable blockchain client that addresses the unique challenges faced by constrained IoT devices.
- *Expected Outcome*: Demonstrating a practical, efficient blockchain integration approach for IoT devices with limited resources, enhancing network interoperability and security in more constrained environments.

These two phases of development represent a comprehensive approach to integrating blockchain technology across a range of IoT devices, from more powerful development boards to the more

constrained IoT environments, thereby expending the scope of blockchain's applicability in the IoT domain.
Building upon the integration of blockchain in IoT devices, the focus shifts to the exploration of multi-chain solutions, a growing field with lots of potential in the IoT sector.

Investigation of Multi-Chain Solutions for IoT

- *Objective*: To explore and assess the viability of multi-chain ecosystems in IoT contexts.
- *Method*: Implementation and evaluation of a multi-chain ecosystem, such as Polkadot, specifically in automotive IoT context.
- *Expected Outcome*: Providing valuable insights into the application of multi-chain solutions in IoT, potentially catalyzing their adoption in IoT-based industries.

Finally, the research aims to chart the future course, identifying potential challenges and opportunities in the rapidly evolving multi-chain technology landscape, especially in the context of IoT.

Future Challenges and Directions in Multi-Chain Technology for IoT

- *Objective*: To pinpoint and outline future research avenues and challenges associated with the application of multi-chain technology in IoT.
- *Method*: Analyzing the outcomes from implementing an automotive use case within a multi-chain framework.
- *Expected Outcome*: Establishing a foundation for future research in multi-chain technology, underscoring its importance and potential difficulties in the context of IoT environments.

These research objectives collectively aim to bridge the existing gap between blockchain and IoT, with a particular emphasis on their application in the automotive sector. The thesis endeavors to provide a comprehensive guide on the integration and potential of blockchain in IoT through detailed analysis and practical implementations.

1.5 Thesis Organization

The thesis is structured to provide a thorough examination of the integration and optimization of blockchain technology within the Internet of Things, with a particular focus on the automotive industry. It progresses from foundational concepts to specific implementations and future perspectives in a clear and systematic manner.

Chapter 2, "State of the Art", lays the groundwork by discussing the exponential growth in computing power, the emergence of IoT, and Decentralized Ledger Technology, with a focus on

blockchain. This chapter sets the context for understanding the potential and challenges of integrating IoT with blockchain technology.

Chapter 3, ”BIoT: Blockchains for IoT”, moves into the practical aspects of this integration, examining the strategic orientation needed to address fundamental challenges and to exploit new opportunities. It particularly addresses scalability, performance, interoperability, and energy consumption, centering on our automotive use case at a single-organizational level.

Chapter 4, ”BIoT: Practical Blockchain Client Integration on IoT”, goes into the technical details of implementing blockchain technology on IoT devices. It covers the necessity of off-chain integration for constrained devices and explores the components of a blockchain client, its implementation in our IoT context, the trade-offs and requirements for an effective system.

Chapter 5, ”Improving BIoT: Towards a Multi-Chain Ecosystem for IoT”, addresses the issue of heterogeneity among blockchain platforms and IoT devices. It introduces the Polkadot network and Substrate Framework as solutions for creating a standardized and interoperable blockchain ecosystem, aiming to show how a unified ecosystem can enhance the IoT domain. An interoperable blockchain ecosystem has the potential to create transparent, secure, and efficient communication between automotive services, IoT devices (vehicles) and blockchain platforms.

The thesis concludes with Chapter 6, ”Conclusion and Perspectives”, summarizing the main findings, contributions, and implications of the research. It reflects on the progress made in integrating blockchain with IoT, the challenges overcome, and suggests potential directions for future research in the BIoT domain.

This organization from basic concepts through to technical implementations and onto solutions for BIoT integration challenges provides a comprehensive overview. The automotive use case offers a practical approach through which these discussions are applied and evaluated, contributing to the academic field of blockchain and IoT integration.

2 State of the Art

2.1 Introduction

Since its apparition in 1965, Moore's Law has remarkably predicted the exponential growth of transistor density on microchips, driving the rapid advancement of computing power [21]. Over the years, the number of transistors per square millimeter has doubled approximately every two years, advancing technology from having 100 transistors per square millimeter in the 1970s to a staggering 121 million transistors per square millimeter in 2022.

Consequently, the declining manufacturing costs, due to mass production, of microchips have led to their integration into everyday objects, triggering a rise of the Internet of Things (IoT). In the present day, these interconnected "things" not only collect and exchange data but also interact with humans, creating a seamless network of devices. This new interconnected digital world allows for massive data collection, new forms of optimization, and automation without constant human intervention.

In parallel with the rise of IoT, this state of the art also explores the emerging domain of Decentralized Ledger Technology (DLT), particularly blockchain. DLT has emerged as a technology with the potential to revolutionize trust in the digital realm, transforming existing technological and organizational structures. By providing a decentralized and secure means of recording transactions, blockchain has paved the way for various innovative applications beyond this so-called cryptocurrencies transactions.

This chapter aims to provide an overview of the Internet of Things (IoT) and a comprehensive study of Decentralized Ledger Technology, focusing on blockchain technology. We will dive into the current state of both IoT and blockchain, exploring their foundational concepts and potential applications. Moreover, we will explore how blockchain technology can be integrated into the context of IoT, addressing the challenges that arise in terms of performance, security, governance, and other aspects.

In this state of the art, we will highlight the possible relationships between IoT and blockchain (BIoT), as well as the trade-offs that must be considered to harness their combined potential effectively. By understanding the landscape of these two domain cutting-edge technologies, we can lay the groundwork for exploring innovative solutions new opportunities and challenges. The novel multiple-blockchain (*multi-chain*) ecosystems will also provide a new path to improve BIoT thanks to interoperability between blockchains.

2.2 IoT: A Heterogeneous Ecosystem

2.2.1 IoT Ecosystem Architecture

The Internet of Things (IoT) is a heterogeneous ecosystem of interconnected objects containing computing, connectivity, and sensing capabilities. Based on various studies and reports, it is predicted that the number of connected IoT devices will continue to grow in the coming years, reaching an estimated 30 billion or more by 2025 [22] [23] [24]. IoT is a term that encompasses many types of device and used in several domains, such as home automation, vehicles, WiFi/network gateways, industrial automation, but are also devices related to the natural world, people and even animals (Figure 2.2.1).

The increasing adoption of IoT is supported by advancements in technology, including wireless and wired networks, cost-effective hardware, advanced analytics, machine learning, and artificial intelligence. However, many challenges are still present – such as management, costs, and cybersecurity – slowing down the large-scale IoT beyond pilot projects or Proof-Of-Concepts (PoC).

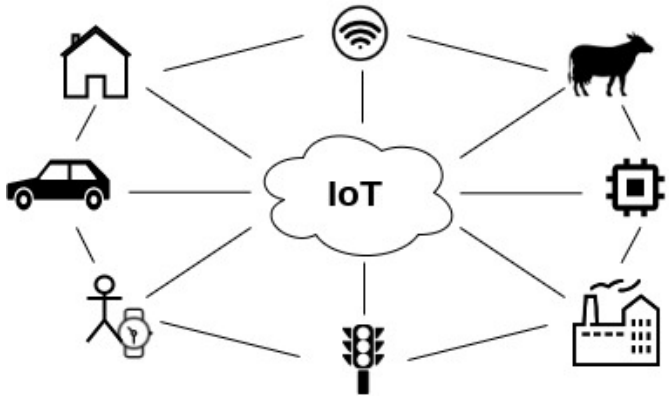


Figure 2.2.1: What are IoT devices ? An interconnected network of devices surrounding us.

The IoT ecosystem is a combination of devices, connectivity, data storage, and applications. The devices range from small micro-controllers embedding basic sensors and communication modules (e.g. device sending temperature in a room), to single board computers with powerful computation capabilities (e.g. the IoT in a vehicle sending collected sensor data from its surrounding).

2.2.1.1 IoT Network Protocols

To transmit collected data or receive data, IoTs are connected using various wireless networking protocols (e.g. WiFi, Bluetooth, Zigbee, Sigfox, LoRaWAN, 4G, 5G). Each protocol is designed to match a specific use case with its physical environment, and can be summarized in Table 2.2.1.

The huge amount of generated data have to be stored and processed. One possible approach is to use edge computing, which processes information closer to the edge of the network. This reduces the load on centralized servers, saves bandwidth, decreases latency, and enhances privacy. This is achieved through local processing on end-devices, nearby edge devices, or fog computing. Practically speaking this done for example by using pre-processing methods such

as real-time video analytics [25] or fog computing in LoRaWAN gateways using Single-Board-Computer (e.g. Raspberry Pi) [26].

2.2.1.2 IoT Network Architecture

There is globally three or four layers (Figure 2.2.2) in the IoT network architecture [32] [33]: the perception layer, network layer, middleware (i.e. processing) layer, and finally the application layer.
The perception layer are end-devices that sense their environment and transmit the data using the network layer to a middleware layer. The tasks are typically data collection, conversion (and local processing), device identification, and finally transmission.
The middleware layer is in charge of handling protocol communication packets conversion, followed by the data storage and processing. It also typically handles the devices management and configuration with additional security middleware.
The application layer defines the application or services that end-users can interacts with. This layer is the interface between the underlying hardware and software components of the system and the applications end-users which utilize the data or services provided by the system. The application layer functionalities are typically a user interface, data display, decision-making, automation, and custom solutions.

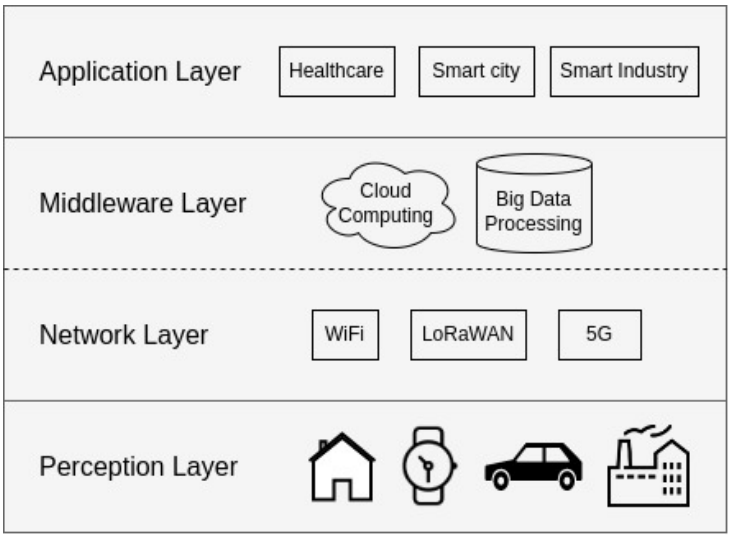


Figure 2.2.2: Four layers of the IoT network architecture (perception, network, middleware, and application layer)

2.2.2 IoT Devices Architecture

Various electronic architecture configurations are employed in the development of IoT devices. They can be broadly categorized into three distinct classes based on their computational power: Class A, Class B, and Class C. A summary of these classes are in Table 2.2.2, but keep in mind that each class is arbitrary defined, and inspired heavily on the compilation of the work of [34] [35] [36].

Class A: These devices prioritize ultra-low power consumption, often working with CPUs like the ARM Cortex-M0 or an 8bit microcontroller, which typically operate at clock frequencies in the range of tens to hundreds of MHz (e.g., 20-200 MHz). They typically have RAM capacities

Protocol	Characteristics	Applications	Power (mW)	Distance Range (meter)
Blue-tooth	Short-range wireless communication protocol, relatively high data rate [27].	Commonly used in wearables, healthcare devices, and home automation.	1-50	Up to 100m
Zigbee	Low-data rate, and close proximity (i.e., personal area) wireless ad hoc network [27].	Used in home automation, smart energy, and in medical devices.	10-100	Up to 100m
LoRaWAN	Long Range (LoRa) and low-power wireless protocol, designed for wide area networks (WANs) and operates in the sub-gigahertz frequency band [28].	IoT applications that require wireless battery operated things in regional, national or global network. Ex: smart city applications, agriculture, and supply chain tracking.	10-100	Up to 15-20km
Wi-Fi	High data rate, connection concurrency, security.	IoT applications that require high data rate and are not battery driven, like smart home devices and video surveillance [29].	50-100	Up to 100m
Sigfox	A global communications service provider for the IoT. It uses a unique technology that enables scalable and energy-efficient data transfer [30].	Global IoT, such as logistics and supply chain, agriculture, and smart city applications.	10-50	Up to 30km in rural areas
Thread	Secure and reliable mesh network with no single point of failure. It's designed for home automation application [31].	IoT devices in the connected home. It's been used in products like Google Nest.	10-100	Up to 100m indoors
5G	High data rate, high system capacity, and huge device connectivity.	Autonomous vehicles, smart cities, fully connected homes and buildings, advanced manufacturing, and evolved health services [3].	100-500	100m up to multiple Kms

Table 2.2.1: IoT wireless networking protocols, their applications, power rating, and usage distance range.

in the order of kilobytes (e.g., 32 KB). Due to their minimalistic design, their average current consumption could be as low as 100µA to 200µA when active, and even lower during sleep modes. With a sufficient capacity, such as a 500mAh battery, their life-cycle can extend to several years [37].

Class B: These devices have a greater computational power, often utilizing CPUs like the ARM Cortex-M3 or M4, which may operate at clock frequencies up to 1 GHz. Their RAM capacities could reach several megabytes (e.g., 1-2 MB). Their average current consumption could be in the range of 1mA to 10mA when active. To operate correctly, they may require energy harvesting techniques, like solar cells or piezoelectric generators, to compensate for the increased power demand. As they offer a balance between performance and power efficiency, devices like certain advanced STM32 models, some ATmega or ESP series microcontrollers fall into this category [38][29].

Class C: Devices in this category often resemble single-board computers with high computational power. These might be powered by processors such as the ARM Cortex-A series, which operate at multi-GHz frequencies. They come equipped with RAM capacities ranging from hundreds of megabytes to several gigabytes. Their current consumption is relatively higher, often lying in the range of 500mA to 2A or more when fully active. For example, devices like the Raspberry Pi or RISC-V boards are in this class. Their computational prowess makes them suitable for complex tasks, such as edge computing or real-time analytics, but they may require a regular or continuous power source [39].

The environmental condition of an IoT device is highly correlated to its power capacity (which is confirmed by Table 2.2.2 and Table 2.2.1). An IoT located in a harsh environment require a more optimized power consumption than an IoT located in a easy access location where battery swapping or main grid power access is pretty straightforward. Additionally, a device power consumption is correlated to its processing power. Based on efficient IoT energy study [40], the power consumption depends highly on:

- Data-type/bus-width optimization (best is to use data type $\leq$ bus-width)
  E.g. An 8-bit microcontroller processing an 64-bit integer isn't recommended.
- Dedicated hardware units usage (use most as possible)
  E.g. crypto IP module (Intellectual Property module), mathematic modules, ...
- The usage of the right sampling-rate, transmission-rate, and clock frequency.
  It is best to use fastest sampling-rate, transmission-rate, and clock frequency.
- IoT software optimization during compilation: fastest code or the smallest binary is not necessarily the most energy efficient.

In the study of Merabtane and Benabadji [37], a very low power photovoltaic panels data logger is realized using an 8-bit microcontroller with only serial communication, a Real Time-Clock-and-Calendar (RTCC), no external interface, and programmed using assembler code. System designed with such optimization can be classified in the Class A, and could be deployed using battery and environment energy harvesting techniques for very long durability.

As contrast, Jabbar et al. [41] smart home solution and design utilizes a ESP8266 board (32-bit Tensilica Xtensa 8MHz) with multiple I/O, interface and Wi-FI communication capability (to send data to a server). The ESP8266 board is known to draw up to 70-80mA of current when transmitting data [42], without any additional connected sensor modules. This work is similar to Chanthaphone Sisavath and Lasheng Yu [38] smart home gateway IoT implementation which used an NXP LPC1769 microcontroller (ARM Cortex-M3 120MHz based), *xbee* module

Parameter	Class A	Class B	Class C
Computational Power	Ultra-low	Moderate	High
Example CPUs	ARM Cortex-M0, STM32L-series, 8bit-microcontroller	ARM Cortex-M3 or M4, Some STM32L-series or STM32F-series	ARM Cortex-A series, BCM2837 (Raspberry Pi 3 B+)
CPU Frequency	Up to hundreds of MHz	Up to 1 GHz	Multi-GHz
RAM	Kilobytes (e.g., 32 KB)	Several megabytes (e.g., 1-2 MB)	Hundreds of megabytes to several gigabytes
Storage	Limited onboard EEPROM, small external flash storage	More extensive flash storage, SD card support	Onboard eMMC storage, SD card support, SSD support via USB interfaces
Current Consumption	100µA - 200µA	1mA - 100mA	500mA - 2A or more
Applications	Basic sensor nodes, wearable health monitors [37]	Smart home automation, industrial monitoring systems [38][29]	Video surveillance systems, IoT gateways, edge computing, real-time analytics [39]
Connectivity	Zigbee, LoRa, BLE	Wi-Fi, BLE, cellular	Wi-Fi, Ethernet, cellular
Environmental	Designed for low power, specialized versions for harsher conditions	Outdoor usage, some models offer waterproofing or resistance against contaminants	Controlled environments, rugged versions with heatsinks or fan-based cooling for industrial or outdoor usage
Security Features	Basic security features	Enhanced security with hardware-based encryption or secure boot	Advanced security with secure chipsets, encrypted storage, and sophisticated OS-level protections

Table 2.2.2: Overview of IoT architecture classes based on their computational power

(Zigbee communication), an *wifibee* module (Wi-Fi communication, 32-bit processor), and a flash storage module (W25Q18FV chip) for webpage data. The overall gateway would require more than 100mA of current or more depending on the device features, thus using external or main power grid due to its energy consumption. The Class B or Class C can be associated with such devices.

In video analysis and real-time detection situations the system architecture would typically require high data processing [39] and a Class C can be attributed. This is translated in using a System-On-Module (SoM) such as Raspberry Pi boards with a System-on-Chip (SoC) like the BCM2837 (containing 4 ARM Cortex A53 CPUs). These IoT devices are often accompanied with an Operating System (OS), where programs can run concurrently, deploy specific device drivers, using shared libraries and other standard applications.

2.2.2.1 Why Has IoT So Much Success?

IoT is part of the *4th Industrial Revolution*. A little bit of history shows us that the *1st Industrial Revolution* (around 1780s) is the mechanization of the world where came the combustion engine (automobile and planes) [43]. Than, came the *2nd Industrial Revolution*, around the 1870s, and is about the electrification, creation of assembly lines and mass production [44]. This was followed, in the 1960-1990s, by automation with the rise of electronics, telecommunication and computers: The *3rd Industrial Revolution*, also called a *Digital Revolution* [45]. The control of nuclear energy in the late 1900s ended this 3rd revolution. Next, starting in the year 2000s, is to the *Information Age* that we are now living in [46]. This last industrial revolution encompasses advancements in artificial intelligence, robotics, automation, biotechnology, and IoT. IoT provides a link or even an integration, of digital technology into the physical world. The industry and machines all around us are customized and personalized for each individuals.

Enhancing connectivity and interoperability between systems will not only allow to accelerate productivity and increase efficiency, but also reduces costs and creates new business opportunities. By 2030 the world would have tripled the number of connected devices in 10 years (10 billion in 2020). The economic ($5.5 trillion to $12.6 trillion in value globally [47]) and social (positive impact on society using IoT targeting only specifics needs, IoT changes aren't destructive, and a progress in the living standards [48]) impact is already, and will be, incredibly significant. The widespread adoption of IoT has the potential to revolutionize industries across the globe, transforming the way we live, work, and interact with the world around us.

2.2.3 Current Trends in IoT

IoTs are constantly evolving thanks to new technologies reducing computing cost, increasing of the battery capacities, and optimization of data transmission and processing.

Edge computing: As the number of IoT devices continues to increase, so does the volume of data they produce. Edge computing reduces bandwidth and latency needs by processing data closer to where it's generated instead of transmitting it to a central location or cloud for processing. This is especially important for applications that require real-time data processing. Edge computing is characterized in terms of high bandwidth, ultra-low latency, and real-time access to the network information that can be used by several applications [49].

Artificial Intelligence (AI) and Machine Learning (ML): are becoming increasingly integrated into IoT devices, enabling more advanced data analysis and decision-making capabilities. This can help to predict trends, reduce network load, improve operational efficiency, and personalize

user experiences. The AI trend is rapidly followed by an increase of interest to embed as best as possible AI and ML into the IoT domain using various optimization or new hardware specific architecture models [50] [51] [52] [53]. AI impacts edge computing, 5G connectivity and many other related IoT domains as it can fit often a multitude of use cases.

5G Connectivity: The roll-out of 5G networks will greatly enhance the performance of IoT devices. 5G provides faster data speeds, lower latency, and the ability to connect a lot more devices at once, which is key for large-scale IoT deployments [3] [4].

IoT in Healthcare: IoT is increasingly being used in the healthcare sector for remote monitoring, telemedicine, and tracking patients' health [54] [55]. This trend has been accelerated by the COVID-19 pandemic, which has highlighted the benefits of remote healthcare solutions [56].

Industrial IoT (IIoT): Industrial sectors are leveraging IoT technology to improve operations and efficiency. The 4th Industrial Revolution, synonym of *Industry 4.0*, arises from the merge of IoT and Cyber-Physical Systems (CPSs), a new approach that uses large-scale Machine-to-Machine and IoT with increased automation, communication and monitoring for a more productive future [57] [58]. The IIoT domain includes predictive maintenance of machinery, real-time monitoring of supply chains, and automation of various processes.

Blockchain for IoT (BIoT): Blockchain technology is being increasingly used in IoT to secure data sharing, device identity verification, and to maintain the integrity of the device and data history. Many studies and surveys on specific IoT trends (IIoT, 5G, smart grid, …) discuss the possible integration of blockchain in the domain [59] [20] [60] [61].

As we have seen, IoT domain comprises many different devices with a very heterogeneous system architecture. Based on recent research for BIoT, this thesis follows a continuum in providing an overview of the possible implementations of blockchain with IoT using different devices and novel blockchain technologies.

2.3 Blockchain

2.3.1 The Rise of Blockchain

As illustrated in the timeline Figure 2.3.1, decentralization is not a new concept. It emerged in the 1980s following centralized computing mainframes, enabled by exponential advancements in technology that reduced transistor size (Moore's Law), amplified computational power, and decreased component costs, which led to democratizing home computers. Decentralization was initially introduced as a solution to the issues posed by mainframes. Its goals include distributing computational power, reducing costs, diversifying systems, and enabling local (individual) participation. This approach can reduce communication congestion (towards a mainframe), enhance security, and attempt to improve the overall management of the computer ecosystem.

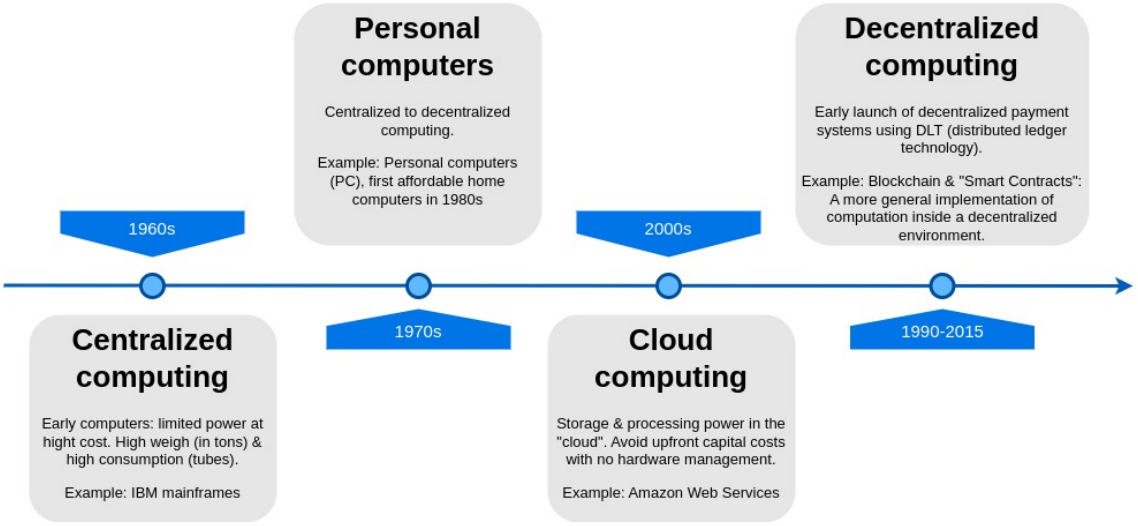


Figure 2.3.1: Computer and data processing evolution from 1960s to today: IBM Mainframe System/360 in 1963 [62], Altair 8800 in 1974 [63] and Commodore 64 in 1982 [64], AWS cloud computing in 2006 [65] and Windows Azure in 2010 [66], BitTorrent in 2001 [67] and Bitcoin in 2008 [68].

After 2005, desktop computers, servers, and embedded devices became capable of interconnectivity through any network. Mobile phone antenna progressed from kilobits per second throughput (2G) to multi-megabits per second connectivity (3G), further augmented by the 4G generation. Servers worldwide experienced a similar evolution after the introduction of commercial fiber cables [69]. This increase in internet speed between connected countries, led continents being closer to each other more than ever before. This reduce in latency and amplified speed, enabled implementation of novel applications, ideas [70], and positive social-economic impact [71].

These innovative concepts could now incorporate instantaneous information exchange using the internet. Fast data exchange with real-time properties are crucial in decentralized systems, such as large databases, that must keep data up-to-date with rest of the network. This necessitates critical components for "real-time synchronization" and agreements among each databases in the network.

As the speed and reliability of internet connections increased, personal computers started to

be gradually replaced by *cloud* solutions. Today, a limited number of organizations are managing cloud infrastructures (e.g. Amazon Web Services, Google Cloud Platform, Microsoft Azure...), which, politically and economically, constitutes a centralized system. In the present day, decentralized computing and the concept of blockchain aims to restore trust, economic fairness, and stability through decentralization and the establishment of universally pre-established consensus rules.

2.3.2 To Centralize or to Decentralize ?

When trying to exchange information or value, you can either handle the task yourself, or use participant(s) to execute that task. In the last case, you will have primarily two choices: use one trusted third party, or trust all participants which are following the same rule. In other words, the trust third party is abstracted by the rule (i.e. algorithm) each participants rely on.

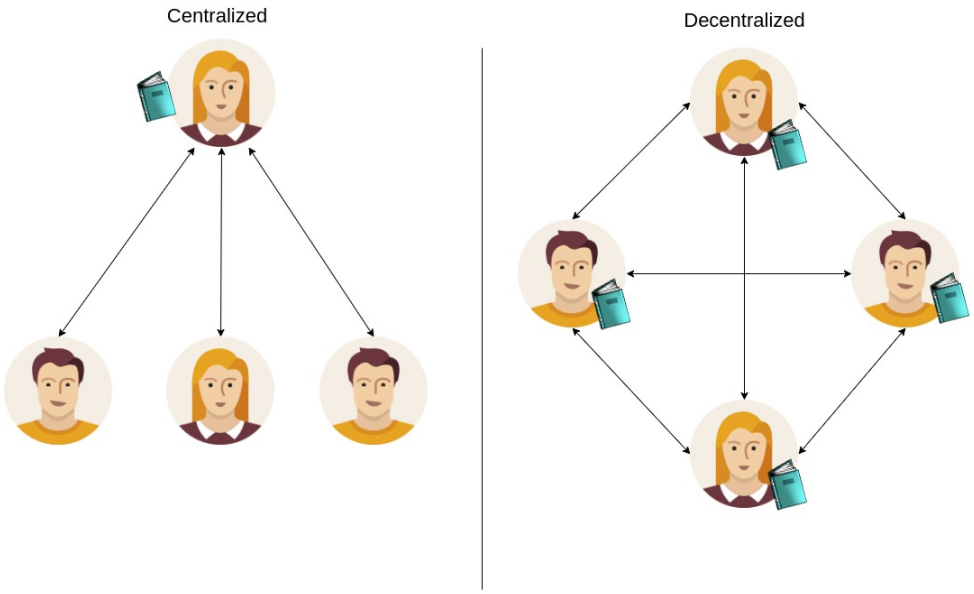


Figure 2.3.2: Difference between centralized and decentralized information

Lets imagine a data ledger containing some asset transactions, depicted in Figure 2.3.2, on the left, a centralized authority has the trusted data ledger and all other participants has to rely on this centralized authority. In contrast, on the right, all the participants have a version of the data ledger and have to cooperate together to agree on the ledger new data (that will then be distributed among participants, thus stored in a decentralized manner).
But questions come to mind when using a decentralized system: *how is the data stored securely in the ledger?* And *how can all participants agree and synchronize the ledger data?* Blockchain comes to the rescue...

2.3.3 What is Blockchain?

Distributed Ledger Technology (DLT) is a concept that refers to a digital system for recording the transaction (e.g. of assets) in multiple places at the same time. Blockchain is one particular distributed ledger with a unique structure made using a chain of cryptographically linked blocks. Unlike traditional databases, a blockchain has no central data store or administration functionality,

making it a shared, distributed, and synchronized digital database spread across multiple sites, countries, or institutions [72].

One of the primary motivations behind the development of blockchain is the **removal of trusted intermediaries and central authorities** to improve the efficiency, security, and transparency of data systems. A blockchain provides a way to record, manage, and execute digital interactions with transparency and auditability. This has been particularly influential in the financial sector, where transaction opacity and intermediaries are important.

The idea of digital cash and electronic payment is old and can be traced back to David Chaum's paper "Blind Signatures for Untraceable Payments" [73] in 1983. However, the first formal concerns of a real possibility to create a functional digital currency appeared in 1996 when Laurie Law, Susan Sabett, and Jerry Solinas [74] published "How to make a mint: the cryptography of anonymous electronic cash". The authors presented core concepts of electronic currencies: prevent multiple spending, the cryptography used for a secure system, and the basics features of such system (transferability, divisibility).

Later in 2008, *Bitcoin* was proposed by anonymous author(s) Satoshi Nakamoto [68], and is the first successful blockchain capable of managing a peer-to-peer electronic cash system. It provide a pseudonymous, decentralized, immutable, and secure digital currency solution, potentially creating a new economy. It introduced the practical implementation of a blockchain: a ledger in which transactions are grouped into blocks and each block is interconnected through a unique identifier of the previous block, forming a chain-like structure.

Bitcoin could be seen as a decentralized computing machine that only process its native currency. The natural evolution of blockchain was to expend its computational capabilities to run custom applications. In 2014, Vitalik Buterin introduce the concept of smart contract within a new blockchain called *Ethereum* [75]. Ethereum implements a layer within the blockchain capable of executing custom business logic.

The concept of smart contracts was first introduced by Nick Szabo in 1997 [76] and is discussed further in Section 2.3.9. The smart contracts innovation implemented in Ethereum started the era of Blockchain 2.0, where custom programs could be executed inside the blockchain, allowing the creation of custom application using blockchain and smart contracts as its core logic, called a **decentralized Application** (dApp). In other words, a blockchain with smart contracts capabilities can be visualized as a distributed, decentralized, trustless computing machine [77].

In 2018, a third evolution of blockchain (Blockchain 3.0) comes forward to solves interoperability and scalability issues (Figure 2.3.3). Blockchain 3.0 is also about the rise of dApp and increased adoption of the technology [78].

In upcoming years, *Blockchain 4.0* technology is expected to be business-usable, compatible with the next Industry 4.0, and implement Artificial Intelligence (AI) concepts [79] [61].

2.3.4 Blockchain Architecture

The most common data structure is a chain of blocks – a blockchain. However, each blockchain has their own data requirements and global structure. They are generally summarized in 3 blockchain components [80]:

- The blocks: contains transactions, consensus configuration, protocol data, and any other information about the blockchain at the given time. A block is a way to organize data in a distributed ledger. Blocks are linked to the previous block using its hash (Figure 2.3.4).

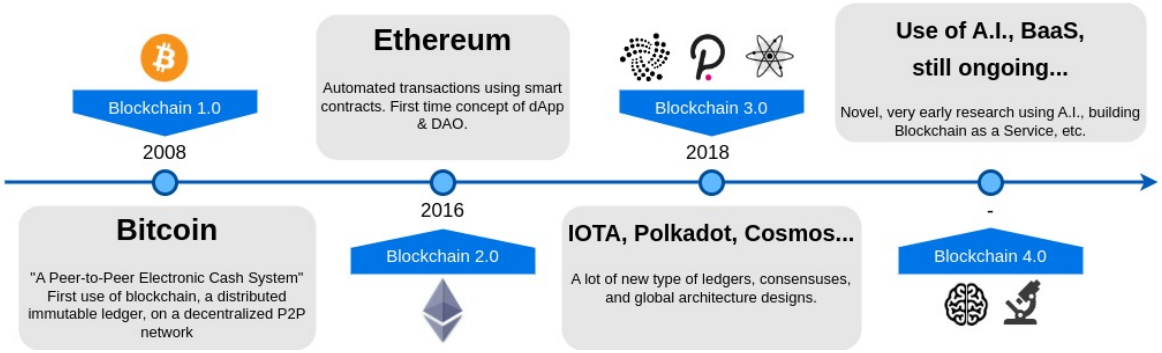


Figure 2.3.3: Blockchain technology evolution from 2008 to today

- The transactions: are instructions to modify the blockchain state (e.g. Alice sends 10 coins to Bob). Transactions are encapsulated in the blocks and thus secured with the same hashing mechanism as blocks.
- The consensus mechanism: is a protocol designed to achieve agreement among nodes on the validity of transactions and the order in which they are added to the blockchain. It allows to agree on the current ledger state.

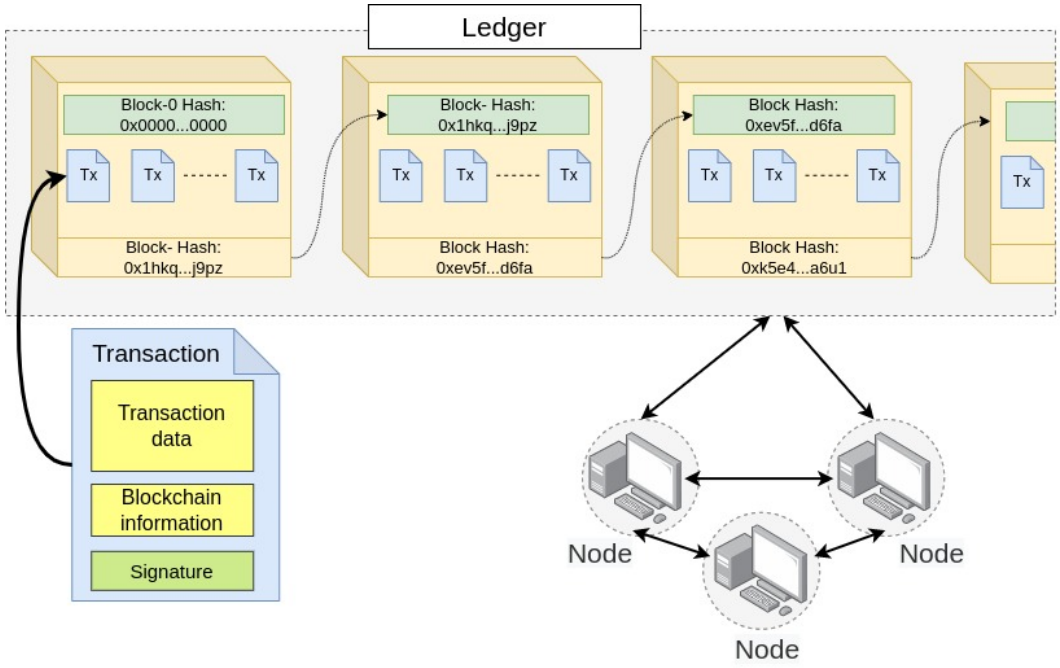


Figure 2.3.4: Blockchain data structure. A shared ledger containing a chain of blocks that encapsulates transactions

The *blockchain state* is the representation of **all** the stored data within the blockchain network at a given time (i.e. at a given block number). The state differs from blocks: the chain of blocks contains raw information and the blockchain state represent the collective information and status of every transaction, account balance, and any other ledger data at a given time. The state is the most analogous concept related to ledger, all useful information can be found in the state, it is the database of the blockchain.

Depicted also in Figure 2.3.4, every time a new block is created, a hash of the previous block

is included in the chain. The chain of blocks is entirely secured because modifying a previous block would break the chain of blocks.

An alternative of blockchain is to use Directed Acyclic Graphs to create an other DLT with a different data structure and consensus. The Tangle [81] is the name describing IOTA Directed Acyclic Graph (DAG) transaction settlement and data integrity layer. It is the first layer designing the structure that holds IOTA system together. The Tangle algorithm is based on the reputation of nodes: a transaction is validated by validating two previous transaction of two different nodes. Depicted in Figure 2.3.5, a transaction is not only linked to one previous transaction, but is linked an confirmed by multiple transactions (e.g. transaction n°2 is confirmed by n°5 and 6, transaction n°6 is a tip of the DAG). The key difference between blockchain and DAG based DLTs is the potential for parallelism. A blockchain has a ground-truth in a chain-like model, and a DAG has multiple paths to certify the validity of a transaction.

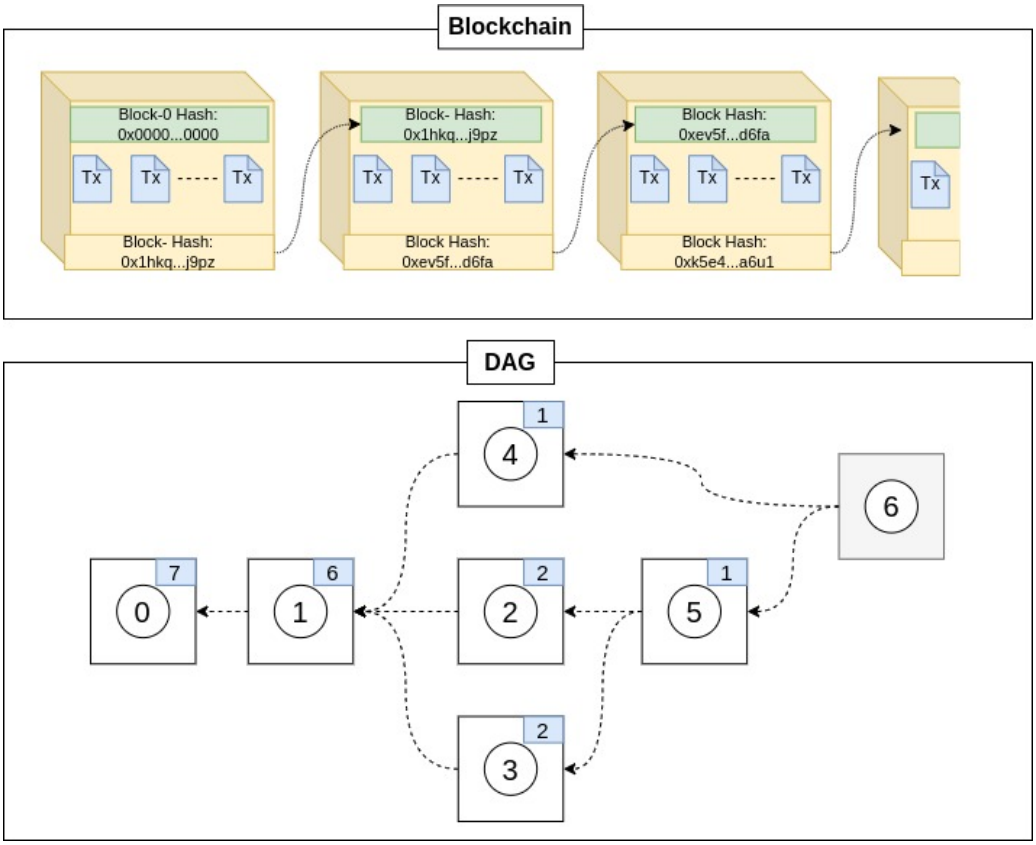


Figure 2.3.5: Blockchain is a synchronous chain of blocks. In a Directed Acyclic Graph (DAG) structure, transactions are confirmed/linked with edges by new other transactions that are yet not confirmed (tip), creating a reputation and allowing parallel transaction finalization.

One of the latest alternative to blockchain structure is hashgraph: The hashgraph technology is patented and used by the public ledger Hedera [82]. Hashgraph has its own consensus and works with a gossip protocol where the gossip history is represented as a directed graph. Hedera claims a throughput, up to 500.000 transactions per second and to be limited mainly by the network bandwidth but has to still be verified by independent research.

Important notice: For the rest of the thesis, we will only use blockchain-based DLTs. DAG-based DLTs offer many advantages, but at the time of writing doesn't offer stable smart contract

implementation, which is an absolute component requirement of my research around BIoT.

2.3.5 Types of Blockchains

As this stage of the thesis, you may understand that blockchains comes in many shape, architectures and designs. There are several types of blockchains, each with its own unique characteristics and architecture. The common types and category of blockchains are:

- Public Blockchains: These are open and permissionless blockchains where anyone can participate, validate transactions, and contribute to the network. Bitcoin and Ethereum are examples of public blockchains [68] [75].
- Private Blockchains: Also known as **permissioned blockchains**, are restricted-access networks where participants must obtain permission to join and perform transactions. Private blockchains are typically used within organizations to maintain privacy and control over the network. Hyperledger Fabric, Hyperledger Sawtooth, Hyperledger Besu, and using a private configuration of the official Ethereum client Geth, are popular private blockchain platforms [83] [84] [85] [86].
- Consortium Blockchains: These blockchains are governed by a consortium or a group of organizations that collaborate and share control over the network. Consortium blockchains are often used in industries where multiple organizations need to work together, such as supply chain management [87] or banking. These types of blockchains are similar platforms as private blockchains but will allow multiple organizations to participate to block creation in the consensus.
- Hybrid Blockchains: These blockchains combine the features of both public and private blockchains. They allow for a public-facing network while maintaining certain restrictions on access and governance. Hybrid blockchains provide a balance between transparency and privacy. XinFin [88], for example, uses a hybrid network architecture combining Ethereum and a private Quorum chain.

In the realm of blockchain technology, public blockchains are characterized by their openness and transparency, whereas permissionless blockchains emphasize the absence of access control, allowing anyone to join and actively participate in the network's consensus mechanism. Public blockchains can exist in either permissionless or permissioned forms, whereas permissionless blockchains are always public by nature.

Conversely, private blockchains are defined by their restricted-access environment, limiting participation to a closed group of participants. On the other hand, permissioned blockchains emphasis access control, necessitating permission from a central authority or administrator in order to join the network. Private blockchains are always permissioned, while permissioned blockchains can be either private or public.

Important notice: The use case and context of this thesis sets the requirements of the blockchain characteristics into using private permissioned blockchains (more details on the use case in Section 1.2.3). Thus, this thesis will explore in the next chapters the usage of private permissioned blockchains.

2.3.6 Blockchain Fault Tolerance

We present here, two key concepts used in the domain of private and consortium blockchain consensuses: Byzantine Fault Tolerance and Crash Fault Tolerance.

2.3.6.1 Byzantine Fault Tolerance (BFT)

Byzantine Fault Tolerance (BFT), in the context of DLTs, is a property of a distributed computing system that enables it to reach consensus despite the presence of malicious nodes, which may act contrary to the protocol specifications. BFT has been initially explained by Leaslie Lamport, Robert Shostask, and Marshall Pease [89] in 1982, with the "Byzantine Generals Problem" and contained practical implementation details. This concept is particularly relevant in the context of blockchain technology, where decentralization and security are paramount. BFT mechanisms are critical in ensuring that blockchain networks function correctly even when some participants are unreliable or actively trying to subvert the network. These algorithms are designed to provide security guarantees in the face of adversarial conditions, making them suitable for high-stakes environments where trust is decentralized.

In private blockchains, BFT takes on an even more critical role due to the restricted nature of these networks. Private, or permissioned, blockchains operate on the assumption that access to the network is limited to certain verified participants. This controlled access model presupposes a certain level of trust and known identities, which contrasts with the anonymous and open-access nature of public blockchains. However, the risk of Byzantine behavior remains pertinent as insider threats or network breaches can still cause substantial harm. BFT in private blockchains is tailored to mitigate such risks by implementing consensus protocols that are less energy-intensive and more efficient than those used in public networks, such as PBFT or its variants. The efficiency of BFT in private blockchains is paramount, as these are often designed to support high transaction throughputs for enterprise use cases.

2.3.6.2 Byzantine Fault Tolerance (BFT) vs Crash Fault Tolerance (CFT)

In contrast to BFT, Crash Fault Tolerance (CFT) addresses a different aspect of system resilience. While BFT is designed to handle both malicious and arbitrary failures, CFT specifically targets scenarios where system components fail due to crashes or non-malicious reasons [90]. In CFT, the primary concern is ensuring that the system continues to operate even when some of its components cease to function correctly. This form of fault tolerance is often simpler than BFT as it does not need to account for potentially deceitful or malicious behavior within the system.

Public blockchains, which are open and decentralized, inherently face a higher risk of malicious behavior due to their anonymous nature. This makes BFT a necessary feature for these blockchains to ensure security and reliability in an environment where participants cannot be trusted. On the other hand, private or permissioned blockchains, which operate in a more controlled environment with vetted participants, often find CFT sufficient. In these blockchains, the primary concern is not malicious behavior but rather the reliability and stability of the network. CFT provides an adequate level of security for these environments, where the likelihood of Byzantine faults is substantially lower.

Furthermore, the choice between BFT and CFT impacts the system's performance and complexity [91]. BFT algorithms, such as PBFT, are generally more complex and require more computational resources. This is due to the need for intricate mechanisms to identify and mitigate the

effects of malicious nodes. In contrast, CFT algorithms are typically less resource-intensive, as they do not need to contend with the possibility of deceitful participants. This difference often leads to CFT being more suitable for environments where efficiency and speed are prioritized, and the threat of malicious actors is minimal.

2.3.7 Blockchain Consensus

The consensus ensures an agreement between all the nodes (also called peers) in the blockchain network. Behind the term 'consensus' lies an algorithm. Each algorithm differs from blockchain to blockchain. The consensus rule is a subjective term, prone to misleading conclusions, because it can be implemented with different consensus algorithms. For example, the consensus rule Proof of Authority can exists in different algorithm such as "Aura", "Clique", "IBFT", "DPoA", and "Tendermint" [92] [93] [94] [95]. This distinction also applies to other consensus rules, such as Proof of Stake (PoS) and Proof of Work (PoW).

The purpose of consensus in a blockchain network is to secure the global Distributed Ledger Technology (DLT) state. It ensures a common agreement among nodes and is designed to tolerate a certain level of malicious or faulty nodes.

Jeff Nijsse and Alan Litchfield propose a state diagram for a distributed computing system such as blockchain [96]. Figure 2.3.6 shows that there are two possible states i) nodes propose a new block, ii) nodes agree on a block. The consensus has to answer **who** create the new block proposal and **how** nodes in the blockchain agree on the new accepted/posted block.

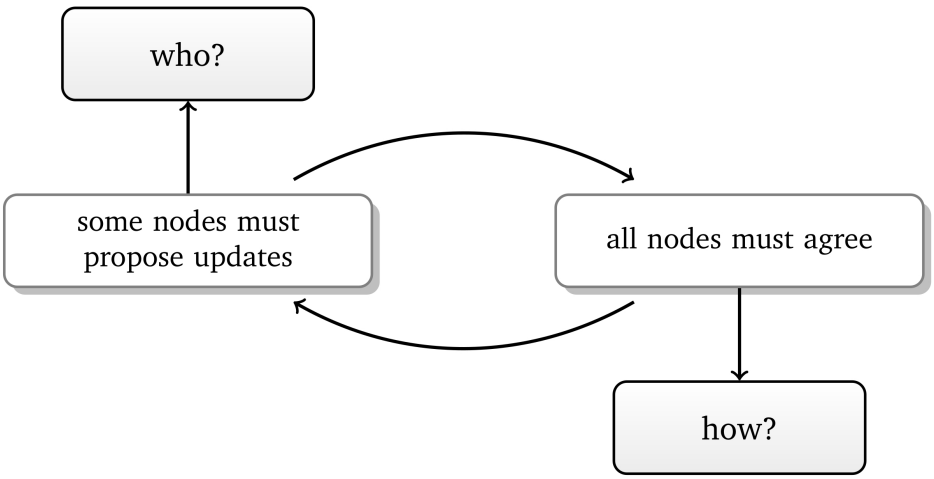


Figure 2.3.6: A state diagram for a distributed computing system including a blockchain. A consensus algorithm answers the questions of how? and who? [96]

As a result, we see that a consensus is a central pillar when designing a blockchain. In public blockchains, there are commonly two type of consensuses used – PoW and PoS – which provide a probabilistic agreement on the posted block (i.e data is confirmed with increasing probability as more blocks are added in the blockchain).
In commonly used private and consortium blockchains consensuses such as PBFT and PoA, the agreement is deterministic: it guarantees that data is written and committed to the blockchain as soon as the block is posted. While there are many variations under different names, we will only focus on the ones listed in sections bellow, which have a relevant use in this thesis.

ℹ In the context of blockchain, *finality* refers to the point at which a transaction or block becomes irrevocably confirmed and permanently recorded on the blockchain (i.e. immutable). The term finality is a property of the blockchain and can be – but not always – inherited from the consensus algorithm. In probabilistic consensuses, the finality is possible using finality gadgets, such as Casper [97], that creates immutability checkpoints. In deterministic consensus, finality is direct (i.e. direct-finality) when blocks are published.

2.3.7.1 Proof of Work (PoW)

PoW consensus mechanisms are designed to ensure that a blockchain node has consumed a sufficient amount of energy or computational work to add a new block to the network [**?**]. In PoW DLTs, miners are nodes performing the computational work to get rewarded in return (the mining concept is close to real world mining for precious metals). The difficulty level for adding blocks is adjusted based on the total computational power of the network, aiming to maintain a consistent block time. PoW was proposed in 1997 and formally presented in 2002 by Adam Back's paper "Hashcash - A Denial of Service Counter-Measure" [98]. This computational effort is what helps secure the network and prevent malicious actors from easily altering the blockchain's history.

With great hashing power comes great responsibility: When an imbalance occurs in the computational power of the Bitcoin network, with a single entity or colluding group controlling more than 51% of the total power, it poses a significant risk. Such a situation enables the controlling actor to produce blocks containing malicious information, and since they hold the majority of computational power, other nodes in the network are likely to accept these malicious blocks as valid.

2.3.7.2 Proof of Stake (PoS)

PoS consensus mechanisms are designed to validate and add new blocks to a blockchain based on the ownership or "stake" of participants in the network [99]. Instead of relying on computational power, PoS assigns the right to create blocks and validate transactions to participants who hold a certain amount of the native cryptocurrency or have a stake in the network. In PoS DLTs the nodes participating in the consensus are called validators.

However, a potential challenge in PoS is the "nothing at stake" problem. It refers to the risk of participants attempting to create multiple forks of the blockchain since there is no cost associated with doing so. To mitigate this, various PoS implementations employ mechanisms such as slashing penalties and the requirement to lock up a certain amount of cryptocurrency as collateral.

PoS and PoW are probabilistic consensuses. PoS can be combined with finality gadget such as Casper [97] to introduce economic finality into the blockchain. More details on the comparison of PoW and PoS block creation flow in Appendix 8.1.1.

2.3.7.3 Proof of Elapsed Time (PoET)

PoET was developed by Intel and first implemented by Hyperledger Sawtooth [100], is an energy-efficient consensus mechanism for permissioned blockchain networks. Utilizing Intel's Software Guard Extensions (SGX) for secure, random wait times generated by SGX's trusted execution environment, PoET ensures fair block production. This Nakamoto-style consensus

differs from Proof of Work as it can create consensus without high energy cost. Nodes compete to create blocks based on the shortest wait time, verified through SGX, fostering an equitable and scalable network. This approach allows broader participation with minimal resource use, enhancing security and efficiency in consortium blockchains where trust is prioritized.

2.3.7.4 Practical Byzantine Fault Tolerance (PBFT)

There are many implementation of BFT consensuses, but the most common is PBFT (implemented by Hyperledger) [100] and Tendermint [95]. Miguel Castro and Barbara Liskov created PBFT [101] as a replication algorithm that is able to tolerate Byzantine faults. PBFT guarantees to withstand and correctly reach consensus in presence of $N/3$ failures of any kind (N being the number of participants).

PBFT by Hyperledger is a deterministic consensus algorithm that focuses on fault tolerance and is often used in consortium DLTs. It requires a predefined set of trusted nodes and achieves consensus through multiple rounds of voting and message exchanges.

Tendermint (the Cosmos Network version) is a consensus engine that combines elements of PBFT and PoS. It uses a round-based voting process, where a set of validators takes turns proposing blocks and voting on their validity. Tendermint is commonly used in consortium DLTs, offering fast finality and Byzantine fault tolerance.

2.3.7.5 Raft Consensus

Raft [94] is a consensus algorithm mostly recognized for its simplicity and understandability, primarily used in distributed systems to manage a replicated log. Developed as an alternative to Paxos [102], Raft breaks down the consensus problem into three relatively independent subproblems: leader election, log replication, and safety. It ensures that each of these subproblems is addressed in a way that is both easy to understand and implement.

In the Raft consensus process, the nodes in a network are divided into followers, candidates, and a leader. The leader is responsible for managing the log replication across the network. If followers do not hear from the leader in a given timeframe, they can become candidates and initiate a new leader election process. The design of Raft ensures that there is only one leader at any given time, and the log entries are replicated in a consistent order across all nodes. Its simplicity, compared to other consensus algorithms, has led to its widespread adoption in various distributed systems (e.g. database and blockchain [103]). Raft is a Crash Fault Tolerant (CFT) system: It assumes that nodes may fail by crashing, which is a less hostile model compared to Byzantine Fault Tolerant (BFT) systems like PBFT. Therefore, while Raft is highly effective in environments where nodes are reliable and the possibility of malicious behavior is low, it may not be suitable for systems that require protection against Byzantine faults, such as in certain blockchain applications.

Raft vs. PBFT consensus: They are both designed to address different kinds of fault tolerance (see Section 2.3.6.2). Raft is a consensus algorithm that falls under the category of Crash Fault Tolerance (CFT). It is primarily designed to ensure consistent replication of logs and maintain a unified system state when nodes fail or crash without exhibiting malicious behavior. On the other hand, Practical Byzantine Fault Tolerance (PBFT), as a BFT system, is designed to function correctly even in the presence of malicious nodes. PBFT can handle not only crashes but also arbitrary malicious behaviors, making it well-suited for environments where trust among participants cannot be guaranteed. PBFT's complexity, however, tends to be higher compared

to Raft, as it must process a larger number of messages to achieve consensus and ensure security against Byzantine faults. This makes PBFT particularly relevant in consortium blockchain networks, where the risk of malicious participants could exist.

2.3.7.6 Proof of Authority (PoA)

Similarly to PBFT, PoA exists in many algorithms ("Aura", "Clique", "IBFT", "DPoA", ...) [92] [93]. Some key elements of PoA-based algorithms are:

- Predefined Validators: PoA algorithms rely on a fixed set of preselected validators or authorities. These validators are typically known and trusted entities within the network. They have the authority to create and validate blocks.
- Block Production Rotation: PoA algorithms often employ a rotation mechanism to determine which validator is responsible for block production at a given time. This rotation ensures fairness and prevents any single validator from monopolizing block creation.
- Identity-Based Consensus: Instead of relying on computational work or stake, PoA algorithms base consensus on the identities of the validators. Validators are typically identified by their unique addresses or cryptographic keys. Consensus is reached by validators signing and validating blocks using their identities.
- Block Validation by Validators: Validators in PoA algorithms independently validate blocks proposed by other validators. They ensure that the proposed blocks adhere to the network's rules and protocol. Validators perform checks such as verifying transactions and confirming block consistency.
- Finality Mechanisms: Some PoA algorithms incorporate specific finality mechanisms to guarantee the immutability of blocks [104].
- Efficient and Scalable: PoA algorithms are often chosen for their efficiency and scalability. They do not require extensive computational power or energy consumption, making them more lightweight compared to traditional Proof of Work (PoW) algorithms.

2.3.7.6.1 AURA (Authority Round) is a consensus algorithm used in the Parity Substrate framework and specifically implemented in the Polkadot network [92]. It is a variation of the PoA consensus mechanism that aims to provide a more decentralized and secure network. It operates under the assumption that all network authorities are synchronized. Authorities manage two local queues: one for transactions and another for pending blocks. Transactions issued across the network are gathered in the transaction queue. During each time step, the designated leader compiles these transactions into a block and broadcasts it to other authorities for round block acceptance. Each authority then circulates the received block among themselves. A block is added to the block queue if all authorities receive and accept the same block. Blocks from non-leader authorities are rejected.

2.3.7.6.2 Clique Clique [105] is a Proof of Authority (PoA) algorithm used in Geth [85], a GoLang-based Ethereum client. It operates in epochs marked by a sequence of committed blocks, with each new epoch beginning with a transition block that identifies the network's authorities. This block helps new authorities synchronize with the blockchain.

The algorithm calculates the current step and leader based on the block number and authority count. In each step, the leader issues a block that is immediately added to the chain by all authorities. While other authorities can also propose blocks, the likelihood of forks is reduced (but not absent) since non-leader blocks are delayed randomly, making the leader's block usually the first to be accepted.

2.3.7.6.3 CUBA Competing Utilitarian Byzantine Agreement (CUBA), made by Cyril Naves [106], is a consensus mechanism tailored for consortium blockchain networks, emphasizing scalability, performance, failure resistance, and security. It was developed in parallel of this thesis and is part of the SIM project. CUBA incorporates a blend of philosophical principles and technical strategies to enhance network efficiency and integrity. It initiates by distributing transactions to dynamic quorums-groups of nodes organized by their contributions, reflected in utilitarian scores. Nodes within these quorums undergo a structured consensus process to agree on partial blocks through proposal, voting, and finalization stages. Subsequently, an inter-quorum phase integrates these partial blocks into complete blocks, validated and agreed upon network-wide. Nodes' behaviors and contributions are continually measured, updating their scores and influencing quorum reorganization. This adaptive mechanism ensures CUBA optimizes network performance and security, promoting a cooperative environment that dynamically adjusts to changes in node behavior and network conditions, maintaining the resilience and efficiency of the consortium blockchain.

2.3.7.7 Delegated Proofs

To tackle scalability concerns, certain consensus mechanisms utilize a delegated approach, such as Delegated Proof of Authority (DPoA) [107], Delegated Proof of Stake (DPoS) [108], and Delegated Byzantine Fault Tolerance (dBFT) [109]. These mechanisms delegate the responsibility of block validation to a limited group of delegates. In the realm of politics, delegation is a fundamental aspect of representative democracies where citizens elect/votes representatives to make decisions on their behalf. This form of delegation is prevalent in numerous democratic countries around the world. The risks regarding delegation in DLTs and politics are similar: centralization, corruption and influence, representation and diversity, and last, trust.

In addition to addressing scalability concerns, these delegated consensus mechanisms offer several advantages. Firstly, by entrusting block validation to a limited set of delegates, these mechanisms can achieve higher transaction throughput and processing speeds compared to traditional consensus protocols.
However, the use of delegated mechanisms can also enhance network security. By selecting a trusted group of delegates (verified human or organization), the consensus protocol can mitigate the risk of malicious activities and prevent potential attacks. This approach reduces the likelihood of network disruptions, ensuring a more stable and reliable blockchain network.

Furthermore, delegated consensus mechanisms enable more effective governance and decision-making within the network. The limited set of delegates can collaborate and make consensus-related decisions more efficiently, leading to smoother protocol upgrades and the implementation of necessary changes.

It is clear that DPoS presents an intriguing proposition for blockchain implementations in IoT systems, such as the automotive sector. Its scalability and energy efficiency are advantageous, aligning with the demands of automotive use case for fast processing and environmental sus-

tainability. The governance model offered by DPoS, emphasizing controlled, transparent delegate selection, could, if properly designed, address significant concerns around centralization and influence, fostering a trusted network environment fit for the IoT.
The security achieved by carefully choosing trusted delegates directly addresses the automotive industry's high-security requirements, indicating that a well-designed DPoS system could greatly enhance network security. Additionally, the possibility of more efficient governance and decision-making within a DPoS framework might lead to easier protocol upgrades and changes, which is important for the quickly changing automotive technologies. Therefore, DPoS, even with initial concerns, appears to be a potentially useful consensus mechanism for this area, provided that issues related to delegation and the risk of centralization are properly managed.

ℹ Considering the specific needs of certain blockchain networks, especially those focused on efficiency and governance simplicity, Proof of Authority (PoA) is preferred in this thesis over Delegated Proof of Authority (DPoA). PoA's reliance on a small number of trusted validators simplifies the consensus process, allowing for faster transaction validation without the complexities of delegate elections. This approach is particularly beneficial in environments where transaction speed and streamlined decision-making are critical. While DPoA aims to distribute decision-making more widely, the efficiency, governance simplicity, and security offered by PoA make it a more suitable choice for networks that value these attributes over the broader decentralization promised by DPoA.

2.3.7.8 GRANDPA

GHOST-based Recursive ANcestor Deriving Prefix Agreement (GRANDPA) is a particular algorithm because it is not a consensus but rather a finality gadget [104]. The finality gadget is an abstraction that runs along any block production mechanism (e.g., AURA) providing provable finality guarantees. The combination of AURA and GRANDPA will provide users of the blockchain the ability to verify the exact status of their transactions. Also, the finality gadget allows for a separation of safety and liveness in order to speed up recovery from failures. It also enables a fast recovery process when the network heals and promotes decentralization by making progress in chain growth even when finalization is slow.

2.3.8 Discussions About Consensuses

2.3.8.1 Consensuses Blockchain Application

Simply put, we can categorize blockchain applications into two primary consensus types based on the level of restriction on node participation in block creation: public and permissioned. In [110] and [96] consensus protocols comparison, we can highlight that the consensus application correlates with fault tolerance: consensuses targeting permissioned applications have considerably less fault tolerance than public consensuses. Additionally, most public consensuses are probabilistic and private/permissioned ones are mostly deterministic. The need of a finality in public blockchain made finality gadgets essential, thus creating hybrid consensuses (PoS + Casper, PoA + GRANDPA, etc.) [97] [104].

These first comments are to be expected because in public blockchain consensuses the algorithms are designed to allow any nodes to participate in the new block proposition agreement, thus requires to be very tolerant to faulty nodes. In permissioned blockchain consensuses, most

of the blockchain nodes are known or controlled to a limited number of participants, thus requiring less fault tolerance.

The resources used in each category is also remarkably different [96]. Private consensuses rely mostly on a voting system, and public blockchains all rely on a more scarce resource such as tine, economic value (coin/tokens), or processing clock-cycles. The resources used in the consensus algorithm imply the consumption of some amount of energy.

2.3.8.2 Ecological Impact and Consensus

As we just saw before, most public consensuses require a scarce resource that indirectly consume more energy such as PoW which rely on a proof of consumed processing clock-cycle. Derived from [111] the authors in [112] created an Indicative Energy Consumption (IECon) that express the energy consumption for most common consensus algorithms (Figure 2.3.7). The energy consumed directly correlates with CO2 emissions, thereby affecting our ecological footprint [113]. The graph should be treated with a degree of caution as the data pre-dates 2021, and is subjective.The energy consumption of each blockchain consensus depends not only on the clean energy ascertained by the entire network and the possibility of clean energy in the consensus incentive, but also on the practical implementation of a consensus in a blockchain. Nonetheless, it is a good first glance estimation for the level of "*pro green*" in consensuses. The IECon chart is confirmed for some public blockchain consensuses by the work of Platt et al. [114].
The authors in [114] provide an analysis on the energy footprint of blockchains depending its processed transaction per second (i.e. transaction throughput). All comparison relates on the energy consumption of transactions and is compared to VisaNet payment network. We will only use Bitcoin and Ethereum data to discuss here the energy impact. Bitcoin PoW consumes 3691.407 kWh/tx and VisaNet 0.00358 kWh/tx: Bitcoin uses 1031119 times more energy per transaction than VisaNet !
This gap is not as important for Ethereum blockchain (PoS consensus) which use between 0.8 and 155 times more energy per transaction than VisaNet. The range is high because the authors express two type of Ethereum throughput projections. We see that PoS in Ethereum blockchain already improved drastically compared to Bitcoin PoW. The decentralization and security is of course different between each blockchain technology, but the result is that Ethereum has the potential to compete with VisaNet in terms of energy per transaction used. Ethereum transaction throughput scaling however is still significantly lower then VisaNet [115]. Bitcoin and Visanet achieve an average of 7 and 24000 transactions per second, which is a considerable difference in processing capabilities. This significant difference shows the practical compromises between the decentralized design of blockchain technologies and the operational efficiency of centralized payment systems.

An other report on Bitcoin energy consumption [116] shows a more nuanced overview of the cryptocurrency compared to a Gold asset and banking system. The report shows that the banking system uses 263.72TWh of energy per year, Gold uses 240.61TWh of energy per year, and Bitcoin uses an estimate of 113.89TWh per year. The authors doesn't question the properties of Bitcoin as it shows sufficient interest and features compared to other assets, but rather if "*Is the Bitcoin network's electricity consumption an acceptable use of energy?*". The following question –partially answered by [112] – may arise in this thesis work: "*Do private (blockchains) consensus algorithms justify its power consumption compared to common databases ?*". However, the first

and current issue is rather if private blockchains perform sufficiently well to compete with current systems.

In this thesis, the targeted consensuses are PoA, as showed in Figure 2.3.7 are the less energy consuming blockchain design (so called ”pro green blockchain A”). The benefits of decentralization and interoperability are considered higher than an ”overly powerful” centralized system precessing capabilities.

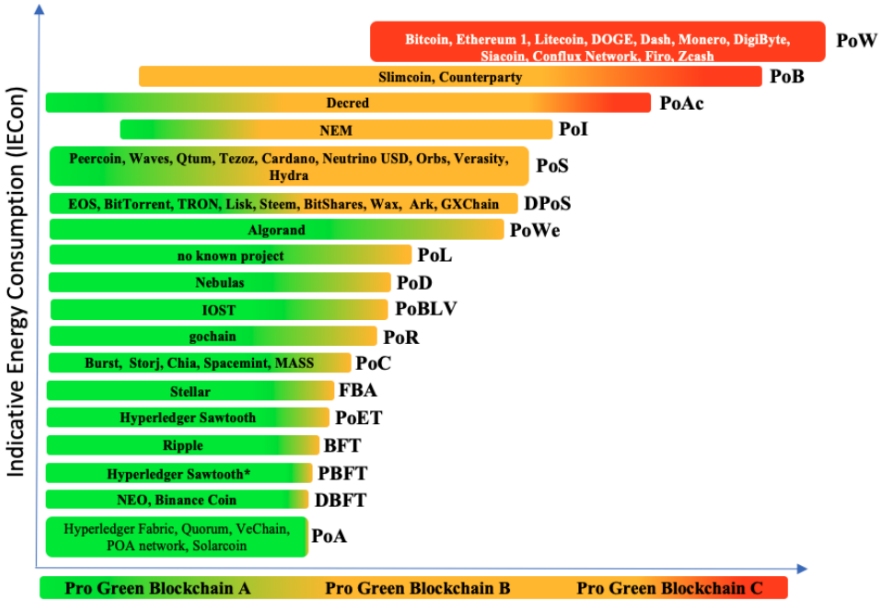


Figure 2.3.7: Indicative Energy Consumption (IECon) Chart [112]

2.3.9 Smart Contracts in Blockchains

2.3.9.1 What is and isn't a Smart Contract ?

In our exploration of the blockchain ecosystem, it is essential to understand the concept of smart contracts – a term often used, but frequently misunderstood. A smart contract, in essence, is a self-executing contract with the terms of the agreement embedded within a code that is stored and replicated on the blockchain network [76]. These contracts automatically execute transactions once pre-set conditions are met, minimizing the need for intermediaries and enhancing transparency and efficiency.

However, while the name might suggest otherwise, a smart contract is not 'smart' in the sense of holding intelligence, nor it is always legally recognized as a 'contract' in the conventional sense (Figure 2.3.8) [117] [118]. Understanding what does and does not constitute a smart contract is crucial in navigating the opportunities and challenges they pose, a topic we will explore in the following section.

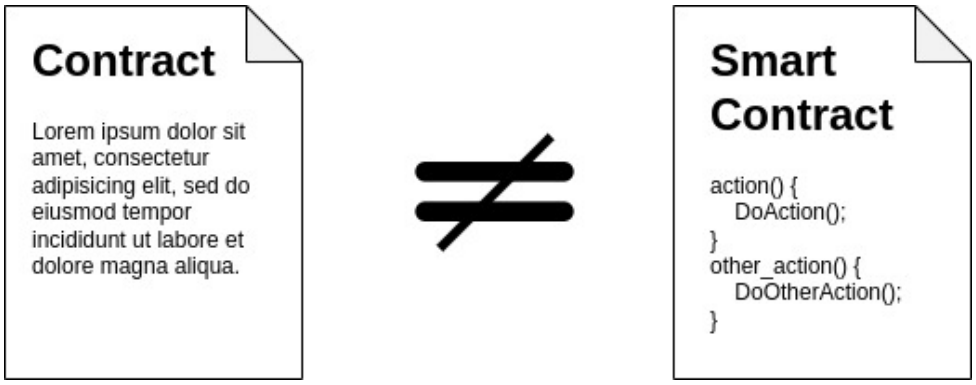


Figure 2.3.8: A legal contract isn't a smart contract. A smart contract is written in code by a programmer and a legal contract is text written by a legal professional or the involved parties, and must be signed.

2.3.9.2 Blockchain State and Smart-Contracts as a Moore Machine

The blockchain ledger is a probabilistic state machine when use of stochastic consensus mechanism such as PoW and PoS (i.e. "who wins the next block ?"). However in deterministic consensus mechanisms such as PBFT [101], the entire blockchain can be represented by a deterministic finite-state machine.

Additionally, a smart contract program is designed to compile in a set of instructions that is executed by the smart contract executor before building a new block (i.e. a Turing complete system, such as Ethereum [119]). The deterministic behavior of these, constitute again a finite-state machine (FSM) based on the current state of the blockchain and incoming transactions. The state transition is clocked by the creation of blocks, which is in turn clocked by the blockchain consensus. Jamsheed Shorish [120], provides an in-depth formalization of blockchain as a state machine, but a simple analogous representation can be described using Moore machine [121], where: the inputs are transactions, the transition logic are smart contracts logic, the state memory is the state database, and the resulting current state is driven by the consensus (Figure 2.3.9). Overall, output current state changes are stored in the new block each time consensus is reached.

The I8X platform [122], a blockchain-based research publishing platform, is an example of

use case that took a two-step approach to create smart contracts (based on Mavridou's et al. work [123]): first transforming their use case to FSM, than encode their processes into a smart contract. The difference between their work and the representation in Figure 2.3.9 is that we consider the entire blockchain as a FSM and not only the smart contracts.

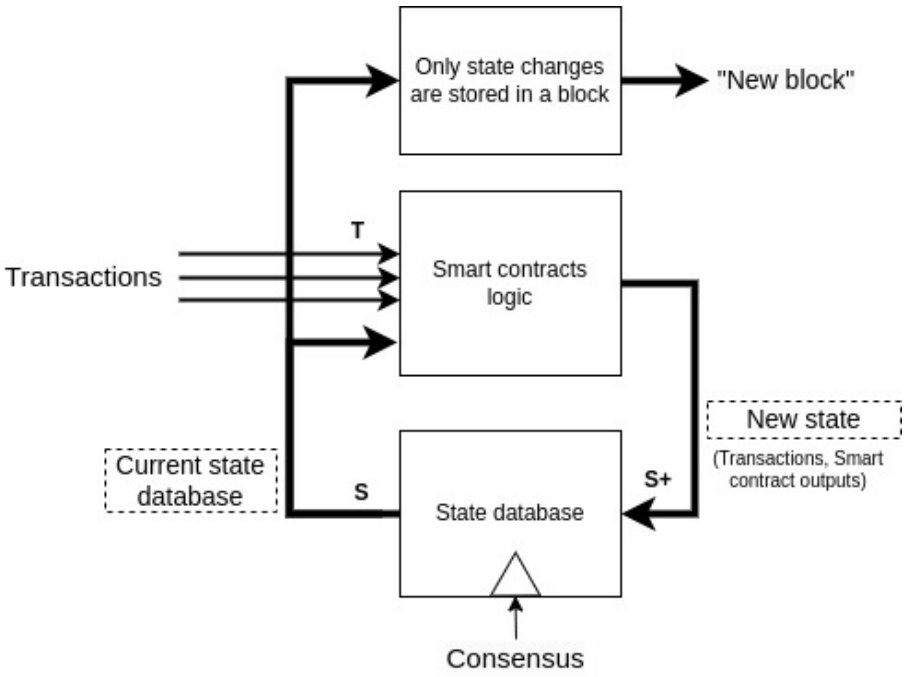


Figure 2.3.9: Blockchain and Smart Contracts Acting as a Moore state machine.

2.3.9.3 Who You Gonna Call? Smart Contracts !

Smart contracts are leveraged in a wide array of applications where trust, transparency, and automation are key requirements. They serve as an ideal solution when multiple parties, possibly unknown to each other, are involved and there's a need for an immutable, transparent system to verify and enforce the terms of a deal. Moreover, smart contracts come into play when an operation relies on a sequence of actions that can be automated based on predefined rules:

- Supply Chain Management: In the realm of supply chain management, smart contracts can enhance traceability and accountability [124] [125] [87]. They allow for automatic tracking of goods and validation of milestones, making the process more efficient and transparent.
- Financial Transactions: Smart contracts are highly suitable for financial transactions, including those involving cryptocurrencies [126]. They can automate and simplify processes in trading, lending, insurance, and more, reducing the need for intermediaries and ensuring swift, secure transactions [127] [128].
- Real Estate and Asset Management: Smart contracts can streamline real estate transactions and asset management by automating processes such as transfer of property titles or assets based on the fulfillment of contractual terms [129].
- Decentralized Autonomous Organizations (DAOs): DAOs, organizations ruled by programming code, greatly benefit from smart contracts. They provide a trustless system for decision-making and fund allocation in these decentralized entities [130] [131].

However, smart contracts may not be the best solution when there's a need for legal interpretation, human judgment, or when the involved parties don't have access to the necessary technological infrastructure. Furthermore, their use should be carefully evaluated considering the potential risks, such as vulnerability to coding errors or hacks, and the irreversible nature of transactions once executed on the blockchain.

2.3.9.4 Type of Smart Contracts

In practice, smart contracts can be understood through two primary perspectives, depicted in Figure 2.3.10):

- (1) On one hand, we have the concept of virtual machine (VM) [1] code execution: smart contracts are binary-like programs that are executed on a virtual machine in the blockchain. In this situation, smart contracts are compiled, sent to the blockchain, and than transactions can interact with it.
- (2) On the other hand, there's application-specific code execution, where smart contracts are designed specifically for a particular blockchain application, running in the core of the blockchain node. The application-specific smart contracts are located (almost hard-coded) inside or outside the core of the blockchain node. In each case the smart contracts modules are communicating with the heart of the node.

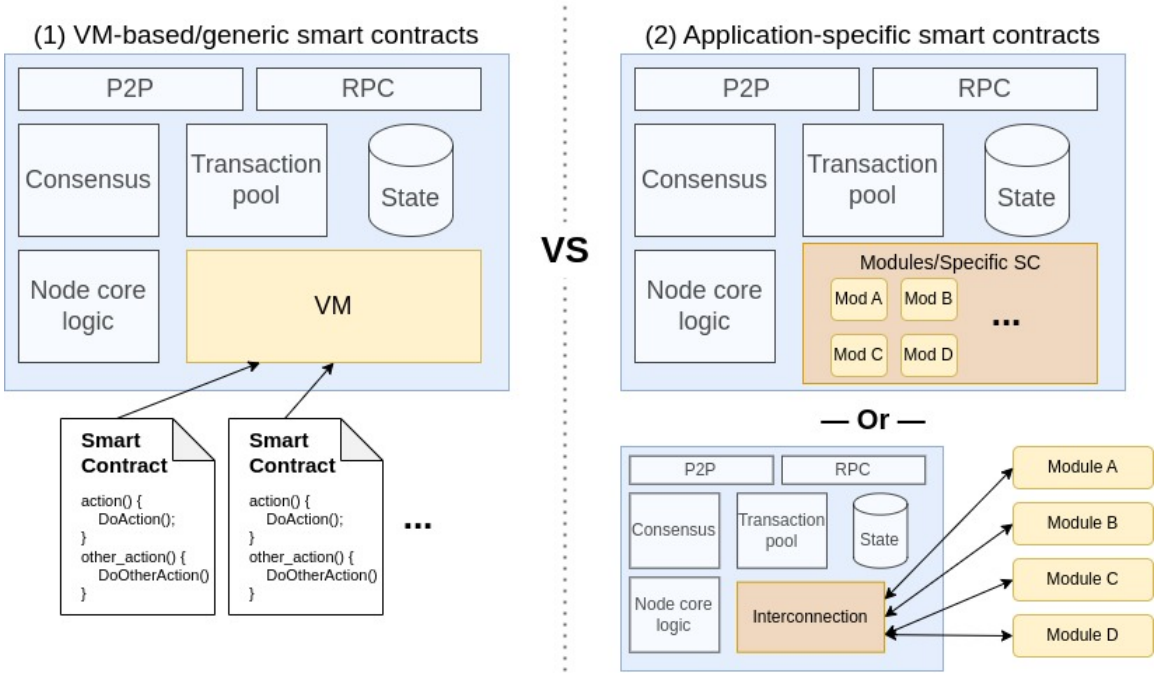


Figure 2.3.10: Two type of smart contracts: VM-based (1) and application specific (2) smart contracts.

2.3.9.4.1 VM code execution or on-chain contracts Refers to the use of a VM inside the blockchain to process and execute smart contracts. Essentially, these are programs that run

[1] Virtual machine: Virtualization or emulation of a computer system. In the case of smart contracts, the VM is specifically built to execute a set of instructions that can change the blockchain ledger state.

on the blockchain itself, thus automated and require no external action. The Ethereum Virtual Machine (EVM) is a well-known example of this. It allows developers to write smart contracts in a high-level language like Solidity, which is then compiled into EVM bytecode to be run on the EVM (Example in Appendix 8.2.2).

This execution model offers significant advantages, including the ability to write smart contracts that are blockchain agnostic (i.e., not tied to a specific blockchain, but tied to the virtualization technology), and the ability to create contracts that interact with each other, enabling the creation of complex decentralized applications (dApps). However, this flexibility also comes with increased complexity, and contracts executed in a VM are susceptible to bugs and security vulnerabilities if not properly written and audited.

Some of the common smart contract languages are:

- **Solidity**: This is the main programming language used for developing smart contracts on Ethereum-based blockchains (i.e. EVM-based), one of the most popular platforms for smart contract deployment [132]. Solidity is statically typed, supports inheritance, libraries, and complex user-defined types, making it a versatile choice for developers.
- **Rust and Ink!**: For the Polkadot and Kusama platforms, smart contracts can be written in Rust using the Ink! library [133]. Rust offers numerous tools for writing safe and correct code, making it suitable for smart contract development. The code is compiled into Wasm bytecode and executed in WebAssembly (Wasm) Interpreter (`wasmi`) for Polkadot and Substrate blockchains [134].
- **Michelson and Ligo (Tezos)**: Michelson is the native language for smart contracts on the Tezos blockchain [135]. It is a stack-based language and is designed with formal verification in mind, aiming to prevent bugs and other issues. However, due to its low-level and somewhat hard-to-use nature, higher-level languages like Ligo have been developed. Ligo is more accessible and user-friendly, making it easier for developers to write secure smart contracts on Tezos [136].

Examples of smart contracts written in each of these three languages – Solidity, Rust with Ink!, and Michelson/Ligo – are provided in Appendix 8.2.1. Solidity and Rust/Ink! are more high-level and user-friendly, offering syntaxes and structures familiar to those used in modern programming languages. Solidity, resembling JavaScript, is particularly popular for its ease of use and comprehensive documentation, making it a go-to choice for a majority of blockchain developers. Rust with Ink!, on the other hand, leverages Rust's robust safety features, such as its ownership model and type safety, bringing enhanced security and efficiency to blockchain programming.

2.3.9.4.2 Application specific code execution Application-specific code execution refers to the implementation of smart contracts specifically designed and optimized for a particular blockchain application (also called *installed* contracts). These smart contracts are typically written in a programming language native to the specific blockchain platform and are designed to fulfill specific tasks within that ecosystem.

This approach offers greater efficiency and performance, as the contracts are highly optimized for the specific application they are intended for. It can also provide improved security (relative to VM-based smart contract code), as the narrower scope of these contracts can limit potential attack vectors. However, the specificity of these contracts can also be a limitation. They may

lack the flexibility of VM-based smart contracts and may not be able to interact with contracts on other blockchains, limiting their interoperability.

Domain specific blockchain core modules (aka smart contracts) languages are:

- **Chaincode (Hyperledger Fabric)**: Chaincode is used for creating smart contracts, also known as "chaincodes", on Hyperledger Fabric, a platform for distributed ledger solutions [137]. Developers have the flexibility to write chaincode in Go, JavaScript, or Java, allowing them to choose the language that best suits their needs. This versatility empowers developers to build smart contracts that execute business logic and interact with the blockchain network efficiently and securely.
- **Transaction Processor (Hyperledger Sawtooth)**: The transaction processor in Hyperledger Sawtooth is responsible for receiving, validating, and processing transactions on the blockchain network [100]. It acts as an intermediary between client applications and the underlying blockchain network. Developers have the flexibility to write transaction processors in their preferred programming language, such as Python, JavaScript, Go, Java, and more, making it adaptable to different use cases.
- **Modules (Substrate-based Blockchains)**: Specific smart contracts on Substrate-based blockchains, also known as "pallets" or "modules" are written in Rust [138]. Rust's safety and performance characteristics make it an excellent choice for developing secure and efficient blockchain applications. By leveraging Substrate's modular architecture, developers can use Rust's expressive syntax and powerful tooling to build robust smart contracts that seamlessly integrate with the blockchain network.

2.3.9.4.3 Comparison of application-specific and VM-based smart contracts In evaluating application-specific versus VM-Based Smart Contracts, as outlined in Table 2.3.1, some key distinctions emerge. Application-specific Smart Contracts offer tailored performance and control, optimized for specific blockchain applications. However, their uniqueness requires considerable development resources and may limit interoperability. In contrast, VM-Based Smart Contracts provide a standardized environment with established infrastructure and tools, enhancing resource efficiency and community support. While VM-Based contracts provide a level of flexibility and ease of development, they may not achieve the same level of performance than application-specific contracts. The security and upgradability in both types of contracts are subject to their respective development and ecosystem contexts, with costs varying based on network demands and structural considerations (i.e. dependant on economic incentive). The choice of the type of smart contract largely depends on the project's specific needs and resources. Application-specific contracts are better suited for specialized, resource-intensive projects where high performance and customization are critical [83] [137] [138]. VM-Based contracts are more suitable for general-purpose applications where ease of development, flexibility, and resource efficiency are key considerations [139] [79].

Aspect	Application-Specific Smart Contracts	VM-Based Smart Contracts
Flexibility	Highly customizable for specific needs.	Constrained by VM's predefined functions.
Control	Full control over the blockchain's parameters.	Limited control within the VM's standardized environment.
Development	Requires building a unique infrastructure.	Utilizes established VM infrastructure and tools.
Security	Custom security can be a double-edged sword.	Relies on the VM's widespread security measures.
Performance	Optimized for the contract's specific use case.	Limited by the VM's general-purpose design.
Interoperability	Custom-designed for specific chains; can be made interoperable with effort and supporting protocols.	High within the same VM ecosystem; standardized for easy contract-to-contract interaction.
Upgradability	Flexible upgrades possible with planning.	Require hard-fork or depends on implementation.
Cost	Variable and potentially optimized.	Blockchain network demand and fee structure (e.g. gas fees).
Resource Needs	Significant for development and ongoing maintenance.	Reduced by leveraging the VM's existing network.
Community Support	Varies with the chain's adoption and support.	Generally robust due to a larger user base.
Use Cases	Suited for niche, complex applications.	Suitable for general use with established demand.

Table 2.3.1: Comparison of Application-Specific and VM-Based Smart Contracts

2.3.9.5 Bridging Code and Clarity: The Ricardian Contract

We mentioned earlier that smart contracts are not legal contracts, but in 1996 Ian Grigg introduced the concept of Ricardian contracts (term derived from economist David Ricardo) [140] [141]. A Ricardian contract is a digital document that combines legal language and machine-readable code, enabling both humans and machines to interpret it (Figure 2.3.11). It aims to bridge the gap between traditional legal agreements and smart contracts executed on blockchain or other digital platforms. Initially, Ricardian contracts were not specifically designed for use on blockchains, but the blockchain community has since repurposed the idea.

In pratice, in a Ricardian contract, the legal terms and conditions are written in natural language, making them understandable to humans. Additionally, the contract includes machine-readable code, typically written in a programming language, that defines the specific actions and behaviors associated with the contract. This code is executed by the software platform – or a smart contract in the case of blockchains – to enforce the agreed-upon terms.

This type of contracts provide transparency and clarity by ensuring that the terms and conditions of the contract are accessible and understandable to all parties involved. Additionally, they enable automation and self-execution of contractual obligations, enhancing efficiency and

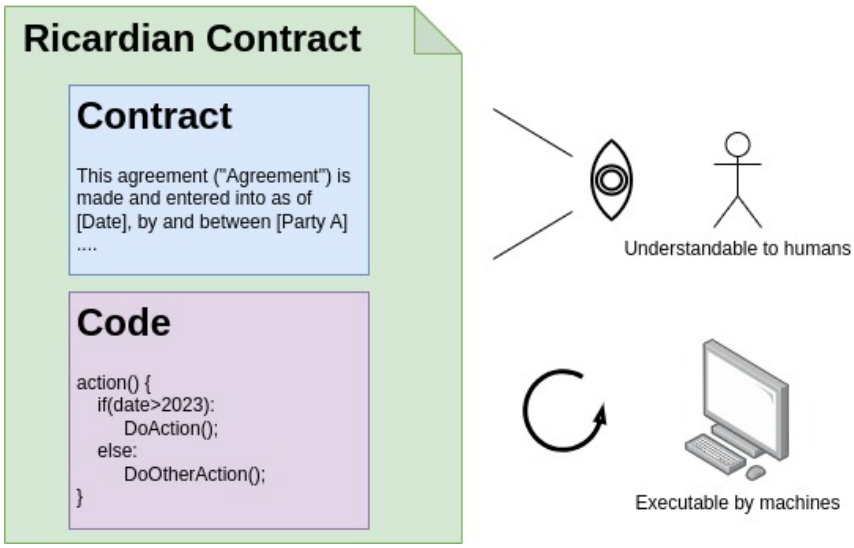


Figure 2.3.11: Schematic Representation of a Ricardian Contract: Integrating Machine Code and Human-Readable Elements

reducing the need for intermediaries. Additionally, the potential of this type of contracts in the blockchain realm could provide legally enforceable agreements and provide a bridge between traditional legal systems and DLT.

The first large-scale usage of these types of contracts where done by the OpenBazaar online market, a decentralized censorship-resistant platform [142]. To the best of our knowledge, the only blockchain implementation of Ricardian contracts where done by the EOS.IO in 2019 (*EOS Ricardian Contracts*) [143].

2.3.9.6 Decentralized Applications (dApp)

Decentralized applications, commonly known as **dApps**, is the combination of smart contracts with applications running on decentralized networks. A dApp is essentially a software application that operates on a distributed network of computers, typically utilizing blockchain technology [144].

Even if the word *dApp* didn't exist at the time, this type of applications already existed since the beginning of internet (e.g. Tor, BitTorrent, Napster, ...) using P2P protocols and open-source software. These applications where decentralized and it is only volunteering that maintained them and P2P network [67] [145] [146].

Bitcoin proved rapidly (from 2009 to 2014) that cryptocurrency had a place in the digital world. In 2014, the novel proposal of Buterin was to leverage applications with a blockchain and its cryptocurrency by introducing the smart contract concept. From that point in time, dApps have been popularized mainly by Ethereum blockchain with the concept of web3 as a new version of internet allowing to read, write, and own data [147]. These new dApps are using blockchain and smart contracts as the backbone of their business logic.

In the open systems interconnection (OSI) Figure 2.3.12 we can observe the layers required to build a blockchain-based dApp. The dApp is at the frontier of the user interface and application. The dApp exists because its logic is encoded in the smart contract but requires also to have designed a matching interface (i.e. a web app, a desktop app, ...).

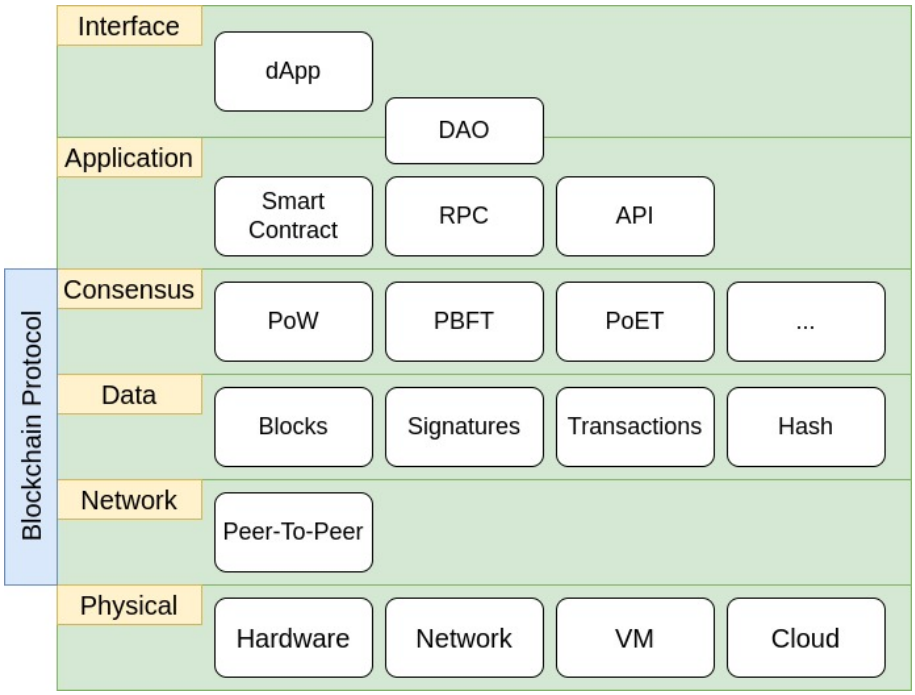


Figure 2.3.12: Layers that constitute blockchain and dApp and DAO: physical, network, data, consensus, application, and the interface layer.

2.3.9.7 Smart Contracts Challenges

The concept of smart contracts embodies the process of encapsulating business logic within a programming language. In essence, it involves the abstraction of all pertinent actions, conditions, and exceptions within a software program, thereby mitigating the potential for human interpretation and extreme circumstances, including situations involving case of force majeure. Smart contract related issues constitute the one of the most risks in blockchain security [20].

The initial challenge lies in the subjective translation of decisions by the programmer who develops the smart contract. Given that the programmer lacks legal expertise, they must take into account all potential scenarios and application use cases. Consequently, the program is susceptible to both misinterpretation of the use case and the inadvertent limitation of the smart contract's scope of actions, thereby deviating from its intended purpose.
A perfect example is given by Sarah Green [148], which demonstrate how complex a simple statement containing a condition can be:

> *"Go to the shop and buy a newspaper. If there are any eggs, get a dozen."*
> Human result in the event the shop has eggs: purchase of one newspaper and a dozen eggs. Computer result in the event the shop has eggs: purchase of a dozen newspapers.

Some individuals, but not all of the coders, would interpret the instruction in buying a dozen eggs, and some individuals would understand it in buying a dozen newspapers.

A challenge emerge from the intrinsic immutability of smart contracts once deployed on the blockchain. Unlike traditional software applications that can be updated or patched to address bugs or add new features, smart contracts are designed to be immutable and tamper-resistant [149]. While immutability is a desirable characteristic for maintaining the integrity and trustiness

of transactions, it also poses challenges when new issues arise or when changes need to be made. If a flaw or vulnerability is discovered in a deployed smart contract, it cannot be easily modified or corrected without going through a complex and potentially aggressive process. However, modern smart contract have found tricks to bypass this immutability using proxy contracts [150].
This lack of upgradability and flexibility can be particularly problematic when it comes to legal or regulatory changes. Laws and regulations are subject to constant evolution, and if a smart contract is not designed to accommodate such changes, it may become non-compliant or ineffective over time.

An additional challenge relates to the complexities of ensuring the security and resilience of smart contracts. Smart contracts are executed on decentralized networks, making them vulnerable to various types of attacks and exploits. These can range from coding vulnerabilities that allow malicious actors to manipulate the contract's logic to external attacks that target the underlying blockchain infrastructure. In 2016, *The DAO* (Decentralised Autonomous Organisation) was hacked and costed about $60 million, stolen by the hacker [151]. The DAO hack caused the majority of the Ethereum community to hard-fork the blockchain (i.e. the blockchain ledger was separated into two networks, Ethereum and Ethereum Classic).

Ensuring the security of smart contracts requires thorough code auditing, rigorous testing, and adherence to best practices in secure coding. A related security issue is the Parity wallet bug, which in 2017 froze $146 million being inaccessible [152].
Additionally, active monitoring and vulnerability management are essential to detect and mitigate ongoing threats. However, these security measures can be resource-intensive and require specialized expertise in the DLT domain, adding to the overall complexity and cost of developing and maintaining smart contracts.

It is worth noting also that the type of smart contract (i.e. its compatibility with the most dominant technology which is EVM-compatible contracts) is determining the probability to attract new projects to choose a blockchain platform. Based on the work of Ruizhe Jia and Steven Yin [153], when given the choice to adopt an EVM-compatible blockchain or an incompatible blockchain, the solution is to either: i) directly subsidize the new entrant firms, or ii) offer better features (e.g. lower transaction costs, faster finality, or larger network effects). The results shows that it is easier for EVM compatible blockchains to attract users through direct subsidy, while it is more efficient for EVM-incompatible blockchains to attract users through offering better features/products.

2.3.10 Key Challenge in Blockchains

What limits the performances of a DLT ? What holds back mass adoption of blockchain solutions?

The mass adoption of blockchain technology is primarily hindered by technical, regulatory, and societal factors. The inherent complexity, thus lack of understanding of the technology, discourage its acceptance among the general public. Scalability issues may also limit transaction capacities and create an obstacle to practical use.

Additionally, one other challenge with blockchain is that it that DLT has a hard time dealing with a lot of transactions at once. This problem, called scalability, makes it difficult for blockchains to grow and support more users. The "blockchain trilemma", introduced by V. Buterin [154], is a way to express this challenge in blockchain: trying to get security, scalability, and decentralization all at the same time. Security means keeping the blockchain safe from fake transactions and

making sure no one can change the data once it's written. Scalability is about handling more transactions faster, so the system can grow. Decentralization means that no single person or group has control over the entire blockchain.

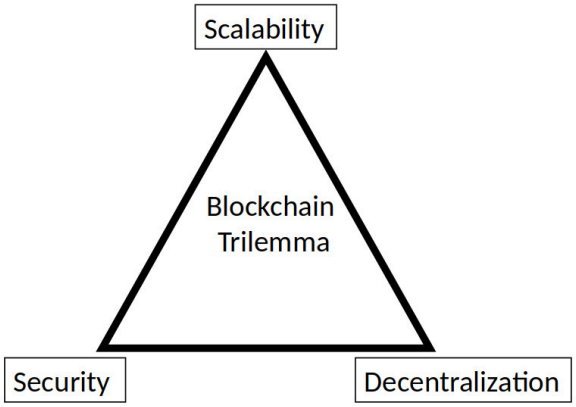


Figure 2.3.13: Blockchain Scalability Trilemma: Decentralization, Scalability, Security

As depicted in Figure 2.3.13, the trouble is that improving one of these usually means compromising on another. Decentralization, which spreads control across a network and requires consensus from multiple participants, can slow down transactions as more people use the system. This makes it tough for a blockchain to scale up and maintain its decentralized nature, as every additional transaction can potentially increase the time needed to reach consensus. Meanwhile, decentralization enhances security by protecting against centralized points of failure and control, making it resistant to censorship and coordinated attacks. However, this widespread distribution of control can also mean slower responses to new, unexpected security threats since changes require agreement from a larger group. On the other hand, putting in place strict security measures can slow down a blockchain. High-level security needs complex calculations and checks to prevent fraud, which can make the whole system slower, especially as the number of transactions climbs. This is the core issue of the scalability problem: how to let a blockchain grow without losing the high security that makes it trustworthy.

Additionally, concerns about security (of the blockchain or smart contracts) may raise safety questions and hinder adoption of the technology. The total amount of losses due to exploits is valued at more than 40 billion USD (based on june 2022 BTC and ETH prices) [155]. In a survey by Dipankar Dasgupta, John M. Shrein and Kishor Datta Gupta [156], the authors extracted eight categories of security issues. The author classification of security concerns, reveals many vulnerabilities that each can be derived into different exploit implementations. In their work [156], they also provide an extensive overview of blockchain potential for many industry sectors and human daily life challenges.

Law and regulations are also an obstacle for DLTs. Regulatory uncertainties across different jurisdictions create an environment of hesitation among businesses and consumers. Due to inherited nature of blockchain, GDPR regulation may seem impossible. However, the work of Michael Kolain and Chrisitan Writh [157] proves otherwise. The authors provide an architectural blueprint and a methodology to translate legal into technical requirements in a comprehensible way (e.g. consent is translated to *shall be asked once for approval* and *be able to withdraw their consent*).

The volatile nature of associated markets, like cryptocurrencies, adds an element of financial risk, and the current lack of user-friendly interfaces can discourage non-tech individuals.

Based on previous discussion, an extension of the above trilemma (Figure 2.3.13) would include additional aspects such as:

- Privacy
- Confidentiality
- Interoperability
- Usability
- Compliance
- Developer Experience
- Finality

In the world of multi-chain, every transaction finds its own chain. In the DLT realm, the proliferation of multiple blockchain networks is proof to the innovation and versatility offered by this technology. However, these networks have each their distinctive protocols, consensus mechanisms, and token economics (tokenomics), smart contracts implementations, offer various solutions across diverse industries, including finance, supply chain management, healthcare, and more. The increase in the number of blockchain networks, introduces the interoperability concept.

2.4 Blockchain Interoperability

Blockchain interoperability is a critical capability that allows distinct blockchain systems to communicate, share information, and transact seamlessly [158] [159]. This feature is essential in realizing the full potential of blockchain technology across various sectors by facilitating the exchange of assets and data between diverse platforms.

As depicted in Figure 2.4.1, the current blockchain landscape comprises a multitude of networks, each operating independently. This separation often results in siloed ecosystems where interaction and asset transfer between different blockchains are not inherently possible. Interoperability seeks to bridge these isolated networks, facilitating a level of collaboration and interaction.

Blockchain interoperability implies that each network acknowledges and validates transactions not only based on its rules and ledger state but also considering those of the other involved networks [158]. Therefore, an interoperable transaction on one network must take into account the ledger state of the other network to be valid [160]. Some of the key concepts of today latest novel designs can be summaried in a paper by Thomas Hardjono, Alexander Lipton and Alex Pentland, who discuss a design philosophy for interoperable blockchain systems early in 2018 [161]. The authors use the design philosophy of the Internet and apply it to blockchain technology. This perspective is interesting due to the Internet agnostic data exchanges concept: Blockchain interoperability was focused mostly on currency interoperability (and so is the paper's goal), but with this Internet concepts recycling it could shows an interest for more generic transaction interoperability between blockchain networks.

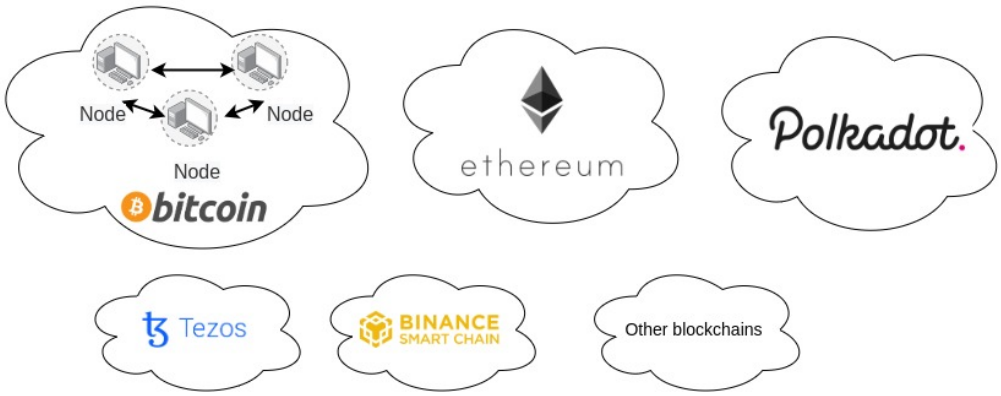


Figure 2.4.1: A landscape of heterogeneous blockchain networks

The work of Schulte et al. [162], analyses the current state of the art in blockchain interoperability. The authors state that there are three functionalities of a blockchain: send tokens from one address to an other address, execute a smart contract, and guarantee the validity of data stored in a blockchain. The paper separates clearly that interoperability of blockchain is split into the three different functionalities. This suggests that a fully interoperable blockchain has to implement three interoperable functionalities.

In this thesis **we are focused on a specific type of cross-chain communication that allows the interoperability of smart contract executions and guarantee the validity of data stored in a blockchain.**

2.4.1 Interoperability Mechanisms in Blockchain

Blockchain interoperability is a complex concept comprising of various mechanisms and practical implementations technologies. We list here some of the key interoperability mechanisms [159].

2.4.1.1 Public Connectors

Public Connectors provide interoperability between cryptocurrency systems [163]. In this category falls: Atomic Swaps, Sidechains/Relays, and Notary Schemes.

Atomic Swaps are allowing for the direct exchange of cryptocurrencies across different blockchains without intermediaries, using smart contracts. This technique enables users to transact across various blockchain platforms without relying on centralized exchanges [164].

Sidechains are independent blockchains connected to a parent blockchain (mainchain). They allow for secure transfers of assets and data between the sidechain and the mainchain, enhancing the scalability and functionality of the parent blockchain [163].

Notary Schemes are protocols where trusted entities validate and verify cross-chain events, issuing cryptographic proofs to ensure transaction integrity and consensus [163]. It is probably the simplest way to exchange information between blockchains, however it requires a trusted intermediary. Example: Imagine sending a digital token from Blockchain A to Blockchain B. A Notary Scheme would involve a trusted validator confirming that the token has left Blockchain A, and then informing Blockchain B to credit the account there, ensuring a secure and verified transfer.

2.4.1.2 Blockchain-of-Blockchains

Blockchain-of-Blockchains are blockchain frameworks that provide reusable modules to build application-specific blockchains (customized blockchains) that interoperate between each other. The created blockchains will reuse data, network modules, consensus concepts, incentive mechanisms, and possible contract layers. Due to the thesis direction, an in-depth study of the these most popular blockchain frameworks in done in Section 2.4.2. Most popular blockchain-of-blockchains platforms are Polkadot and Cosmos [165][166] and are presented in the next Section 2.4.2.

2.4.1.3 Hybrid Connectors

Hybrid Connectors represent a category within blockchain interoperability solutions, distinct from Public Connectors or Blockchain-of-Blockchains. These connectors focus on creating a *blockchain abstraction layer*, enabling uniform operations for decentralized applications (dApps) to interact across different blockchains without the need for distinct APIs. This category encompasses several sub-categories, including Trusted Relays, Blockchain-Agnostic Protocols, and Blockchain Migrators. Simply put, a hybrid connector is required when interoperability is not for assets, but some arbitrary data [158].

Trusted Relays typically operate in environments where blockchain registries facilitate the discovery of target blockchains, often in permissioned blockchain settings. These relays serve as trusted intermediaries, directing cross-blockchain transactions [159].

Blockchain-Agnostic Protocols offer technology-neutral frameworks for inter-blockchain communication, enabling interactions between various distributed ledger systems. However,

they might require modifications to existing blockchain source codes for integration and may not guarantee backward compatibility [159].

Blockchain Migrators focus on transferring data between different blockchains, resembling notary schemes with a centralized party overseeing the migration process. This includes the migration of blockchain states, potentially enabling the transfer of cryptocurrencies and other assets across blockchains [159].

Hybrid Connectors aim to address the challenges of blockchain interoperability by offering solutions that range from trust-based relays in controlled environments to more decentralized, agnostic protocols. These connectors are essential in enabling diverse blockchain ecosystems to interact seamlessly.

2.4.2 Interoperable Blockchain Ecosystems (Blockchain-of-Blockchains)

2.4.2.1 Polkadot, Comsos, and Avalanche

Polkadot, Comsos, and Avalanche are the most popular solution ar the current time of writing (2023). We examine the current state of development within these ecosystems by analyzing the number of projects and their categories (Figure 2.4.2)[2].

Polkadot [165] has been notably focused on low architecture level development and shows close to 200 projects. It's worth noting that a project in Polkadot corresponds to a parachain, effectively creating a new blockchain environment for applications. Considering that each parachain can host many application projects, the entire Polkadot ecosystem can easily encompass over 2000 sub-projects. Polkadot's ideology revolves around stability and a long-term vision, making it an attractive platform for developers seeking a solid foundation for their projects.

Cosmos [166], with approximately 250 blockchain projects in its ecosystem, also showcases a strong development focus. Especially, the most popular Cosmos-SDK-based blockchains host over 100 applications each, highlighting the active development and adoption of new applications within the Cosmos ecosystem. This trend signifies the ecosystem's vitality and its ability to attract innovative projects.

In contrast, Avalanche [167], a relatively new blockchain platform introduced around 2020, has already collected an impressive 400 projects. These projects cover various domains, including tools, blockchain nodes, and diverse applications. Avalanche's rapid growth signifies the platform's appeal and potential as a promising ecosystem for developers.

While analyzing the number of projects in these ecosystems provides valuable insights into their development activities, gauging actual user engagement presents a more complex challenge. The metric of user activity, especially in terms of real-world transactions, can be obscured by the prevalence of automated software, which often accounts for a significant portion of transaction volumes. Therefore, a comprehensive evaluation of Polkadot, Cosmos, and Avalanche should not only consider the quantity of user interactions but also delve into the quality and innovation of the projects, the level of developer engagement, and the extent of community involvement. Such a multifaceted approach ensures a more accurate assessment of each ecosystem's overall vitality and potential for future growth.

[2]Sources: https://cosmos.network/ecosystem, https://core.app/discover/projects/all/, https://parachains.info/, and compiled with project: https://github.com/lucgerrits/blockchain_projects_metrics.

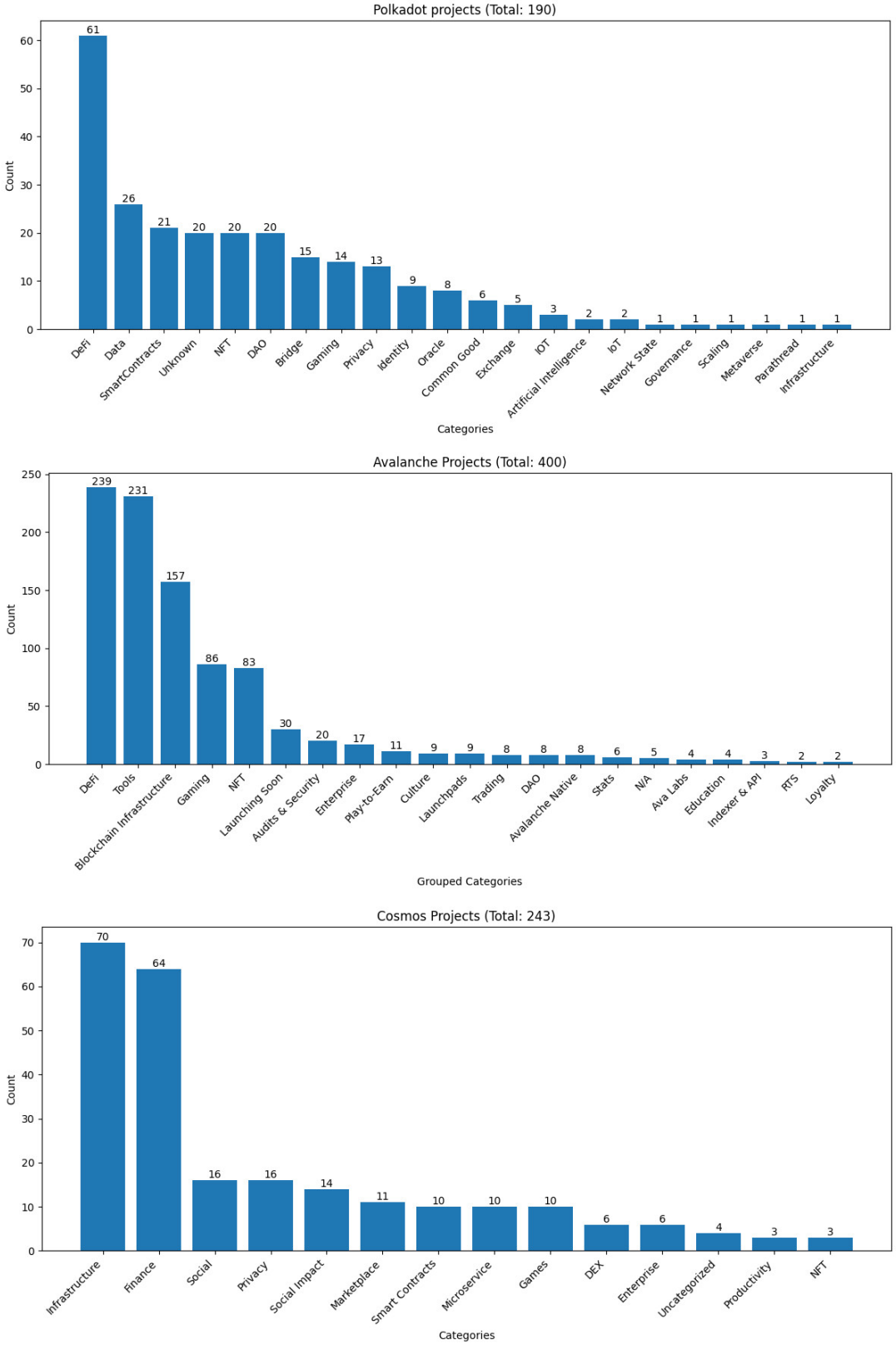


Figure 2.4.2: Project Distribution Across Categories in Polkadot, Cosmos, and Avalanche Ecosystems (August 2023)

2.4.3 Analysis of Blockchain-of-Blockchains Models

There are very few blockchain projects that are actually ”fully” interoperable. Moreover, even fewer blockchain projects provide a dedicated software development kit (SDK) to facilitate the creation of custom blockchains. Establishing standards within a network of multiple blockchains is essential for seamless communication, making interoperable blockchains reliant on a complete standard software stack. As you can see in Table 2.4.1, we compare the three most potential and advanced interoperable multi-chain technologies (Polkadot, Cosmos, Avalanche).

Table 2.4.1: Comparative Table: Polkadot vs. Cosmos vs. Avalanche Blockchains

Feature	Polkadot	Cosmos	Avalanche
Launch Year	2017	2016	2020
Consensus Algorithm	Nominated Proof-of-Stake (NPoS) + GRANDPA	Tendermint (Byzantine Fault Tolerance)	Avalanche (Ava)
Interoperability	Yes	Yes	Yes
Governance Model	On-chain	On-chain	On-chain
Scalability	Multi-chain architecture (Parachains)	Multi-chain architecture (Zones)	Customizable blockchains (Subnets)
Security Model	Shared security model	Independent security for each zone	Customizable security (Variable Fees)
Consensus Upgrades	On-chain using governance	On-chain using governance	On-chain using governance
Native Token	DOT	ATOM	AVAX
Smart Contract Platform	Yes (Polkadot Parachains support EVM)	Yes (Cosmos SDK supports various VMs)	Yes (EVM and Avalanche-X)
Cross-Chain Swaps	Yes (through XCMP protocol)	Yes (through IBC protocol)	Yes (through AWM & Avalanche Bridge)
Framework or SDK	Substrate	Cosmos SDK	Avalanche-CLI
Development Language	Rust	Go	Go
Initial Focus	Interoperability and shared security	Interoperability and scalability	High-throughput and low fees

In the Blockchain-of-Blockchains architecture, various advantages and challenges are brought to light, as demonstrated in the comparative analysis of Polkadot, Cosmos, and Avalanche (see Table 2.4.1). The emergence of multi-chain models such as Polkadot, Cosmos, and Avalanche marks a significant evolution in blockchain interoperability, aiming to address the limitations of traditional single-chain architectures. These models offer a blend of scalability, flexibility, and improved cross-chain communication but also come with their own set of challenges. The

Table 2.4.2 offers a detailed overview of the advantages and disadvantages of various blockchain interoperability projects, focusing on the Blockchain-of-Blockchains models, and other multi-chain solutions.
In this comparative analysis, Polkadot and Cosmos stand out as leading projects in the Blockchain-of-Blockchains space, each offering unique features and facing distinct challenges. Polkadot, known for its shared security model and efficient cross-chain communication, with an advantage in interconnecting various blockchain applications, while Cosmos, with its emphasis on chain sovereignty and the IBC protocol, provides to projects a prioritization on independence and robust inter-chain interactions.
The table also evaluates other solutions like more agnostic interoperability protocols – highlighting standardization benefits – and more specific solutions such as bridges, wrappers, and middleware, more adapted for asset transfer and data management.

2.4.3.1 Advantages of Blockchain-of-Blockchains Models

- **Enhanced Scalability:** By distributing the overall transactions load across multiple chains, architectures like Polkadot and Cosmos effectively manage congestion and improve throughput (e.g. parachains and hubs).
- **Customization:** These platforms allow for the creation of tailored blockchains, enabling specific governance models and consensus mechanisms to suit diverse project needs (e.g. Substrate SDK and Cosmos SDK).
- **Inter-chain Communication:** Facilitated by protocols like Polkadot's XCM and Cosmos' IBC, these models excel in enabling asset and data exchange between different chains within their ecosystems.
- **Security and Governance:** Polkadot's shared security model pools resources for collective network protection, while Cosmos offers independent governance for each chain, catering to varied security needs.

2.4.3.2 Disadvantages of Blockchain-of-Blockchains Models

- **Complexity:** The intricate nature of multi-chain networks can pose challenges in development and user experience. Each platform has their own (reusable) data structure, consensuses, language, etc. Both platform solution are progressing toward development *homogeneity* (i.e. Cosmos and Polkadot only support Tendermint-based blockchains and Substrate-based blockchains respectively) [159].
- **Limited External Interoperability:** While robust within their ecosystems, connecting with external blockchains often requires additional mechanisms (due to their homogeneity). However, each ecosystem have active development for inter-ecosystem interoperability using public or hybrid connector mechanisms [168].
- **Security Risks:** Shared security models may propagate vulnerabilities across the network (i.e. systemic risks), while independent/sovereign security models in ecosystems like Cosmos could leave smaller chains more vulnerable (i.e. each chain carry the responsibility of its own security). This key difference between platforms is probably a top priority when deciding which one to use.
- **Governance Issues:** Coordinating governance across multiple independent chains can be a complex and ongoing task. Example: Polkadot has questionable centralized governance due to limited central voters [169].

The two key characteristics to highlight: **Polkadot** is notable for its shared security and efficient cross-chain communication and is a prime choice for building interconnected blockchain applications. **Cosmos** emphasis on chain sovereignty, their IBC protocol, and is suited for projects prioritizing independence and inter-chain interactions.

	Type	Advantages	Disadvantages
Projects	Polkadot	-Enables transfer of any type of data across different chains -High scalability due to its multi-chain architecture	-Its shared security model can potentially lead to systemic risks -Relatively complex governance model
	Cosmos	-Promotes sovereignty, allowing individual chains to retain control over their governance -Offers high scalability through its unique hub and zone model	-Inter-blockchain communication can be complex and challenging -No shared security, so individual chains are responsible for their own security
Other Solutions	Interoperability Protocols	-Standardization allows for easy interaction between chains -Can enhance security and efficiency -Easy development using interoperability frameworks	-Might not support all types of data or transactions -Dependence on the robustness of the protocol -Dependence to the interoperability framework
	Bridges	-Facilitate direct asset and data transfer between different chains	-Can be technically complex to establish -May face security risks if not properly implemented -Usually case-specific (token/coins) bridges
	Wrappers	-Allows tokens to be used across different chains	-Depend on the security of the wrapping contract -Wrapped tokens require trust in the custodian of the original tokens -Mainly works only for tokens/coins
	Middleware	-Facilitate communication between different chains and external services -Can manage data effectively	-Can introduce additional complexity and potential points of failure -May face scalability issues if not properly designed -Usually very service-specific

Table 2.4.2: Advantages and disadvantages of blockchain interoperability projects and other multi-chain solutions

2.4.4 Choosing the Right Blockchain

Drawing from the existing state-of-the-art literature and exploring various blockchain configurations, it becomes evident that choosing the right blockchain can be a challenging task.

To determine the necessity of implementing a blockchain in your specific use case or industry, it is essential to refer to relevant works, such as Wüst et al. [111], which provide valuable insights into the decision-making process. This initial step helps ascertain whether a blockchain solution is justified and provide highlights on the type of blockchain that might be suitable (permissionless, public permissioned, private permissioned). However, once you have confirmed the necessity of blockchain technology, the next crucial task is to select the appropriate blockchain technology or DLT based on your requirements.

In early 2018, Scriber et al. [170] propose a framework for determining blockchain applicability. The framework isn't targeting any blockchain technology in particular but provide a form like a quiz to decide the fitting level of a project towards blockchain technology. Similarly to Scriber et al., Staderini et al. [171] provide a requirements-driven methodology to select and configure a blockchain. Their work propose a method to decide the blockchain project requirement (i.e., *Yes* or *No*), its category (public permissionless, public permissioned, consortium, or private), and finally its consensus configuration. In 2021, Almeshal et al. [126] did a literature review of blockchain for businesses focusing on suitability evaluations frameworks. The authors shows the latest comparison of decision models for blockchain suitability evaluation.

While multiple practical DLT decision-making studies are available (Table 2.4.3), only a few of them consider cutting-edge interoperable blockchains, such as Cosmos, Polkadot, or Avalanche.

In the current landscape of research, interoperable blockchains offer unique advantages, facilitating scalability, compatibility, and cross-chain interactions. However, integrating such advanced solutions also adds complexity to the decision-making process. Additionally, independently from the use case domain, systematic review and surveys highlight the interoperability challenge [174] [128].

Only specific use cases study discuss the need of interoperable blockchains [175]. However, a common criteria for choosing a blockchain with an interoperability protocol is the requirement of cross-blockchain interactions allowing directly a scaling and increasing the reach of the project application.
Interoperable blockchains are gaining attention in various specific use cases, which emphasize the need for cross-blockchain interactions [175]. For instance, industries like supply chain management, finance, and healthcare are exploring the potential benefits of interoperability [176]. The primary criteria guiding the adoption of blockchains with interoperability protocols is **the capability to enable seamless cross-blockchain interactions**. This feature not only allows for direct scaling of projects but also significantly expands the reach of their applications (additional users, other project utilities, cryptocurrency liquidity, etc.). Consequently, the consideration of interoperability becomes crucial for projects seeking to maximize the potential impact of blockchain technology, as seen in the latest work of [173] in 2020.
In one of the latest (March 2023) blockchain interoperability study [177], authors show that most of the developing interoperating schemes are for cryptocurrencies only. However, they also show that relays are most popular (*go-to*) solutions when flexibility is required, because it can transfer any form of data while also providing decentralization.

Moreover, the work of Mehrdad Salimitari, Mainak Chatterjee, and Yaser P. Fallah [178] is a unique and very related work to this thesis as it discuss in depth potential consensus algorithms

Author	Title	Year	Public blockchains	Private or consortium blockchains	Inter-operable blockchains	Note
Büyük-özkan, Tüfekçi [172]	A decision-making framework for evaluating appropriate business blockchain platforms using multiple preference formats and VIKOR	2021	N/A	Fabric, Quorum, Multichain, Sawtooth, Hydrachain	N/A	Only entreprise-focused blockchains.
Farshidi et al. [173]	Decision Support for Blockchain Platform Selection: Three Industry Case Studies	2020	Ethereum, NEO, Cardano, Stellar, Bitshares, QTUM, ICON, IOTA, Factom, Lisk, Wanchai, Neblio, Zilliqa, Komodo	Ethereum, R3 Corda, JPMorgan Quorum, Hyperledger, BigChainDB, Multichain, Hydrachain, Chain, Symbiont, Stratis, OpenChain,	Cosmos	Applying their blockchain study on three specific industry use case.
Saraf, Sabadra [124]	Blockchain platforms: A compendium	2018	Ethereum	Ethereum, Corda, Hyperledger, Sawtooth, Iroha, Burrow, Indy, Fabric	N/A	Early comparison of most famous technologies to help projects to decide which blockchain to use.

Table 2.4.3: Latest related work on blockchain decision-making

that can be used for resource constrained IoT networks. Their study shows that Proof-of-Elapsed-Time (PoET), PBFT, and Tangle are the most appropriate for IoT networks. Additionally, they also state that PoS, DPoS, Proof-of-Importance (PoI), dPBFT Stellar, Ripple, Tendermint, OmniLayer, RapidChain, and Raft are partially appropriate consensus methods. However, this work mention IoT Network based blockchain, and thus imply that the IoT will participate in the consensus, and in our thesis work, only off-chain blockchain-IoT integration is taken in account due to the high limitation of on-chain integration.

2.5 Case Study: Automotive Industry Use Case

The thesis gravitates around an automotive industry use case where IoT inside vehicles are communicating to services using blockchain technology. Based on use case described previously, we apply here blockchain decision-making methods to define the type of blockchain platform to use, the potential consensus to target, and thus the feasibility of an blockchain-powered automotive industry. The use case problem can be summarized into:

> $\longrightarrow$ Allow multiple independent services around the connected vehicle, to communicate and exchange information, using a trusted system that doesn't require any central trusted-third-party.

Based on the decision-making literature, we can defined the platforms to target for an exploration of suitable blockchain integration within the use case.

2.5.1 Need of Blockchain ?

First is the question of "*Do we really need a blockchain ?*", which can be deduced using Wüst et al. [111] work:

- Yes we need to store a state
- Yes there are multiple writers
- No we can't use only one online trusted-third-party
- Yes writers of the ledger are known, and no the writers are not to be trusted always
- Yes public verifiability is required

Consequently, a private permissioned blockchain is the appropriate technical solution to apply in our use case problem. This type of blockchain can be achieved using any of the permissioned blockchain presented before (Ethereum in private network, Sawtooth, Fabric, etc.). However, this indirectly means that a unique blockchain technology would have to be adopted by all the involved organizations in the use case (i.e. usage of a single blockchain, a *solo chain*).

2.5.2 Consensuses Requirements and Need of Multi-Chain

It has been shown with our previous work [179] that interorganizational digital servitization requirements would force a blockchain-based technological solution use a different architectures than a solo chain. Indeed, each organization have specific requirements (system governance, consensus, data management, etc.), thus using one blockchain to create industrial interoperability (i.e. interoperability for multiple organizations) is impossible. This is why interoperable blockchain-of-blockchain architecture is more suitable for the use case. Each service can build and manage it's own system (i.e. blockchain), participate in the ecosystem (i.e. multi-chain platform ecosystem), and be interoperable with other systems using a trustless cross-system communication (i.e. multi-chain platform cross-communication).

In this research, we pose the hypothesis that each ecosystem within our automotive industry use case will utilize a uniform blockchain technology across services. This assumption allows for an in-depth exploration of interoperability within homogenous blockchain architectures, where similar protocols and structures are employed across different entities and applications.

By focusing on a standardized technological framework (i.e. Polkadot), the thesis will uncover the challenges and implement a solution for integrating and managing IoT communications

within a multi-chain ecosystem using the Substrate standardized framework (a detailed analysis of Polkadot and Substrate framework in Chapter 5). This approach not only simplifies the complexities often associated with multi-chain interoperability, but also offers insights into the optimization of blockchain functionalities in a cohesive, technology-unified environment (i.e. case-specific blockchain development). This hypothesis serves as a foundation for evaluating the potential benefits and challenges of a streamlined, technology-consistent approach to blockchain interoperability in industrial applications.

2.6 IoT in the Context of Blockchain

2.6.1 Overview of BIoT

When we talk about the integration of Blockchain and the Internet of Things (BIoT), we're referring to the fusion of blockchain technology with IoT devices, IoT network infrastructure, or the application layer that connects IoT and blockchain. This concept isn't new; in fact, as early as 2015, IBM's Institute for Business Value was already exploring this in their report titled "Device Democracy" [180].

As highlighted by IBM, one of the fundamental advantages of blockchain technology with IoT, is its ability to provide a trustless intermediary environment. Unlike traditional systems that rely on trusted third parties like banks or centralized servers to validate transactions, blockchain's decentralized and consensus-based architecture allows IoT devices to have secure interactions without the need for a central authority.

Blockchain technology in the context of IoT can offer numerous benefits, including enhancing secure data sharing, enabling device identity verification, and ensuring the integrity of both devices and their data history. For secure data sharing, blockchain's immutable and decentralized nature provides a robust mechanism to record and verify data transactions. Each data exchange between IoT devices can be securely recorded as a block in the blockchain, preventing unauthorized access and tampering [20][60]. This ensures that the data generated by IoT devices remains trustworthy and accurate, which is particularly vital in critical applications like healthcare [54], supply chain management [125] [87], and industrial automation [58].

Moreover, integrating blockchain in IoT devices enables enhanced device identity verification. Each device can have a unique identity stored on the blockchain, eliminating the risks of counterfeit devices and ensuring that only trusted devices can participate in the IoT network [59].

The trustless nature of blockchain in BIoT adds an extra layer of security and transparency to the ecosystem, making it more resilient to potential attacks or single points of failure. This also provide participants with increased confidence in their interactions and data exchanges within the network.

In conclusion:

> *"The benefit of integrating a DLT with existing applications lies in the formation of a new distributed trust "layer". "*[181]

2.6.2 BIoT Key Related Works

Caro et al. (2018) developed a system called AgriBlockIoT to show how blockchain can be used to track food products in the supply chain. This system is focused on making the supply chain more transparent, reliable, and easy to audit. The authors talk about how technologies like RFIDs and Wireless Sensor Networks are changing the way we can monitor food conditions from production to consumption. However, they also point out that many current systems depend too much on centralized cloud infrastructures, which can create problems with transparency, security, and data management. AgriBlockIoT uses blockchain to solve these problems by providing a secure and transparent way to store information. It can work with different IoT devices and blockchain types like Ethereum and Hyperledger Sawtooth. The authors compared different blockchain types to see how they perform in terms of speed, CPU load, and network usage, providing valuable insights into their pros and cons. This work is a significant first step

in understanding how blockchain can improve the reliability of IoT systems in the food supply chain [125].

A very interesting article related to this thesis is the study by Pincheira et al. (2021) [182]. The authors focus is on the integration of constrained IoT devices with a public blockchain infrastructure to address water management in precision agriculture. The use case involves the implementation of a system architecture where constrained IoT devices, used for measuring water consumption, act as direct data-source actors on a public blockchain infrastructure. These devices interact with smart contracts that represent the interests of different water management stakeholders and regulate the distribution of incentives amongst farmers practicing sustainable water usage. The architecture is divided into three modules: the Device module, the Gateway module, and the Blockchain module. The Device module is responsible for converting sensed values, such as water consumption, into blockchain transactions. These transactions are then sent to their corresponding smart-twin counterparts in the blockchain. The Gateway module acts as a relay component between the device and the blockchain layer, ensuring the secure and accurate transmission of transactions. The Blockchain module houses all the smart contracts representing the smart-twins and the distributed application itself, providing a decentralized interface for developing IoT applications. The Ethereum network was employed as the public blockchain for the real implementation of this use case, and six different types of IoT platforms, including 8-bit micro-controllers like the ATM from the AVR family typically used on Arduino-like boards, and 32-bit boards from the ARM and MIPS families, were individually assessed for their impact on the IoT devices in terms of energy, processing time, and available memory. The findings reveal that solutions based on the proposed architecture can be implemented with only a **6% additional energy** allocation compared to the normal operations of the IoT devices. This study by Pincheira et al. is significant as it not only demonstrates the feasibility of integrating cost-effective IoT devices directly with a blockchain but also emphasizes the role of blockchain in fostering sustainability and trust in water management systems in the agricultural sector.

Andoni et al. (2019) [127] explore the potential of blockchain technology in the **energy sector**, highlighting its ability to overcome challenges and unlock opportunities. The authors systematically review blockchain's applications in energy domain, focusing on its adoption and facilitation of peer-to-peer energy trading, decentralized marketplaces, IoT applications, and its role in promoting decentralization, digitalization, and decarbonization in energy systems. Their study categorizes and reviews 140 blockchain projects and startups, exploring their relevance, potential, opportunities, challenges, and limitations in energy applications. The authors emphasize blockchain's ability to redefine digital trust, remove intermediaries, and disrupt conventional governance, contributing to process optimization and the emergence of novel business models. They highlight the growing interest of energy companies in blockchain for achieving sustainability and a low-carbon transition. The authors also point out the role of blockchain in improving operational efficiency, transparency in energy systems and markets, offering benefits to both companies and consumers. A key element in this field are the numerous IoT applications that could enable a blockchain-based energy trading smart cities, thus playing a significant role in its smart grid systems. The study underscores the necessity for blockchain to overcome challenges and market barriers to prove its commercial viability and secure mainstream adoption. Other smart energy grid studies highlight the potential of blockchain in IoT-enabled smart grid application and challenges overview [59].

2.6.3 BIoT Challenges and Limitations

The integration of IoT and DLT, such as blockchain, is gaining traction across various sectors, including but not limited to healthcare, supply chain, and agriculture. While this convergence promises enhanced security and decentralization, it also introduces a set of challenges and limitations (Figure 2.6.1). Based on related works, this section go through these challenges, aiming to provide a comprehensive understanding of the difficulties faced when merging these two technologies.

- Scalability and Performance : The scalability and performance of blockchain in IoT environments are critical concerns. The consensus mechanisms, such as Proof of Work, are resource-intensive and may not be suitable for IoT devices with limited computational capabilities [183] – however, it is not relevant in this thesis because we only target permissionned blockchains. The transaction throughput of blockchains is often lower compared to centralized solutions, potentially leading to bottlenecks in high-rate IoT environments.
- Integration Complexity and Interoperability: The integration of diverse IoT devices and platforms with various blockchain protocols necessitates addressing compatibility and interoperability issues. The heterogeneity of IoT devices and the variety of blockchain protocols can complicate the seamless interaction between different components in BIoT systems. Standardized protocols and mechanisms are essential to facilitate integration [179].
- Data Privacy and Security: While blockchain can enhance the security of IoT by providing a tamper-resistant ledger and making data sources identifiable, it also introduces new challenges related to data privacy and security. The immutable nature of blockchain raises concerns about data privacy and compliance with regulations such as GDPR. The public nature of some blockchains can expose sensitive IoT data to unauthorized entities. Network privacy and transaction confidentiality are also significant concerns [184].
- Energy Consumption: The energy consumption associated with blockchain, especially those utilizing resource-intensive consensus algorithms, is a significant concern. The first blockchain, Bitcoin [**?**], introduced in 2008, relied primarily on proof of energy consumption for its consensus mechanism. Since then, various solutions have emerged to address this issue. A blockchain node is unlikely to be running on an IoT device, but the ecological aspects of the technology are at the heart of political and environmental discussions. Moreover, the implementation of blockchain, including the choice of cryptography for transactions, directly influences the energy requirements of IoT devices when they interact with it. In essence, **IoT devices must run a 'client program' that aligns with the blockchain's operations**.
- Cost Implications: The financial cost of implementing and maintaining blockchain can be substantial, impacting the economic viability of BIoT solutions. These costs include transaction fees (in the case of public blockchains), additional energy consumption costs, and the expenses associated with the development of blockchain-compatible IoT devices.
- Adoption and Regulation: The slow pace of adoption can be explained to the perceived complexity of blockchain technology and misconceptions about its applicability. Overcoming this resistance and building trust by creating standard and interoperable systems

are essential to bring acceptance up and promoting the integration of blockchain in IoT. Additionally, the current regulatory landscape is fragmented and underdeveloped, creating uncertainties and compliance challenges for developers and organizations. The lack of universal compliance standards hinders the deployment of worldwide acceptable BIoT solutions, potentially impacting innovation and investment in this field.

- Skill Deficit: The integration of blockchain with IoT requires specialized knowledge and skills. However, very few individuals truly understand how blockchain functions, especially in conjunction with IoT. This presents hiring challenges and necessitates focused educational efforts to bridge the skill gap in this interdisciplinary field.

Addressing these challenges is pivotal for unlocking the full potential of the IoT-DLT convergence, ensuring its sustainable and secure integration into various fields. Ongoing research and development efforts are focused on mitigating these challenges and exploring alternative consensus mechanisms, enhancing interoperability, security, and developing standardized protocols and frameworks [185] [186].

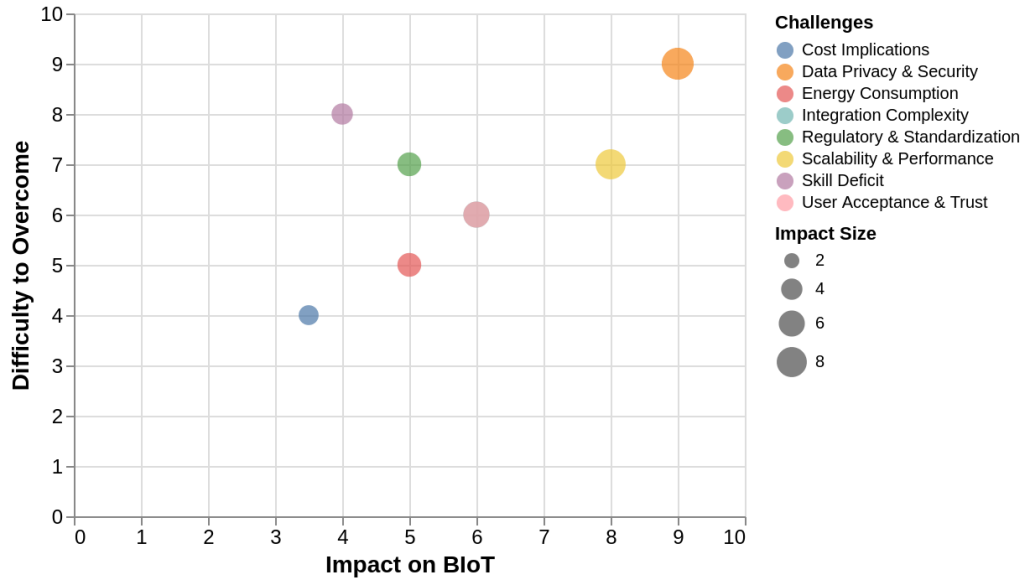


Figure 2.6.1: Difficulties in overcoming blockchain challenges are plotted based on their potential impact. Very difficult and high-impact challenges are rated 10, while low difficulty and low impact challenges are rated 0. The **scores are given by the thesis author** based on his personal judgment and comprehensive research in the field. This visual representation aids in quickly identifying and prioritizing the issues demanding immediate attention in the blockchain ecosystem.

2.6.4 Cryptography

Cryptography is the science and art of securing information by transforming it into an unreadable format, known as ciphertext, using mathematical algorithms and keys [187]. This process ensures that only authorized parties with the corresponding decryption key can decipher and access the original plain text information. The key properties of cryptography is to allow:

- Secrecy: Keep data private.
- Integrity: Ensure data remains unaltered.
- Authentication: Verify sender or recipient.
- Non-repudiation: Prevent denial of actions.

2.6.4.1 Cryptography in Blockchains

Any blockchain or DLT in general will at some point use cryptography techniques to create, manage, and secure its ledger, it is almost a pillar of blockchain technology (Security, Decentralization, and Transparency).

Asymmetric-key cryptography (or "public key") is the type of cryptography used in blockchains. It uses a pair of keys – a public key and a private key. Information encrypted with the public key can only be decrypted with the corresponding private key, and vice versa. This ensures that data remains secure and confidential. Public key cryptography is used to authenticate participants in a blockchain network. When a user wants to send a transaction or interact with the network, they can sign their data with their private key. Others can verify the authenticity of the sender by using the sender's public key.

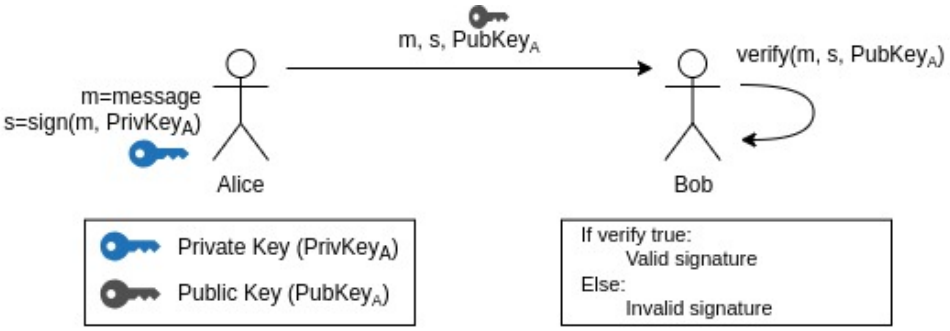


Figure 2.6.2: Asymmetric-key cryptography

Digital signatures, which are based on asymmetric cryptography, are used to verify the integrity and authenticity of transactions and data in the blockchain. As depicted in Figure 2.6.2, when a transaction is created (the message m) by Alice, it is signed with its private key ($PrivKey_A$). Anyone can use Alice's public key ($PubKey_A$) to verify that the signature (s) matches the transaction data, ensuring that the transaction has not been tampered with.

In a blockchain system, participants may not trust each other. Asymmetric cryptography allows users to keep their private keys secret while sharing their public keys. This separation of keys simplifies key management and enhances security.

As described in Table 2.6.1, the algorithms used in blockchains are various in complexity, key size, and targeted usage. In terms of security, each algorithm are proved to be secure up to a certain degree depending on the key size. We have to notice that behind security there are aspects hidden such as privacy or confidentiality. Any blockchain that uses public-key cryptography, is

Cryptographic Algorithm or Signature	Description	Common Usage	Key Size (Bits)	Year
RSA (Rivest–Shamir-Adleman)	Uses large prime numbers for secure data transmission and digital signatures	Secure communication, digital certificates	1024 to 4096	1977
ECC (Elliptic Curve Cryptography)	ECC encapsulate multiples elliptic curves (e.g. secp256k1, ed25519, Curve25519, etc.)	Ethereum and other DLT	256 to 521 (curve-dependent)	1985 (ECC)
ECDSA (Elliptic Curve Digital Signature Algorithm)	Relies on elliptic curve mathematics for digital signatures	Bitcoin, Ethereum, and other cryptocurrencies	256 to 521 (curve-dependent)	1991 (ECDSA)
EdDSA (Edwards-curve Digital Signature Algorithm)	Based on Edwards curves for efficient and secure digital signatures	Some modern blockchain implementations	256	2015
Schnorr Signatures	Offers simplicity and security in digital signatures	Taproot upgrade in Bitcoin	256	1989 (Schnorr)
Lamport Signatures	Quantum-resistant cryptographic scheme using one-time hash functions	Experimental in blockchain for quantum resistance	128 to 256	1979 (Lam-port)

Table 2.6.1: Comparative Table: Cryptographic algorithms and signatures in blockchains

pseudonymous because all transactions are signed and are identified by a public key. When some cryptocurrency is deposited on the wallet (i.e. the customer's wallet public key), and then later sends this cryptocurrency anywhere else it will be possible to route back this asset (using the intrinsic immutability of blockchain).

If a person deposits some cryptocurrency – using a centralized exchange that requires KYC[3] verification – into a wallet (using the wallet's public key) and later sends it to another wallet, it's possible to trace back the movement of this asset because of the immutable nature of blockchains. If the cryptocurrency founds are used for illicit activities, it would also be possible to know the funds owner identity because KYC information are often shared with governmental authorities for regulatory purposes [188]. Identities could also be traced back even due to leaked databases, services, or hacked websites [189].

Note: While privacy and confidentiality are crucial considerations in blockchain technology, an in-depth exploration of this topic is beyond the thesis scope given its vast and complex nature.

Hashing algorithms play a crucial role in blockchains by ensuring data integrity and security. Table 2.6.2 presents a comparative overview of various hashing algorithms commonly employed in blockchain applications.

SHA-256 and **SHA-512**, both part of the SHA-2 family, are widely used due to their balance of output size and computational efficiency. SHA-256, in particular, is the backbone of many cryptocurrencies, including Bitcoin, where it secures blockchain integrity through its proof-of-work mechanism. **SHA-3**, an evolution of the SHA family, offers a different hash function design, providing an additional layer of security and is considered more resistant to cryptanalysis than its predecessors.

Keccak, is the underlying algorithm of SHA-3, is specifically adapted for the latest Ethereum version, focused on security. **BLAKE2**, another notable hashing algorithm, focuses on high-speed performance, making it an attractive option for newer blockchain implementations seeking efficiency alongside robust security.

Hash Algorithm	Description	Output Size (Bits)	Year
SHA-256 (Secure Hash Algorithm 256-bit)	Cryptographic hash function for data integrity	256	2001
SHA-512 (Secure Hash Algorithm 512-bit)	Variant of SHA-2 with longer output	512	2001
SHA-3 (Secure Hash Algorithm 3)	A member of the SHA-3 family of hash functions	Variable (e.g., 224, 256, 384, or 512 bits)	2015 (SHA-3)
Keccak (SHA-3)	Designed for security and efficiency in Ethereum 2.0	256 (default)	2005 (SHA-3), 2020 (Ethereum 2.0)
BLAKE2	Focuses on privacy and security in hashing	Variable (e.g., 256 or 512 bits)	2012

Table 2.6.2: Comparative Table: Hashing Algorithms in blockchains

Post quantum cryptography concerns by the National Institute of Standards and Technology (NIST) [190] are strongly discussed during the last 20 years due to the fast quantum computing

[3]KYC (Know Your Customer) is a standard in the investment industry that ensures advisors can verify a client's identity, their client's investment knowledge, and financial profile.

development. The current standards and cryptographic methods are being tested and show that the impact from large-scale quantum computer makes RSA, ECDSA, ECDH, and DSA no longer secure. Thus, new algorithms (encryption and signature generation) are proposed and selected such CRYSTALS-KYBER, CRYSTALS-DILITHIUM, FALCON, and SPHINCS+ in 2022 [191]. Blockchain project will have to take in account these concerns to be quantum-resistance by design. The authors in [192] propose solution for these future type of blockchain with a 5-step end-to-end framework applicable to most blockchain networks. The authors emphasis the urge of implementing – or at least planning – quantum-resistant algorithms because we maybe can't predict the time that large scale quantum computers are capable to break current standards, but it will be very likely possible at some point in time. The survey study by Dasgupta et al. [156] confirms these concerns on the expire time limits of the used cryptography in blockchain.

2.6.4.2 Cryptography in IoT Context

One of the most evident challenges emerging from the IoT security aspect is the great increase in the number of interconnected devices, leading to a larger attack surface for potential adversaries. These devices, often designed for specific tasks, might have limited computational power, making the deployment of advanced encryption techniques challenging [193]. Moreover, unlike personal computers or smartphones that receive frequent updates, many IoT devices have prolonged operational lifetimes without regular software upgrades. This lack of updates makes them more susceptible to cryptographic attacks that might emerge in the future. Additionally, devices might be deployed in physically accessible and unsecured locations, further adding vulnerability to potential threats.

To address the unique challenges posed by IoT devices, various cryptographic solutions have been developed and adopted:

- Lightweight Cryptography (LWC): Given the computational limitations of many IoT devices, lightweight cryptographic algorithms are designed to be efficient, requiring less power and memory. They target specifically resource-constrained devices, ensuring a certain level of security and performance. The authors in [194] discuss and expose many of the LWC for the IoT. The survey provide the also the hardware and software implementation performances of LWC algorithms along the top 10 most efficient LWC algorithm depending on the targeted metric (energy, latency, memory, …).
- Physical Unclonable Functions (PUFs): These are used for device authentication in IoT. PUFs generate unique device-specific responses based on inherent physical variations in the device. They can be used to derive cryptographic keys, ensuring device authenticity without the need for stored secrets [195].
- Elliptic Curve Cryptography (ECC): ECC offers a stronger security-to-key-length ratio compared to traditional algorithms like RSA. This makes it an attractive option for IoT devices since shorter key lengths require less computational power and storage. Notably, ECC-based digital signatures can be generated and verified with reduced overhead, which is vital for IoT device performance.
- Hash-Based Signatures: For IoT devices with severe limitations, hash-based signatures might be a feasible option. They are simple, have a smaller memory footprint, and can be more resilient against quantum attacks.

Kromes et al. research [196] underscores the significance of blockchain cryptography integration within IoTs, by demonstrating how a specific hardware architecture model can accelerate the traditionally time-consuming cryptographic operations intrinsic to blockchain, thereby making it more feasible to implement this technology for constrained IoT devices applications.

The author's proposed model, developed in SystemC-TLM, is particularly notable for its ability to significantly reduce the execution time of cryptographic processes, which are essential in blockchain technology. Furthermore, the hardware-centric approach put forward by the authors addresses one of the primary concerns in IoT security: the limitations imposed by the devices' computational capacities. By optimizing the hardware to better support the cryptographic demands of blockchain technology, their model presents a path toward implementing more secure, efficient, and reliable IoT systems without overloading the devices.

However, as identified by the authors, the application of blockchain cryptography in IoT isn't without its challenges. Concerns related to the blockchain performance scalability, IoT energy efficiency, and the secure management of cryptographic keys must be addressed as research in this area continues (e.g. usage of PUFs to store the keys). Despite these obstacles, their work emphasizes the powerful advantages of integrating advanced cryptographic solutions in IoT, highlighting the potential to enhance the security and trustworthiness of interconnected devices.

2.7 Conclusion

This chapter has provided an extensive overview of the current state of the art in the domain of the Internet of Things (IoT) and Decentralized Ledger Technology (DLT), and particularly on blockchain technology. As we've explored, the landscape of IoT has grown exponentially, and is characterized by an increasingly interconnected digital world. This interconnectivity facilitates unprecedented data collection and automation capabilities, extending across various domains from personal devices to industrial applications.

Similarly, blockchain technology has shown its transformative potential in establishing trust in digital interactions, significantly influencing both technological and organizational paradigms. Beyond its well-known application in cryptocurrencies, blockchain's decentralized, transparent, and secure nature makes it suitable for numerous applications, potentially revolutionizing sectors like finance, healthcare, supply chain, and, pertinent to this discussion, the IoT sector.

Furthermore, the intersection of IoT and blockchain presents a strong argument for enhanced security, privacy, and interoperability among the abundance of IoT devices. However, this convergence is not without challenges, which necessitate further exploration and innovation to overcome issues related to scalability, practical integration, energy efficiency, and regulatory considerations.

In summary, the advancements in IoT and blockchain technology highlight a pivotal moment in our digital era. The private and public sector is currently exploring the feasibility of IoT-Blockchain integration. Numerous questions surrounding the practical implementation of this novel technology are only beginning to be addressed. The following Chapter 3, explores our specific use case, its implementation on private blockchain platforms, and multiple performance studies. The Chapter 4 discuss the practical implementation and provide key findings to improve BIoT by exploring the blockchain client integration directly on constrained electronic devices. Concurrently, the shift in our interaction and trust towards our digital ecosystem brings also the question of systems interoperability, which is discussed in Chapter 5. Interoperability in IoT devices and blockchain systems are two different concepts, and the thesis focuses rather on the BIoT interoperability by employing interoperable blockchain-of-blockchain ecosystems.

3 BIoT: Blockchains for IoT

3.1 Introduction

As we explore the complex domains of the Internet of Things (IoT) and blockchain technology, we saw that a notable integration potential is emerging. The previous chapter established a comprehensive foundation, examining the distinct, yet interconnected realms of IoT and blockchain, with a particular focus on interoperability.

However, the focus of this chapter and the next one, shifts toward the practical of this integration. This convergence represents more than just a combination; it's an strategic orientation designed to address fundamental BIoT challenges and limitations we expressed (Section 2.6.3) and unlock new opportunities (with a focus mainly on *scalability and performance*, *interoperability*, and *energy consumption*).

The practical implementation of BIoT remains largely in the proposal or PoC stage, with several critical aspects still unaddressed for industrial-level BIoT integration. In this chapter, we embark on an in-depth exploration, scrutinizing the key components of BIoT. This involves identifying the specific requirements of an industrial use case and ensuring these needs are adequately met on both the blockchain side (i.e., blockchain's intrinsic performances), and – in the next chapter – the IoT side (i.e., IoT computational performance, communication methods, etc.). This chapter is about first creating a common smart contract and then extracting specific blockchain performances for a practical context use case.

ℹ Please note that at this stage of the thesis, our exploration will primarily center on the automotive use case at the single-organizational level. The Chapter 5 will expand this scope to include multiple organizations, offering a macroscopic overview of BIoT's full potential.

3.2 SIM Use Case Description

Most of the thesis revolves around the Smart IoT for Mobility (SIM) automotive use case. In this section, we describe this project in detail and explain how it will provide a pathway that is sufficiently bounded to establish an environment to explore BIoT solutions, blockchain, and infrastructure architecture design.

3.2.1 In-Depth SIM Use Case Requirements

For this chapter, we'll need to know the use case requirements in more detail: *Who and How are the interactions between actors ? What, How, and How much data is exchanged ?*
To avoid misinterpretation, in the context of blockchain and for the rest of the thesis we differentiate throughput into three categories: the blockchain **transaction input throughput**, the blockchain **transaction output throughput**, and the **network throughput** (for read-only data).
The *data writes*, or input transaction throughput, refers to the rate at which transactions that record or modify data are generated and submitted to the blockchain.
In contrary, the *data reads*, or network throughput, is the retrieval rate of data from the blockchain that does not necessitate the creation of a new transaction.
The output throughput is the rate that data is processed by a blockchain, a metric used only to study the results of blockchain performances (more described in Section 3.4.1.2).

3.2.1.1 Vehicle: Data Writes

To ensure that the accident use case is managed effectively, the vehicle's generated data rate needs to be estimated carefully. A smart interconnected vehicle will generate various data points—such as speed, location, mechanical status, and more (around 140 data fields based on an Android Automotive OS specification [197]).

ℹ We consider "transaction" also as a unit. A transaction is an digital object containing multiple fields such as a nonce, public key, signature, and a message. The message is a generic element that can be a payment, a call to a smart contract, or anything else. This field is arbitrary, thus the transaction data length vary from one blockchain to an other. To provide a level of abstraction, we keep "transaction" as a unit an won't use the "bit" unit.

To estimate the input transaction throughput we will only consider that a transaction is sent for the following types of events:

- Critical events: Such as accidents which must be recorded instantly. The accident record is stored on the blockchain.
- System updates: Firmware or software updates that change the vehicle's operation or capabilities. A trace of the updates is inserted on the blockchain.
- Regular maintenance records: Including scheduled services, part replacements, and general maintenance activities. We consider that all these interactions with the vehicle should be recorded on the blockchain (i.e. Smart Service Book).

For each of these events, a blockchain transaction must be created to ensure the data's immutability and traceability. The following Table 3.2.1 describe the average transaction emission rate for each type of events, for a vehicle.

Event Type	Estimation Basis	Average interaction with blockchain
Critical events	Road traffic accidents	$50,000$ per year
System updates	Manufacturer's schedule	$(4+12) \times N_{vehicles}$ per year
Regular maintenance records	Service intervals	$(1+10) \times N_{vehicles}$ per year

Table 3.2.1: Average transaction emission rates for different vehicle event types with concrete estimation

Traffic safety reports from French road safety observatory (ONISR) sets the average road traffic accidents per year around 50 000, with a car fleet in France of around 40M ($= N_{vehicles}$) [198][199].

System updates are not daily occurrences, they can depend on the manufacturer's schedule or critical updates that are required. Let's assume there's a major update quarterly and minor updates monthly.

Regular maintenance typically occurs at set intervals, such as every 10,000 km or once a year, whichever comes first. Let's assume an average vehicle drives 15,000 km per year. Additionally, the maintenance will replace parts and initiate checks that would create transactions, we will estimate this to 10 interactions with the blockchain that create a transaction each.

The total average transaction throughput for vehicles in France can be calculated by summing the transactions for critical events, system updates, and regular maintenance records. Given that $N_{vehicles}$ represents the total number of vehicles and that the car fleet in France is approximately $40M$ vehicles, we get the Equation 3.2.1.

$$\begin{aligned} \text{Annual critical events} &= 50,000 \\ \text{Annual system updates events} &= (4+12) \times 40M \\ \text{Annual maintenance events} &= (1+10) \times 40M \\ \text{Annual average total Writes} &= 1,080,050,000 \text{ transactions} \end{aligned} \tag{3.2.1}$$

We then normalize this to get the average transaction throughput by dividing the total annual throughput by 365 (days) or 31536000 (seconds) in Equation 3.2.2.

$$\begin{aligned} \text{Average blockchain input throughput} &= \frac{1,080,050,000}{365} \\ &= 2,959,041.10 \text{ transactions/day} \\ &= 34.25 \text{ transactions/second} \\ \text{Average blockchain input throughput (critical-only)} &= \frac{50,000}{365} \\ &= 136.99 \text{ transactions/day} \\ &= 0.0016 \text{ transactions/second} \end{aligned} \tag{3.2.2}$$

These values represents an estimation of the average interactions (per second) of a car fleet has with the blockchain for our use case. Given the growth in the vehicle industry and the

digitization of processes, 34.25 transactions/second is a relatively low estimation and could easily be 10 times this value (340 transactions/second).

While the critical-only events input throughput accounts for a mere fraction of the overall transaction volume, this does not reduce the *accidents* value importance.
Car manufacturers and OEMs consider not only the necessity of recording critical events but also a broader **potential for data monetization and other purposes**. The vast amount of vehicle data points, beyond accidents, offers a rich source for analytics and business intelligence, which is central to the automotive industry's evolving revenue models. By leveraging all the vehicle data in a blockchain environment, manufacturers can unlock new opportunities for service offerings, customer insights, and operational efficiencies. This view highlights why using blockchain for more than just accident data makes sense for car manufacturers: it **opens up new ways to make money and improve services** by using all the data cars generate.

3.2.1.2 Vehicle: Data Reads

For managing the data generated by vehicles effectively, the network's capability to handle large-scale read operations must also be addressed. We define, *data reads* interactions that encompasses the rate of data retrieval from the blockchain, which does not change the blockchain's state but is critical for the operation of smart vehicles and related services.

ℹ Similarly to transactions, we consider "reads" as a unit. The "read" doesn't have a proper unit because it totally depends on the retrieved data: The insurance could be querying an 32 bit identifier (ID) or an entire table of IDs.

Data reads, in the context of the accident use case includes:

- Verification of accident data by insurance companies
- Real-time access to vehicle status by mechanics or emergency services
- User requests for vehicle history or status updates
- Regulatory compliance checks and audits

Given the high dependability on real-time data for immediate services and safety assurance, the blockchain architecture must support high data read throughput. For the purpose of our analysis, we will consider both synchronous and asynchronous reads. Synchronous reads happen in real-time and are usually initiated by an immediate need for information, such as during an accident verification process. Asynchronous reads are scheduled or occur as background processes, for instance, regular compliance checks.

The following hypothetical model (Table 3.2.2) shows how the demand for data reads might be distributed across different services for a single vehicle.

With assumptions based on the average vehicle usage patterns and service requirements, we can estimate the data reads throughput a blockchain has to support. However, unlike write operations, **reads can be off-chain using archives database of the indexed blockchain**, allowing for greater scalability and less strain on the blockchain network. It is also possible to scale read performances by adding blockchain nodes that are light-nodes or peer-only nodes, containing a live copy of the ledger. It is relatively unnecessary to explore the read performances in this thesis because it relates more on network performances and very large database management (in the case of a centralized blockchain-indexed database). The study by Kalajdjieski et al. shows an overview of the possible databases that can be used to index and process blockchain data [200].

Service Type	Read Frequency	Estimated Reads per Year
Insurance Verification	After accidents	$2 \times N_{critical\_events}$ (Let's consider just 2 Insurance data reads from the blockchain, could be much more.)
Vehicle Status Access	Daily operational checks	$365 \times N_{vehicles}$
Vehicle History Requests	User-driven, irregular	Assumed $5 \times N_{vehicles}$ per year
Compliance Checks	Regular authority audits	Assumed $2 \times N_{vehicles}$ per year

Table 3.2.2: Estimated read demands for different services related to a single vehicle

However, to grasps the necessity for read-performances in blockchain, we illustrate, assuming an average car fleet in France of $40M$ vehicles, the annual read operations for various services would result in Equation 3.2.3.

$$\begin{aligned} \text{Annual reads for Insurance Verification} &= 2 \times 50,000 \\ \text{Annual reads for Vehicle Status Access} &= 365 \times 40,000,000 \\ \text{Annual reads for Vehicle History Requests} &= 5 \times 40,000,000 \\ \text{Annual reads for Compliance Checks} &= 2 \times 40,000,000 \\ \text{Average Total Reads} &= 14,700,100,000 \text{ reads} \end{aligned} \tag{3.2.3}$$

When normalized to per second we obtain Equation 3.2.4.

$$\begin{aligned} \text{Average blockchain read throughput per second} &= \frac{14,700,100,000}{31,536,000} \\ &= 466.31 \text{ reads/second} \end{aligned} \tag{3.2.4}$$

This projected estimations underscores the necessity for high-performance blockchain solutions capable of supporting both the high write and read demands of the use case. All blockchains have an internal database to store and manage transactions, blocks, and other state related information. However, a blockchain encapsulates many other components (as we have seen in the blockchain OSI model). In opposition, "barebone" databases are specifically optimized only for data management, thus their performances in data writes/insert are better than blockchains but can have similar performances in data reads (Bergman et al. [201] comparison of latencies in permissioned blockchains and distributed databases). To draw a contrast, databases like MySQL rely on a combination of factors including disk I/O, memory usage, and query optimization for read/write performances [202], blockchains predominantly employ a consensus mechanism for data writing, which introduces inherent latency related to consensus.

Furthermore, the estimated use case reads is also indicative of the network infrastructure requirements that must be in place to sustain the data flow, emphasizing the need for blockchain infrastructure designs that are both efficient and scalable enough to meet the strict requirements of the automotive industry.

3.2.1.3 Vehicle IoT Device Constraints

From the electronics point of view, the SIM project has to consider the operational constraints imposed by the hardware within these interconnected vehicles. These constraints primarily include computational capacity, power consumption, memory size, and network connectivity, which are all critical for the successful integration and performance of blockchain technology in an IoT environment.

The vehicles are equipped with various sensors and processors that continuously collect data and perform computations. However, the devices used in such vehicles are often limited in terms of processing power and energy availability, given the need to optimize for lower power consumption and longer operational periods without compromising on performance.

- **Computational Capacity:** The blockchain clients will perform cryptographic operations essential for securing transaction data. Optimizing the software to run these operations within the client's resource limits is vital to prevent overloading the vehicle's computing hardware. It is also recommended to use specific hardware resources for secure cryptographic operations.
- **Power Consumption:** The blockchain clients must be designed to be energy-efficient. Since vehicles have limited energy reserves, it's important for client operations to conserve power to maintain the vehicle's overall battery life.
- **Memory Size:** The client software will require memory for transaction operations and potentially for storing transaction histories in some cases. Given the memory limitations typical of vehicle IoT devices, the client's storage footprint must be minimized.
- **Network Connectivity:** The clients need to maintain a constant and secure connection to the blockchain network for transaction submissions and data retrieval. The challenges of maintaining connectivity in a dynamic driving environment require robust network management within the client software.

The blockchain integration in the automotive IoT must be adapted to embedded systems with a focus on lightweight client design. This involves creating blockchain client applications that provide essential features such as transaction creation and signature verification.

In previous work part of the SIM project, R. Kromes created and developed an IoT architecture model dedicated to blockchain applications [203]. Kromes emphasizes the necessity for a harmonized architecture that accommodates the limited computational and energy resources of IoT devices while ensuring secure and reliable blockchain operations.
The proposed architecture made in SystemC-TLM modeling language takes advantage of dedicated hardware accelerators specifically designed to handle the cryptographic operations inherent to blockchain technologies. These accelerators are tailored to execute hashing, encryption and digital signature algorithms, more efficiently than general-purpose processors. By offloading the most resource-intensive tasks to specialized hardware, the architecture seeks to balance the trade-off between performance and power consumption, which is critical in the automotive IoT landscape.

3.2.1.4 Vehicle Data Automation

Vehicle data automation in the context of the SIM project refers to the implementation of smart contracts for automating the processes related to vehicle data management using a blockchain network. In the current phase of this thesis, three principal entities are considered:

the OEM, which refers also to the car manufacturer, the vehicle itself, and the driver operating the vehicle.
The design of smart contracts includes several functions, including a basic authentication procedure and management of the automobile fleet, as well as the tracing and storage of accident reports. It is assumed that the car manufacturer will be responsible for registering identities within their system. Consequently, the smart contract is endowed with methods that determine actors or vehicles additions to the smart contract's state.

The permission architecture adopted throughout this chapter is described as follows: The "Blockchain admin" is conceptualized as either a human or a software tool with the authority to delegate permissions to a manufacturing entity. A "factory" within this system has the capability to enroll car identities into the smart contract's state. A "car" has the ability to execute methods within the smart contract to report accidents. Lastly, the "driver" identity (i.e. public key) is associated to the accident when they are reported.

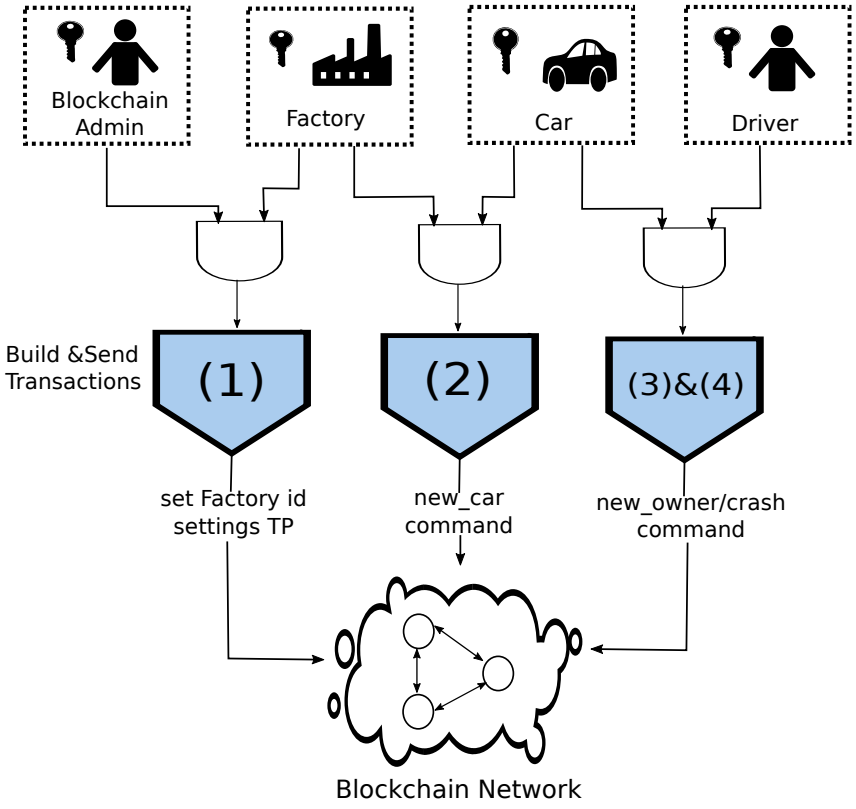


Figure 3.2.1: Hierarchical permission architecture for smart contract interaction

The Figure 3.2.1 depicts the permissioned architecture within the smart contract framework of the SIM project. It shows the Blockchain admin at the top level, who has the authority to assign permissions to factories. The factories, in turn, have the responsibility of registering vehicle identities in the smart contract state. Cars are authorized to use smart contract methods for accident declaration, and the driver's identity, indicated by a public key, is recorded in association with the accident reports. This structure ensures a clear definition of roles and responsibilities for secure and efficient management of vehicle data transactions on the blockchain. This architecture constituted the initial step towards realizing the use case for smart contracts and implemented in our work [204], which aimed to establish a genuinely decentralized implementation of the accident use case. We achieved the first proposal using Hyperledger Sawtooth blockchain

software and IPFS decentralized storage solution. Additionally, specific modules were developed to facilitate the integration of blockchain with the storage system.

Moreover, without relying on vehicle images, video cameras, or data from radar or LIDAR systems, the quantity of information generated approximates to 128 kbits [205]. While this amount may appear negligible, a simple calculation reveals that the annual increment in storage is considerable. Given the estimated data production, the annual storage increase can be calculated as follows: $128\text{kbits} \times 50,000 = 6.4$ Gbit/year. Adding only an 1Mbits image will increase the data storage requirement by $1\text{Mbits} \times 50,000 = 50$ Gbit/year, thus on-chain storing for the entire vehicle fleet ($40M$) is unrealistic.

Now that we defined the landscape of the use case requirements which the thesis focuses on, we will now proceed to the blockchains technologies used to explore the practicability of automotive blockchain-based vehicle and vehicle accident management.

3.3 Main Studied Blockchains

3.3.1 Hyperledger Sawtooth and Fabric

Hyperledger Sawtooth and Hyperledger Fabric are two prominent frameworks within the Hyperledger suite, each tailored for specific enterprise blockchain applications.

Hyperledger Sawtooth is a permissioned blockchain framework, known for its modularity and flexibility [84]. It supports various consensus algorithms, like Practical Byzantine Fault Tolerance (PBFT) and Proof of Elapsed Time (PoET), and allows for the customization of its core components. Sawtooth's architecture, featuring a modular validator and transaction processors, focuses on security, scalability, and adaptability, making it well-suited for diverse business requirements (see Figure 3.3.1).

In contrast, Hyperledger Fabric offers a unique approach to enterprise blockchain solutions. Fabric's architecture is highly modular and includes features such as channels for private transactions and chaincodes for smart contracts [83]. This design enhances scalability, privacy, and performance in business contexts. Fabric's permissioned nature allows for controlled network access, and its pluggable consensus mechanism caters to varied operational needs.

In Sawtooth, the validator plays a critical role in transaction validation and maintaining a uniform state across all nodes. Similarly, Fabric employs a modular architecture, where components like identity services, ledger storage, and consensus mechanisms can be customized. Chaincode in Fabric, analogous to smart contracts, allows for business logic encapsulation and can be deployed independently on different channels.

Sawtooth's consensus engine, a separate module, works alongside the validator to manage consensus processes. Sawtooth also offers a REST API for external communication, contrasting with Fabric's diverse communication protocols that cater to different network and client requirements.

Hyperledger Sawtooth's adoption of Intel SGX for its PoET consensus mechanism offers a secure and efficient alternative to the computational demands of Proof of Work algorithms. However, security concerns around Intel SGX have prompted ongoing research and updates [206]. Hyperledger Fabric distinguishes itself with also a pluggable consensus feature, offering enterprises the flexibility to select and tailor consensus mechanisms to fit their requirements (Kafka, Raft, or a specific ordering service in Fabric v2.x).

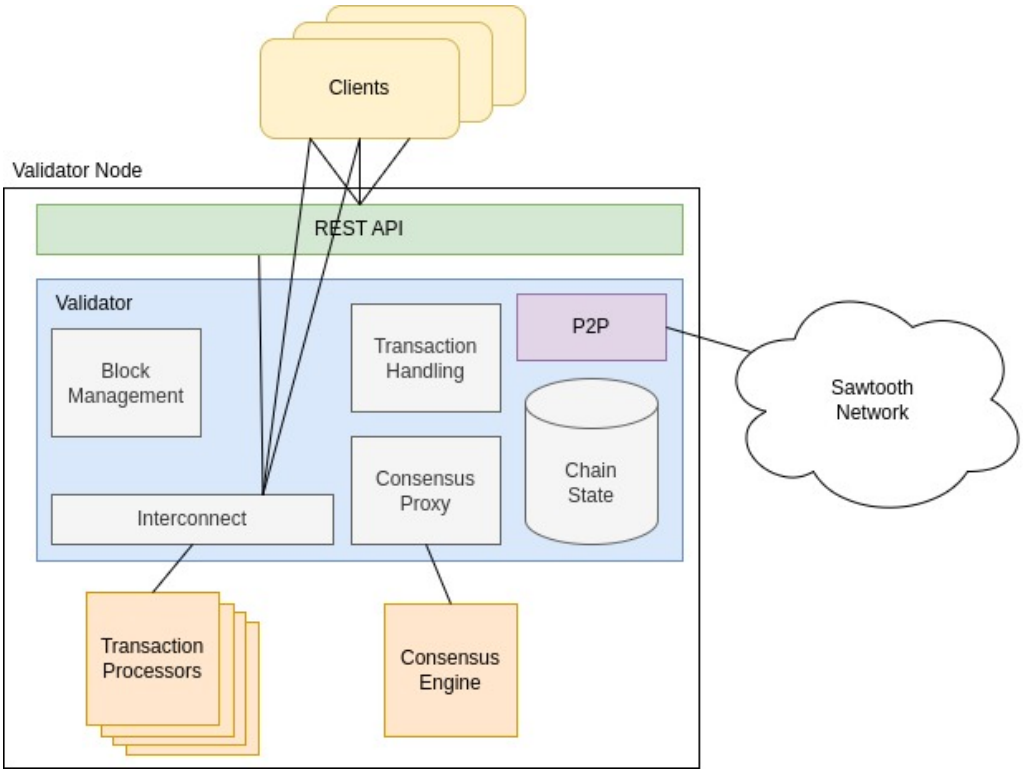


Figure 3.3.1: Hyperledger Sawtooth node and network structure

Both Sawtooth and Fabric emphasize transaction processing efficiency, with Sawtooth utilizing transaction processors for specific transaction types and Fabric using chaincodes for smart contract implementation. Fabric achieves efficiency through its unique 'execute-order-validate' architecture, which allows transactions to be processed before the final network-wide agreement, thus reducing latency and improving throughput by parallelizing transaction execution and consensus.

3.3.2 Ethereum

Ethereum is a public, open-source blockchain platform that features smart contract functionality [85]. It is widely recognized for its role in pioneering decentralized applications (dApps) on the blockchain. Ethereum operates on a global scale and is not confined to private or enterprise use, making it fundamentally different from permissioned blockchains like Hyperledger Sawtooth.

The public Ethereum platform is powered by its own cryptocurrency, Ether (ETH), which is used to perform various functions within the network, such as compensating participating nodes for computations performed. The inherent flexibility of Ethereum's programming language, Solidity, allows developers to write and deploy a wide range of smart contracts and dApps that can automate complex transactions and create decentralized autonomous organizations (DAOs).

However, Ethereum can be deployed in a private network configuration, which allows it to be used for enterprise applications requiring privacy and permissioned access. This versatility makes Ethereum adaptable for both public and private use cases. Within a private context, Ethereum allows the creation of specialized smart contracts and permissioned processing flows that are optimized for the specific performance and security requirements of the enterprise.

The *Geth* implementation (Figure 3.3.2) of Ethereum is one of the most popular for its cus-

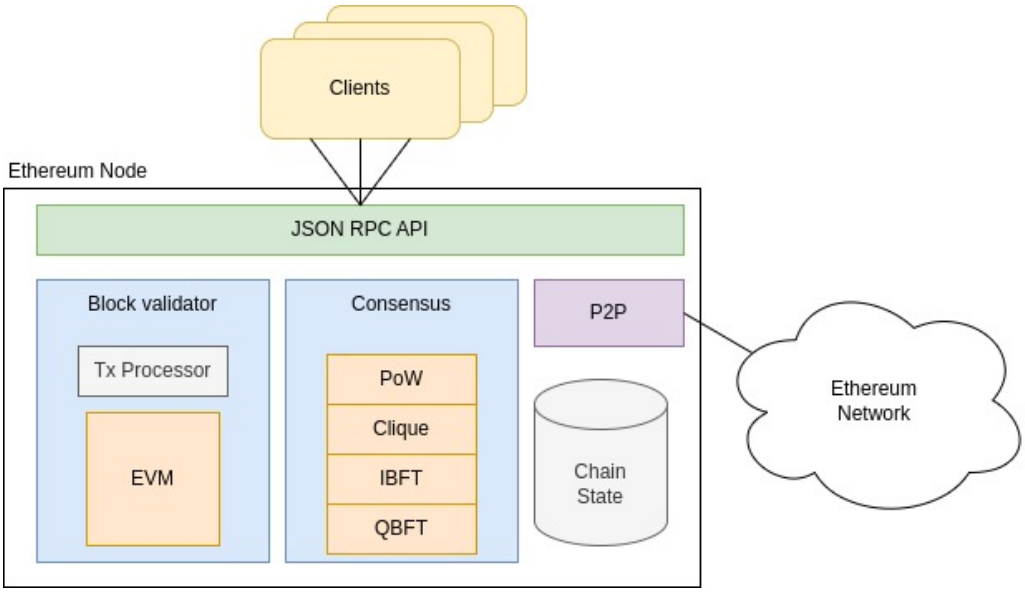


Figure 3.3.2: Generic Geth/Besu Ethereum node structure: Possible to choose PoW, Clique, IBFT, or QBFT consensus

tomization options and control features [85]. It provides the functionality to participate in the public Ethereum network or to launch a private network with a different consensus mechanism, such as Proof of Work (PoW), Clique (for proof of authority systems). With Hyperledger Besu [86], implementation of Ethereum it is possible to choose Clique, Istanbul Byzantine Fault Tolerance (IBFT), or Quorum Byzantine Fault Tolerance (QBFT) consensuses.

Geth and Besu encompasses a full Ethereum node that validates blocks, manages the blockchain, and ensures data consistency across the network. In a private network, both Ethereum implementations allows for significant customization, enabling enterprises to define their own governance policies, transaction verification protocols, and network authority mechanisms. The choice of consensus in a private network impacts its performance and security, with Clique and IBFT offering faster block times and reduced computational overhead compared to the traditional PoW used on the public Ethereum network.

The *Geth* implementation (Figure 3.3.2) of Ethereum stands out as one of the most favored due to its wide range of customization options, community support, and control features [85]. It enables users to either join the public Ethereum network or establish a private network that operates with a distinct consensus mechanism, including Proof of Work (PoW) and Clique (for proof of authority systems). Furthermore, with the Hyperledger Besu [86] version of Ethereum, users have the flexibility to select from Clique, Istanbul Byzantine Fault Tolerance (IBFT), or Quorum Byzantine Fault Tolerance (QBFT) as their consensus mechanisms.

Both Geth and Besu offers blockchain node capabilities that are crucial for block validation, blockchain management, and maintaining data consistency across the network. When deployed in a private network setting, these Ethereum implementations allows extensive customization. This adaptability can give organizations the possibility to have specific options their governance models, transaction verification processes, and network authority structures. The consensus mechanism chosen for a private network directly influences its performance and security. Notably, Clique and IBFT are known to facilitate quicker block times and lower computational demands in comparison to the conventional PoW mechanism that governs the public Ethereum network.

3.3.3 Substrate-Based Blockchain

In the evolving landscape of blockchain technology, Substrate emerges as a revolutionary framework that empowers developers to design customized blockchains tailored to specific needs. This innovation is particularly crucial in the context of application-specific blockchains, which demand a high degree of customization to efficiently serve their intended purposes.

Substrate was developed by Parity Technologies, the same team that created the Polkadot network, with the intention of providing a robust framework for blockchain innovation [207]. It is intricately linked to Polkadot through its native compatibility with the network's interoperability protocol, allowing Substrate-based blockchains to seamlessly connect with Polkadot and other chains in the ecosystem. This relationship enables shared security and inter-chain communication, positioning Substrate as an essential tool for developing the next generation of interoperable blockchain systems.

Substrate is a cutting-edge Software Development Kit (SDK) that offers a rich set of tools for developers to build blockchains that can operate independently or within the shared security ecosystem of the Polkadot network. The core of Substrate's innovation is found in its modular design, enabling developers to create blockchains with an extraordinary level of freedom.

The architecture of a Substrate-based blockchain is inherently modular, comprising of several components (i.e. *pallets*) to deliver a fully functional and customizable blockchain solution [138]. At the heart of this architecture lies the concept of pallets, which are the core elements of the Substrate runtime.

Runtime Composition and Pallets:
By using the FRAME (Framework for Runtime Aggregation of Modularised Entities) environment, developers can select and configure pallets. Substrate's runtime is an aggregation of pallets/modules, interchangeable pieces that encapsulate a set of functionalities or business logic. These pallets range from essential system functions to specialized business logic like asset management and consensus mechanisms. Depicted in Figure 3.3.3, each pallet can be seen of as a reusable component that can be selected and configured to fulfill specific roles within a new blockchain core logic. There are around 80 pallets available in the FRAME SDK, however many parachain projects and open-source organizations have created many more application-specific pallets [208].

For instance, the Balances pallet manages the ledger of account balances within the blockchain, while the Timestamp pallet records the time at which transactions occur. Together, and combined with others, these pallets could form the backbone for one new blockchain's operational logic, providing the essential services that allow the blockchain to function effectively.

General overview of a Substrate-based blockchain node:
As depicted in Figure 3.3.4, a Substrate-based blockchain node can be represented into two primary components: the *core client* and the *runtime*.

- **Core Client and Outer Node Services**: The client acts as the operational shell of a Substrate node, coordinating with outer node services to manage network activities and maintain the blockchain's connectivity and consistency. These services include P2P, transaction pool, consensus-building, and execution of RPC calls.

  Key activities performed by client services include:
  - **Storage:** A key-value store to persist blockchain state data.

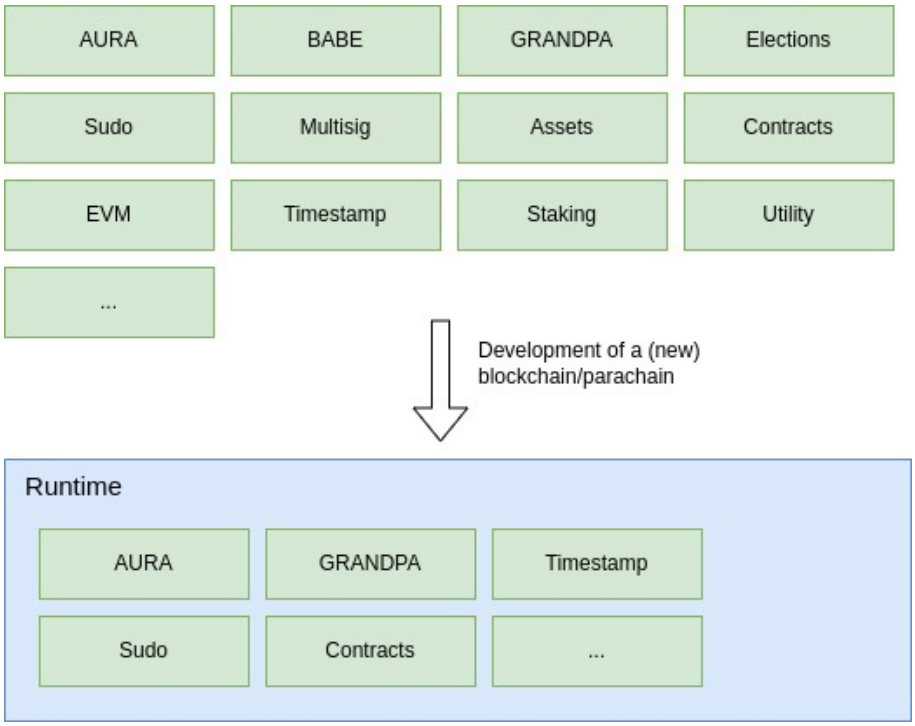


Figure 3.3.3: Substrate pallets: Used to create the blockchain runtime

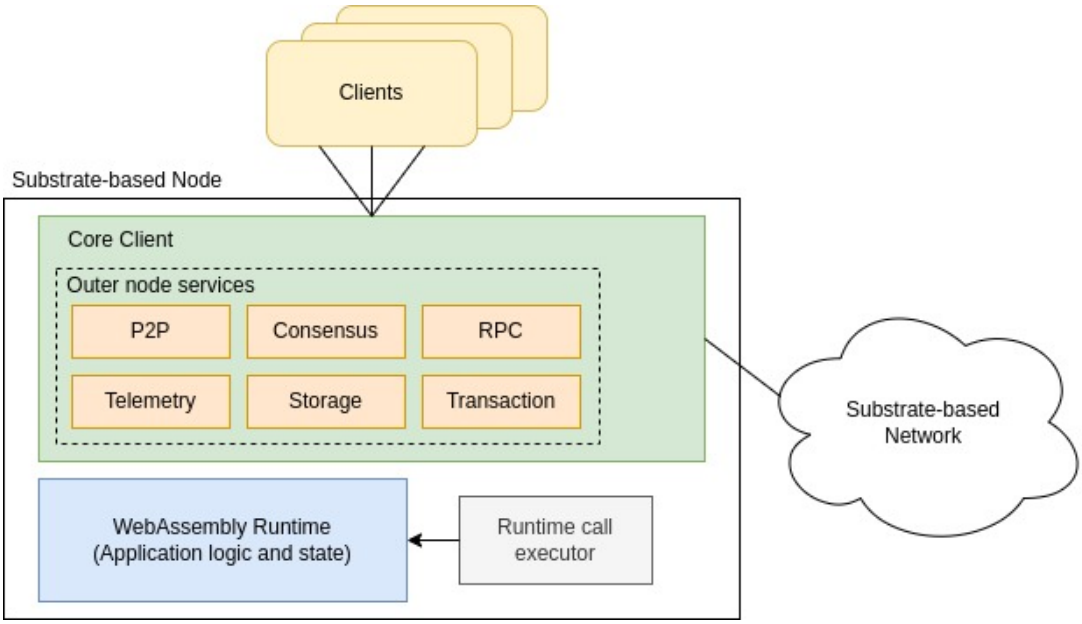


Figure 3.3.4: Substrate-based blockchain node structure

- **Peer-to-peer (P2P) networking:** Using libp2p library, for a decentralized and resilient mesh of nodes.
- **Consensus:** The outer node communicates with other network node participants to ensure agreement of the blockchain state.
- **RPC API:** Facilitates interactions with the blockchain through HTTP and WebSocket protocols, allowing external clients and developers to query and interact with the network.
- **Telemetry:** Collects and shares node performance metrics.

- **Execution environment:** Chooses between WebAssembly or native execution for the runtime, dispatching dispatching calls to the selected runtime.

- **The Runtime**: The runtime is the component that executes the state transition function of the blockchain. It validates transactions, implements changes to the state, and determines the blockchain's evolving logic. The runtime is responsible for all on-chain related executions, processing all transactions and storing the resulting state in a format ready for retrieval and verification by the client services.

  Substrate's runtime encapsulates the blockchain's business logic within a Wasm byte code environment, stored on-chain which this enables several features:
  - **Forkless Upgrades:** Permits the upgrade of blockchain functionality without network forks.
  - **Multi-platform Compatibility:** Ensures the runtime can operate across different computing environments due to the portability of Wasm.
  - **Runtime Validity Checking:** Offers an intrinsic mechanism to verify the integrity and correctness of the runtime code.
  - **Validation Proofs:** Facilitates consensus on the Polkadot relay chain by providing proof of state transitions.

Intercommunication between the client and runtime is achieved through runtime APIs, which enable the client to pass requests to the runtime. This framework for blockchains is a highly appreciated solution for this thesis: it allows customization, by reusing core blockchain components for faster and seamless development/testing. Existing standard in the framework allows to also implement cross-communication with compatible blockchains.

3.3.4 EOSIO

EOSIO is a blockchain platform designed for the development and operation of decentralized applications (dApps). It was developed by Block.one and first released in June 2018 [209].

ℹ EOS.IO project has been discontinued by Block.one mid-thesis, in the year 2021. A fork of the ecosystem and platform by maintainers, called Antelope, has been done in an attempt to continue development and provide support for existing applications [210].

EOSIO's architecture is characterized by its emphasis on scalability and efficiency, which is fundamentally different from typical blockchain structures. This is achieved through a combination of unique elements defined bellow.

Consensus Mechanism: Delegated Proof of Stake (DPoS): At the core of EOSIO's architecture is its Delegated Proof of Stake (DPoS) consensus mechanism. This system elects block producers (BPs) through a community-driven voting process, where EOS token holders cast their votes to select a limited number of BPs. This mechanism ensures high transaction throughput and reduced latency, making EOSIO highly scalable.

Free Transaction Model and Resource Staking: EOSIO introduces a novel approach to transaction costs, allowing users to interact with dApps without direct fees. Instead, it utilizes a resource staking model where users stake EOS tokens to access network resources (CPU, NET, and RAM). This model promotes a more user-friendly experience compared to traditional transaction fee models.

Governance Structure: EOSIO's governance is community-driven, with token holders having a say in network decisions, including BP elections and protocol updates. This democratic approach fosters a more engaged and participatory community, though it has also been a topic of debate regarding centralization concerns.

EOSIO Node Overview: The EOSIO node (Figure 3.3.5), **nodeos**, is the core software of EOS.IO blockchain node. Depending on its configuration it can handle smart contracts, validate transactions, and produce and confirm blocks. Around *nodeos* are **plugins**. Some plugins are mandatory, such as *chain_plugin* (process and aggregate chain data on the node), *net_plugin* (provides an authenticated P2P protocol to synchronize nodes persistently), and *producer_plugin* (loads functionality required for a node to produce blocks). Plugins allow the EOS node to be modular. Unlike Hyperledger Sawtooth, EOS node plugins are compiled and packet in *nodeos*. Partitioning software features allows building different blockchain nodes for different purposes.

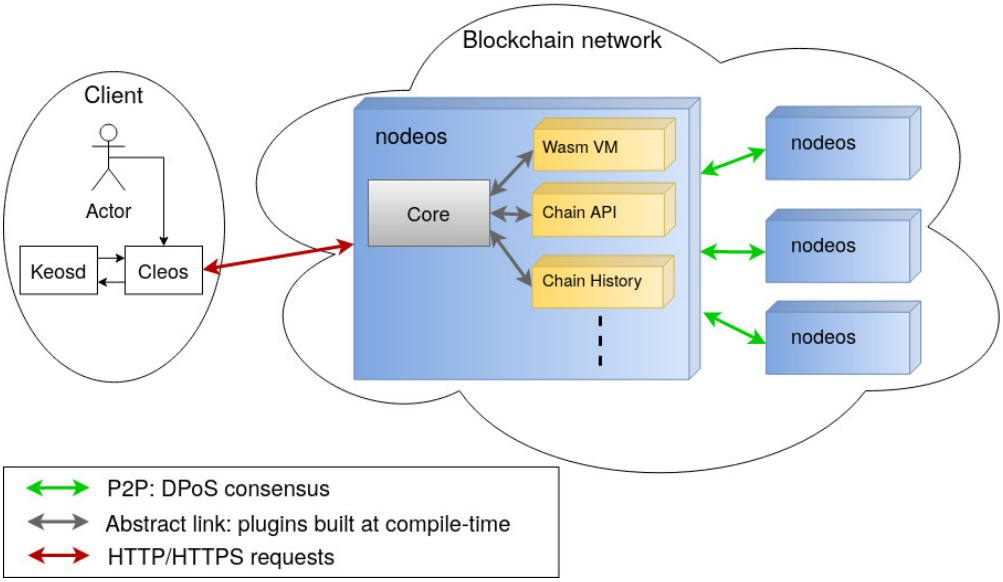


Figure 3.3.5: EOSIO blockchain node structure

A standout feature of EOSIO is its high transaction throughput. Thanks to its Delegated Proof of Stake (DPoS) consensus mechanism and the efficiency of block producers, EOSIO can handle thousands of transactions per second (TPS), significantly outperforming many traditional blockchains. This scalability is essential for applications requiring fast and reliable transaction processing, such as financial services, gaming, and social media platforms.

The innovative resource management system in EOSIO, where users stake EOS tokens to access network resources (CPU and NET), and purchase RAM, presents a unique approach. This system not only ensures equitable resource allocation but also incentivizes token holding and network participation. By eliminating per-transaction fees, EOSIO enhances user experience and accessibility, particularly for frequent transactions.

Smart Contracts: EOSIO's smart contracts supports various programming languages (Rust, Go, Python, ...), with C++ being the primary one (called also EOS++ in their ecosystem), allowing developers to build complex and efficient contracts. The contracts are compiled in WebAssembly (Wasm) and generate an ABI to interface with it (Figure 3.3.6).The smart contract Wasm is executed in the Wasm VM.

EOSIO has a similar smart contract design to Ethereum, using a blockchain layer inside the node core that execute contract code in a virtual machine (VM). The difference is the contract and VM execution language.

EOSIO has an implementation of Ricardian Contract (cf. Section 2.3.9.5) features when the plugin is compiled with *nodeos*. The Ricardian Contract is written in a plain text file using a template specification to include variables and conditions in the contract document. It is in the on-chain ABI file that the Ricardian Contract template is stored.

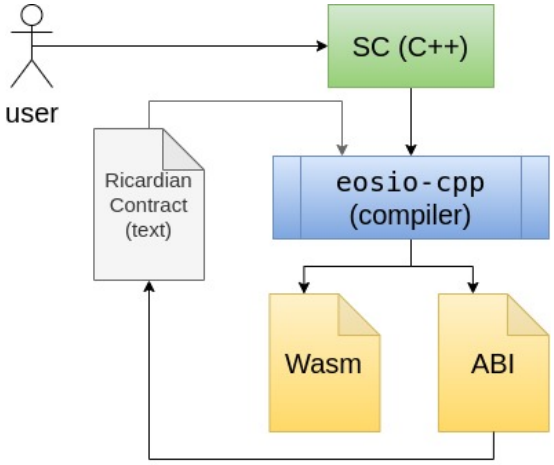


Figure 3.3.6: EOSIO Smart Contract Compiler: Wasm & ABI file

⚠ An important notice is that ABI can be bypassed when executing transactions. Messages and actions passed to a contract do not necessarily have to conform to the ABI. The ABI is a guide, not a gatekeeper.
From this notice, we deduce that when displaying the compiled Ricardian Contract template to a user, the data displayed is **for information purposes only**. It does not take part of the agreement coded in the SC.

3.3.5 Discussion: Selected Blockchains in Smart Mobility and the Automotive Industry

The studied blockchain platforms each possess unique attributes that align with specific requirements and challenges of smart mobility in smart cities and the automotive industry. This alignment is critical for selecting the appropriate blockchain technology for specific applications within these domains.

Hyperledger Sawtooth and Fabric: These permissioned platforms are ideal for environments where data privacy and controlled access are paramount, such as in collaborative automotive sectors involving OEMs, suppliers, and service providers. Hyperledger Sawtooth's modularity adapts well to various throughput and latency needs in automotive applications, while Fabric's private transaction channels and smart contracts are suitable for intricate supply chain management and manufacturing processes.

Ethereum: Ethereum stands out as a versatile blockchain platform for smart mobility in smart cities, due to its configurable nature (public/private/consortium), standardization, and robust community support. Its standardized development environment, primarily through Solidity smart contracts, has become a benchmark in the blockchain domain, facilitating easier adoption and ensuring reliability and compliance, which are crucial for example in the automotive sector.

Additionally, Ethereum's expansive and active community offers a lots of resources, shared knowledge, and collaborative innovation, driving continuous improvement and technological advancement. This robust ecosystem and flexible framework make Ethereum suitable for a range of smart mobility applications.

Substrate-Based Blockchain: The high customization potential of Substrate is beneficial for smart city-specific blockchain solutions, enabling integration across diverse services, participants, transport modes and IoT devices. Its interoperability, especially with a relay chain (Polkadot), enhances integration among various urban services, essential for comprehensive smart city management. With an appropriate design, Substrate can provide the ground layer for any specific blockchain application. The framework already has shown its efficiency in parachain projects to tackle specific issues, including one BIoT use case [211].

EOSIO's Legacy: Despite its discontinuation, EOSIO's architecture, designed for scalability and efficiency, serves as a model for platforms needed in high-throughput and real-time data processing applications, which are crucial in mobility scenarios. Its scalability, enabling the handling of vast volumes of real-time IoT data, can potentially improve the realm of BIoT applications. This scalability, and with its low transaction fees, makes it a potential economical choice for BIoT applications. Additionally, the Ricardian Contract feature had lot of potential that would allow the legal world to interface with the technical world.

At its inception, EOSIO's innovative resource management approach held the promise of propelling blockchain technology into a new era of adoption. However, history has shown that the platform's overall trajectory has not aligned with this initial enthusiasm.

3.4 Blockchain Performances

The performance of blockchain technology is critical to its effectiveness and to fit the use case with applications demanding high throughput and rapid data exchange in smart mobility of smart cities. This section will explore how performances of blockchain are measured and what metrics are commonly used to study and evaluate blockchain systems.

3.4.1 How is Blockchain Performance Measured?

Blockchain performance measurement involves quantifying the efficiency and effectiveness of a blockchain system in executing its intended functions: creating a decentralized, immutable distributed ledger. It typically includes various metrics that reflect the system's ability to process transactions, handle data storage, manage network communication, and execute smart contract code. Performance is not only about speed and throughput but also about the system's reliability, security, and scalability, which are tightly linked to the used consensus mechanism.

3.4.1.1 Transaction Latency

The transaction latency is the delay from the moment a transaction is submitted to when it is first acknowledged by the blockchain network. The term "acknowledged" in latency can be interpreted as the delay a transaction takes to: get in the transaction queue (unconfirmed or un-finalized) or to be finalized in the chain ledger (becoming immutable).

3.4.1.2 Transaction Throughput

Transaction throughput measures the blockchain's capacity to process a certain number of transactions within a given timeframe, generally quantified as transactions per second (TPS).

Input vs Output TPS: In the thesis we differentiate the transaction per second that are sent to a blockchain system (Input TPS) and the transaction per second processed by the blockchain (Ouput TPS). A transaction *processed by the blockchain* means that the transaction has been included in a block and that the block is finalized.

3.4.1.3 Transaction Finality

Finality in blockchain refers to the point at which a transaction can no longer be altered or reversed. Finality time is the measure of the time taken from when a transaction is submitted to when it is considered final and immutable. This metric is especially pertinent in scenarios requiring assured and timely settlements, providing a real-time and deterministic behavior. It is similar to transaction latency when measuring the delay of a finalized transaction.

As expressed by V. Buterin [154], the scalability trilemma depends on how much of security and decentralization you are willing to sacrifice. Thus, the consensus which controls these two properties of a blockchain has a major impact on its performances. Additionally, other blockchain properties or configuration options will have an impact and drastically change the chain behavior.

3.4.2 Blockchain Properties which Impacts Performances

The performance of blockchain systems is intrinsically linked to their underlying properties and configurations. These configurations properties, including consensus parameters, network design, and system governance, can have profound effects on the efficiency and scalability of a blockchain.

3.4.2.1 Choosing the Consensus and Its Parameters

Consensus parameters play a fundamental role in the performance of a blockchain. They define the rules and processes through which nodes agree on the state of the blockchain. Different consensus algorithms, such as Proof of Work (PoW), Proof of Stake (PoS), and Proof of Authority (PoA), each have distinct performance implications. PoW, for instance, is known for its high energy consumption and slower transaction finality compared to PoS or PoA, which offer faster processing times and lower energy requirements.

However, it is important to notice that some PoA algorithms are possible to ”tweak”, such as the number of nodes, round duration (or time interval), sync duration, or even ban duration for inative miners.

ℹ Only private consensus rules are studied in the context of this thesis, thus we will mostly look at PoA algorithms.

3.4.2.2 Network Size and Node Distribution

The size of the blockchain network and the geographical distribution of its nodes can significantly affect performance. Larger networks with nodes spread across vast distances may experience increased latency due to the time required for data to travel/spread between nodes. However, such distribution can also contribute to improved security and resilience against localized network failures. In private blockchain networks, the number of nodes highly impacts the performances. This is due to the algorithms which are BFT variants and require a high number of exchanged messages (e.g. PBFT has $\mathcal{O}(n^2)$ with $n =$ number participating nodes) between nodes to reach consensus.

3.4.2.3 Block Size and Block Interval

Block size and block interval are crucial parameters that determine transaction throughput. Larger block sizes can hold more transactions but may lead to longer propagation times across the network, potentially increasing the risk of forks. Conversely, shorter block intervals can increase throughput but may also result in higher rates of orphaned blocks and chain reorganizations. This trade-off between block size and interval is depicted in Figure 3.4.1. We arbitrarily define an approximate ”safe zone” when block interval and block size have appropriate values (both not to high or low).

3.4.2.4 Network Bandwidth and Latency

The physical infrastructure of a blockchain, such as network bandwidth and internet latency, limits the speed with which information is disseminated across nodes. High bandwidth and low latency networks are essential for maintaining a high-performance blockchain, especially when handling large volumes of transactions.

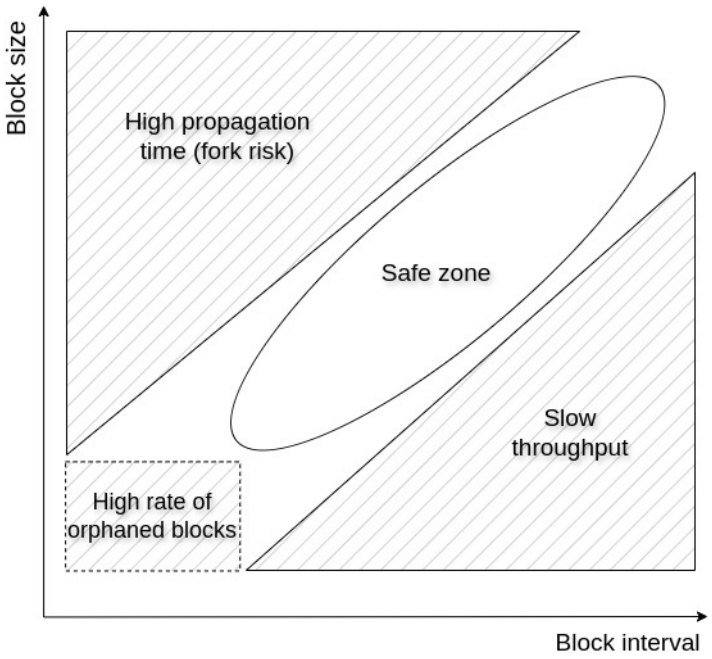


Figure 3.4.1: Block Size vs Block Interval: The generated block interval have to be spaced enough so the consensus won't create orphaned blocks and the block size can't be to high or the network latency will cause consensus forks.

3.4.2.5 Storage and Computation Limits

Blockchain systems are also constrained by the limits of storage and computation, especially for nodes that must store a full ledger and validate transactions. As the ledger expands, the requirements for storage space and computational resources increase, potentially affecting the system's performance and its ability to scale. To mitigate these challenges, pruning and sharding are two approaches that are employed, though they must be applied with caution to avoid compromising availability. Pruning involves removing all historical data up to a specific point, thus maintaining the efficiency of the node by decreasing the size of the ledger. Sharding, on the other hand, requires splitting the ledger into smaller, more manageable pieces that are distributed across the network. This technique improves scalability by enabling nodes to handle only a portion of the total data (i.e. distributing storage).

3.4.2.6 Smart Contract Complexity

The complexity of smart contracts has a direct effect on the performance of the blockchain. Contracts that are more complex necessitate greater computational resources for execution. This can lead to slower transaction processing times and higher operational costs. The complexity of these contracts contributes to the overall block size and the intervals between blocks. The duration allocated for the creation of a block includes the time required for executing these smart contracts. In this context, different approaches to transaction execution, such as "smart" or "fee priority based" methods, can be implemented to manage how transactions and smart contracts are processed within the given block creation time frame.

3.4.3 Benchmark Model and Testing Protocol

3.4.3.1 Benchmark Model for the Accident Use Case

In the thesis work, multiple benchmarks are created to push the limits of different DLT softwares. A general methodology is used to benchmark each blockchains, depicted in Figure 3.4.2: create a contract, deploy it, set initial values in the blockchain, and finally do the test. The blockchain will either probed live by a database or data is retrieve after the test.

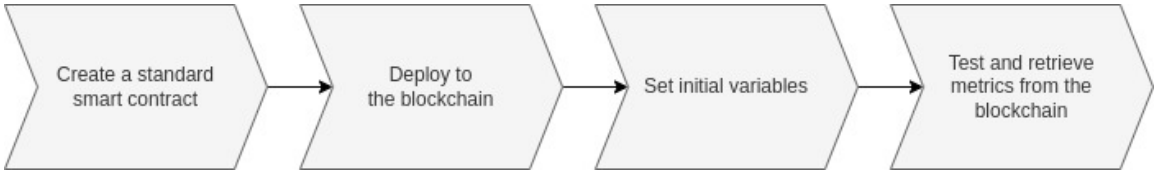


Figure 3.4.2: General benchmark methodology design to test the blockchain performances

The smart contract business logic that we started to describe in Section 3.2.1.4 is applied in the testing of the benchmark. The Algorithm for such contract is described in Algorithm 1.

3.4.3.2 Accident Use Case Smart Contract Storage

When designing the basic contract functionalities, we observe that the smart contract requires at least the following storage tables: *Factories*, *Vehicles*, *VehiclesStatus*, *AccidentCount*, *IpfsStatus*, and *Accidents*.

It is important to detail the complexity of each storage table to understand the impact on the overall contract performance. The storage design and access patterns are critical for ensuring that the blockchain remains efficient under different operational loads.

- **Factories[]** This table stores information about each factory, including its identifier and the block number when it was registered. The access pattern is typically write-heavy during the initial phase as new factories are set up and read-heavy when operations related to vehicles are performed.
- **Vehicles[]** The vehicles table keeps records of individual vehicles, linking them to their creating factory and the block number of their creation. The performance is influenced by write operations when new vehicles are registered and read operations during vehicle queries and updates.
- **VehiclesStatus[]** VehicleStatus is an index of the current state of vehicles, such as ‘active‘ or ‘non-active‘. The table is updated when the vehicle is created and put on market and only if there is a change in the vehicle’s status.
- **Accidents[]** This table logs each reported accident’s data hash. It is a critical component for applications needing traceable and unalterable historical data.

We now deduce, from the pseudo-code (Algorithm 1) and the contract’s storage details, the execution complexity on storage operations for each of the contract methods:

Algorithm 1 Smart contract pseudo-code for managing manufacturer's vehicles

```
1: function CREATE_FACTORY(origin, factory_id)
2:     Input: The origin (must be root/admin) and the factory ID
3:     Output: A new factory entry in the blockchain state
4:     Ensure origin is root/admin
5:     if factory_id is not stored then
6:         Store factory_id in Factories with current block number
7:         Emit 'FactoryStored(factory_id, origin)' event
8:     else
9:         Throw 'FactoryAlreadyStored' error
10:    end if
11: end function
12: function CREATE_VEHICLE(origin, vehicle_id)
13:    Input: The origin (factory account) and the vehicle ID
14:    Output: A new vehicle entry in the blockchain state
15:    Ensure origin is a signed factory account
16:    if vehicle_id is not stored then
17:        Store vehicle_id in Vehicles with factory account and current block number
18:        Emit 'VehicleStored(vehicle_id, origin)' event
19:    else
20:        Throw 'VehicleAlreadyStored' error
21:    end if
22: end function
23: function INIT_VEHICLE(origin, vehicle_id)
24:    Input: The origin (vehicle account) and the vehicle ID
25:    Output: Vehicle status set to initialized
26:    Ensure origin is a signed vehicle account and matches vehicle_id
27:    if vehicle_id exists then
28:        Set vehicle status in VehiclesStatus to initialized
29:        Emit 'VehicleInitialized(vehicle_id)' event
30:    else
31:        Throw 'UnknownVehicle' error
32:    end if
33: end function
34: function REPORT_ACCIDENT(origin, data_hash)
35:    Input: The origin (a vehicle account) and the data hash of the accident
36:    Output: A new accident entry in the blockchain state
37:    vehicle_id ← ensure_signed(origin)
38:    Ensure that the vehicle_id is registered
39:    count ← Get the accident count for vehicle_id or start at zero if none exist
40:    accident_key ← Create an accident key using the vehicle_id and accident count
41:    Ensure that the accident_key is not already used to store another accident
42:    Store the data_hash in Accidents with the accident_key
43:    Set the IPFS status for the data_hash as unchecked (0)
44:    Increment the accident count for vehicle_id by one
45:    Emit 'AccidentStored(data_hash, vehicle_id, count)' event
46: end function
```

$$\begin{aligned}\text{create\_factory}() &= \mathcal{O}(1\ \text{read} + 1\ \text{write})\\ \text{create\_vehicle}() &= \mathcal{O}(1\ \text{read} + 1\ \text{write})\\ \text{init\_vehicle}() &= \mathcal{O}(1\ \text{read} + 1\ \text{write})\\ \text{report\_accident}() &= \mathcal{O}(2\ \text{read} + 3\ \text{write})\end{aligned}$$

3.4.3.3 Accident Use Case Smart Contract Code Implementation

The translation of the accident use case into code can be implemented using multiple programming languages. Solidity, one of the most common languages for smart contracts, is used for Ethereum-based contracts (using the most widespread contract execution environment: the EVM). The following code, presented as Code 3.1, is an example of how this use case might be implemented in Solidity.

Code 3.1: Solidity Coded Accident Use Case Smart Contract

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;

/**
* @title VehicleManagement
* @dev Vehicle Management for accident use case
* @custom:dev-run-script scripts/deploy_with_ethers.ts
*/
contract VehicleManagement {
  // State variables
  address public admin; // root/admin address
  mapping(address => uint256) public factories; // Factories table
  mapping(address => Vehicle) public vehicles; // Vehicles table
  mapping(address => bool) public vehicleStatus; // VehicleStatus table
  mapping(bytes32 => bytes32) public accidents; // Accidents table
  mapping(address => uint256) public accidentCount; // Accident count for each vehicle
  // Struct to keep vehicle information
  struct Vehicle {
    address factoryId;
    uint256 creationBlock;
  }
  // Events
  event FactoryStored(address indexed factoryId, address indexed origin);
  event VehicleStored(address indexed vehicleId, address indexed origin);
  event VehicleInitialized(address indexed vehicleId);
  event AccidentStored(bytes32 dataHash, address indexed vehicleId, uint256 count);
  // Modifiers
  modifier onlyAdmin() {
    require(msg.sender == admin, "Only admin can perform this action.");
    _;
  }
  constructor() {
    admin = msg.sender; // Set the deployer as admin
  }
  // Functions
  function create_factory(address factory_id) external onlyAdmin {
    require(factories[factory_id] == 0, "Factory already stored.");
    factories[factory_id] = block.number;
    emit FactoryStored(factory_id, msg.sender);
  }
  function create_vehicle(address vehicle_id) external {
```

```
42      require(msg.sender == admin || factories[msg.sender] != 0, "Only factories can create
        vehicles.");
43      require(vehicles[vehicle_id].factoryId == address(0), "Vehicle already stored.");
44      vehicles[vehicle_id] = Vehicle({factoryId: msg.sender, creationBlock: block.number});
45      emit VehicleStored(vehicle_id, msg.sender);
46    }
47    function init_vehicle(address vehicle_id) external {
48      require(vehicles[vehicle_id].factoryId != address(0), "Unknown vehicle.");
49      require(msg.sender == vehicles[vehicle_id].factoryId, "Vehicle does not match origin.");
50      vehicleStatus[vehicle_id] = true;
51      emit VehicleInitialized(vehicle_id);
52    }
53    function report_accident(address vehicle_id, bytes32 data_hash) external {
54      require(vehicles[vehicle_id].factoryId != address(0), "Vehicle is not registered.");
55      require(vehicleStatus[vehicle_id], "Vehicle is not active.");
56      // Create a unique key for the accident
57      bytes32 accident_key = keccak256(abi.encodePacked(vehicle_id, accidentCount[vehicle_id]));
58      require(accidents[accident_key] == 0, "Accident already stored for this vehicle and count.
        ");
59      // Store the accident data
60      accidents[accident_key] = data_hash;
61      // Increment the accident count
62      accidentCount[vehicle_id]++;
63      // Emit the accident stored event
64      emit AccidentStored(data_hash, vehicle_id, accidentCount[vehicle_id]);
65    }
66    // Helper functions
67    function isVehicle(address vehicle_id) external view returns (bool) {
68      return vehicles[vehicle_id].factoryId != address(0) && vehicleStatus[vehicle_id];
69    }
70    function getAccidentCount(address vehicle_id) external view returns (uint256) {
71      return accidentCount[vehicle_id];
72    }
73    function getAccidentData(address vehicle_id, uint256 count) external view returns (bytes32) {

74      bytes32 accident_key = keccak256(abi.encodePacked(vehicle_id, count));
75      return accidents[accident_key];
76    }
77    // Function to change admin (additional helper function)
78    function changeAdmin(address newAdmin) external onlyAdmin {
79      admin = newAdmin;
80    }
81  }
```

There are many other possible implementations using other blockchain smart contract languages or even smart contract equivalent features (e.g. different storage model).

The Hyperledger Sawtooth blockchain has a unique model of transaction processor (TP) that allows to build our business logic in many languages (Python, Java, ...). The TP is running in a containerized environment or directly in parallel of the validator. Under these circumstances, the code is also versioned: it can be upgraded for continuous development and improvement of the smart contract.

In a Substrate-based blockchain, the Solidity smart contract (Code 3.1), could be implemented in two ways:

- Using traditional smart contract: Solidity if the blockchain is EVM compatible or in Ink! language contract if using Substrate-native contracts.
- Or by creating an application specific *pallet*, based on the same business logic.

ℹ The implementation of the use case into a specific pallet has also been made for a Substrate-based blockchain (in Rust language). The pallet code can be found in Appendix 8.2.3.

3.4.4 Private Blockchain Performances Result

In this section, we present the findings from a series of performance tests aimed at evaluating the practicality and efficiency of Byzantine Fault Tolerant (BFT) consensus algorithms within private blockchain networks (more in Section 2.3.6.1). These tests are essential in understanding how different BFT-implementing blockchain platform – Hyperledger Sawtooth, Ethereum, and Substrate – handle the custom build accident use case to judge their real-world applicability. Specifically, the aims of the performance tests are as follows:

- Explore the practical implementation of the use case in each blockchain software, providing a concrete measure of their feasibility and deployment.
- Test the transaction processing limits of a given blockchain network to gauge their maximum throughput and identify any potential bottlenecks.
- Study the influence of the number of nodes on the blockchain's transactions per second (TPS) performance, offering valuable insights into the scalability and network resilience of these blockchain solutions.

The results of these tests are intended to offer an in-depth understanding of how BFT consensus can be optimized for private blockchain applications, setting a benchmark for future developments in the field.

3.4.4.1 About Blockchain Network Deployments

Following the methodology laid in previous work [212], the deployment uses a Kubernetes cluster composed of six cloud instances – partitioned into three master nodes and three worker nodes – to manage the blockchain network. Rancher orchestrates the cluster operations, providing a streamlined management layer that simplifies the deployment and scaling process. An overview of the infrastructure architecture is in Figure 3.4.3.

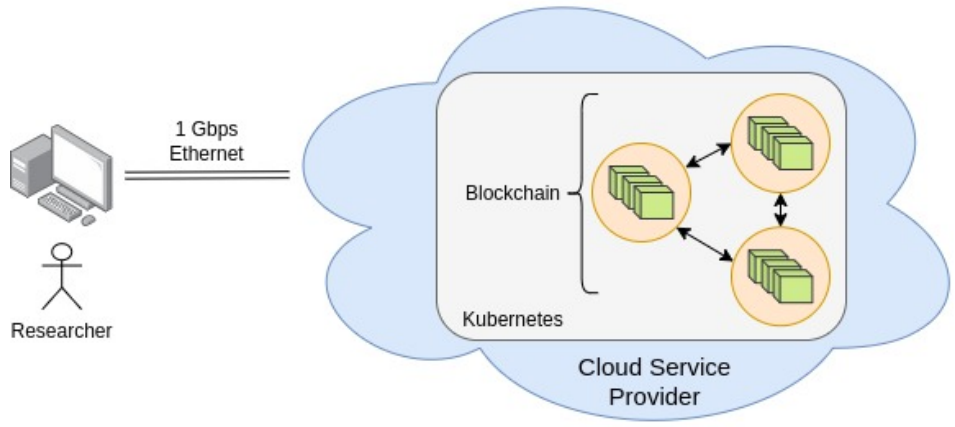


Figure 3.4.3: Cloud deployment infrastructure architecture

The worker nodes, designated to run the blockchain network, are collectively composed of 282 GB RAM and 192 CPU cores, distributed evenly across them, with each worker equipped with 94.2 GB of RAM and 64 CPU cores. To reduce latency due to cluster management tools,

a significant amount of resources is provided to Kubernetes masters. The master nodes are allocated with 48 GB of RAM and 24 CPU cores in total.

Each instance is initially equipped with 50 GB for general storage. An additional 500 GB of HDD storage is allocated per worker node, increasing each node's total storage capacity to 550 GB, specifically to accommodate the ledger's storage requirements for validator transactions.

To simulate a real-world scenario of transactions flow, such as vehicle accidents reports, we created a customized set of benchmarking programs. These programs are designed to generate and send transaction data, effectively mimicking the behavior of vehicles transmitting accident reports to the blockchain at a specific rate.

This setup provides a thorough evaluation of the blockchain's performance, reflecting real-world conditions. The established infrastructure demonstrates the applicability, scalability and feasibility of blockchain deployment for real-life scenarios.

3.4.4.2 Benchmark Program Input TPS Generation and Data Collection

To prevent misunderstanding the configuration of benchmarks and the method used to generate metrics, we describe the programming and data collection methods.

3.4.4.2.1 Benchmark Program Input TPS Generation: Due to its heterogeneity, each blockchain benchmark requires to develop a corresponding client application (detailed analysis of the client applications in Chapter 4). To generate consistent results, the benchmark has to send transactions at a constant TPS rate to the system under test. This constant rate is achieved by creating a benchmark program where two situation are encountered: 1) single threaded method and 2) multi-threaded method. The benchmark was made using Node.js, which behind the scenes use a JavaScript runtime environment that processes incoming requests in a loop, called the event loop. JavaScript is a scripting language and does not create a compiled program. Node.js is using non-blocking I/O calls allowing asynchronous code to be executed. This detail is import to explain the two methods and their limitations.

1) Single-thread program + asynchronous code: For relatively small transaction rate – under 100 transactions per second – the program can use asynchronous single-threaded method, using a loop function with a 10 milliseconds wait time in each iterations. In other words, the Node.js event loop is fast enough to handle the non-blocking I/O calls from the script, thus allows to process such transaction rate. However, when trying to send at faster rates, a single iteration will take multiple milliseconds (e.g. 5ms) and thus reaching a threshold of $\frac{1}{0.005} = 200$ iteration per second. Simply put, a single iteration will thus take longer then the delay time to send more then 200 transaction per second. To prevent this issue, multi-threaded method has to be applied for higher TPS.

2) Multi-thread program: When generating multiple hundreds to thousands of TPS the entire benchmark program has to be multi-threaded to distribute the workload (without sharing memory or blocking code by preparing TPS distribution beforehand) and prevent code execution latency in the loop iterations.

By spawning multiple threads that can produce 100s of TPS, the global program could achieve for example: 20 threads $\times$ 100 TPS $= 2k$ TPS. The number of threads is tightly linked to the machine's hardware executing the benchmark because the number of CPU and CPU cores. The impact of other applications on the machine can also reduce the "true" maximum multi-threading capabilities.

In the case of blockchain performances, Hyperledger Sawtooth benchmark have used the first method using single-thread program and asynchronous code because results shows that Output TPS reach a limit that doesn't require improving the benchmark with more TPS. However, in the case of Ethereum and Substrate-based, multi-threading benchmark program was applied due to their higher performances.
Note: For our benchmarks, the performance we get from Node.js is sufficient. However, a more fast and optimized language such as Rust or C/C++ creates a compiled program that could create much more powerful benchmarks.

3.4.4.2.2 Data collection of each blockchain tests is similar by scanning all blocks and transactions included after the tests. Data inside blocks is the only real information that we are interested in because network throughput is highly dependent to the software and architecture used which could create additional buffering or transaction pools (i.e. *mempool* or *non-final pool*).

3.4.4.3 Hyperledger Sawtooth

Figure 3.4.4 illustrates that Hyperledger Sawtooth achieves a maximum throughput of 25 transactions per second with a 4 node network. Expanding the network demonstrates a clear performance trade-off: transaction speed is reduced to 13 transactions per second as the number of nodes increases. This performance degradation can be explained in two ways: due to the constraints of the PBFT consensus mechanism at scale and the single-threaded nature of Sawtooth's core (version 1.2 at time of testing), which relies heavily on Python for significant parts of its operation.

The latency optimizations carried out using the Rust language have streamlined transaction processor execution, yielding some of the lowest latencies possible. However, the potential for consensus configuration to significantly elevate transaction rates is limited, as confirmed by discussions with Hyperledger developers. The throughput bottleneck is largely attributed to the 'sawtooth-core' limitation in handling block production, which is currently bound by single-thread execution.

These findings suggest that the performance issues of Sawtooth are not a result of hardware under-utilization but are inherent to the software's current architectural state. With the forthcoming Sawtooth version 2.0, we anticipate a higher transaction throughput because of the expected re-coding in Rust with multi-threading capabilities.

Additionally, this work on cloud-based deployments of Hyperledger Sawtooth show significant performance improvements compared to earlier benchmarks on local computer setup, highlighting the benefits of using cloud infrastructure for blockchain applications [204] [212].

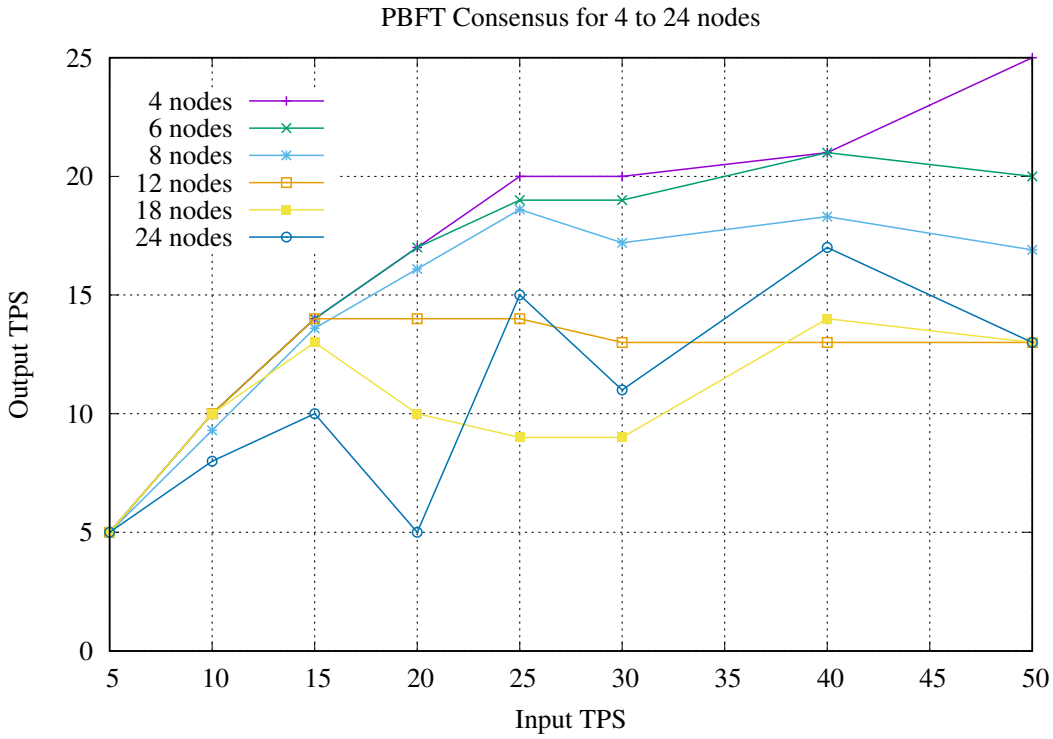


Figure 3.4.4: Hyperledger Sawtooth PBFT Consensus Performance

3.4.4.4 Ethereum

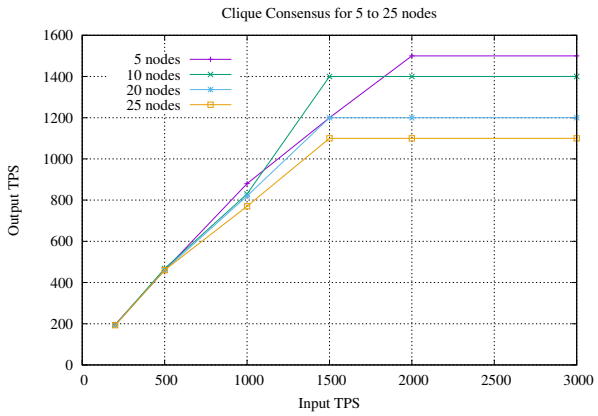
Ethereum, as a pioneering platform for smart contracts and decentralized applications, has been experimented with various consensus algorithms to balance between decentralization, security, and scalability. Three noteworthy consensus mechanisms adopted in Ethereum's diverse ecosystem are Clique, Istanbul Byzantine Fault Tolerance (IBFT), and Quorum Byzantine Fault Tolerance (QBFT). These algorithms are specifically tailored for private or consortium network contexts, emphasizing fault tolerance and performance efficiency.

Clique, a Proof of Authority (PoA) based algorithm, stands out for its reduced computational complexity and improved transaction time, making it suitable for networks where participants are semi-trusted entities. IBFT and QBFT, both adaptations of the classic BFT consensus mechanism, offer improvements over Clique in terms of finality and consistency, ensuring that once transactions are committed, they cannot be reversed, which is paramount for use cases requiring high levels of trust and data immutability.

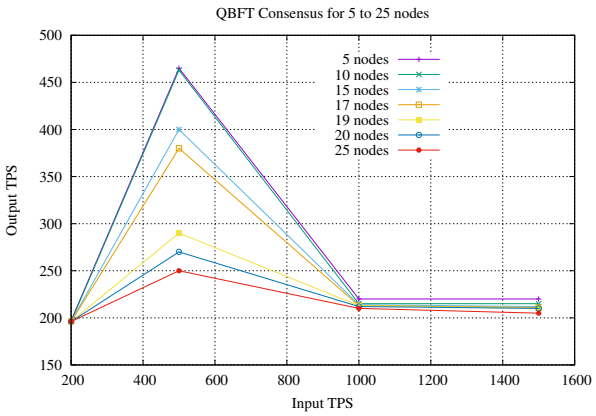
Simply put, the key insight on Clique, IBFT (Istanbul BFT), and QBFT (Quorum BFT) tests with Ethereum are:

- **Clique (Proof of Authority - PoA):** We find that Clique's throughput increases with the number of nodes until reaching a saturation point, where the output stabilizes at approximately 1500 TPS for an input of 2000 TPS (Figure 3.4.5a). This performance ceiling is attributed to the limitations of the Ethereum Virtual Machine (EVM). Despite its high throughput, Clique shows lower consistency due to the frequency of forks and chain reorganizations within the network (see paragraph 3.4.2.3 for explanation).

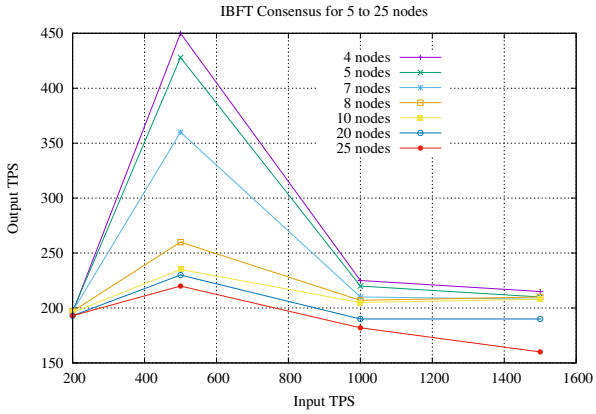
- **IBFT and QBFT:** Our performance measurements of IBFT reveal an optimal throughput of



(a) Ethereum Clique Consensus Performance



(b) Ethereum QBFT Consensus Performance



(c) Ethereum IBFT Consensus Performance

around 460 TPS for an input rate of 500 TPS, with performance decreasing as more nodes participate in the consensus due to computational overhead and increased message communication (Figure 3.4.5c). Meanwhile, QBFT maintains similar throughput levels (Figure 3.4.5b) without the stalling issues observed in IBFT, demonstrating improved algorithmic efficiency and network resilience. Both algorithms experience a performance drop at higher input rates, more pronounced for IBFT at 1000 TPS input.

- **Performance vs. Nodes:** We observe a geneal trend where increasing node participation decreases transaction throughput for Ethereum blockchains.
  With 10 or more participating nodes, all consensuses Output TPS decrease drastically. The worst case is for IBFT where already at 7 nodes, the Output TPS drops more than 20%. In QBFT, the network can have up to 15 nodes with only 13% Output TPS decrease. Clique Output TPS drops 20% with a network of 20 nodes.

3.4.4.5 Substrate-Based Custom Blockchain

A Substrate-based blockchain is created specifically for our use case, with two pallets (i.e. core modules) called *sim_renault* and *sim_renault_accident*. This blockchain isn't only one consensus because Substrate distinguishes consensus block authoring (called AURA) and finality (called GRANDPA). The Output TPS is extracted from the included – finalized – chain block data. The resulting benchmark, depicted in Figure 3.4.6, shows a similar decrease in Output TPS when

increasing the number of nodes as in Sawtooth and Ethereum platforms.

Using Substrate, we observe a maximum Output TPS of 476 at an Input TPS of 600 transaction per second using a network configuration of 4 consensus participating nodes. The combination of Figure 3.4.6 and Figure 3.4.7 show the importance of the network configuration for high throughput using Output TPS and the number of failed transactions. The benchmark logs shows that for small network configuration the pending transaction queue reach maximum capacity fairly easily (10k), thus leading to failed transactions. However, with a big network the transactions are spread among the nodes leading to less failed transactions, but still reaching a limitation in the Output TPS of maximum 300 TPS.

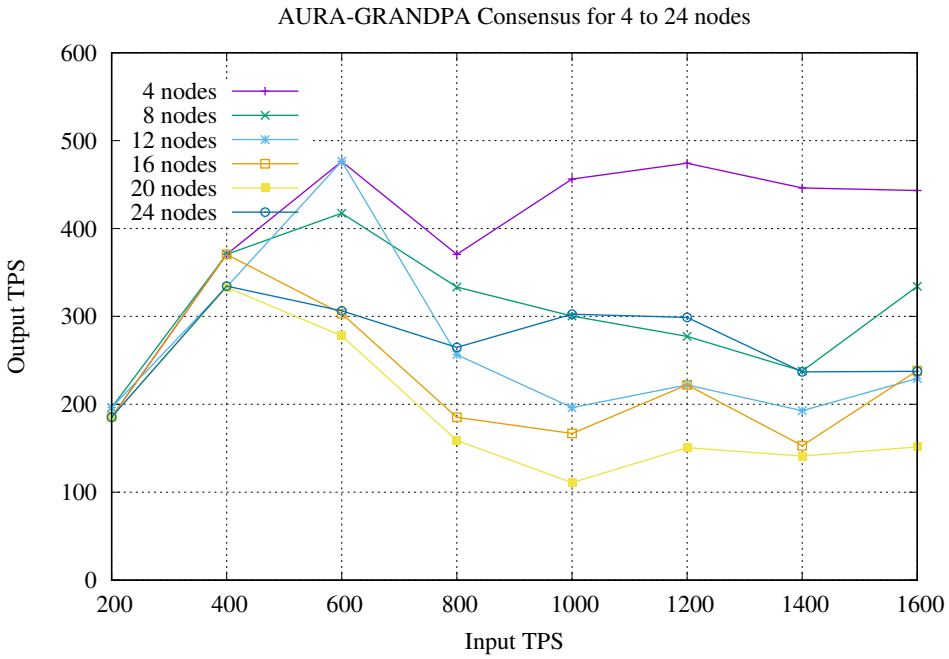


Figure 3.4.6: Substrate-based AURA + GRANDPA Consensus Performance. More nodes decrease the Output TPS, expect for 24 nodes. At high node count, the network can absorb better the total sent transactions, leading to less failed transactions due to node full pending queue.

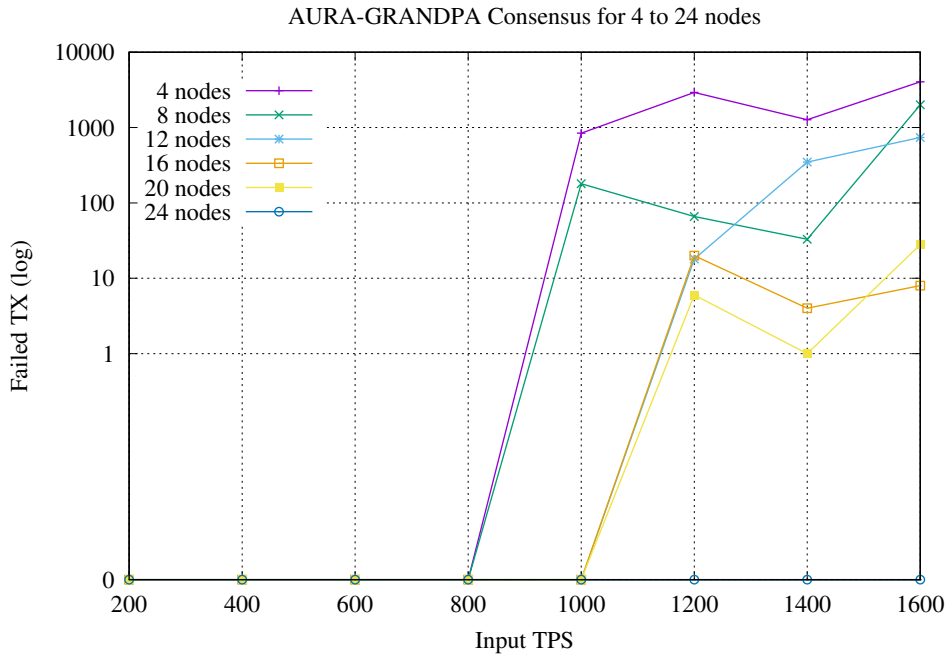


Figure 3.4.7: Failed Transations for Substrate-based AURA + GRANDPA Consensus Performance. More nodes in the consensus leads to less failed transactions. With a small node count the overall network can receive and absorb less transactions in the pending queue. This demonstrate that network configuration and transaction pool (i.e. *mempool*) are crucial.

3.5 Conclusion on Private BFT Consensuses

In our analysis in Figure 3.5.1, we evaluate the maximum normalized Output TPS performance for all previously tested BFT consensus blockchain variants at their optimal Output TPS, effectively excluding most implementation-related limitations. We adopt a coefficient scale for Output TPS, with 1 representing the highest TPS performance and 0 the lowest, for each blockchain. This Output TPS Coefficient indicates performance fluctuations as node count increases in the consensus mechanism, and how these changes manifest.

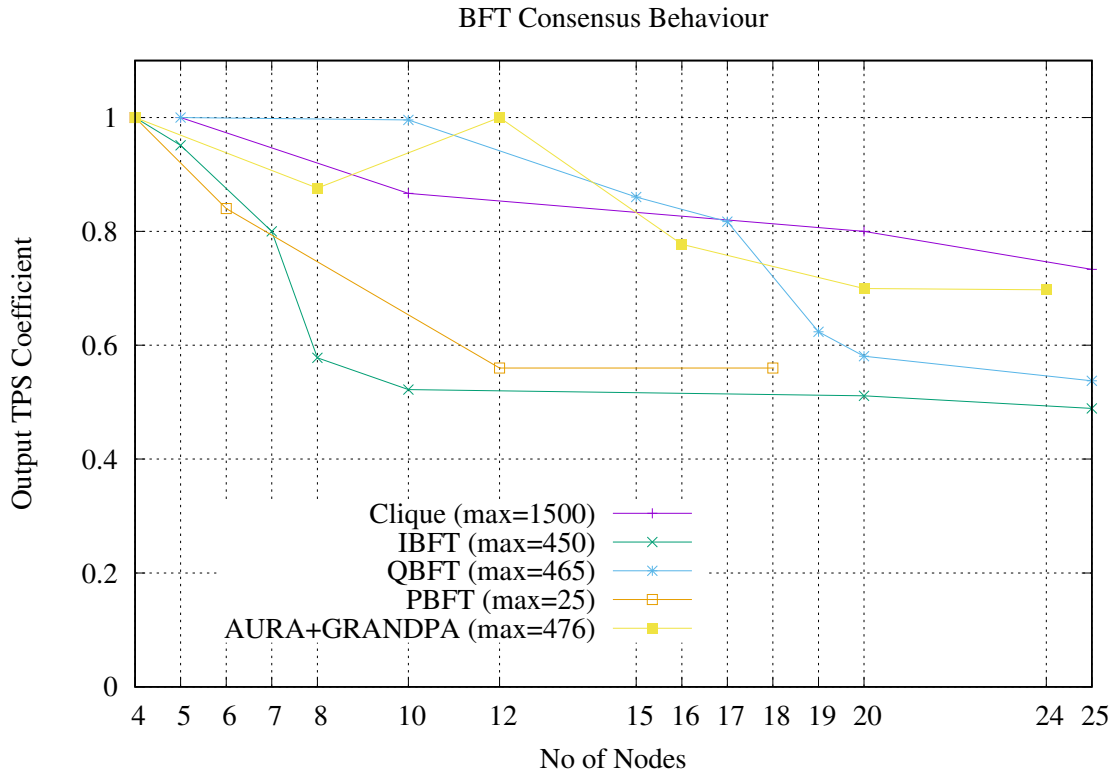


Figure 3.5.1: Normalised Output TPS behaviour of BFT consensuses

Within the BFT consensus group, we observe that optimal performance typically occurs with 4 to 12 nodes. For QBFT and AURA+GRANDPA, both performance degradation begins gradually at around 12 nodes, while for Clique, the transaction throughput processed is comparatively lower. In this node range, we see the most efficient transaction processing across the blockchain spectrum. This efficiency aligns with the preferred range for validator participant numbers in the accident use case. However, once the node count exceeds 12, all BFT consensus blockchains start to show a decline in performance, eventually stabilizing at a stabilized output TPS rate. This trend can be traced back to the core principle shared by all BFT algorithms: communication overhead, which is necessary to avoid Byzantine faults. Despite variations in the number of phases and leader election methods, all BFT consensus algorithms display similar patterns of behavior. The blockchain platform and code design implementation is responsible for the overall Output TPS, from fastest to slowest being:
Clique $>$ AURA+GRANDPA $>$ QBFT $>$ IBFT $>$ PBFT.

We can now establish that choosing Substrate AURA+GRANDPA seems to be the best fit for the rest of this thesis: it provides enough security, determinism, scalability, prevent forks, and

good performances. Also the software modularity is such that customization is accessible using stable/enterprise grade components.

The next chapter is a continuum of the use case exploration focusing on the IoT/vehicle side. We will develop blockchain client applications not only for Substrate, but also for other blockchains in order to finally express the common question a potential enterprise would ask: interoperability.

4 BIoT: Practical Blockchain Client Integration on IoT

4.1 Introduction

We have completed a preliminary testing phase of the blockchain platforms within the SIM project's scope, it is now time to go more in-depth on the implementation of the blockchain onto the IoT device itself. We will discover in this chapter *what is a blockchain client? what the client is composed of?* and the *how to do a practical implementation of them in an IoT device?* and most importantly *how to improve a client to implement it on IoT devices?*

Previous work on the SIM project has shown that only off-chain integration is possible for constrained devices, thus it is the blockchain client that has to be implemented [203].
A quick reminder on why we choose an off-chain design of BIoT is the following.

The on-chain IoT approach integrates IoT devices directly into the blockchain network, allowing them full participation and benefit from blockchain features. However, this method requires continuous synchronization with the network, leading to high energy consumption due to the need for constant connectivity.
Although the device can hold a copy of the blockchain, it will face challenges of the ever-increasing storage and high networking bandwidth requirements.
Additionally, Y. Khacef [213] provides a list of today's vehicle sensors and analyzes the estimated data quantity required in a hypothetical accident report. This estimate is around 1.05 GB for a record containing 10 seconds of data prior to the accident, accentuating the need for an off-chain IoT approach.

In the off-chain IoT approach, IoT devices interact with the blockchain without storing a copy of it, thereby reducing the need for constant connectivity, increasing storage, and power consumption. Devices operate via an interface to the blockchain, performing offline transaction creation and signing. This method demands compatibility with the communication protocol and the ability to perform certain cryptographic computations for blockchain identification. Although this reduces power requirements, it requires devices with enough computing power to handle the cryptographic tasks and networking connectivity to exchange information with a blockchain node. The trade-off is that while power consumption is optimized and devices create secure offline digital signatures, they do not maintain a full blockchain copy and thus do not fully participate in the network.

4.2 What Is a Blockchain Client?

In the context of this thesis, the term **client** refers to the software that is responsible for the preparation, creation, and transmission of valid transactions for a specific blockchain platform. Essentially, a blockchain client is an aggregation of libraries compiled into an executable form, and is a binary or a script file. For our study's purpose, the client program application is integrated as a binary within the IoT device's firmware – the machine code written into the microcontroller's memory. Figure 4.2.1 illustrates the progression from code development to hardware implementation, highlighting the procedure of embedding the blockchain client into the IoT hardware.

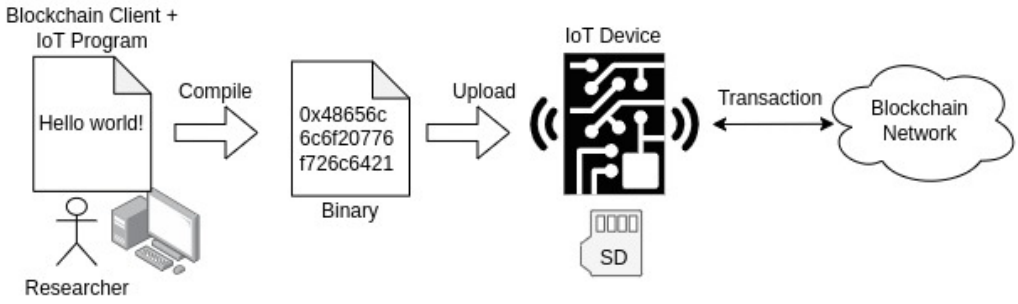


Figure 4.2.1: From code to microcontroller: Device programming – writing, compiling, uploading, and executing

Each blockchain has its unique specifications, necessitating distinct implementations of Remote Procedure Call (RPC) protocols (or API endpoints) and cryptographic algorithms. The heterogeneity of blockchains means that a client must be matching to the specific RPC interface and cryptography standards of the targeted blockchain.

For instance, as seen in the state of art, some blockchains may employ SHA-256 for hashing, while others might use Keccak-256. Similarly, the signing of transactions could involve different elliptic curve algorithms like secp256k1 or EdDSA. As such, if we want to be compatible with any blockchain, the client's software libraries must encompass a broad range of cryptographic functionalities and be adaptable to various blockchain protocols. This requirement adds a layer of complexity to the development of blockchain clients for IoT devices, thus for this thesis we study the client implementation for Hyperledger Sawtooth, Ethereum, EOSIO, and Substrate-based blockchains.

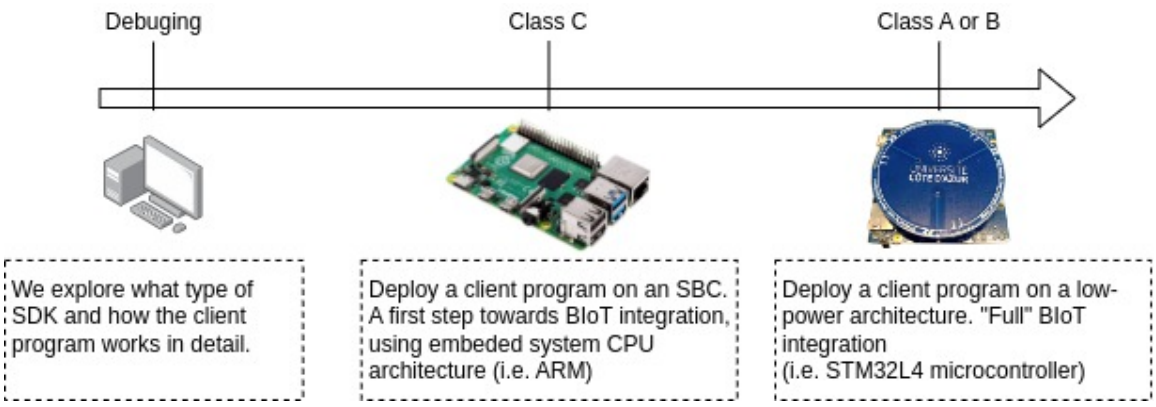


Figure 4.2.2: Thesis evolution of BIoT integration

As depicted in Figure 4.2.2, the aim in this BIoT integration exploration is to find a way to implement these blockchain clients onto IoT devices that fit Class A or Class B (cf. Table 2.2.2 p. 34). First, we will develop solutions that can be deployed on a Class C device, which has an OS with hardware comparable to that of a regular small computer, exploring the minimal requirements of a client program. Next, we progressively shift towards increasingly constrained devices to reach the limits for BIoT integration.

4.2.1 Blockchain Transaction Format

It is worth noting that transactions (and blockchain storage) are always formatted in a blockchain-specific format. Three of the most used data structures are the following.

RLP (Recursive Length Prefix) is a serialization method used in Ethereum for encoding structured data, primarily for network communication and data storage [214]. It encodes nested arrays of binary data, and it's the primary method for serializing and transmitting data in Ethereum. It's used for encoding transactions, blocks, and other objects within Ethereum.

Protocol Buffers (Protobuf) is a method developed by Google for serializing structured data, similar to XML or JSON [215]. It is useful in developing programs that communicate with each other over a network or for storing data. Protobuf involves defining data structures in a .proto file, compiling this file into language-specific classes (almost any), and then using these classes in your program to serialize data into a compact binary format for efficient storage or transmission. Deserialization allows the conversion of this binary data back into usable class instances. This process offers efficient and compatible data handling across various applications. Hyperledger Sawtooth uses Protobuf.

SCALE codec (Simple Concatenated Aggregate Little-Endian) is a binary encoding and decoding system primarily used in the Polkadot and Substrate frameworks for blockchain development [216]. It's designed for high-efficiency data serialization.
The SCALE codec works by defining data structures, then using compact binary encoding for efficient serialization. It features variable-length encoding for integers to minimize data size. The serialized data is suitable for network transmission or storage, and can be deserialized back into its original structure. SCALE emphasizes efficiency and compactness, crucial for blockchain performance and storage requirements.

4.2.2 Blockchain Communication Endpoint Protocol

In the context of blockchain clients, the communication protocol used to interact with the blockchain network is often referred to as an RPC (Remote Procedure Call) endpoint. Two commonly employed RPC endpoint protocols are Websocket (WS) and Hypertext Transfer Protocol (HTTP), each offering distinct advantages and considerations.

Websocket (WS) is the preferred choice when real-time communication and immediate updates are required. WS enables bidirectional communication between the client and the blockchain network, reducing latency and allowing for push notifications of blockchain events. However, implementing Websockets can be complex, and the continuous maintenance of open connections may consume more system resources.

On the other hand, Hypertext Transfer Protocol (HTTP) serves as a more straightforward and well-established RPC endpoint protocol. It utilizes widespread support across various platforms and devices, ensuring potential compatibility with a broad range of IoT devices, including those with limited resources. Nonetheless, HTTP typically involves periodic polling by the client to check

for updates, potentially introducing latency and increased network traffic. Unlike Websockets, HTTP does not offer real-time push updates.

The choice between Websocket and HTTP as the RPC endpoint protocol relies on the specific requirements of the application and the capabilities of the IoT device. Some blockchain platforms provide support for both protocols, granting developers the flexibility to choose based on project's needs.

It's important to note that, ultimately, the protocol used is not relevant. **A transaction in the context of blockchain functionality remains the same, regardless of the transport protocol.** The focus should be on selecting the protocol that best suits the application's real-time communication needs and the IoT device's resource constraints.

4.3 A First Step Toward BIoT with Hyperledger Sawtooth

The current available mature and stable Hyperledger Sawtooth SDKs are in Python, GO, Rust, and JavaScript language (date: 2023-11-01). As introduced in sections before, we will start our exploration using a relatively powerful development board (Raspberry Pi 3 B+) and deploy a Sawtooth blockchain client on it.

4.3.1 Car Transaction Processor (Cartp)

In the case of this blockchain a transaction processor (TP) is made called *cartp* (car transaction processor), written in Rust language. This TP will handle transactions for the previously described use case (cf. Section 1.2.3 p. 23). Also, we describe the transaction family associated as follows: cartp manages transactions by including three primary command types: *NewCar*, *NewOwner*, and *Crash*. Each command type corresponds to a specific kind of transaction within the blockchain.

- **NewCar:** This transaction type is used to register a new car on the blockchain. It requires details such as the car's brand, type, license, and a unique identifier.
- **NewOwner:** This transaction type handles the change of ownership of a car. It records the new owner's details, including their name, address, and country, along with the car's identifier.
- **Crash:** This transaction type is used to record an accident involving a car. It includes details like the accident ID, a signature, and a public key.

The Crash transaction in the *cartp* transaction family requires the following data fields:

- **Command Type (cmd)**: Set to `Crash`.
- **Car ID (car_id)**: A unique identifier for the car involved in the crash.
- **Owner ID (owner_id)**: The identifier of the car's owner at the time of the crash.
- **Accident ID (accident_id)**: A unique identifier for the crash incident data, which is an IPFS CID[1].
- **Signature (signature)**: A digital signature associated with the crash transaction.
- **Data Public Key (datapublickey)**: The public key associated with the data of the crash.

[1] IPFS is a decentralized file system which store its contents using a Content Identifier (CID).

4.3.2 In-depth Sawtooth Client Application

To understand how Sawtooth client works and how we could later implement it for BIoT cases, we have to detail the client application[2]. The client communicates using a REST API, thus requiring the typical IP/TCP networking stack, that we will implement using standard CURL library.

Next, we have to detail the transaction creation flow, that works as follows in Sawtooth:

1. **Load (or create new) cryptographic private-public keys**:
   Sawtooth use ECDSA with the secp256k1 curve parameter. The program has to include the standard secp256k1 library coded in C [217], and load the keys for later use.
2. **Build a transaction that is matching the TP's transaction family**:
   Based on the previous described TP family, we have to provide a couple data fields in the transaction Protobuf.
3. **Sign the transaction**:
   Use the loaded cryptographic keys to sign the serialized transaction (DER encoded format). The resulting serialized compact signature is used as *signature* (64 bytes).
4. **Send the transaction** :
   The last step is to feed the serialized Protobuf transaction message into the CURL library and send it.

4.3.3 Code Profiling

To study the importance of each function costs in our client application we apply a profiling method using Valgrind. Valgrind is an instrumentation framework for building dynamic analysis tools that can generate a call graph of the entire program. The obtained call graph opened with KCachegrind (Figure 4.3.1) is a call tree that also contains the number of CPU cycles which is also calculated based on the parent call, thus providing a percentage of usage per called function relative to the parent function.

In our Hyperledger Sawtooth C++ client application, profiling results reveal that the initialization phase consumes 9.85% of the total program cycles. Additionally, the analysis shows that constructing the transaction requires as much as 29.86%, while transmitting the transaction accounts for 59.86% of the program cycles. Based on these findings, it becomes clear that, even with an optimized client application such as the one created here, **a significant portion of the program's resources is dedicated to managing networking-related tasks.**

4.3.4 Practical Energy Consumption

In this part of previous work [19], we analyzed the energy consumption patterns of the Hyperledger Sawtooth C++ client program, executed on a Raspberry Pi 3 B+ device. This study involved monitoring the current consumption in relation to the size of the data being processed. We present our findings in Table 4.3.1, illustrating both the scenarios with and without the idle mode of the Raspberry Pi. It is worth noting that at that time of this work, the client application was implemented to send all accident data directly on-chain, and not off-chain as stated in the beginning of this section.

Our observations reveal a nearly linear relationship between data size and current consumption. The Raspberry Pi, when running an idle OS only, typically consumes about 0.52 A. Figure 4.3.2

[2]A version of the implementation is available here: github.com/lucgerrits/HyperledgerSawtooth-cpp-client .

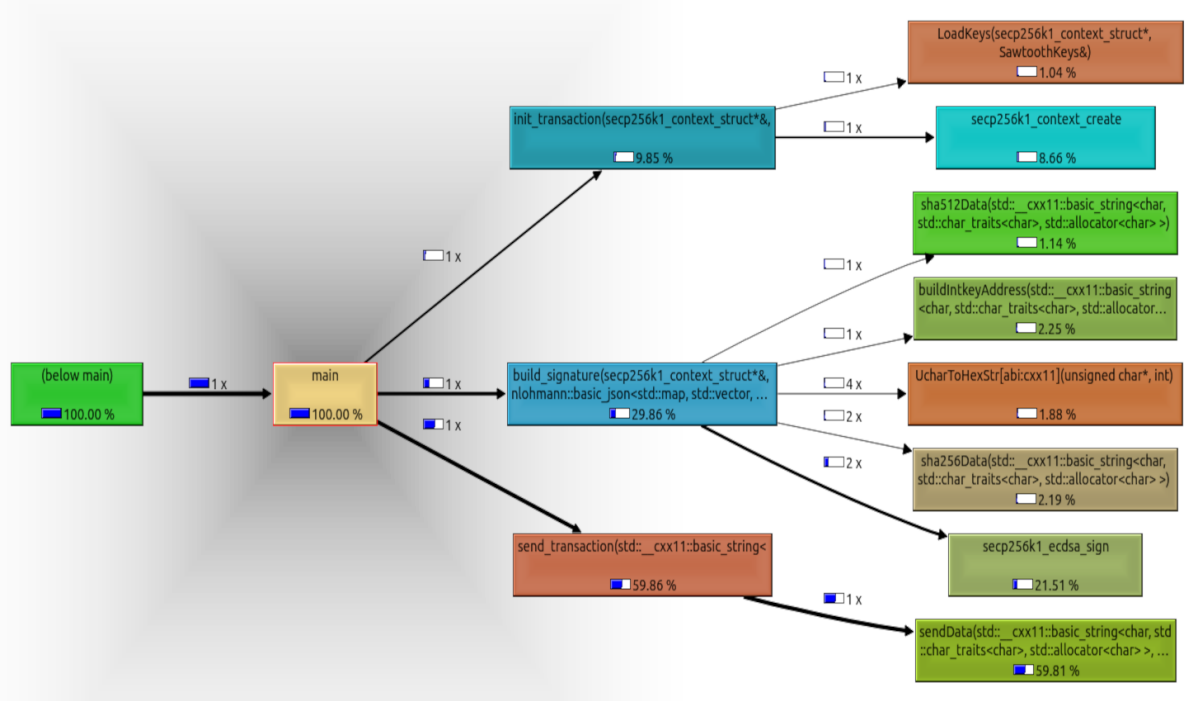


Figure 4.3.1: Call graph of Hyperledger Sawtooth C++ client application

Data size to send	Current consumption with Idle	Current consumption without Idle
< 1 MByte	0.13 A.s / 0.0365 mAh	0.024 A.s / 0.0065 mA.h
1 MByte	3.37 A.s / 0.94 mA.h	0.76 A.s / 0.21 mA.h
2 MBytes	6.29 A.s / 1.75 mA.h	1.51 A.s / 0.42 mA.h
41 MBytes	153.07 A.s / 42.52 mA.h	24.06 A.s / 6.67 mA.h

Table 4.3.1: Current consumption with and without OS according to the data size to send

offers a graphical representation of these consumption patterns, highlighting various stages of program execution such as startup (1), operation (2), and memory release (3). It is interesting to note that the power consumption escalates with an increase in the volume of data processed. This is attributed to the additional computations required by the C++ application as it handles more data.

However, an exception to this trend is the case of processing 41 MBytes of data. This scenario is excluded from our analysis as its execution time is disproportionately higher – about 40 times that of processing 1 MByte of data. The graphical representation helps in visualizing the distinct phases of current consumption, providing a clear picture of the energy demands at each stage of the program's lifecycle.

4.3.5 Energy Optimization

The energy optimization process involves re-evaluating the application's operation with a focus on its hashing algorithms, specifically SHA-512 and SHA-256. This work has been done by R. Kromes [203] in previous research [19], by developing functional models of these hashing algorithms using SystemC. These models are then integrated into the SystemC model of the Raspberry Pi BCM2837, allowing us to simulate and analyze the application's performance in a

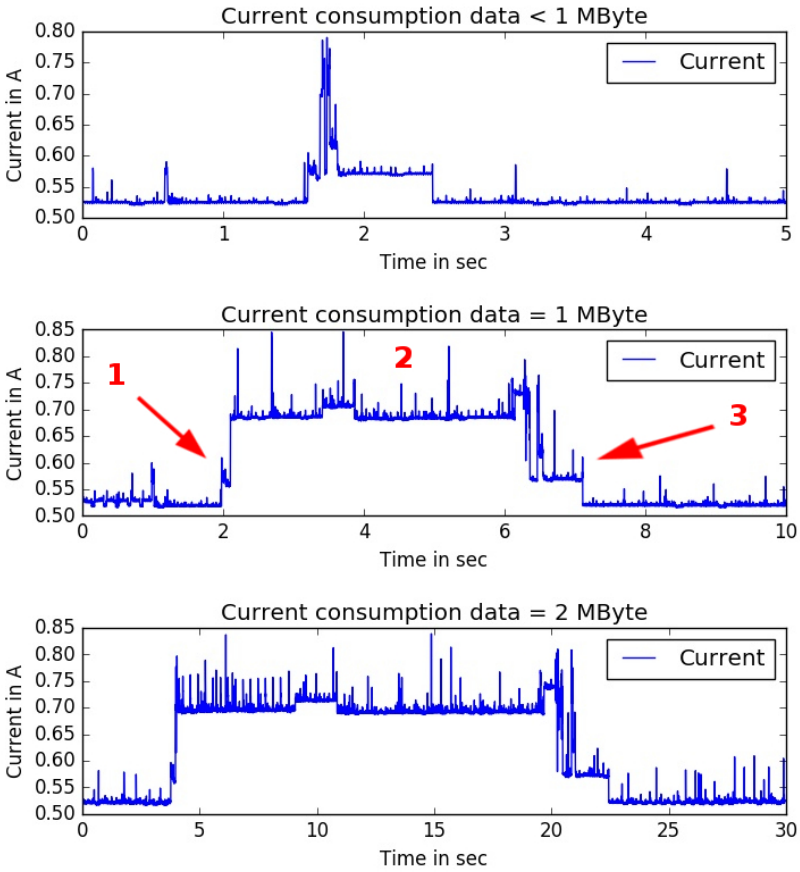


Figure 4.3.2: Current consumption of Hyperledger Sawtooth C++ client program on Raspberry Pi

simulated environment.
This approach enables us to execute the client program with the newly implemented hashing models. The aim is to observe how these modifications impact the overall energy consumption and efficiency of the application. The results shows a potential acceleration gain of 112 for SHA-256 and 293 for SHA-512 cryptographic algorithms, thus potentially a very interesting reduction in energy consumption.

4.3.6 Conclusion On Sawtooth

This initial implementation and previous analysis provide valuable insights into the energy dynamics of blockchain-based applications in the context of BIoT. The findings demonstrate that while the Hyperledger Sawtooth C++ client application is functional and cryptographic operations optimization is feasible, there are significant opportunities for optimizing energy consumption in the areas of data processing at the networking level.

The profiling and optimization steps highlight the importance of efficient code design and the potential impact of algorithms choices on the energy footprint of blockchain applications. A key element that has hasn't been improved is the IoT communication layer which is computing intensive.

The next blockchain customer application will provide a deeper insight into the level of com-

plexity and differentiation of blockchain platforms.

4.4 Other Blockchain Client Applications Analysis

4.4.1 Ethereum: Client Application Using Go

Sending transaction to an Ethereum blockchain can be performed using scripting or compiled program languages. We create a similar application[3] implementation as Sawtooth, however using Go language and `go-ethereum/ethclient` library.
The profiling of the application (Figure 4.4.1) shows that the preparation of transaction takes 93.1% (10,34% to connect to the HTTP endpoint and create a context + 13.79% to get gas price + 3.45% to get the account nonce + 37.93% to load the private key + 27.59% to sign the transaction). Additionally, the program use 3.45% to send the transaction.

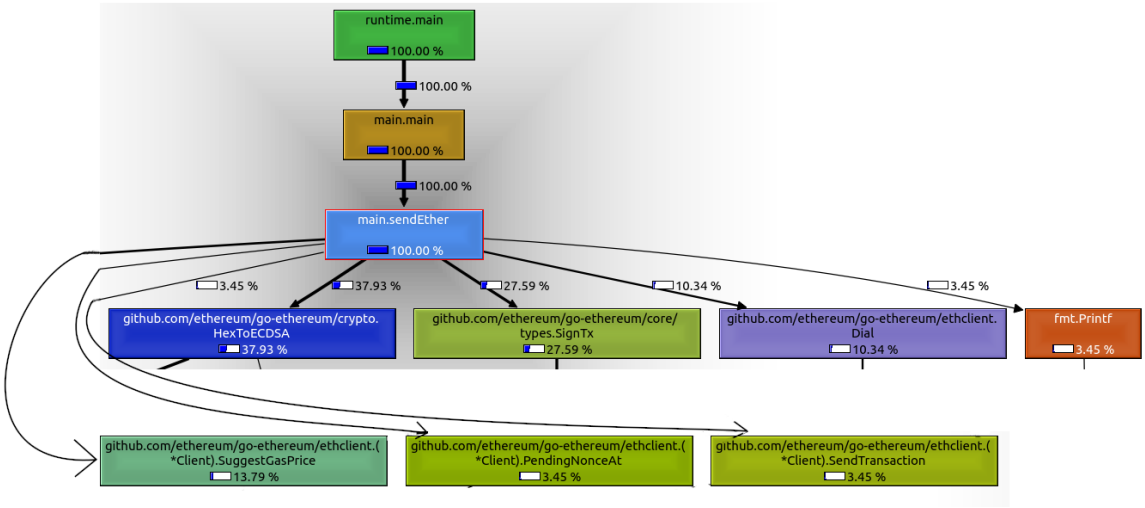


Figure 4.4.1: Call graph of Ethereum Go client application

The Ethereum Go client's interaction with the blockchain involves critical preparatory steps before executing a transaction. This is highlighted by examining the `tcpflow` logs, which reveal the sequence of HTTP requests made by the client. The resulting application shows that the library requires two elements before sending a valid transaction: `eth_gasPrice` and `eth_getTransactionCount` to retrieve the current gas price and the transaction nonce. Also, the program network usage shows 622 Bytes received and 662 Bytes sent.

Initially, the client requests the current gas price with `eth_gasPrice`. This step is crucial for setting an appropriate transaction fee, ensuring timely processing in the network. Following this, it makes a `eth_getTransactionCount` request to retrieve the nonce for the sending account, crucial for transaction ordering and security.

Once these details are fetched, the client sends the transaction via `eth_sendRawTransaction`, including the signed and serialized transaction data. Similarly as Sawtooth, this sequential approach underlines the complexity in blockchain interactions, highlighting the need for careful orchestration.

[3] A version of the implementation is available here: github.com/lucgerrits/ethereum-go-client-tests.

4.4.2 EOSIO: a C++ Client Application

The EOSIO C++ client application[4] is very similar but still has variation of the Hyperledger Sawtooth program. The key differences are:

- Required to use an ABI interface to interact with the smart contract (*abieos* library)
- The hashing algorithm used are SHA256 and RIPEMD160
- Transaction signatures are required to be canonical and in a specific base58 format
- Use the JSON RPC endpoint to send the transaction

To build a valid transaction the program has to retrieve beforehand the node information, latest block, and the smart contract code and ABI.

Overall, the implementation [218] results in the following call graph in Figure 4.4.2. Using a

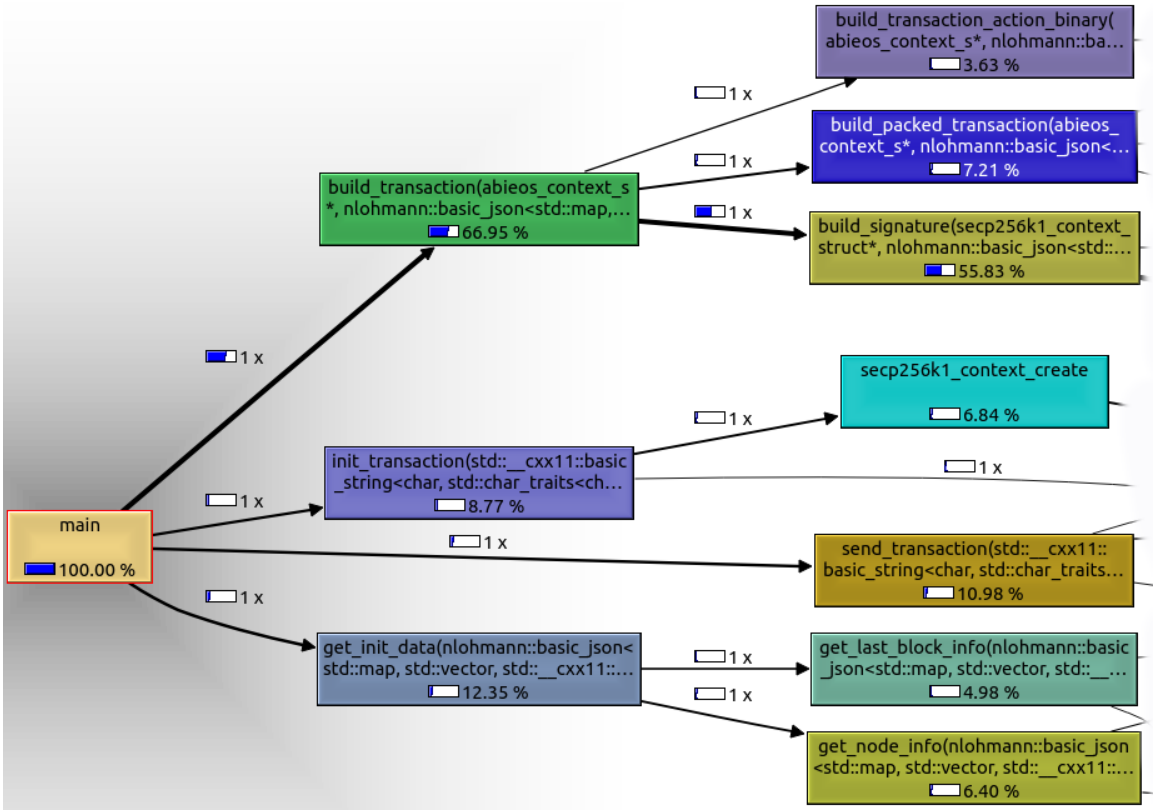


Figure 4.4.2: Call graph of EOSIO C++ client application: Main function called

global overview of the call tree with *main* function as parent, the *build_transaction* step cost 66.95% of the entire program. Meaning the most used functions are in this step. Relative to the main function, the sign function (*secp256k1_ecdsa_sign_recoverable()*) costs 53.39% of the entire program. It is confirmed by selecting *build_transaction* step as parent of the graph (Figure 4.4.3); we can observe that the sign function is again the most used at 79.75%. Building the signature takes 83.39% of the program time relative to *build_transaction*, the Figure 4.4.3 show a zoom on this step by setting it as the relative parent of the call graph.

This concludes that an optimization could be possible by accelerating the client application by using a dedicated hardware accelerator IP for the *secp256k1_ecdsa_sign_recoverable()* function.

[4] A version of the implementation is available here: github.com/lucgerrits/EOS.IO-cpp-client.

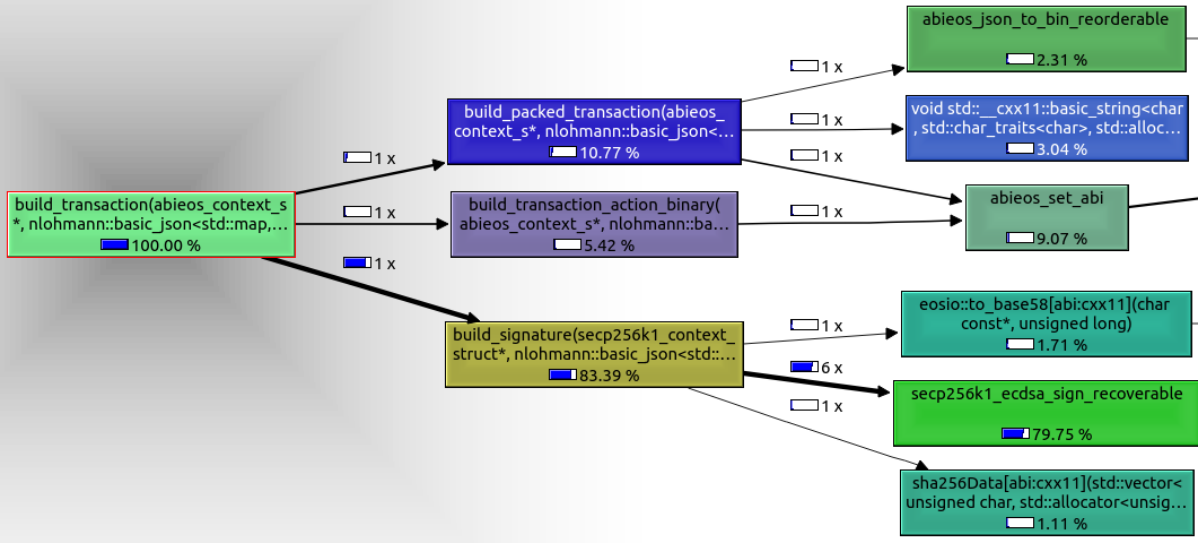


Figure 4.4.3: Call graph of EOSIO C++ client application: Zoom on the transaction build step. The *build_transaction* function is here set as parent of the call tree.

⚠ An important notice on the C++ program is that the signature generation cost is not deterministic. The results shown previously are only results with best case scenario of the signature generation. To generate a valid signature, in EOS.IO, it is required to have a canonical signature. An ECDSA secp256k1 signature can have four different, perfectly valid forms (called Signature Malleability [219]). To prevent the same mistake that Bitcoin has, EOS.IO must have only one valid, canonical form. In order to be canonical, our program has a while loop that iterates a maximum four times with the condition "is canonical". This condition means that the *build_transaction* function cost is minimum when the first iteration is true and a maximum when the function is false three times. The reason for having a maximum of four iteration is that the *secp256k1_ecdsa_sign_recoverable()* function can only have a recover id going from 0 to 3.

4.4.3 Substrate-based Client Application

4.4.3.1 Substrate-based: JavaScript Client Application

A Substrate JavaScript client application[5] is developed to extract also its minimum requirements. Given that JavaScript is a high-level scripting language, it does not provide the same level of detailed profiling capabilities as lower-level compiled languages like C++ or Go. However, the process flow of the client application remains crucial for understanding its requirements.

The transaction process in the Substrate JavaScript client can be outlined as follows:

1. **Connection Establishment**: The client uses WebSocket (WsProvider) to establish a connection with the blockchain node, enabling two-way communication.
2. **Keyring Setup**: It creates a keyring instance to manage cryptographic keys, using the sr25519 cryptographic scheme, crucial for transaction signing.
3. **Account Selection**: The script selects an account (e.g., 'Alice') for conducting the transaction, involving loading the account's private key.

[5] A version of the implementation is available here: bitbucket.org/edge-team-leat/all-private-blockchains-cloud-perf-tests/src/master/Substrate/js-client/.

4. **Transaction Preparation and Signing**: The client prepares and signs the transaction, ensuring its authenticity and integrity.
5. **Transaction Submission**: The signed transaction is sent to the blockchain network, with the client awaiting a confirmation transaction hash.
6. **Disconnection**: Post-transaction, the client disconnects from the blockchain node.

Although JavaScript's high-level abstraction simplifies the development process of blockchain client applications, it necessitates alternative approaches for performance analysis, focusing primarily the network usage of a Substrate-based client program. The analysis of the WebSocket messages shows that 1513 Bytes are sent and 1604 Bytes are received when sending one transaction (detailed messages in Appendix 8.3.2).

4.4.3.2 Substrate-based: Rust Client Application

Another version of the client application written in the Rust language is created[6] and executed on the Raspberry Pi development board. We measure the power consumption of the Raspberry Pi 4B using the Otii Arc power profiler. We also collect all the serial console communication transmissions from the development board using the Otii (i.e. synchronizing messages and power consumption). The resulting[7] power profile is in Figure 4.4.4.

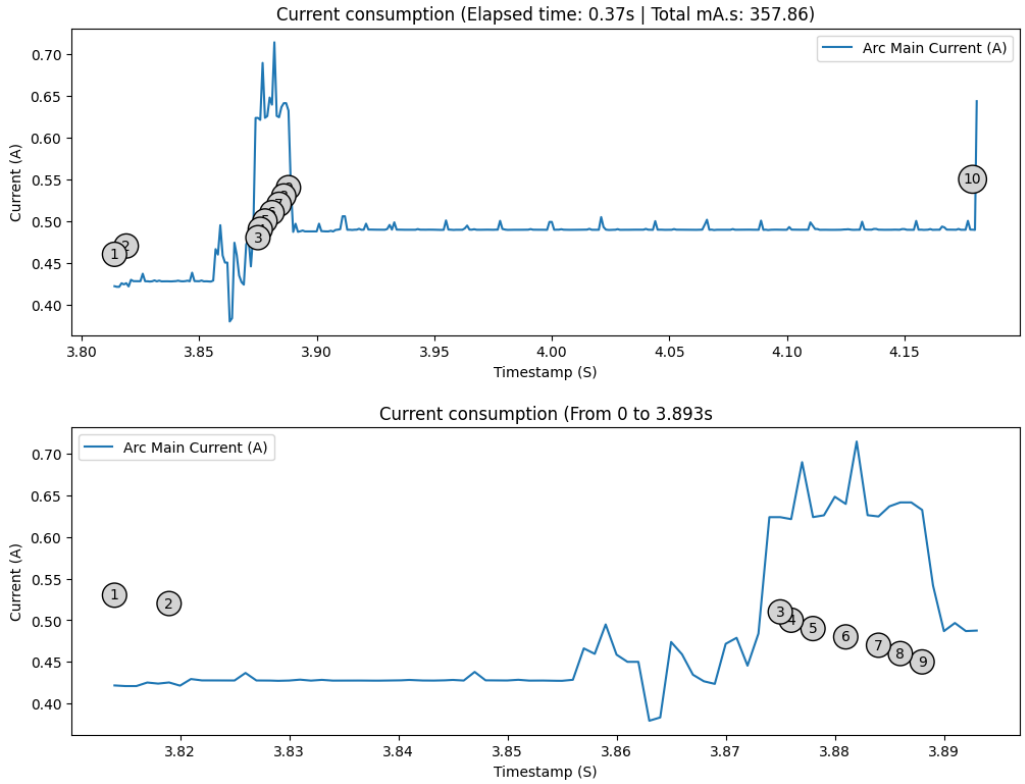


Figure 4.4.4: Current consumption of the Rust Substrate Client Application on a Raspberry Pi 4B.

The legend for Figure 4.4.4 numbers are representing the UART received messages from the Raspberry Pi UART console:

[6] A version of the implementation is available here: bitbucket.org/edge-team-leat/substrate-blockchain-client/.
[7] Measurements and build scripts available here: bitbucket.org/edge-team-leat/mesures-otii-raspberry-pi-4/.

1. `Interacting with node on ws://X.X.X.X:8080`
2. `Start set_storage`
3. `API OK`
4. `end init set_storage`
5. `Set key: "foo" <==> [66, 6F, 6F]`
6. `Set value: "bar" <==> [62, 61, 72]`
7. `Create extrinsic`
8. `End create extrinsic`
9. `Send Tx`
10. `End send Tx`

All UART messages are ending with \x0d\x0a, which represent the \r and \n ASCII characters. As we see from the legend there are nine events recorded starting with interacting with the blockchain endpoint (WebSocket) and finishing with the message `End send Tx`. The program sends a transaction to a substrate based blockchain that sets the key (e.g. foo) to a value (e.g. bar). It uses `substrate-api-client` library in combination with substrate primitives and Raspberry Pi Peripheral Access Library.

It is worth noting that during the tests the impact of the communication method with the blockchain (Wi-Fi or ethernet cable) highly impacts the power consumption between the stage `Send Tx` and `End send Tx`. The Figure 4.4.5 shows the high dependence of the network technology on power consumption: 375.86 mA.s in Wi-Fi and 2172.56 mA.s in Ethernet, almost 6 times more power usage.

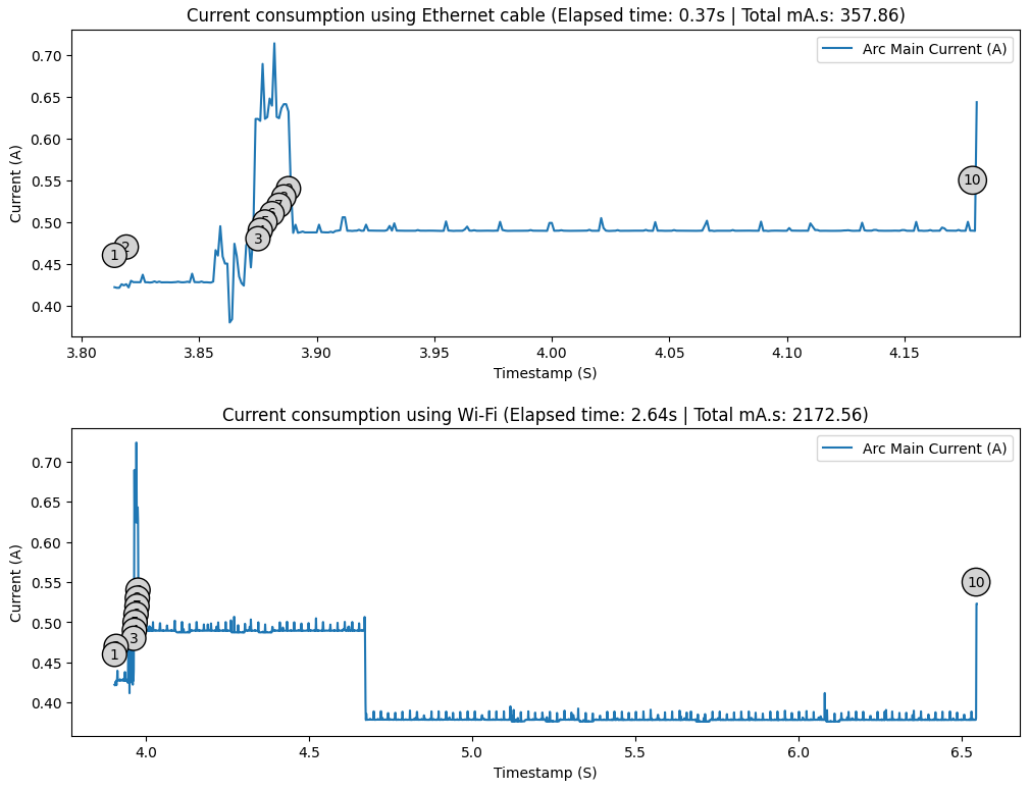


Figure 4.4.5: Current consumption of the Rust Substrate Client Application using Wi-Fi and Ethernet cable using a Raspberry Pi 4B

4.4.4 Conclusion on Specific Client Applications

The comprehensive analysis of various blockchain client applications, including Hyperledger Sawtooth, Substrate, Ethereum, Hyperledger Fabric, and R3 Corda, has provided valuable insights, particularly in the context of Blockchain Internet of Things (BIoT). Some of the measurements that wasn't done in this thesis are issued from previous work by N. Massiéra [220].

Diverse Power Consumption Patterns: The power consumption across different blockchain platforms varies significantly. As illustrated in Figure 4.4.6, there is a wide spectrum of power requirements, influenced by factors such as the complexity of the client application, the network technology used, and the hardware capabilities (Raspberry Pi 3B+ and 4B). Even if the measurements aren't perfectly comparable due to differences in hardware and network situations (i.e. $\pm$latency), the Figure 4.4.6 shows that all platforms are requiring the same order of energy.

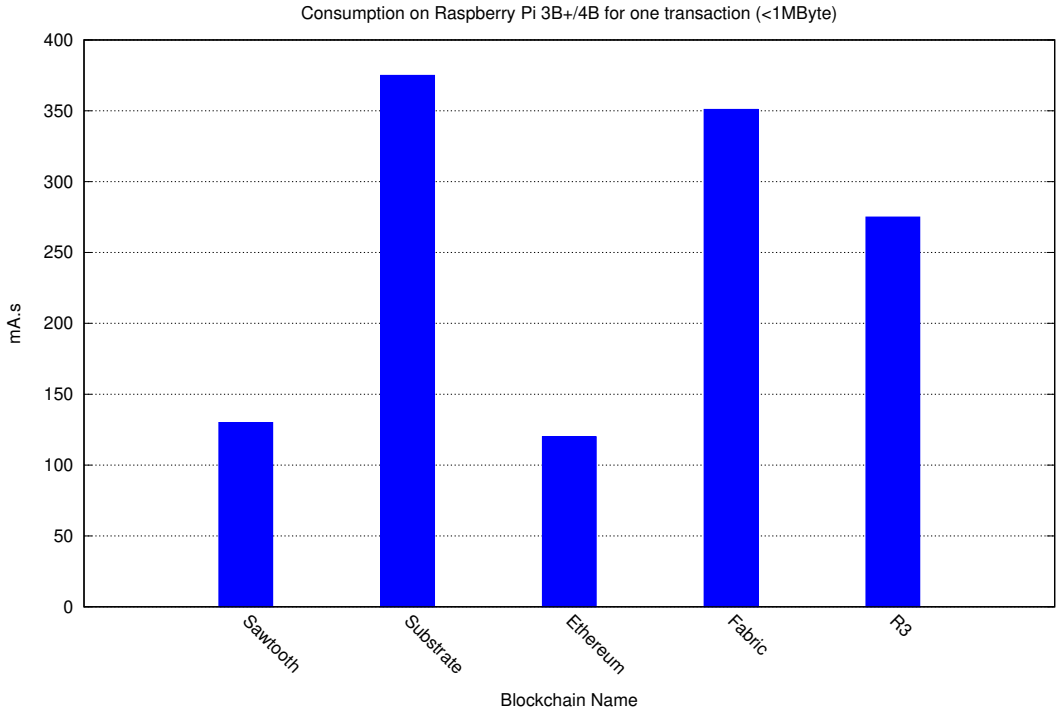


Figure 4.4.6: Summary of the blockchain client applications power consumption using Raspberry Pi 3B+/4B board for Hyperledger Sawtooth, Substrate, Ethereum, Hyperledger Fabric, and R3 Corda platforms.

Complexity of Client Applications: Blockchain client applications, regardless of the specific platform, show a common set of complexities and processing requirements. These typically include establishing network connections, managing cryptographic operations for transaction signing, handling data serialization, and ensuring secure and efficient communication with blockchain nodes. Each platform, while unique in its implementation, shares the challenge of balancing these requirements against the constraints of IoT environments, such as limited network resources and power efficiency.

Impact of Networking and Communication Methods: The communication method with the blockchain network (Wi-Fi vs. Ethernet) significantly affects power consumption. This is particularly evident in the Substrate Rust client application, highlighting the importance of the network layer in blockchain client applications in BIoT contexts, where power efficiency is crucial.

Optimization Opportunities: There is considerable potential for optimizing blockchain client applications. This includes enhancing cryptographic operations' efficiency using cryptographic IP or modules on the device (e.g. ATECC608A chip has ECDSA support), streamlining transaction preparation specifically for constrained devices, and most probably the most important, optimizing network communication protocols.

Anticipation of New Protocols: The analysis anticipates the introduction of new protocols tailored for BIoT environments. A notable example, which is detailed in the subsequent section of this thesis, is the Light Transaction Protocol (LTP). Developed as part of this research, LTP aims to address the challenges of energy and efficiency in BIoT by simplifying transaction formats and reducing the overhead associated with traditional blockchain transactions. Its introduction marks a proactive step towards resolving some of the identified issues in current blockchain client applications.

Considerations for Android Automotive and Other Platforms: A Java-based implementation for Android Automotive OS (AAOS) is presented in this thesis as a proof of concept for blockchain in automotive IoT (Section 4.6 p. 139). Discussed later, it illustrates the practicality of blockchain in a specific use case, highlighting platform-specific challenges and integration considerations in automotive environments.

This analysis of specific blockchain client applications underlines the challenges and opportunities in optimizing blockchain technology for BIoT applications. The varying power consumption patterns, the complexity of the client applications, and the significant impact of networking methods emphasize the necessity for a comprehensive approach in designing more efficient and sustainable blockchain solutions in IoT environments. The progression towards protocols like LTP and adaptations for diverse platforms like Android Automotive represents strategic steps towards making blockchain a viable and efficient solution in the domain of IoT.

4.5 Light Transaction Protocol for Substrate-based and Hyperledger Fabric Blockchains

In this part of the thesis, we turn our attention to an alternative approach for integrating blockchain into smaller, more constrained IoT devices. We present the lightweight transaction protocol (LTP) as a potential solution to transact with one or multiple blockchains, analyze its feasibility and implementation results [221].

4.5.1 A Light Protocol: Alternative Approach to BIoT

From our previous analyses of client applications, it becomes clear that networking components and blockchain-specific data formatting are the most demanding in terms of CPU cycles and memory usage. This complexity is further amplified in IoT scenarios involving communication with multiple blockchains, necessitating highly heterogeneous programming solutions. Additionally, we have observed that all clients require two-way communication to retrieve information from the blockchain to secure the transaction (e.g. account nonce).

There are two possible solutions to these problems: either use more powerful IoT devices that have sufficient computational power and network connectivity (Class C devices), or create a lighter transaction version (compatible for any Class A or B devices) that can later be encapsulated or proxied to ensure compatibility with actual blockchain transactions.

4.5.2 LTP (Light Transaction Protocol) Specification

LTP aims to abstracts the networking and format layer with a generic IoT friendly structure. We have seen previously that multiple times the client application has to exchange with the blockchain node to retrieve the state of the chain (e.g. the nonce, last block hash, ABI or other interface, etc.). If we abstract this into a "standard stateless transaction" we could make the client program less network intensive and also IoT friendly. This work is presented in [221] by proposing a standard format that match constrained device architecture requirements. The proposed work is designed to work with LoRaWAN communication protocol and system architecture.

4.5.2.1 LTP Proposal

Depicted in Figure 4.5.1, we propose a unique transaction payload format constructed with the Protobuf to define data structure (in a *.proto* file Code 4.1). The data structure has a matching smart contract that verifies authorized IoT public key and data integrity. The proposed data structure is modular enough to choose which blockchain platform, the smart contract, and action to target. A relay or proxy is required to encapsulate and send the payload in a valid transaction (based on selected blockchain). The IoT payloads can not be altered because of signature generation is done on the IoT directly. However, rate limiting and replay attacks could be possible. To prevent replay attacks, we would have to introduce a nonce generation inside LTP and LTP smart contract.

Base don the common requirements of all blockchains, we define the minimum components to create an agnostic transaction. The data structure of such transaction is built with the following fields:

- public key: IoT device public key, typically the secp256k1 compressed public key of 257-bit integer (33 bytes)

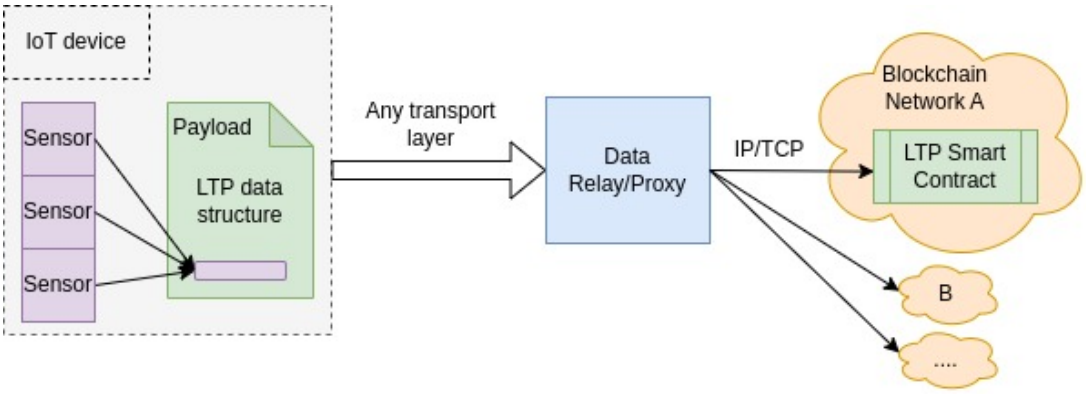


Figure 4.5.1: Design principle of LTP proposal

- signature: the secp256k1 digital signature of 512 bits (64 bytes)
- message:
  - blockchain ID : 1 byte, enumeration of blockchain identifiers
  - smart contract name: 1 to 13 bytes, name or address of the smart contract
  - action ID : 1 byte, enumeration of actions to execute in the smart contract
  - sensors data : customization field, variable length and type

Resulting in the following Protobuf file:

Code 4.1: LTP Protobuf Payload data structure

```
syntax = "proto2";

message Payload {
  required bytes public_key = 1;
  required bytes signature = 2;
  required Message message = 3;
}
enum Blockchain {
  HYPERLEDGER_FABRIC = 0;
  HYPERLEDGER_SAWTOOTH = 1;
  SUBSTRATE = 2;
  ETHEREUM = 3;
}
enum Action {
  SET_DATA = 0;
  GET_DATA = 1; //not used in current implementation
}
message Message {
  required Blockchain blockchain_id = 1;
  required string smart_contract_name = 2;
  required Action action = 3;
  required Sensors_data sensors_data= 4;
}
message Sensors_data {
  required float temperature = 1;
  required float altitude = 2;
  required float pressure = 3;
  required float axis_x = 4;
  required float axis_y = 5;
  required float axis_z = 6;
```

```
}
```

4.5.2.2 LTL: Light Transaction Library

A library coded in C language is made to be able to be deployed on most IoT devices without compatibility issues. LTL provides the core methods to create valid LTP payloads. The Code 4.2 describes the implemented functions for LTL. This library is adapted for the Arduino environment and configured for LoRaWAN protocol limitations.

Code 4.2: Light Transaction Library (LTL) for LTP

```
#ifndef _LightTransaction_H_
#define _LightTransaction_H_
#endif

#include <Arduino.h>            // Includes the Arduino library for functionality related to Arduino hardware.
#include "sha2.h"               // Includes the sha2 header for SHA-256 hash function-related operations.
#include "payload.pb.h"         // Includes the generated Protobuf header for LTP data structures.

// Defining constants used throughout the code.
#define PUBLIC_KEY_COMPRESSED_SIZE 33 // Size of the secp256k1 compressed public key in bytes.
#define SIGNATURE_SIZE 65             // Size of the ECDSA secp256k1 signature.
#define HASH_SIZE 32                  // Common hash size for SHA-256.
#define SHA256_BLOCK_LENGTH 64        // Block length for SHA-256.
#define SHA256_DIGEST_LENGTH 32       // Digest length for SHA-256.
#define PRIVATE_KEY_SIZE 32           // Size of the private key in bytes.
#define NETWORK_SIZE_MAX 242          // Maximum byte size for LoRaWAN payload
#define MESSAGE_SIZE_MAX 140          // Maximum byte size for a LTP message.

// Generates public keys from a given private key
bool generate_public_keys(byte private_key[], byte public_key_compressed[]);

// Hashes data using SHA-256
void hash_data(byte data[], int data_length, byte my_hashed_data[]);

// Signs data using a private key (ECDSA secp256k1)
void sign_data(byte *private_key, byte *hashed_data, byte *signature);

// Encodes a message to Protobuf LTP format
bool encoding_protobuf(byte protobuffer[], size_t *protobuffer_length, const Message my_message);

// Encodes a payload to Protobuf LTP format
bool encoding_protobuf(byte protobuffer[], size_t *protobuffer_length, const Payload my_payload);

// Builds a LTP payload with various parameters
int build_light_payload(byte private_key[], byte public_key_compressed[],
  Message my_message, byte protobuffer[],
  size_t *protobuffer_length);
```

4.5.2.3 LTP Generic Smart Contract

The corresponding smart contract for the LTP data structure is based of five actions and at least four storage tables (Figure 4.5.2). We define these components as follows:

- **init():** A development action to directly store a fixed list of IoT identities in the contract

- **put_data():** Store data on-chain. Only identities stored in the identities table can execute this action.
- **get_data():** Retrieve on-chain data
- **add/rm_identities()**: Add or remove an identity from the authorized list of public keys that can execute *put_data()*.

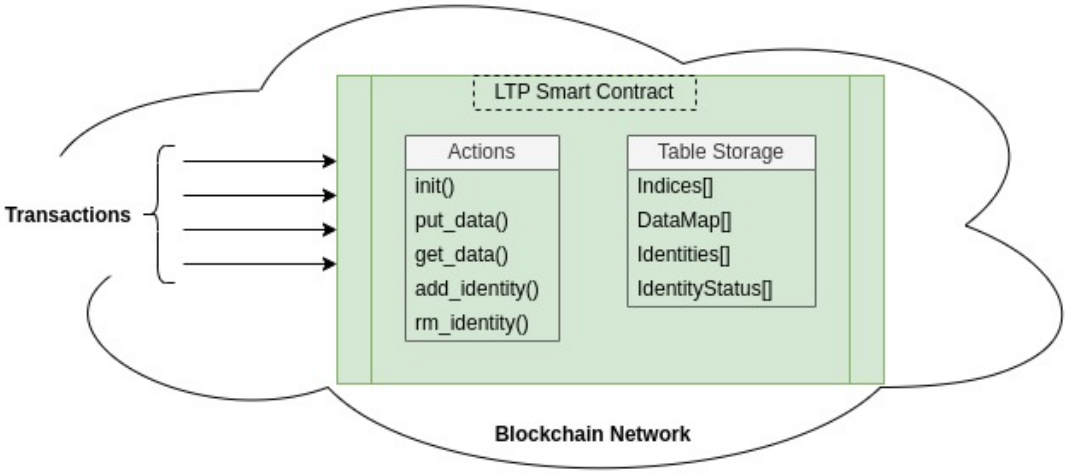


Figure 4.5.2: LTP required smart contract overview

4.5.3 LTP implemented with LoRaWAN

The LTP proposal doesn't necessarily require a specific blockchain platform, nor IoT networking protocol. In this part, an implementation of LTP is done using LoRaWAN, UCA Development Board, and three blockchain platforms (Hyperledger Fabric, Ethereum, and Substrate-based).

A complete overview of the communication protocol is depicted in Figure 4.5.3. The LoRaWAN communication protocol is designed to provide long-range/low-power bi-directional communication with end-to-end security. Simply put, the LoRaWAN network is comprised of end-devices (IoTs), gateways, and application services. LTP integrates itself at the end-device level by including the LTP data structure in the LoRaWAN payload and by implementing a specific module on the gateway level.

⚠ A typical LoRaWAN gateway, capable of constant LoRa transmissions and Ethernet or Wi-Fi communication, often uses a Raspberry Pi as its processor, paired with a Semtech SX1301 or SX1308 transceiver for LoRa operations. This setup requires a mains power supply for reliable functioning as it draws at least continuously 1A of current.

Literature shows that In the works of Pincheira et al. [182] [125], the authors' overall goal is to establish direct communication between the IoT devices and the Ethereum blockchain. The authors deployed an Arduino C library which allows the creation of Ethereum transactions locally in the IoT device. The valid Ethereum transaction creation requires the digital signature of the transaction, which must be done by a private key of a registered blockchain participant. One of the conclusions of this article is that the digital signature procedure should be accelerated because its creation can take a significant amount of time.

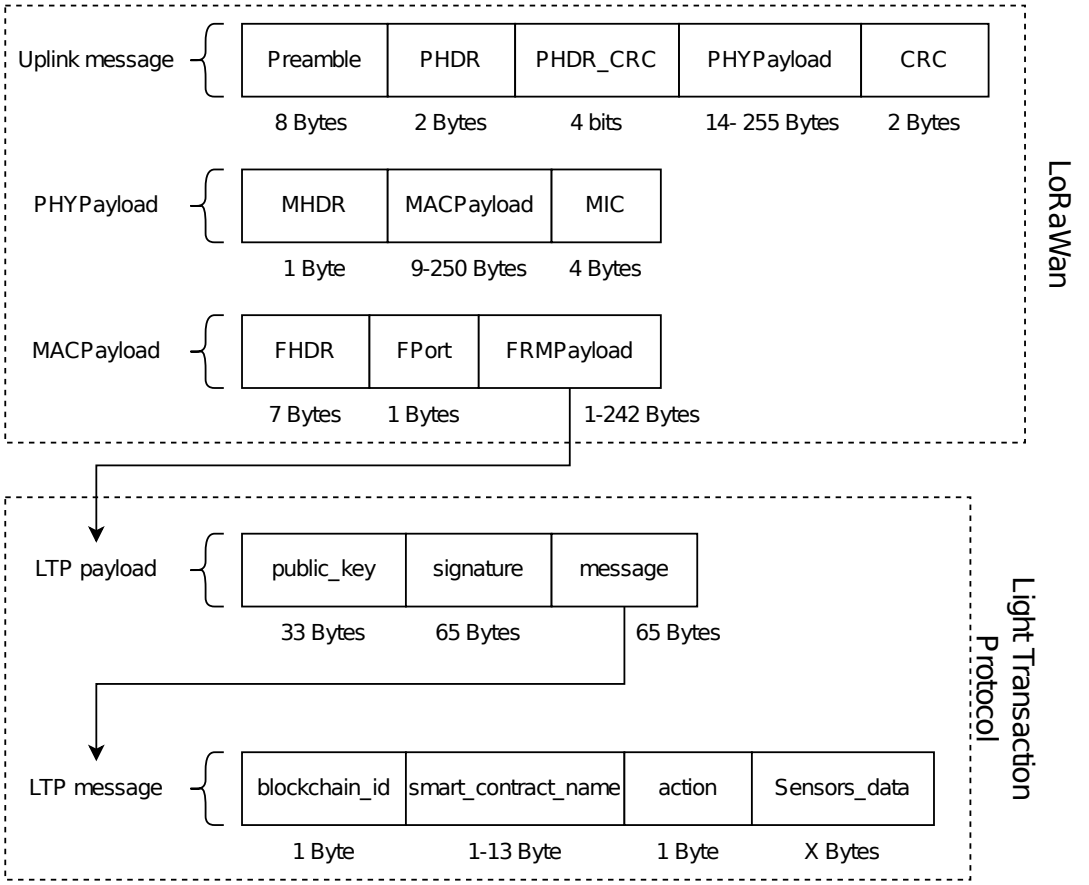


Figure 4.5.3: LoRaWAN and LTP message structure

The LTP implementation uses i) Protobuf data format to serialize the structured LTP data, ii) the C library trezor-crypto for data hashing and signature generation and iii) LoRaWAN library called rfthings-stm32l4[8] which is derived from LacunaSpace code. The signature is based on ECDSA with the secp256k1 curve, thus using **32 bytes private key**, a **33 bytes *public_key*** (ECDSA compressed public key), and a **65 Bytes signature**. The ***blockchain_id* and the *action* are encoded on 1 byte** and are represented in Protobuf by an enumerations type. The ***smart_contract_name* is encoded on 1 to 13 bytes** and represented in Protobuf by a string. Finally, the **sensor data is encoded in a variable *sensors_data* Byte array**.

The global architecture Figure 4.5.4 shows that the gateway is essential as it acts as the data proxy to the blockchains. The gateway contains a list of key-pairs that allows LTP messages to be encapsulated into valid blockchain transactions.

ℹ ChirpStack [222] is an open source project, a manual solution for LoRaWAN deployment, development, and network management. In other words, it provides a solution to deploy a private gateway.
ℹ MQTT [223] is a lightweight protocol that makes easier the communication between client and broker (server) applications using publish/subscribe methods.

The gateway contains a LoRaWAN radio module in a Chirpstack gateway module, and the proposed Gateway module API (i.e. data proxy). The radio module is linked to a Chirpstack gateway

[8] github.com/RFThings/rfthings-stm32l4

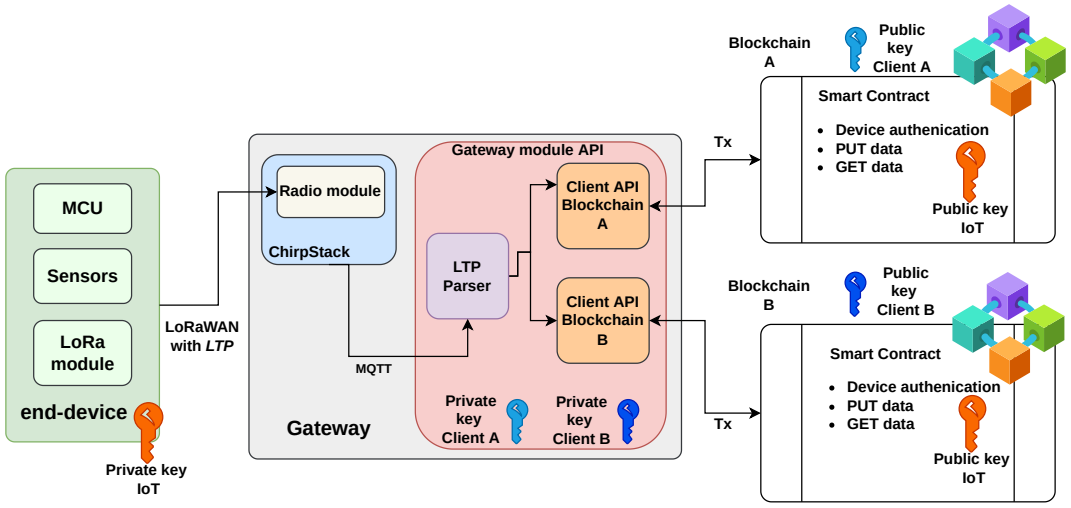


Figure 4.5.4: LoRaWAN and LTP global architecture: End-devices, Gateway (with module API), and Blockchains

module which converts the LoRaWAN packets into UDP/MQTT. Usually the UDP/MQTT packets are forwarded to a back-end (e.g., the Things Network server). However, in our implementation, the back-end is replaced by our gateway module API which creates blockchain transactions and forward them to the correct blockchain platform.

Thanks to an MQTT Client implemented in the gateway module API, the API can receive the MQTT packets forwarded by the ChirpStack module. The packages are set to a generic queue in order to avoid any loss. It is worth noting that the core implementation of the gateway module API is based on Rust programming language. The ThreadPool official Rust library was applied to achieve a multi-threaded environment and thus accelerate tasks execution.

4.5.4 End-Device Energy Analysis

The UCA Development board is equipped with an ultra-low-power Arm Cortex M4 32 bit STM32L476RG processor, 245 KB flash memory, running at 80 MHz, and uses LoRaWAN communication protocol thanks to the SX1262 Semtech module.

For testing purposes, the sent data is generated by two sensors: the HP203B precision barometer and altimeter sensor and the ICM20948 accelerometer. The HP203B retrieve temperature, altitude and pressure data. The ICM20948 retrieve x, y, and z axis accelerations.

4.5.4.1 Measurements

We used the OTII Arc device to measure the energy consumption. The OTII interface allows serial link communication with the development board, which is used to synchronize the software and energy consumption. Measurement (time, current, energy, and serial output) are exported in CSV files with a 4KHz sampling frequency and an accuracy of $\pm(0.1\% + 150\mu A)$. Software on the UCA board is synchronized (for each analyzed functions) with two or three bytes which are sent on the OTII serial link at a bit rate of 115200 bauds, which can add a delay of 0.19ms to 0.29ms on the measurements.

4.5.4.2 LoRaWAN Configuration

The LoRaWAN module transmission of the end device use a spreading factor of 7, an end-point output power (*TXPower*) of 16 dBm, a custom preamble length of 984 symbol, disabled reception, and a transmission frequency of 865MHz. The energy engaged by the LoRaWAN module amplifier can fluctuate according to the measurement environment, thus the experiments are conducted in a fixed environment, avoiding external disturbances.

4.5.4.3 About Data Size

The use of the LTL requires minimum 124 bytes because of the blockchain transaction requirements.
Protobuf encoding consist of key-value pairs, with the key representing two information. Increasing the number of fields in the Protobuf message causes the encoding of the key to vary (i.e. it may be using more than one byte). The remaining 118 free bytes can be used for other purposes. If only floating values are sent, the payload will contain 22 fields, assuming 5 bytes per field (1 for the key + 4 bytes for float representation).

4.5.4.4 Comparison Results

We compare the energy consumption of the runtime using the minimal (4 bytes) and the maximal (88 bytes) payload. The results given in Table 4.5.1 shows that the sign message function (i.e. sign the LTP transaction) has the biggest impact of the LTL energy consumption. The use of the library is always about 4 mJ and does not depend on the data size sent. However, LoRaWAN communication takes 333.17 mJ to send 4 bytes and 375.75 mJ to send 88 bytes. Thus the overall energy consumption depends on this LoRaWAN communication step and not on LTL. Therefore, increasing data size will primarily increase the energy consumption of the LoRaWAN communication.

Testing 10 times sending an LTP message and measuring the current consumption results in the Table 4.5.2. We observe the total energy consumption increases of 19% and the memory footprint increases of 101.6% because of LTP.

		Sensors data size	
		4 bytes	88 bytes
LTL (mJ)	Encoding message	<1	<1
	Hash message	<1	<1
	Sign message	4.19	4.20
	Encoding payload	<1	<1
	Total LTL	4.23	4.24
LoRaWAN communication (mJ)		333.17	375.75

Table 4.5.1: Runtime energy consumption of LTL functions

4.5.5 Gateway Latency Analysis

The gateway module API are deployed and LTP packets processing latency are being studied. Depending on the blockchain, the time for the transaction processing can variate because the module API waits for the transaction finalization. Also, we analyze the incoming packets concurrency impact, the resulting module latencies measurements are depicted in Figure 4.5.5.

	Without LTL	With LTL
Memory Footprint (kBytes)	57.1	115.1
Data sent by LoRaWAN (Bytes)	24	154
Avg ΔT (s)	1.40	1.70
Avg ΔE (mJ)	311.31	370.65
STD ΔE (mJ)	0.22	0.42

Table 4.5.2: Energy, memory footprint, and time impact of LTL (average of 10 measurements)

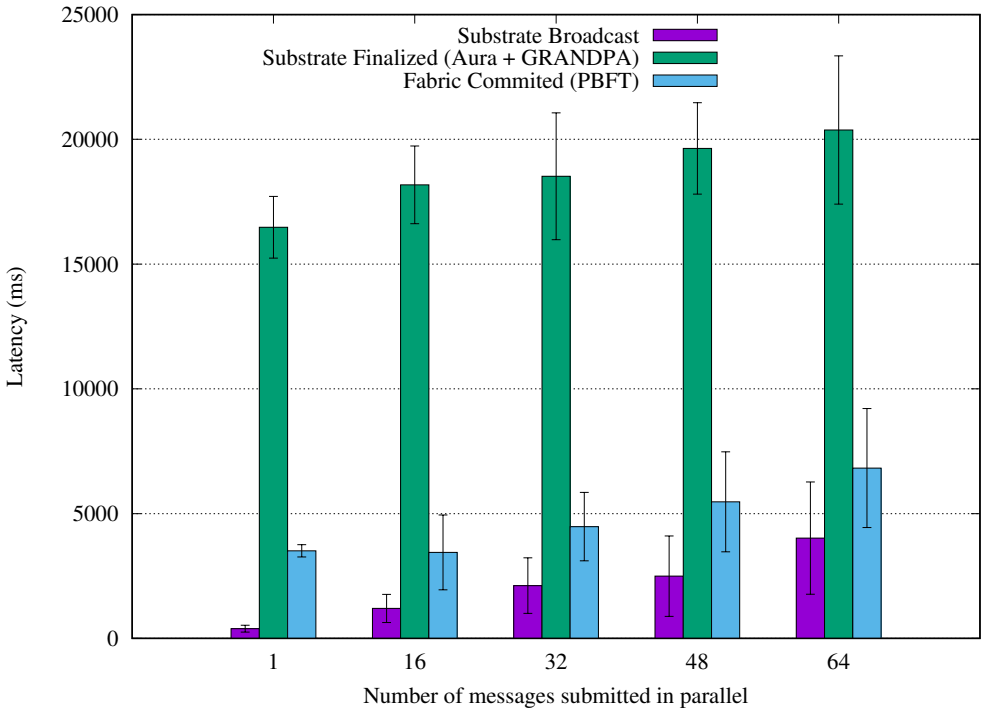


Figure 4.5.5: Gateway Blockchain Module Latencies +/- standard deviation over 5 measurements

The first observation is that the latency of the module increases when submitting more messages to the gateway, independently of the blockchain used. The latency depends on the targeted blockchain (Substrate based blockchain or Hyperledger Fabric) and depends on the blockchain transaction finalization we choose. As explained previously the module will always wait for the HTTP/ws status after sending a transaction.

In the case of a Substrate-based blockchain, a transaction status can either be *broadcasted*, *inblock*, or *finalized*. When only *broadcasting* the transaction, the module does not wait if it has reached blockchain consensus. When waiting the *finalized* status, the transaction has reached the consensus process and thus stored the transaction in the blockchain ledger (i.e. using a block time of 6 seconds, the consensus GRANDPA is reached after 3 block time, thus 18 seconds).

In the case of Fabric, a transaction status is known directly after sending the transaction (*pending*, *failed*, or *committed*), thus when successful, a transaction is *committed* and has reached the consensus (PBFT) process.

The latency of the Substrate broadcast mode is five times lower than the latency in finalized mode. In the latter, the transaction has to wait three block-time before being finalized. However, in broadcast mode, Substrate only validates and queues transactions. Hyperledger Fabric latency is close to Substrate broadcast mode, which is due to Fabric PBFT consensus algorithm. In

other words, the finalized latency in Substrate is higher than in Fabric due to performing two consensus algorithms (Aura and GRANDPA).

4.5.6 Conclusion on LTP with LoRaWAN

A gateway can process multiple messages at the same time (batches of 16, 32, 48 and 64 messages). The increase in latency shows the hardware limits of the gateway that cannot allow the module to increase indefinitely the number of threads. One batch of 64 transaction has an average latency of 6825.06ms (using Fabric). Meaning each message will create a new thread and takes on average 6825.06ms to be committed to the ledger. Thus, sending too many messages will overflow the number of threads and in the same way increase latency drastically.

The results in Figure 4.5.5 are a key element to understand the maximum end-device LoRaWAN uplink duty cycle. If we have a high probability to send multiple messages at the same time, the end-device uplink should at least have a duty cycle greater or equal to the corresponding latency of the number of messages submitted in parallel (e.g., high probability of 16 LoRaWAN messages in parallel requires the end-device uplink duty cycle to be at least 3444.26ms when using Fabric).

4.6 Android Automotive App to Communicate Vehicle Data Using Blockchain

In our exploration of blockchain technology within the automotive domain, we have previously discussed applying and benchmarking blockchain client software on various platforms. These initial investigations laid the foundation for a more ambitious endeavor: deploying **a blockchain client directly within a vehicle**, aligning with the initiatives outlined in the Smart IoT for Mobility project. This transition marks a step towards realizing the project's objectives and addressing the complexities of practical integration of blockchain technologies into automotive environments. The previous work of the thesis and collaboration with Renault car manufacturer, allowed us to propose an Android app implementation for Android Automotive OS. This joined work with Thomas Mabrut (project engineer part of SIM) was published in 2023 Fifth International Conference on Blockchain Computing and Applications [205].
This section prepares the groundwork for further discussions and enhancements concerning the technical details of implementing blockchain clients in vehicles. It highlights the project's contribution to the advancement of automotive technology and data management practices. Chapter 5 explores an extension of this sections work by using multi-chain.

The integration of Android Automotive OS (AAOS) into modern vehicles is part of the evolution towards Software-Defined Vehicles (SDV), necessitating a shift to connected and upgradeable automotive systems. This shift, previously introduced in the thesis introduction in Chapter 1, underscores the automotive industry's trajectory to vehicles that not only enhances the user experience through improved connectivity and functionalities but also serves as platforms for continuous software innovations and service upgrades.
The integration of blockchain technology within the AAOS ecosystem could facilitate an even more interconnected enterprise ecosystem with frictionless inter-organizational communication experience, pushing innovative automotive services and a secure transactional oriented platforms.

Our implementation focuses on efficiently managing vehicle data records in the event of an accident, utilizing blockchain and IPFS. The process involves collecting, formatting, and transmitting vehicle data to ensure secure and reliable record-keeping. The Figure 4.6.1 is an illustration with a vehicle-centric view of the broader Smart IoT for Mobility ecosystem presented in the introduction chapter, providing a simple visual representation of the interconnected system components and data flow.

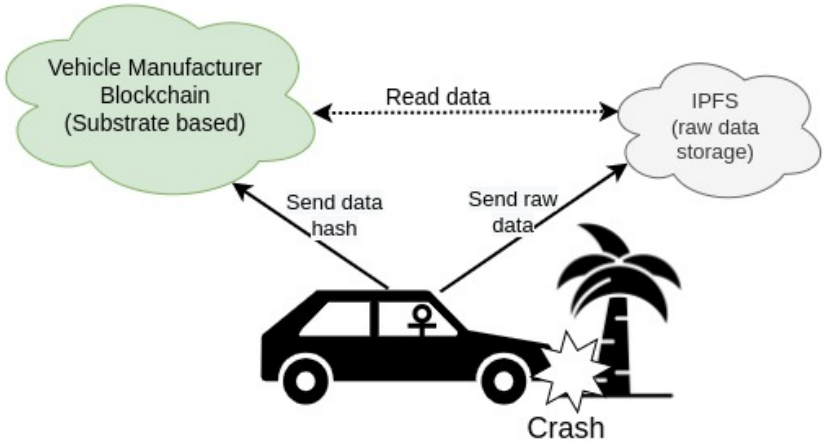


Figure 4.6.1: SIM accident use case: A vehicle-centric point-of-view using an Android Automotive application connected to a blockchain and IPFS.

The project confronts several challenges introduced mostly previously where technology platform diversity is high. Coding ”all” client application to provide full compatibility to communicate with any blockchain is not realist. Also off-chain participation of the vehicle device is a mandatory design due to the unreliable communication a vehicle can have on roads.
While privacy and confidentiality remain critical aspects of data management, it’s important to note that the paper primarily explores the feasibility, scalability, and certain security dimensions of the implementation (which relies on the intrinsic security of blockchain). Nonetheless, as we will describe next, the results of this project is successful in creating a trusted environment for data sharing among collaborative services, addressing scalability through the deployment of blockchain technology, and enhancing security protocols to protect sensitive data.

4.6.1 Overview of the Proposed Implementation

In our proposed implementation[9], AAOS plays a crucial role in interfacing with the vehicle’s infotainment system. It allows for the extraction of a comprehensive set of vehicle information. The system is engineered to collect real-time data metrics that are essential in the event of an accident. These include vehicle speed, acceleration data, GPS positioning, etc. All these data points can be used in drawing a detailed picture of the vehicle’s status before, during, and after an accident occurs.
As we have explained in previous parts of the thesis, the target blockchain is Substrate-based due to its modularity, good performances and potential for cross-communication with other blockchains.

[9]Code available here: github.com/projet-SIM/android-automotive-blockchain-2023

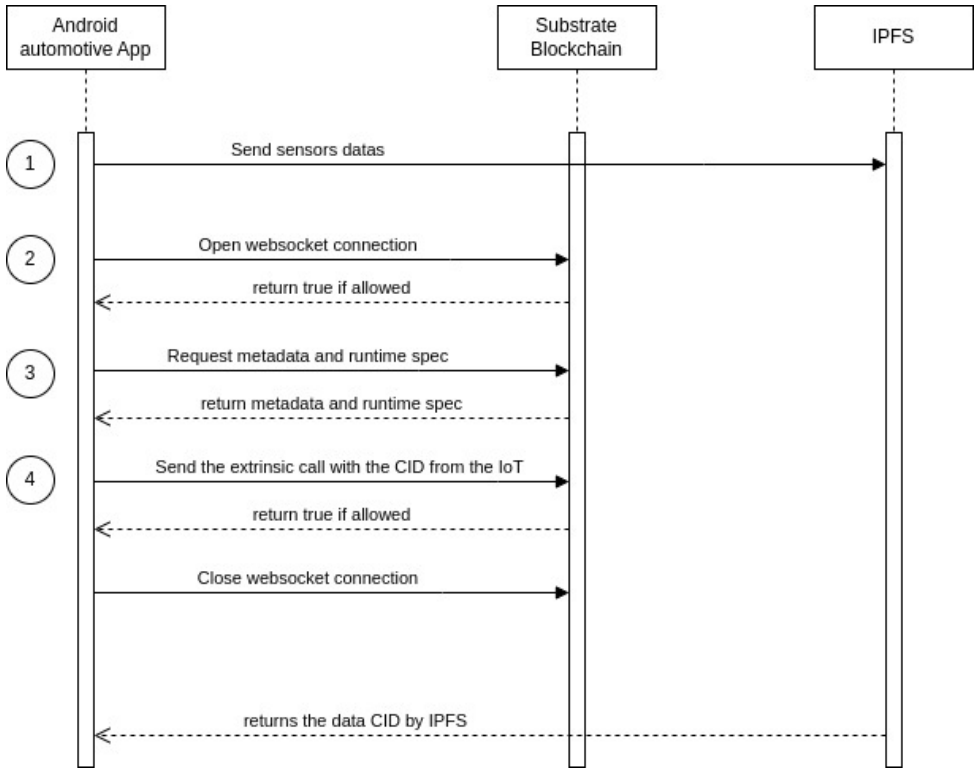


Figure 4.6.2: Interactions between vehicle-blockchain-IPFS after an accident occurs.

In the event of an accident, vehicle data is recorded and serialized using Protocol Buffers (Protobuf). This not only optimizes the data for storage but ensures its integrity during transmission. The sequence of interactions, as illustrated in Figure 4.6.2, starts with the essential step ① by dispatching this serialized data to the InterPlanetary File System (IPFS).
In preparation for potential post-accident connectivity issues, we preemptively generate an offline Content Identifier (CID). This allows to prevent additional delays and speeds-up the first step.

This identifier is unique to each data packet and is essential for retrieving the stored data from the IPFS as it allows to locate it. The subsequent interactions, done by steps ② to ④ in Figure 4.6.2, involve the establishment of a secure WebSocket connection, verifying permissions, and finally sending the extrinsic call along with the CID to the blockchain.

This private blockchain, curated by the vehicle manufacturer, is the repository that ensures the CID-and by extension, the accident data—is immutable and traceable. It is within this secure digital ledger that the integrity of vehicular data is registered, accessible through the structured protocols of smart contract (created in Chapter 3). The data's journey, from the vehicle's sensors to the immutable ledgers of blockchain, is designed in the Substrate Framework to allow in a last part to communicate with other blockchains using Substrate standard cross-transaction protocol.

4.6.2 Android Automotive Application Design

Our Android Automotive application is created to be running on the vehicle's infotainment system to access a broad spectrum of sensor data (136 sensors types to be exact) each producing 32-bit integer values. The application necessitated a data categorization based on the sensor type. Data is divided into two streams: **periodic records** for constant monitoring and

event-driven records for capturing data pertinent to specific incidents, such as accidents, gear shit change, etc.
We record the last 10 events for each category, obtaining the following data size estimation (4.6.1). We define the acronyms TS for timestamp and SD for sensor data.

$$\begin{aligned} &10 \text{ records} * (64 \text{ TS bits} + 5 \text{ SD} * 32 \text{ SD bits}) \\ +&10 \text{ records} * 131 \text{ SD} * (64 \text{ TS bits} + 32 \text{ SD bits}) \\ &= 128000 \text{ bits} \approx 128.0 \text{ kbits} \end{aligned} \tag{4.6.1}$$

In this initial proof of concept, the application doesn't process video or images. These type of data would significantly increase the storage requirements to several gigabytes. Naturally, with today bandwidth speed, sending 128 kbits (over the internet) is relatively fast, but increasing the data will result in significant latency.

To validate the functionality and reliability of our application, we simulate the application's environment using open-source Android Studio software that can emulate an Automotive Operating System through Android Virtual Devices (AVD). Specifically, we employ a system image created by Volvo OEM for the Polestar 2 vehicle (based on API v-29.2, Android 10.0, and supports X86 64 architecture).

The application is a combination of Java code using also a wrapped version of sr25519 Rust cryptography library to sign transactions for the Substrate-based blockchain.
With this emulator it is possible to have an application dashboard interface for presenting various details about the vehicle (speed, GPS ...) as well as information related to blockchain (metadata, transaction status ...). The application also incorporates extra debugging tools, like buttons to manually trigger accident detection.

4.6.3 Performance Analysis and Results

To measure the application's latency, we used two setups. The local setup runs the blockchain and IPFS software directly on a computer with a single blockchain validator node. For the cloud setup, we used a test cluster managed by a cloud service, using three blockchain validator nodes and a private IPFS network for the decentralized storage.

The application's performance was measured in terms of latency, with the local setup achieving a latency time of just 1.065 seconds to the blockchain and a mere 0.023 seconds to send raw data to IPFS. When operating through a cloud setup, these times were understandably longer, at 4.971 seconds and 2.134 seconds respectively, leading to a total latency of 7.105 seconds in the cloud compared to only 1.088 seconds locally. This significant difference underscores the impact of network conditions on the system's responsiveness and highlights the necessity for efficient network infrastructure in vehicular environments to optimize data recording.

Similarily as Chapter 3 Section 3.4.4.5, the blockchain's throughput was evaluated under different input Transaction Per Second (TPS) rates, with the average output TPS demonstrating the system's capacity to process up to 560 transactions per second.
Interestingly, the block time remained consistent at 6 seconds, regardless of the input TPS, illustrating the stability and reliability of the blockchain's consensus mechanism (Aura consensus). Even under a heavy load of up to 2000 input TPS, the system's test time bottomed out at around 12 seconds, indicating that while the blockchain could queue transactions efficiently, a transaction queue limit exists, emphasizing the importance of considering a correctly setup transaction

memory pool size for high-frequency transaction scenarios.

4.6.4 Future Directions and Potential Enhancements

This implementation establishes a foundation for secure and efficient vehicle accident data management. However, as we look towards future technological evolutions, there are multiple potential enhancements that could improve the system. One such area is enhancing security in the application memory, particularly for key management. The integration of hardware security modules (cryptographic IPs) could significantly reinforce the system's defenses against cyber threats, ensuring that the vehicle's keys and that signature generation are protected.

Additionally, the confidentiality and privacy of the data handled by our application are very important. Upcoming implementations could explore more into privacy-preserving blockchain technologies, such as zero-knowledge proofs, to ensure sensitive data is managed securely while complying to the existing (complex) regulatory standards, including the European General Data Protection Regulation (GDPR).

Based on all these considerations, the next chapter introduces a natural progression of the project into the domain of multi-chain ecosystems. Leveraging the Substrate framework's inherent support for interoperability, we will investigate the creation of an ecosystem where blockchains can communicate seamlessly using Substrate's cross-communication protocols. This evolution towards a multi-chain environment not only extends the scope of interaction between different blockchain networks but also offers the potential for enhanced scalability, robustness, and most importantly **collaboration between automotive services**.

The transition to a multi-chain architecture promises to improve the capabilities of all interconnected individual chains. This enables to create an ecosystem where **data, assets, and services can exchange freely in a secure and efficient manner**. Such an ecosystem could facilitate more complex interactions highlighting a potential new generation of automotive services that are more interconnected and capable of leveraging collective resources.
The next chapter is about an exploration focusing on the technical aspects of multi-chain integration, the challenges it poses, and the opportunities it presents.

4.7 Conclusion on Client application Implementations

We explored the implementation and analysis of blockchain client applications within the realm of Blockchain for Internet of Things (BIoT), focusing on the integration challenges, energy considerations, and potential solutions for the SIM project.

4.7.1 Key Findings from Blockchain Client Application Analysis

Here we synthesizes the core insights and significant observations derived from our comprehensive analysis of blockchain client applications in the context of BIoT.

We did a comparison between off-chain and on-chain integration approaches for IoT and blockchain, highlighting that off-chain integration is more suitable for constrained devices due to its lower energy consumption and minimal storage requirements. Furthermore, we identify the role of a blockchain client as the software that generates and transmits valid blockchain transactions, noting that integrating these clients into IoT devices poses complexities due to the need for various cryptographic algorithms and RPC protocols across different blockchain platforms.

Our exploration covers multiple blockchain platforms, including Hyperledger Sawtooth, Ethereum, EOSIO, and those based on Substrate. Each platform presents unique challenges regarding client implementation, protocol compatibility, and cryptographic requirements. Particularly, profiling the Hyperledger Sawtooth C++ client application revealed that networking operations significantly consume computational resources, suggesting that future optimizations should focus on enhancing networking and data processing efficiency.

We proposed the Lightweight Transaction Protocol (LTP) as a solution to facilitate the integration of blockchain with smaller, constrained IoT devices. The design of LTP aims to minimize computational demands by abstracting the networking and format layer into a generic structure that is more amenable to IoT environments. The practical implementation of LTP, using LoRaWAN, demonstrated its viability. Through detailed analyses of energy consumption at the end-device level and latency at the gateway level, we provided insights into the scalability and performance of the LTP system in real-world scenarios.

4.7.2 Limitation and Challenges

While the research on blockchain client applications for BIoT presents promising directions, we are still encountering inherent limitations and challenges. Among these, the trade-off between enhanced security and decentralization versus increased energy consumption remains a significant concern. The research points toward potential energy optimizations but underscores that these advantages come with their own set of compromises.

The process of integrating blockchain clients into diverse IoT platforms presents difficulties, primarily due to the wide variation in device capabilities, communication protocols, and blockchain technologies. This heterogeneity adds a layer of complexity that can hinder, or even prevent, the widespread adoption of these technologies.

Security and privacy emerge as the next primary concerns as well. While blockchain can reinforce security in IoT applications, its immutable nature may conflict with privacy requirements and data regulation policies, particularly where sensitive or personal data is involved. Additionally, the security of IoT devices themselves remains a concern, as they can be vulnerable entry points into the blockchain network.

The reliability of blockchain-based IoT applications is heavily dependent on network connectivity for transaction processing. This reliance poses significant challenges in environments with poor or unstable network conditions, which can jeopardize the reliability and speed of data processing in real-time applications.

Lastly, the diversity of blockchain platforms, each with its own set of unique protocols and standards, complicates the development of a standardized client application approach. This diversity demands substantial customization and adaptation, further escalating the complexity and resource demands for effective deployment.

4.7.3 Future Directions and Considerations

- **Further Optimization and Scaling**: Further optimizations in terms of energy efficiency and computational load are essential, especially for scaling the solution to accommodate more IoT devices and blockchain platforms. This is linked to the work on true data decentralization separating the data management and raw data storage (e.g. IPFS) [204].
- **Security and Robustness**: It is critical to ensure the security and robustness of blockchain clients in IoT devices, future works on enhancing security on both IoT and blockchain side is key for practical industrial implementation. It is worth noting that removing blockchain state information in the LTP transaction, ultimately opens a security issue due to possible replay attack because transactions are therefore indistinguishable from one another (e.g. no transaction nonce). Future work would require to explore more "localized" nonce in the communication protocol (e.g. LoRaWAN frame counter), or other methods to prevent two-way communication directly with the blockchain.
- **Interoperability and Standards**: Given the diversity of blockchain platforms and IoT devices it is crucial to develop blockchain interoperability standards and protocols. This includes standardizing data formats, communication protocols, and security measures. This thesis takes this axis at heart and explored in-depth in the Chapter 5.
- **Real-World Deployment and Testing**: Following the previous work in [205], more extensive real-world testing and deployment of these blockchain-IoT integrations are needed to assess their practical implications, performance in diverse environments, and user acceptance.

To summarize, integrating blockchain technology with IoT devices offers a promising approach for enhancing security, reliability, and functionality in IoT applications. The research conducted provides valuable insights and lays groundwork for future developments in this domain. Continued exploration, optimization, and testing are essential for realizing the full potential of blockchain in IoT (BIoT).

4.7.4 A Final Word on Related Work

The discussion on blockchain interoperability, a pivotal theme of the subsequent chapter, is enriched by insights from the Smart Contract Invocation Protocol (SCIP) study [224]. SCIP offers a framework for integrating heterogeneous smart contracts across varied blockchain platforms. SCIP's approach, while primarily focused on smart contract integration, shares a common underlying principle with the Light Transaction Protocol (LTP) developed in this thesis – **facilitating seamless blockchain interoperability**.
SCIP's methodology in handling diverse blockchain technologies through a uniform interface

can relate to LTP's objectives of enabling efficient, cross-blockchain transactions for IoT devices. This alignment not only highlights the common goals shared by SCIP and LTP in addressing interoperability but also allows us to introduce the latest topic of this thesis in the next chapter. We will explore the complexities of blockchain interoperability in greater depth, focusing primarily on one blockchain platform (Polkadot/Substrate-based), demonstrating the feasibility of the SIM project automobile use case with a proof-of-concept.

5 Improving BIoT: Towards a Multi-Chain Ecosystem for IoT

5.1 Introduction

We have explored previously the intricate potential and integration of BIoT, however a key limitation is the heterogeneity of blockchain platforms and the heterogeneity of IoT devices that have to integrate the necessary client application to communicate with this technology.

A primary challenge in the widespread adoption and effectiveness of BIoT lies in the inherent heterogeneity of both blockchain platforms and IoT devices. The diverse landscape of blockchain technologies presents a fragmented ecosystem where interoperability is limited. This heterogeneity extends to IoT devices, which vary in capabilities, standards, and protocols, further complicating BIoT integration.
The need for IoT devices to integrate client applications compatible with specific blockchain platforms adds another layer of complexity. This fragmented environment hinders the seamless interactions between both organizations and devices, ultimately hindering the realization of BIoT's full potential.

Addressing these challenges requires a shift towards a more standardized interoperable blockchain ecosystem. In this chapter, we go through an in-depth study of Polkadot's network paradigm and the Substrate Framework potential to create a more unified and efficient multi-blockchain environment. The Polkadot protocol, with its unique multi-chain and cross-chain transaction capabilities, emerges as a beacon of standardization in the fragmented blockchain architecture landscape. Its design philosophy and technical base hold a possible key to enabling more united and interoperable interactions between diverse blockchain platforms and IoT devices.

This chapter aims to provide a comprehensive understanding of the multi-chain ecosystem's role in BIoT domain. We will study the Polkadot and Substrate Framework's capabilities by implementing our automotive use case, which will help us explore how a unified blockchain ecosystem can significantly enhance the IoT domain, marking a step forward in BIoT.

5.2 Perspectives of an Multi-Chain IoT Ecosystem

5.2.1 Layer 2 Blockchains and Blockchain-of-Blockchain Multi-Chain Solutions

Within the layered architecture of blockchain technologies (see Figure 5.2.1), Layer 2 solutions and blockchain-of-blockchains address distinct challenges related to scalability and interoperability. Layer 2 solutions, represented by frameworks like Optimism and StarkNet, operate on top of base blockchains such as Ethereum. They are designed to **enhance transaction speed and scale by handling transactions off the main chain and leveraging its security for finality**.

In contrast, blockchain-of-blockchains, as created by Polkadot and Substrate Framework, provide a foundational infrastructure enabling different Layer 1 blockchains to interoperate (called parachain in Polkadot). These systems **facilitate transactions across a network of independent chains through a central relay chain, which coordinates security and cross-chain communication**.

The divergence between these two approaches is architectural: Layer 2 solutions scale by building upon a single blockchain, improving its capacity for transaction processing. Blockchain-of-blockchains scale by connecting multiple blockchains, each potentially tailored for specific use cases, to working all together. This architecture not only spreads transaction loads but also empowers a variety of chains to interact within a single ecosystem.

For applications in the IoT domain, where numerous devices have to transact across multiple platforms, these two distinct architectures present various advantages. Simply put, Layer 2 solutions can offer a streamlined, cost-effective means for high-frequency transactions on a dominant ”main” blockchain. Meanwhile, a blockchain-of-blockchains could enable a more segmented approach, with different processes managed across multiple specialized chains.

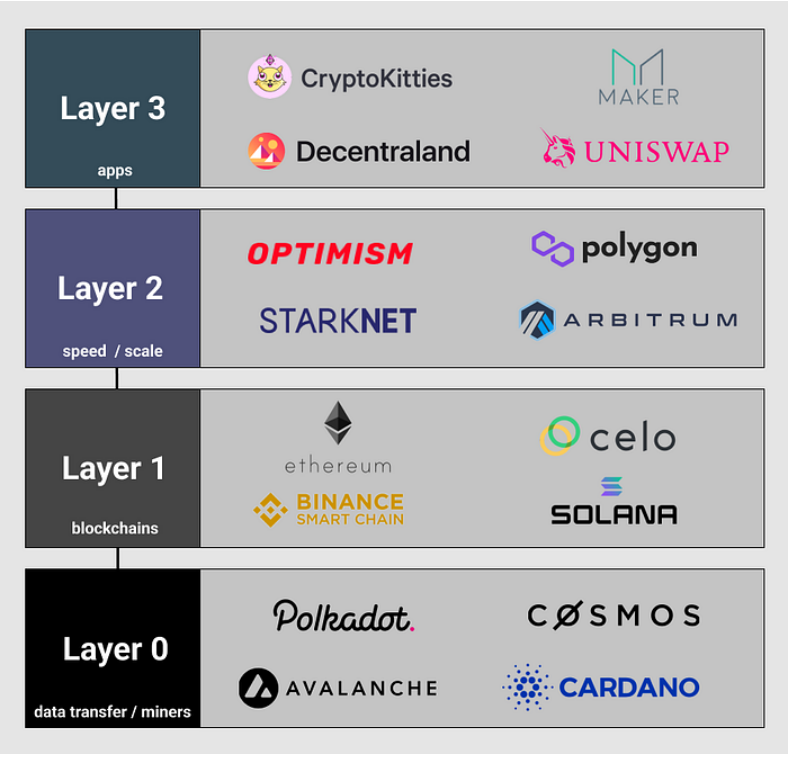


Figure 5.2.1: Blockchain Layers: Layer 0 (data transfer/miners), Layer 1 (blockchains), Layer 2 (speed/scale), Layer 3 (apps) [225]

At first glance, both Layer 2 solutions and blockchain-of-blockchains seem promising for enhancing scalability and interoperability within BIoT ecosystems. However, their suitability for specific applications varies significantly.
For instance, consider our use case in the automotive sector: accident reporting. In such scenarios, immediate and unambiguous data transmission is crucial for insurance and safety purposes. Layer 2 solutions stand out because of their high-throughput and quick finality features. They efficiently process the massive data streams generated by vehicles, but don't allow complex inter-chain data transfers which is by nature possible with blockchain-of-blockchains.

However, the accident use case have diverse stakeholders (manufacturers, insurers, repair shops, etc.) which might operate on different platforms and require robust cross-chain communication, thus a blockchain-of-blockchains approach is more suitable.
This model facilitates a more segmented approach, with different processes managed across multiple specialized chains, allowing for a granular level of control and customization. As we have designed with a Polkadot-based solution, individual parachains are able to handle specific aspects of the accident use case, such as claim processing, vehicle data management, and parts supply chain verification, all coordinated under a shared security model.

It's essential to recognize that most blockchain implementations have an economic model, supported by a native token or coin. These tokens facilitate incentivize participation and manage resources.
Therefore, while our use case is abstracted from these economic considerations, any practical deployment would require a carefully designed economic model to ensure long-term viability, network sustainability, and stakeholder engagement.

⚠ Therefore, when trying to choose between Layer 2 and blockchain-of-blockchains for a BIoT application, such as ours, it is important to weigh the architectural benefits against the economic sustainability of the network. In our case, the need for interoperability among the platform stakeholders led us to lean towards a blockchain-of-blockchains approach.
Although the initial phase of the project may discard the need of a native economic model to avoid complexity, it becomes imperative to acknowledge that the inclusion of a token or coin mechanism is vital in a real-world situation. Tokens play a crucial role in network security, governance, and incentivization, which are fundamental to the long-term success and adoption of blockchain or BIoT in general.

5.2.2 BIoT Practical Related Works

In the multi-chain implementation for the automotive use case, as explored in this thesis, the IoT device's role are building, signing, and sending transactions to a blockchain node. With our approach each IoT device acts as a participant in the blockchain network, engaging in transaction creation and communication with the blockchain, however it doesn't participate in the network consensus. We here describe two projects that are closely related, each with their specialties.

5.2.2.1 Comparative Analysis with IN3 and Helium in an Multi-Chain Automotive Use Case Context: Focus on IoT Client Implementation

The IN3 (INcentivized Node Network) project, deprecated since 2021, presents a different paradigm for IoT interaction with blockchain networks. IN3, primarily designed as a minimal verification client, focuses on providing IoT devices with a lightweight and efficient means to access

blockchain data. Unlike the ”send transaction-only” seen in our multi-chain implementation, IN3 allows IoT **devices to verify blockchain data without needing to maintain a full blockchain client** and allows transaction creation and signing. This distinction is crucial as it positions IoT devices in the IN3 framework more as data validators or verifiers, and can also create transactions.

IN3 operates as a stateless client, which means it does not store the entire blockchain or engage in constant synchronization with the network. Instead, it relies on a set of distributed full nodes for data verification. These nodes are part of the IN3 network and are incentivized to provide accurate and reliable data. To facilitate this, **IN3 creates an additional network layer on top of the original blockchain**, where these full nodes operate to assist IoT devices in data verification and interaction with the underlying blockchain. When an IoT device using IN3 needs to interact with the blockchain, it sends a request to these nodes. The nodes then respond with the necessary data, along with proofs that can be independently verified by the IoT device for authenticity and accuracy.

This mechanism allows IoT devices to access and verify blockchain data without the overhead of managing a full blockchain client. It significantly reduces the resource requirements on the IoT device, making blockchain interaction feasible even for devices with limited computational power and storage.

While our multi-chain implementation lets IoT devices interact by creating and sending transactions to the blockchain, the IN3 project has a more straightforward way of doing things. It lets IoT devices retrieve blockchain data and also make their own transactions, without using too much resources. This makes IN3 well-suited for IoT applications involving constrained devices, while still maintaining security and efficiency in blockchain interactions.

Moving to the Helium Project, it represents another unique approach in the realm of IoT and blockchain integration. Helium’s primary focus is on creating a decentralized network for IoT devices using a novel proof-of-coverage model. This model relies on a **network of Hotspots (gateways)**: physical devices that provide wireless coverage and act as miners on the blockchain. The Helium network uses a native token, HNT, to incentivize network participation and data transfer.

Key distinctions of the Helium network include:

- **Decentralized Wireless Network**: Helium creates a physical network of Hotspots that provide wide-area network coverage for IoT devices. This coverage is particularly useful for low-power devices that require long-range connectivity, such as sensors and trackers in automotive applications.
- **Proof-of-Coverage and Data Credits**: Helium’s novel consensus mechanism not only secures the network but also validates the wireless coverage provided by Hotspots. IoT devices use Data Credits, a secondary currency derived from HNT, to pay for network connectivity.
- **Scalability and Flexibility**: The network’s decentralized nature offers scalability and flexibility, making it well-suited for a wide range of IoT applications. This could fit the automotive scenarios where tracking, monitoring, and data aggregation are crucial.

In comparison to IN3, Helium offers a more hardware-centric approach, with a focus on building a LoRaWAN-compatible physical network infrastructure for IoT. While IN3’s minimal client architecture is efficient for lightweight interaction with commonly used blockchains such as Ethereum mainnet, Helium’s network provides the necessary infrastructure for large-scale IoT connectivity, compatible only with their own network.

Both IN3 and Helium shows the versatility and innovation in blockchain applications for IoT. However, key downsides is the lack of native interoperability in Helium, and IN3 has been deprecated since too long to extract some potential usage today.

Currently an existing parachain has introduced the IoT realm in the Polkadot ecosystem, called Nodle. Nodle provides an approach to IoT connectivity, leveraging the existing infrastructure of smartphones and the Polkadot network's interoperability features.

5.2.2.2 Introduction to Nodle Parachain: Smartphone as IoTs

Nodle is a parachain in the Polkadot ecosystem which represents a significant step in the convergence of IoT and blockchain technology. Unlike IN3's minimal verification client architecture and Helium's decentralized network of Hotspots, Nodle uses the large presence of existing smartphones to create a decentralized IoT network.

Now the key features of Nodle are:

- **Smartphone-Centric Network**: Usage of smartphones as nodes, utilizing Bluetooth Low Energy (BLE) for connectivity. This approach take advantage of the widespread availability of smartphones, turning them into a distributed network for IoT devices. IoT devices can use the smartphone Bluetooth connection as a secure "gateway" to transmit data.
- **Decentralized Data Transmission**: The network facilitates decentralized data exchange and verification using blockchain for better data integrity and security in IoT applications.
- **Tokenomics**: Nodle uses a native token (NODL), to incentivize network participation and transaction processing, mirroring a token economic model similar to Helium's.
- **Interoperability with Polkadot**: Being a parachain on Polkadot, Nodle benefits from the interoperability and shared security features of the Polkadot network, enhancing its potential for additional applications.

Nodle's approach to IoT connectivity (using smartphones) offers a new unique angle for our automotive use case. In urban environments where smartphone density and penetration is high, Nodle's network can be utilized for various purposes related to vehicle connectivity and telematics. For instance, smartphones, through Nodle's network, could act as relay points for transmitting vehicle data. Connected vehicles don't necessarily have internet access, and Nodle can provide a decentralized more open network to these vehicles with good coverage.

The Nodle model use existing smartphone infrastructure which prevent the need for additional dedicated IoT hardware, making it a cost-effective solution for automotive IoT applications. It also simplifies deployment and scaling, as it take advantage of devices that are already in widespread use. However, it's important to note that this model relies on the density of participating smartphones, which could limit or even stop its effectiveness in less urbanized areas. Incentivization towards installing Nodle App to receive compensation when your network is used has to be sufficiently interesting.

In comparison to Helium's LoRaWAN-compatible network, which is designed to cater to a wide range of IoT devices including those in constrained environments, Nodle's smartphone-centric approach might seem limited. However, Nodle can provide faster data transfer with Bluetooth and fills a niche in more urban situations where the density of smartphones is high.

Moreover, Nodle's integration into the Polkadot ecosystem adds another layer of utility, thanks to the interoperability and shared security features of the network. This could potentially allow **seamless data exchange and integration with other blockchain-based systems** within the

automotive industry, similarly proposed and demonstrated by our implementation using two Substrate-based parachains in the rest of this chapter.

5.2.2.3 Conclusion on Related Work

The comparison of IN3, Helium, and Nodle highlights the diverse methodologies in BIoT integration. While IN3 focuses on lightweight blockchain data access, Helium builds a dedicated physical network infrastructure, and Nodle innovates by turning smartphones into a decentralized IoT network. Each approach has its advantages and limitations, pointing to the necessity of a better understanding of our automotive context in the case we are willing to use such existing BIoT platform. Our work provided a demonstration using a methodology comparable to Nodle without usage of smartphones. Even if the implementation is at a PoC level, it showed that interoperability in blockchain is a key feature to potential adoption of blockchain in the automotive industry.

5.3 Shift Towards Multi-Chain Ecosystems

The emergence of new approaches is leading to a shift from existing closed-network blockchains to more open interoperable blockchain technologies, enabling interaction and cooperation among blockchains, users, and devices across numerous blockchain networks.

5.3.1 Quick Overview of Multi-Chain

An overview of interoperability blockchains are in Table 5.3.1. The blockchain ecosystem is an ever-evolving domain, with multi-chain only at its beginning stages. This dynamic field witnesses continuous developments in technology and strategies aimed at enhancing interoperability between different blockchain networks.

Technology	Initial apparition date	EVM compatibility	Specific blockchain development	Interoperability protocol	Architecture Topology
Polkadot	2017	Possible	Substrate Framework	XCM	Relay-chain, parachains
Cosmos	2016	Yes	Comos-SDK	IBC	Zones, hubs, blockchains
Avalanche	2020	Yes	Avalanche-cli	AWM	Three blockchains and Subnets

Table 5.3.1: Comparison of different multi-chain interoperable blockchains

A quick comparison of the platform is as follows:

- Cosmos BCT is a network of multiple blockchains interconnected by the Cosmos hubs. The Cosmos hubs contain different ”zones” that maintain a zone state. IBC (Inter-blockchain Communication) is the communication protocol used to exchange messages between blockchains. Up to 2020, Cosmos was primary limited to asset transfers, but by the end of 2021, a stable version of IBC was released allowing more generic information to be exchanged between blockchain networks. Blockchain bridges can provide interoperability for cryptocurrencies by using two one-way communications (one for each BCT).

- Avalanche is a blockchain platform that operates on a Proof-of-Stake consensus protocol and is designed with a unique architecture consisting of three chains: the Platform Chain (P-Chain) for network coordination, the Exchange Chain (X-Chain) for asset transfers, and the Contract Chain (C-Chain) for Ethereum-compatible smart contracts. The communication between the connected blockchains or "subnets" is performed by the so called Avalanche Wrap Messaging (AWM).

  ℹ At the time of writing, Avalanche emerged as a very good candidate to build and explore solutions related to the thesis project, however, we already have sufficient development using Polkadot.

- Polkadot is built using the Substrate blockchain framework and its purpose is to relay messages (Polkadot is a relay-chain) between multiple Substrate-based blockchains (parachains). The Cross-Consensus Message (XCM) is a message format for its associated protocol, the cross-chain communication protocol of Polkadot. The Polkadot ecosystem has standardized this XCM communication format. This format allows parachains to exchange messages between the relay-chain and parachains and between parachain themselves through the relay-chain. The resulting implemented interoperability protocol is called XCMP.

Polkadot, more precisely the Substrate Framework project, philosophy is to build the ground layer for a multi-chain ecosystem. This explains the deep involvement in focusing more on the global architecture design, architecture primitive modules and inter-communication protocols. The SDK provided by Substrate was rapidly containing most of the necessary tools to build and experiment with private and multi-chain – in the case of the thesis – blockchains. The aim using such platform SDK is to experiment the requirements and implementation of an heterogeneous, scattered industrial use case, bringing them together in a unified trusted ecosystem.

At the time of writing, Substrate Framework finally reached version v1.0.0 of its SDK. This marks a next step on their development, which will now focus more on the application layer.

5.3.2 Interoperability: From a Company Point of View

The concept of interoperability in blockchain technology is frequently misinterpreted due to its evolving nature and the varying needs of different applications. Initially, the focus of blockchain development was on creating one standalone system that everyone would use, with little consideration for how these systems would interact or integrate with new other networks, thus being isolated. As the blockchain landscape matured, especially post-2016, it became evident that for broader adoption and functionality, especially in corporate world, these systems needed to communicate and work together seamlessly. This shift in focus marked the beginning of the era of interoperability in blockchain technology, presenting new potential for organizations.

Organizations considering blockchain technology now have to face the following interoperability options:

1. **Consortium Chains**: This approach involves multiple companies using a shared blockchain network, inherently creating an interoperable service. An example is a group of financial institutions sharing a blockchain to record transactions, ensuring consistent and transparent data across all entities.
2. **Using Public Connectors and Hybrid Connectors**: Companies can utilize public connectors like Atomic Swaps, Sidechains/Relays, or Notary Schemes for interoperability between

different blockchain systems. These mechanisms facilitate asset or data transfer across various blockchain platforms, enabling transactions without relying on centralized exchanges. Similarly, hybrid connectors, such as Trusted Relays or Blockchain-Agnostic Protocols, allow for interactions between diverse blockchains. An example is a supply chain network where a product's journey is recorded on different blockchains, and connectors facilitate data exchange to track the product lifecycle across these platforms.

3. **Native Built-In Interoperability Protocols**: Some ecosystems are designed with inherent interoperability (i.e. Blockchain-of-Blockchains), functional within their specific network. A practical example is a multi-service digital platform where each service operates on its dedicated blockchain, but all services can interact and exchange data as part of a larger, interconnected system. Blockchain platforms like Polkadot or Cosmos exemplify this, where various blockchains within their network can natively communicate.

Each approach presents unique advantages and challenges. Organizations must consider factors such as the level of control, scalability, and the complexity of integration processes when selecting the most suitable interoperability solution for their needs.

The paper by Ballatore et al. introduces a comprehensive exploration into the implementation of a multi-chain blockchain architecture in the automotive industry [17]. This research, using a design science approach, underscores the significance of multi-chain environments for facilitating interorganizational data sharing in the connected vehicle context. The study demonstrates how the Polkadot/Substrate framework, a leading blockchain-of-blockchains platform system, can be utilized to construct a multi-chain ecosystem that balances the decentralization benefits of public blockchains with the control offered by private ones. This approach is helpful in addressing the challenges of data-sharing and interoperability in complex, multi-stakeholder environments like the automotive industry, thereby aligning closely with the principles of effective organizational level blockchain integration. The insights from this paper improve the understanding of blockchain interoperability from a practical standpoint, especially in scenarios where collaboration and data exchange across diverse organizations is critical.

5.3.3 Economic Implications

Adopting multi-chain blockchain architectures in IoT systems has distinct economic implications. The primary focus is on optimizing resource usage and managing costs effectively.

5.3.3.1 Economic Potential

The shift towards multi-chain architectures in BIoT is more than a technological advancement; it announces a transformation in the economic and business dynamics within the IoT sphere. This approach not only opens up a multitude of new possibilities but also represents a fundamental change in how economic and business models are perceived and constructed in a blockchain-enabled world. This change in philosophy is creating opportunities that were previously unfeasible in a non-native interoperable blockchain ecosystem.

Unlocking New Economic Models: The groundbreaking nature of multi-chain structures in BIoT lies in their ability to enable seamless interoperability among diverse blockchain networks. This facilitates more efficient data exchange and collaboration between IoT devices across different sectors. As a result, it becomes possible to explore new economic models based

on (real-time) data sharing and complex multi-party interactions, which were not feasible with isolated blockchain systems.

Enhancing Data Monetization and Asset Management: With improved interoperability and security, multi-chain systems provide a more robust framework for data monetization. They allow for the secure aggregation and trading of IoT data, smart contracts can allow innovative asset management and value creation.

Driving Sector-Specific Innovations: Multi-chain BIoT systems are helpful in pushing forward more sector-specific innovations. By building a flexible and secure foundation, they enable the development of new applications designed to address unique challenges in various domains (compared to general purpose blockchains). This leads to improved operational efficiencies, cost reductions, and enhanced services.

5.3.3.2 Incentivization in Multi-Chain

Similar to public blockchains, multi-chain architectures necessitate effective incentivization mechanisms to encourage active participation and ensure network sustainability. In a multi-chain context, these incentives need to be strategically designed to fit the diverse needs of various stakeholders involved in the network.

Table 5.3.2: Comparison of Incentivization in Classic Public Chains vs. Multi-Chain Architectures

Aspect	Public Chain	Multi-Chain Architecture
Tokenization	Rewards typically in the form of native cryptocurrency for activities like mining or transaction validation.	Utilizes a variety of tokens for specific activities across different chains, plus a common token for the entire ecosystem.
Stakeholder Engagement	Primarily focused on miners or validators, with limited roles for other stakeholders.	Broader engagement strategies, including developers, users, and enterprises, often involving governance tokens or profit-sharing.
Quality of Service	Incentives primarily for maintaining network integrity and security.	Broader incentives encompassing service quality, such as efficient data handling and uptime reliability.
Cross-Chain Collaboration	Not applicable or very limited.	Specific incentives for facilitating cross-chain communication and interoperability.

As Table 5.3.2 shows, multi-chain architectures provide a broader and more comprehensive approach to incentivization compared to public chains (private and consortium chains are omitted because economic mechanism is specific to its use case). While multi-chain systems utilize a variety of tokens across different chains for specific activities, they also often employ **a common token that is integral to the entire ecosystem** (e.g. ”DOT”, ”ATOM”, or ”AVAX” cryptocurrencies). The common cryptocurrency facilitates seamless interactions across the multiple chains, reinforcing the unity, cohesiveness and security of the ecosystem network. However, keep in mind that each blockchain can create its own internal incentive economy with its own token cryptocurrency (e.g. the ”DOT” token used to send cross-chain messages from one blockchain to another, each blockchains containing their own token and token economy).

Stakeholder engagement in multi-chain networks is more inclusive, extending beyond miners and validators to incorporate developers, users, and enterprises. This engagement is often facilitated by governance tokens or profit-sharing models, allowing a wider range of participants to contribute to and benefit from the network.

The role of quality service incentives and cross-chain collaboration is also more pronounced in multi-chain architectures. These incentives are crucial for maintaining high service standards and encouraging the development of protocols and practices that enhance interoperability, thereby increasing the overall functionality and utility of the network.

5.4 Polkadot Relay-Chain and Parachain Model

This part of the chapter is devoted to defining and discussing the multi-chain paradigm that is put forward by the Polkadot platform and the Substrate community in general. Drawing from the preliminary conclusions presented in Section 2.5, and considering the current stability and modularity of the Polkadot/Substrate Framework, this thesis will concentrate on and develop solutions utilizing these technologies.

5.4.1 The Polkadot Multi-Chain Model Using Substrate Framework

This subsection explores the structure and components of a blockchain built using the Substrate framework, with a specific focus on the Polkadot multi-chain model. The Polkadot network is characterized by its unique architecture, comprising the Relay-Chain, Parachains, the GRANDPA consensus algorithm, and the XCM format for cross-consensus messaging. The Substrate SDK, integral to this ecosystem, provides tools for blockchain development, including the Cumulus SDK for creating Polkadot parachains.

At the heart of Polkadot's design philosophy is the idea of interoperability and scalability, aiming to overcome common challenges in blockchain networks. It facilitates communication and connection between multiple blockchains in a unified network. The Relay-Chain coordinates network security and consensus, while parachains allow for specialized and diverse applications. These components collectively achieve a balance between centralized coordination and decentralized innovation.

5.4.1.1 A Relay-Chain, Parachains, Consensus, and Cross-Consensus Messages

A Relay-Chain: First, this blockchain that serves as the central chain of the Polkadot network. It is responsible for maintaining consensus across the network and allowing interoperability between the different parachains connected to it.
Validators nodes in the relay-chain play a key role, ensuring the integrity and security of the overall network.

Parachains: Second, we have these individual blockchains that run in parallel within the Polkadot network. Each parachain can have its unique features and use case, contributing to the overall ecosystem's diversity and functionality.
Collators nodes in parachains are responsible for collecting transactions and producing proposed blocks for the parachains, facilitating the processing of all types of transactions within the network.

GRANDPA: is the consensus algorithm used by Polkadot for finality, to ensure that transactions are immutable once they are added to the blockchain.

XCM (Cross-Consensus Message): is a format used for inter-chain communication within the Polkadot ecosystem. It allows different blockchains, whether parachains or external networks, to communicate and transact with one another seamlessly.

Nominated Proof of Stake (NPoS): The NPoS mechanism is the last component, used for Polkadot's network security. It involves nominators, who secure the network by selecting trustworthy validators, and validators, who are responsible for producing new blocks and validating parachain blocks (vote and stake pools are on-chain). This system not only incentivizes good behavior but also ensures network security and integrity by punishing malicious activities through slashing (penalty mechanism).

5.4.1.2 A Multi-Consensus Ecosystem

The Polkadot network, by design, accommodates a multi-consensus ecosystem, enabling each parachain to choose a consensus mechanism best suited to its unique needs and functionalities. This flexibility is essential in Polkadot's architecture, allowing interconnected consensus-heterogeneous blockchains, each optimized for specific purposes while contributing to the overall network's with specialized features and applications.

Diverse Consensus Mechanisms: In the Polkadot ecosystem, parachains are not restricted to a single, one-size-fits-all consensus model. Instead, they have the liberty to implement various consensus algorithms, such as Proof of Stake (PoS), Proof of Work (PoW), or even more permissioned ones like Proof of Authority (PoA). As depicted in Figure 5.4.1, this diversity in consensus mechanisms encourages innovation and specialization by allowing blockchain developers to tailor their infrastructure to specific use cases. Depending on the use case, the created parachain can be built specifically for high-speed transactions, energy efficiency, enhanced security, or other targeted functionalities.

Lastly, GRANDPA ensures the immutability and security of all transactions on the relay chain. In the event of reverting relay chain blocks through a hard fork, it would potentially necessitate all parachains to undergo a hard fork as well.

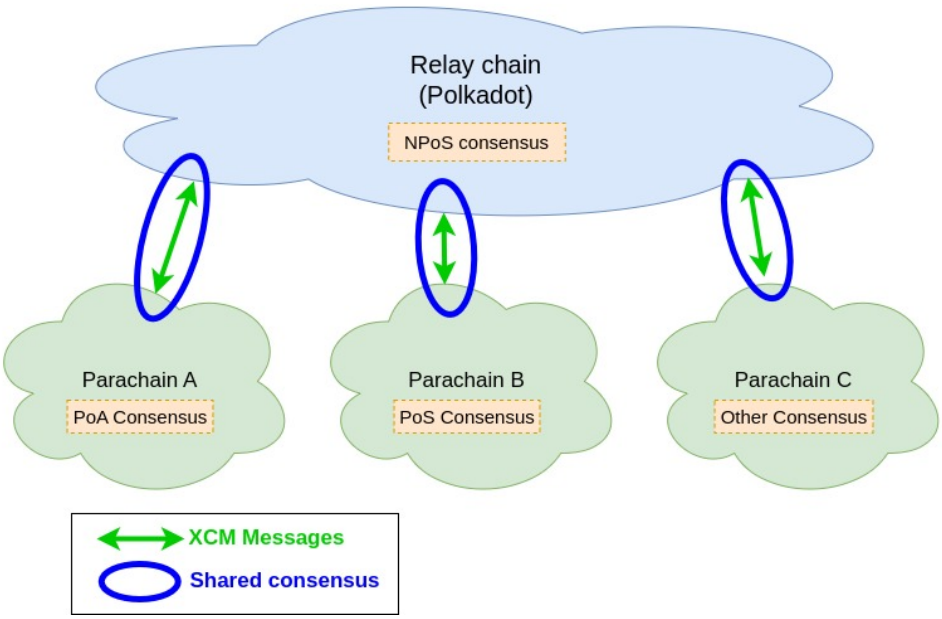


Figure 5.4.1: Illustration of the multi-consensus ecosystem in Polkadot

5.4.1.3 Encoding in Substrate Framework and Polkadot

The Substrate framework employs the SCALE codec (**S**imple **C**oncatenated **A**ggregate **L**ittle-**E**ndian) for efficient encoding and decoding, a vital process for data transmission within the network. This codec facilitates communication between the runtime and the node and is optimized for high-performance and low-overhead encoding and decoding, particularly in environments with limited resources, such as the Substrate WebAssembly runtime.

The SCALE codec is non-self-describing, meaning that complete knowledge of the encoded data's type is necessary for decoding. The `parity-scale-codec` Rust library, maintained by Parity, provides the necessary functionality for encoding and decoding data related to runtime RPCs.
The SCALE codec presents several advantages:

- Its lightweight design, offering a more efficient solution compared to general serialization frameworks like `serde` (serde is the native framework for serializing and deserializing Rust data structures).
- Compatibility with `no_std` environments, allowing its use in Wasm-compiled situations like the Substrate runtime.
- Streamlined support in Rust for adding codec logic to new types, simplifying the process of encoding and decoding (i.e. automatic attribution of en/de-code traits to underling items).

The choice to define a unique encoding scheme in Substrate, instead of reusing an existing Rust codec library, is driven by the need for interoperability among various platforms and languages within the Substrate blockchains. Substrate provide documentation[1] that specifies the encoding rules for most popular data types. The following example in Figure 5.4.2, is a transaction (i.e. extrinsic) encoded in a Substrate-based blockchain.
⚠ As stated before, SCALE is not self-describing, thus all encoded data has to be decoded using the according decoder structure. An extrinsic encoded data is only decoded by knowing the extrinsic structure definition. In other words, a transaction to transfer assets (`balance` extrinsic), can be decoded only if we know the `balance` structure: This is possible by encoding the extrinsic call (called *call index*).

5.4.1.3.1 SCALE Codec Implementations The SCALE Codec has been implemented in various languages, including Python, Golang, C, C++, JavaScript, TypeScript, AssemblyScript, Haskell, Java, and Ruby. This wide array of implementations underlines its adaptability and relevance across different programming environments.

5.4.1.4 Cross-Chain Communications

5.4.1.4.1 Overview of Cross-Consensus Messaging The Cross-Consensus Messaging (XCM) format in the Polkadot ecosystem facilitates transactional interoperability across various consensus systems. XCM allows for both vertical and horizontal message exchanges within the network through Upward Message Passing (UMP), Downward Message Passing (DMP), and the interim Horizontal Relay-routed Message Passing (HRMP), pending the full development of Cross-Consensus Message Passing (XCMP). This setup enables parachains to communicate with the relay chain and with each other (see Figure 5.4.3).

[1] https://docs.substrate.io/reference/scale-codec/

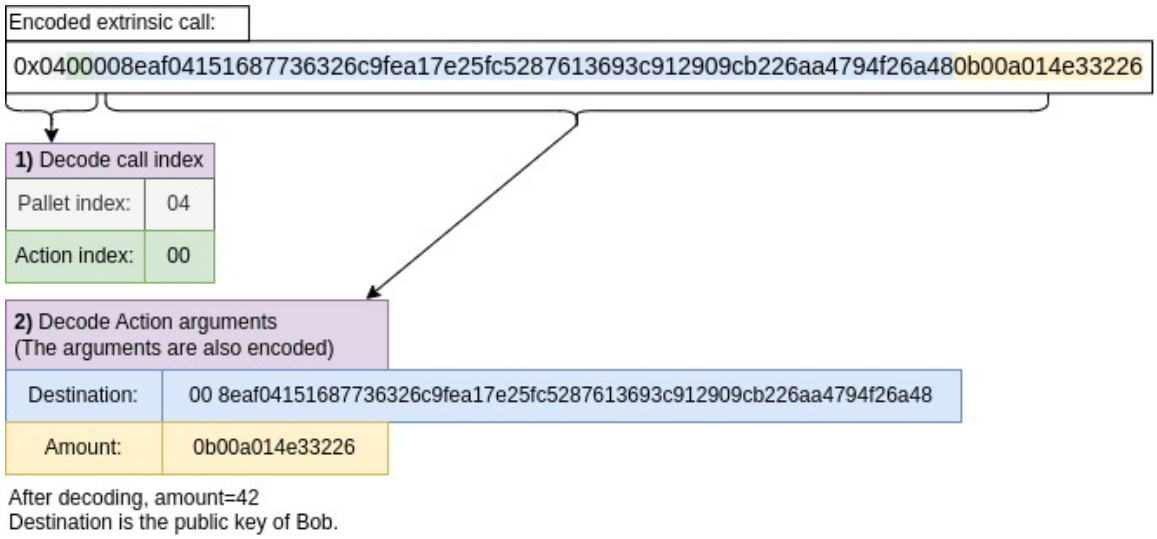


Figure 5.4.2: How to decode an extrinsic data that has been SCALE encoded?

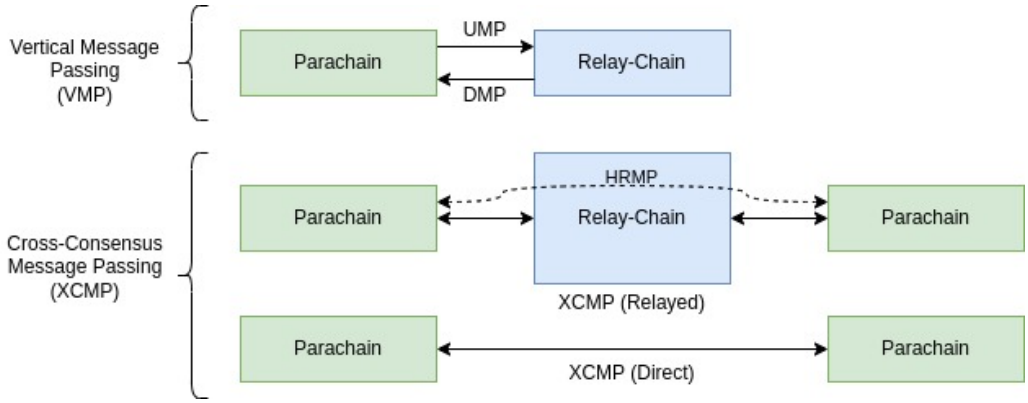


Figure 5.4.3: XCM transport possibilities: parachain-to-relay-chain, parachain-to-parachain relayed, or parachain-to-parachain direct

XCM Mechanics and Execution: XCM messages are asynchronous, absolute, and asymmetric, ensuring efficient delivery and execution without necessitating immediate response. The Cross-Consensus Virtual Machine (XCVM), a register-based state machine, executes these messages. The XCM Executor, integral to the Substrate and FRAME frameworks, interprets and executes XCM instructions with customizable handling options, including message processing barriers and fee configurations.

Defining Resources and Flexibility in XCM: XCM uses 'MultiLocation' and 'MultiAsset' types for versatile and precise definition of locations and assets across blockchain systems. A wide range of instructions, like 'Transact' and 'TransferAsset', provides the flexibility for various operations, including asset transfers and state modifications in different blockchain environments.

XCM's Role in Polkadot's Ecosystem: XCM's adaptability and operational diversity make it a key tool for interoperable blockchain communication within Polkadot. It supports a broad spectrum of tasks, from simple asset transfers to complex cross-chain operations, enhancing seamless integration and communication across different blockchain systems.

5.5 Towards a Multi-Chain Implementation of the Automotive Use Case

This section of the thesis aims to investigate further the implementation of a vehicle use case using blockchain technology, with a particular focus on its integration into the IoT framework. While Chapter 3 examined a single blockchain network used by all organizations, the need for a multi-chain architecture as highlighted in the multi-disciplinary work of M. Ballatore et al. [17], is addressed here. Multi-chain frameworks offer the flexibility and scalability required for complex automotive industry applications. This section extends the work of M. Ballatore et al. as presented in [17], by offering an enhanced implementation of the vehicular use case and providing additional performance results to demonstrate its efficacy.

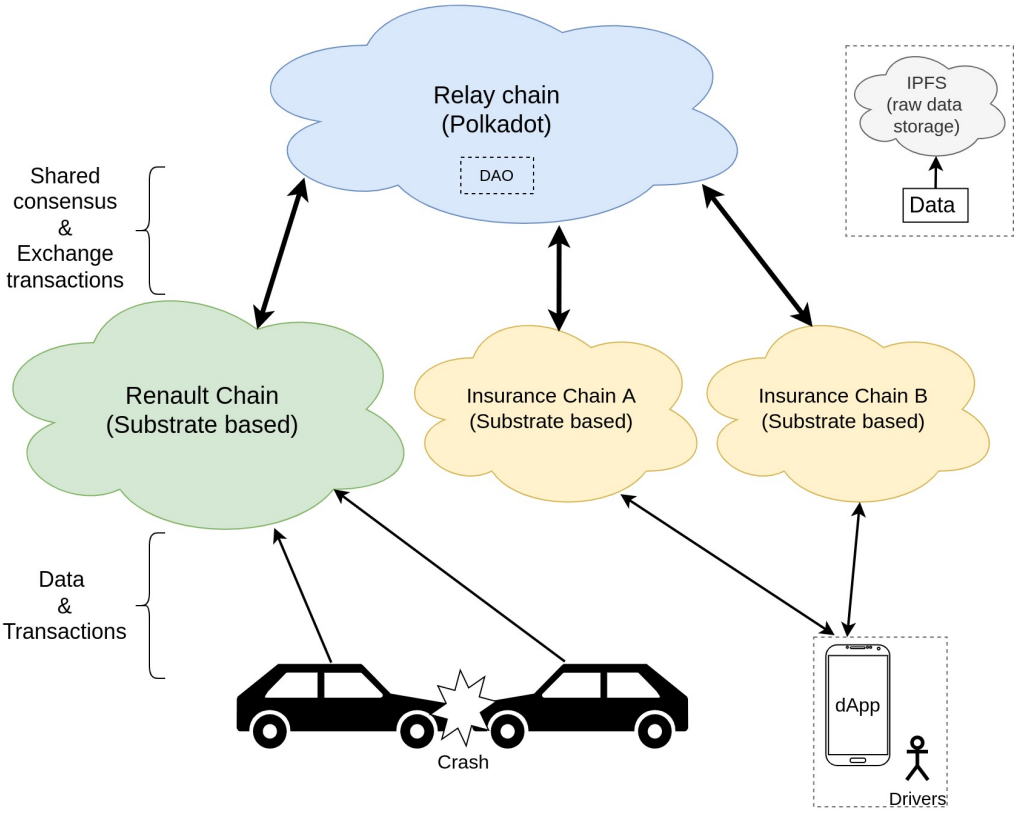


Figure 5.5.1: Two sided vehicle accident use-case within the multi-chain Polkadot ecosystem.

The figure 5.5.1 visualizes the interaction flow for a one-sided vehicle accident scenario within a multi-chain environment. In this model, each chain has a distinct role, enhancing the overall functionality and security of the ecosystem. The OEM Chain, built on Substrate, is the starting point for data capture, where vehicle data such as incident reports and telemetry are initially recorded. Following an accident, the vehicle transmits a data hash to the OEM Chain, which is subsequently stored securely and decentralized through IPFS.
The relay-chain, part of the Polkadot framework, does not manage the data directly but instead provides governance for the various parachains, enables cross-chain transactions, and ensures security across the ecosystem.
The Insurance Chain, another Substrate-based blockchain, initiates requests for vehicle data which are facilitated by the relay-chain's XCM messaging protocol. This ensures secure and

consistent data exchange necessary for processing insurance claims and accident reports.

This use case exemplifies the application of a multi-chain framework, highlighting how different blockchain technologies, including Polkadot and Substrate, work in concert to meet the sophisticated requirements of digital servitization in the automotive sector. It underscores the importance of a well-orchestrated multi-chain environment in enabling complex, secure, and efficient data sharing and processing between various stakeholders within the automotive ecosystem.

5.5.1 Interaction Flow Implementation

The sequence diagram in Figure 5.5.2 details the steps taken after a vehicle accident, showing data communication across the multi-chain architecture.

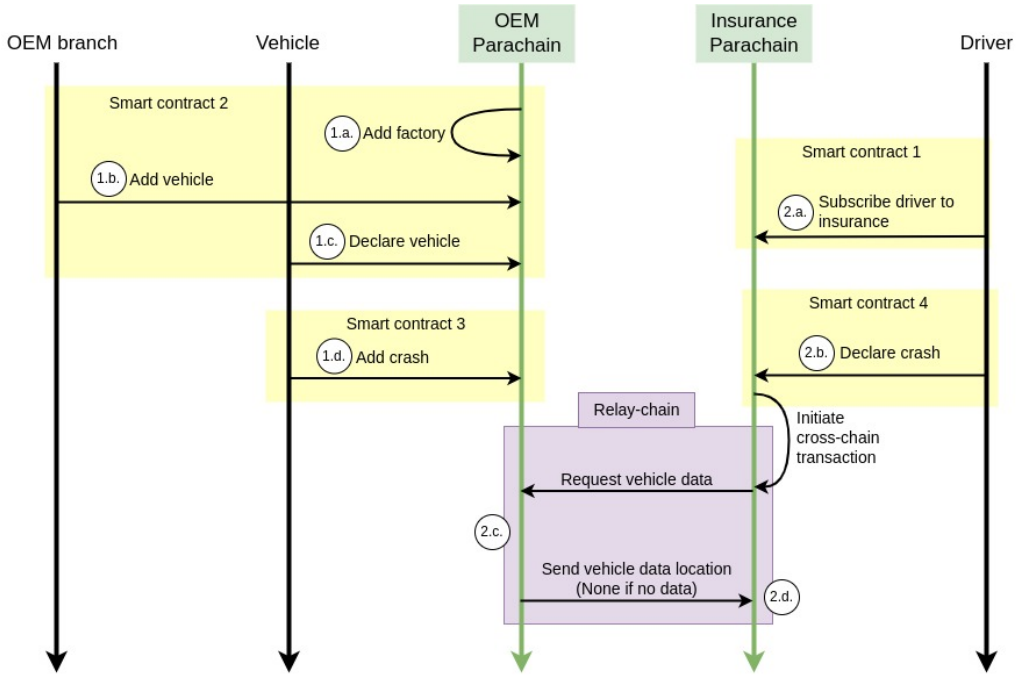


Figure 5.5.2: Interaction flow for the one-sided vehicle accident use case within the multi-chain environment.

The process begins with the vehicle's interaction with the OEM Parachain, where several actions take place:

- **1.a** The vehicle's manufacturing details are added to the OEM Parachain through a smart contract upon production.
- **1.b** The vehicle itself is registered on the OEM Parachain, establishing its identity within the blockchain network.
- **1.c** In the event of an accident, the vehicle's status is updated to "declared", which allows the vehicle to trigger the accident reporting mechanism.
- **1.d** A crash is recorded on the OEM Parachain, through another smart contract.

On the other side, the driver interacts with the Insurance Parachain to ensure proper insurance coverage and accident claim processing:

- **2.a** The driver subscribes to an insurance policy, facilitated by a smart contract on the Insurance Parachain.

2.b Post-accident, the driver declares a crash on the Insurance Parachain, which initiates the insurance claim process.

The relay-chain serves as the backbone of the ecosystem, providing governance and enabling cross-chain transactions:

2.c Upon a accident declaration, a request for vehicle data is sent from the Insurance Parachain to the OEM Parachain via the relay-chain.

2.d The OEM Parachain responds by sending the location (IPFS content identifier CID) of the vehicle data, or a notification if no data is available, back to the Insurance Parachain through the relay-chain, thus completing the cross-chain transaction.

This flow outlines how different parachains operate within the multi-chain setup, demonstrating the system's capability for secure and direct communication after a vehicle accident.

5.5.2 Creation and Deployment of Parachains

The creation of parachains for the automotive use case involves the Substrate framework, which allows for modular development through the use of pallets—reusable components that encapsulate specific blockchain functionality. For this project, a number of custom pallets have been created to address the unique requirements of the automotive industry. Tests in this next sections are using the same cloud infrastructure and hardware resources as previous chapter 3.

5.5.2.1 Development of New Pallet Modules

The development of the OEM and Insurance parachains were based on 'polkadot-v0.9.24', a version of the Polkadot parachain template[2].

OEM Chain (ParaID=2000): The OEM Chain is designed to manage vehicle-related data. The `sim_renault` and `sim_renault_accident` pallets are central to this parachain, handling vehicle manufacturing details, registration, and accident reporting.

Insurance Chain (ParaID=3000): The Insurance Chain focuses on insurance-related processes. Pallets like `sim_insurance` and `sim_insurance_accident` manage insurance policies, claims, and the verification of accident data pertinent to insurance proceedings.

Offchain Worker Pallet: The `ocw_ipfs_file_status` is an offchain worker pallet, which enables the parachains to perform certain tasks outside the realm of the consensus mechanism. This pallet is specifically responsible for interfacing with IPFS, facilitating the storage status of accident data in a decentralized manner.

These pallets are part of the blockchain node's `runtime` and form the backbone of the parachain's business logic. Pallets in the Substrate framework function similarly to smart contracts, providing specific business logic and operations for blockchain applications.

Deployment Workflow: Figure 5.5.3 outlines the deployment process customized for the multi-chain environment, where the orchestration sequence is a fundamental key to ensuring the functionality of the blockchain network.

The multi-chain setup starts differently than a single chain, with the first step being to launch the relay-chain, which has six validator nodes. This step is necessary to set up the governance

[2] github.com/substrate-developer-hub/substrate-parachain-template

and security that the parachains will depend on. After that, the parachains, each with a specific ParaID, are started and connected to the relay-chain with a transaction that registers them. This registration is crucial because it signals to the relay-chain to start validating and finalizing the parachain's blocks.
A period of observation is crucial after this step to confirm that the parachains are not only producing blocks but that these blocks are being validated and finalized by the relay-chain. It ensures that the parachains are operating smoothly within the ecosystem and are fully synchronized with the relay-chain's consensus mechanism.

The genesis state of each parachain is initialized with their respective genesis configuration files, such as `cloud-renault-chain-raw.json` for the OEM Chain and `cloud-insurance-chain-raw.json` for the Insurance Chain. These files contain the initial state and configuration for each parachain. When an upgrade of the chain is required, files such as `cloud-para-2000-genesis` and `cloud-para-2000-wasm` are used, where the latter contains the compiled WebAssembly (WASM) bytecode that enables runtime upgrades on the live network without the need for a hard fork.

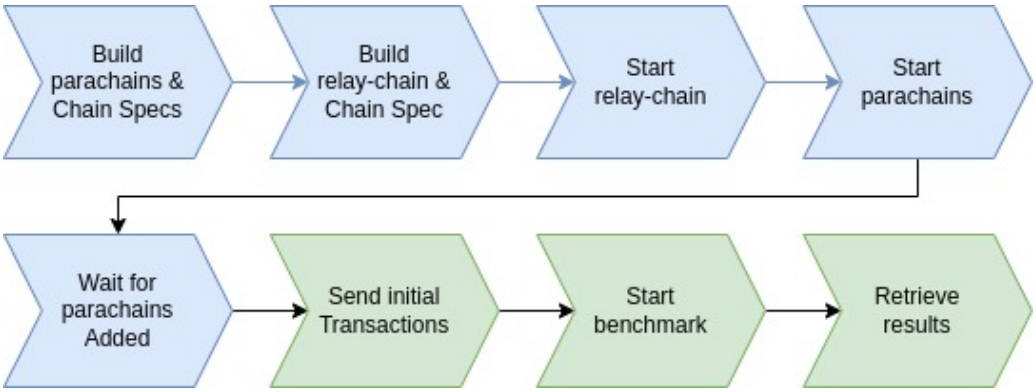


Figure 5.5.3: The deployment process for OEM and Insurance parachains in the multi-chain environment.

After completing the deployment, the next step is to assess the multi-chain network's performance. The following section will cover the benchmarking processes used to test network throughput, latency, and stability. This evaluation is essential to ensure the parachains are fit for real-world situations.

5.5.3 Benchmarking Parachains Performance

Benchmarking is a critical step in assessing the performance capabilities of parachains. It provides valuable data on the transaction throughput and stability of the network under various conditions. The benchmarks conducted for this project focused on measuring the average transactions per second (TPS) for both the OEM and Insurance parachains.

Tests were conducted using 1, 2, and 3 collators to examine the scalability and performance under different network loads. All tests use 6 relay-chain validator nodes.

The data, as depicted in Figure 5.5.4, shows three distinct graphs corresponding to the number of used collators for each parachain. It is evident from the results that an increase in the number of collator nodes corresponds to a rise in the average output TPS due to overall more processing power and more mempool capacity. Notably, the OEM parachain consistently outperforms the

Insurance parachain in terms of output TPS. It is critical to mention that the OEM and Insurance chains execute different substrate pallets, which inherently influences their performance metrics.

An important observation is that, regardless of whether 50 or 2000 TPS are fed into the parachains, the maximum output TPS reached with three collators is 43 TPS for the OEM and 35 TPS for the insurance chain. This indicates that above a certain input threshold, the processing capacity of the parachains reaches a ceiling.

The increase in processed transactions with the addition of more collators will be further explored by analyzing block time metrics in subsequent sections. Additionally, the variability in performance can also partially be explained by block time variations, as indicated by the standard deviation, it suggests that the output TPS could peak at nearly 60 TPS for the OEM and almost 50 TPS for the Insurance chain, under optimal conditions. With these performances, we can express the maximum accidents of the use case. The base requirements expressed previously (Equation 3.2.2, p. 87) are 34.25 transactions per second. This means that, the two parachain's processing rate is sufficient to handle the estimated transaction load from critical events, system updates, and regular maintenance records as outlined in the SIM use case.

The insights gained from these benchmarks are valuable in optimizing the parachains for improved throughput and reliability, facilitating their deployment for potential production environments.

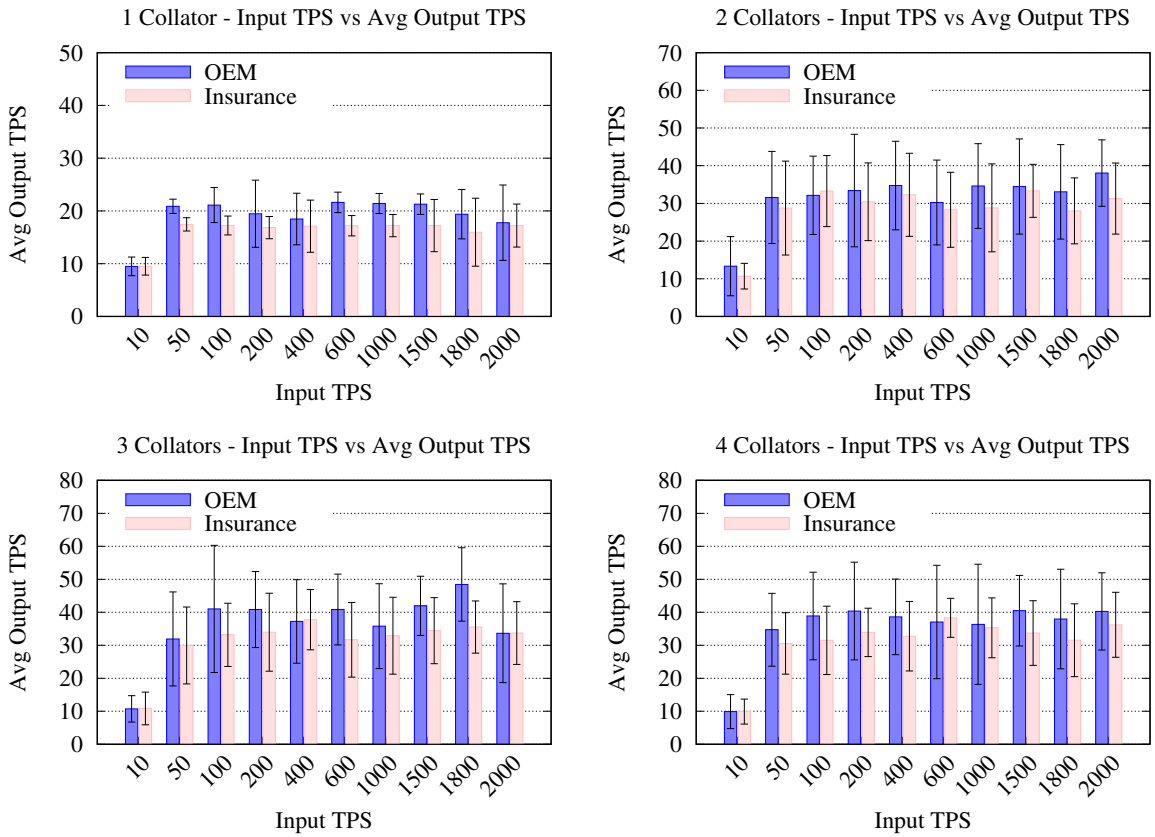


Figure 5.5.4: Benchmarking results showing Input TPS versus Average Output TPS for the OEM and Insurance parachains with varying numbers of collators

5.5.4 Analysis of Block Time During Benchmarks

The benchmark results depicted in Figure 5.5.5 provide insight into the block time performance as it relates to the number of collators within the network. An understanding of the parachain's chain specification and node runtime configuration is essential for interpreting these results.

In the tested parachains, the chain specs define that there are 4 block authors designated for block production. This setting is critical when considering the AURA consensus mechanism, which inherently requires a waiting period if a designated block producer is not available. When operating with fewer than four collators, as in the one and two collator configurations, the benchmarks reveal longer block times, which can be partly attributed to this waiting period within the consensus algorithm.

The 'SLOT_DURATION' parameter, set to 12 seconds, represents the target block time that the network aims to achieve under optimal conditions. However, the actual block time can vary due to the consensus mechanism's waiting periods, especially when the number of active collators is less than the number of designated block authors.

For a single collator, the average block time remains around 50 seconds, indicating that the network is often in a wait state due to the lack of multiple block producers. With two collators, the average block time improves to 20-25 seconds but still exceeds the 'SLOT_DURATION' target, likely due to occasional waiting periods when the two other expected collators are not available to produce a block.

With three collators, the block time decreases further, varying between 15 to 19 seconds, closer to the 'SLOT_DURATION' target. This reduction illustrates the direct impact of increased collator numbers on decreasing block times, as the likelihood of having a collator ready to produce a block increases, thus reducing wait times and improving overall efficiency.
With four collators we obtain a much closer blocktime of 12 to 16 seconds, showing that 'SLOT_-DURATION' target is reached. Blocktime can be more than 12 seconds due to additional delays that can apply when sending important input TPS, such as network latency, network load, or transaction processing latencies.

While the benchmarks show a closer alignment to the 'SLOT_DURATION' target with three collators, another significant benefit of increasing the number of collators is the improved absorption of transaction load across the network. Put differently, this improvement is partially due to the larger collective mempool, which allows more transactions to be held and processed by the network.
With more nodes available to produce blocks, the network can handle a higher volume of transactions without reaching capacity, which would typically lead to transaction rejection. Consequently, as the number of collators increases, the average output TPS also rises (See Figure 5.5.4). The blocktime and TPS measurements are experimental, and more testing should be considered when realizing an industrial real-word scenario. Different node configuration (e.g. more collators or more parachain nodes) are essential to get an accurate estimation for real-world scenarios. This correlation between the number of collators and network throughput – however not linear – is a key consideration for scaling the network, as it directly impacts the network's ability to handle peak loads and maintain high availability and reliability for users.

In practice, this means that a well-dimensioned network of collators is essential not only for achieving desired block times but also for ensuring the network can accommodate the transaction demands of a real-world automotive environment. Therefore, strategic scaling of collator numbers is crucial for the long-term efficiency and success of the parachains in

production.

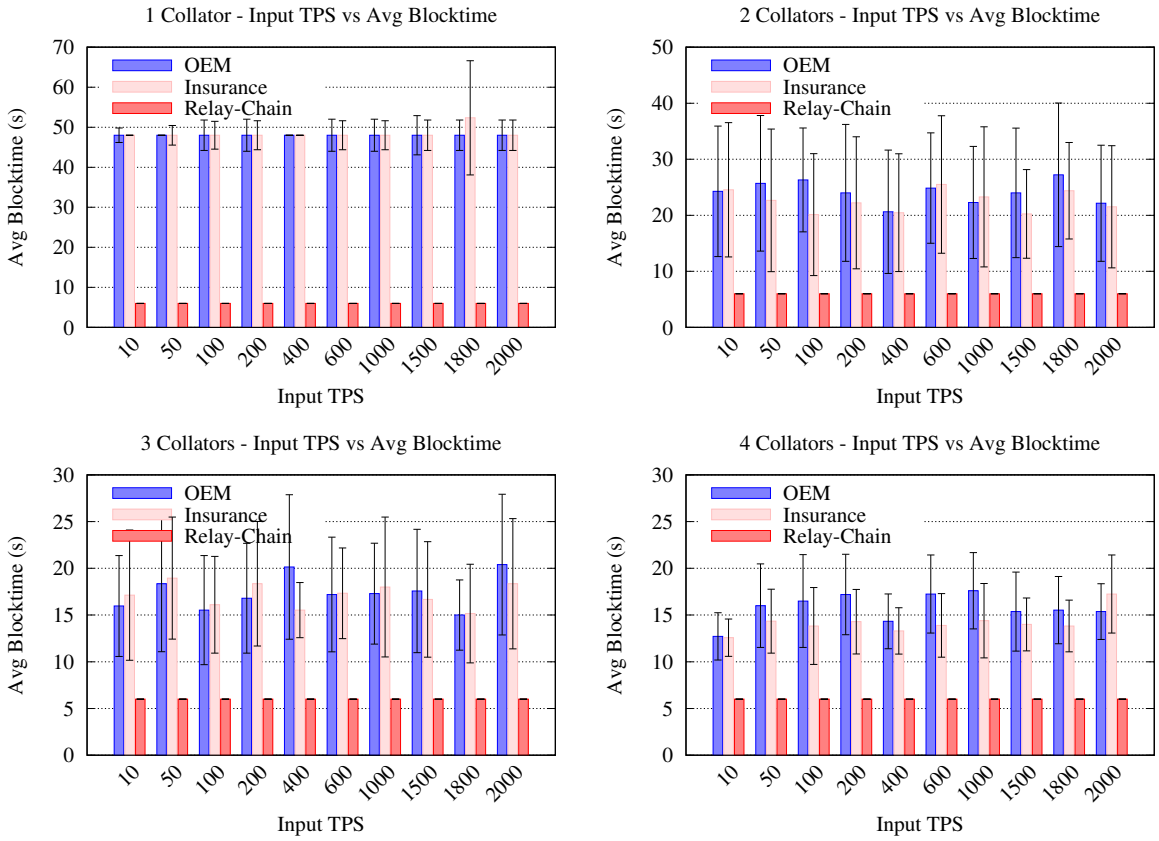


Figure 5.5.5: Benchmarking results for block time against varying input transactions per second (TPS) for both OEM and Insurance parachains, across different collator configurations.

5.5.5 Latency in Cross-Communication Transactions

Understanding the latency in cross-communication transactions is important for evaluating the performance of a Polkadot-based multi-chain environment, particularly in applications like automotive systems involving OEM and Insurance parachains. We use a 4 collator nodes configuration per parachains and 6 relay-chain validator nodes.

Analysis of Transaction Latency: The transaction process encompasses several stages, each with measurable latencies. These stages provide insights into the network's operational efficiency. Depicted in Figure 5.5.2, the latency is a metric retrieved by starting with interaction **2.a.** to **2.d.**. The Code 5.1 shows the log of timestamp-messages, which help us for the following analysis by doing timestamp differences.

- *Transaction Initiation and Readiness:* After the transaction was created, it is broadcast and reach the 'Ready' status. This is the start time from where a driver start to wait for processing an accident declaration towards the insurance.
- *Inter-Parachain Communication:*
  - Next, the duration from the transaction being processed by the Insurance chain is 5.26 s, this stores the transaction and start requesting vehicle data.
  - Then, the time for the OEM parachain to receive the vehicle data request and send a reply is 17.26 s.
  - Finally, the Insurance parachain received the vehicle data after 38.10 s.
- *About Transaction Finalization:* The driver accident report transaction was included in a block and finalized 29.26 seconds later in the Insurance chain. These timings highlight the network's latency of 29.26 s, a relatively slow finalization time for the transaction. However, we see that even if the transaction isn't finalized, the cross-chain message still operated, and by the end of transaction finalization, the Insurance chain already received the vehicle data.

The measured latencies offer an understanding of the timeframes involved in different transaction stages within the Polkadot network. This information is useful for assessing the network's suitability for an application that require timely data exchanges, such as those in our automotive use case.

An important note to keep in mind is that XCM is asynchronous, thus "waiting 29 seconds" does not block the parachains and the relay-chain operations and block production. It is the delay for inter-parachain message exchange, and in our case the delay for the insurance to receive the data after the accident declaration. Compared to current systems, this process automation is beneficial for all parties concerned.

Code 5.1: Console log latency test of cross-transaction message. Each message is prefixed with a UNIX timestamp spaced with a tabulation.

```
1 1708431935528 Connected to OEM chain.
2 1708431936392 Connected to Insurance chain.
3 1708431936406 Transaction created.
4 1708431936817 Transaction status: Ready
5 1708431942078 Insurance: palletSimInsuranceAccident.AccidentStored
6 1708431942078 Insurance: palletSimInsuranceAccident.RequestData
7 1708431954080 OEM: palletSimRenaultAccident.ReceiveVehicleDataRequest
```

```
1708431954080 OEM: palletSimRenaultAccident.SendVehicleDataRequestReply
1708431954344 Transaction included at block hash 0
    x14e170593014f5cb090b7f6ffb80ba5e268151eb0cf57766fefc11356081694b
1708431974921 Insurance: palletSimInsuranceAccident.ReceiveData
1708431983610 Transaction finalized at block hash 0
    x14e170593014f5cb090b7f6ffb80ba5e268151eb0cf57766fefc11356081694b
```

5.6 Conclusion of Multi-Chain Implementation

In the automotive industry, using a multi-chain blockchain system, especially for integrating IoT, leads to a system where different blockchain networks, each with their own specific purpose, **work together to improve the overall function and security**. As shown in Figure 5.5.1, in a Polkadot-based multi-chain configuration, a car accident scenario is presented. In this scenario, the OEM and an Insurance Chain is built on the Substrate framework. The OEM Chain is key for capturing initial data, including vehicle incident reports. Post-accident, the OEM chain becomes "*a repository*" of the data hash transmitted by vehicles, safeguarding this information in a decentralized manner through IPFS.

At the same time, the relay-chain, a central part of the Polkadot framework, controls the ecosystem. It does not handle the data itself, but rather manages the operation of various parachains. Its crucial role includes **enabling transactions between chains and supporting the security** throughout the ecosystem. Additionally, the Insurance Chain becomes only active when declaring accidents, and than triggering the vehicle data request procedure. These requests are sent through the relay-chain's messaging system, ensuring safe and efficient sharing of information necessary for processing insurance claims and accident reports.

This multi-chain setup shows how different blockchain technologies can work together effectively, each adding value to meet the complex needs of digital services in the automotive industry. This approach not only shows the importance of a well-designed multi-chain system but also the potential of such a structure in enabling sophisticated, secure, and efficient data exchange and processing among various parties.

6 Conclusion and Perspectives

6.1 Thesis Conclusion

In this thesis, we have conducted an in-depth exploration of the integration of blockchain and Internet of Things (BIoT), specifically examining its application in the automotive industry through the *accident use case*. Our research journey has unveiled numerous complexities and significant potential. Figure 6.1.1 outlines the evolution of this thesis.

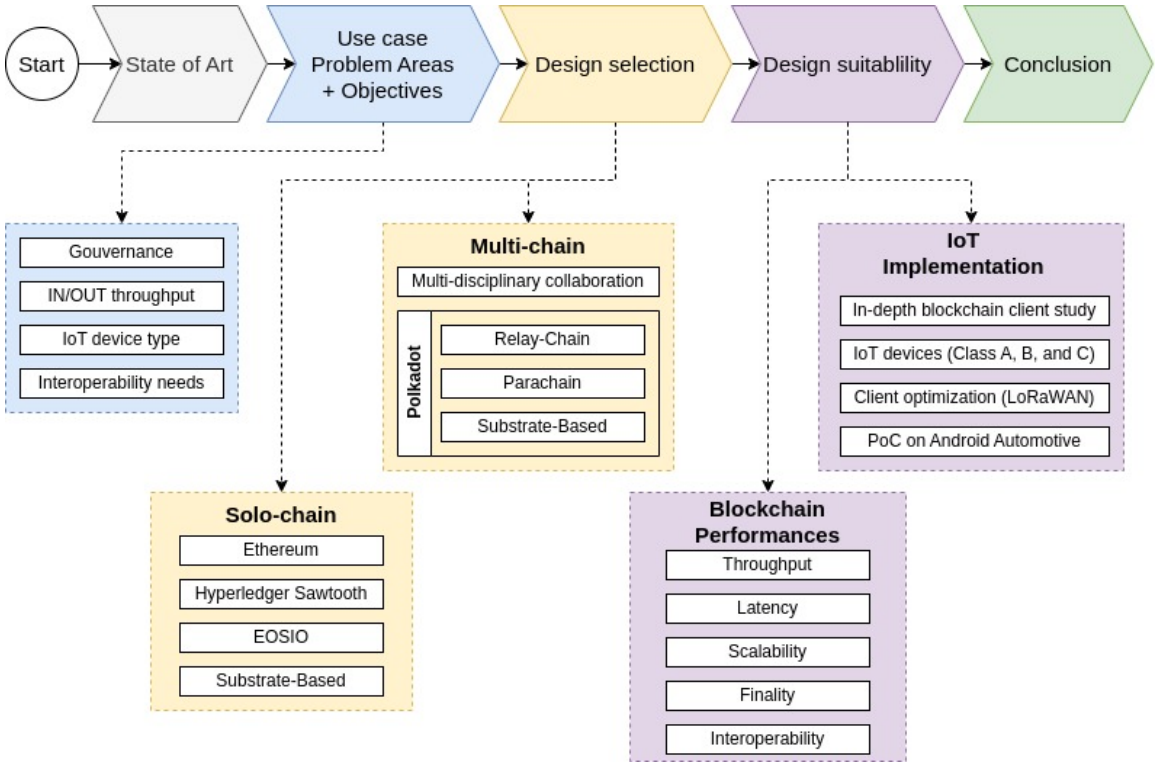


Figure 6.1.1: Outline of the thesis contribution and challenges

We have observed that the convergence of these technologies has the potential to reshape the landscape of the IoT realm, and consequently, domains such as the automotive industry. Globally, the key accomplishment of our research has been the successful demonstration of integrating blockchain technology within various IoT devices, improving the security and reliability essential for many practical IoT-based use cases.

Firstly, the state of the art highlights the importance of exploring a collision of two new technologies to improve existing IoT solutions. Blockchain is a relatively young type of ”database” that has been under design since the 2000s and has begun to operate by creating a ledger of

financial transactions since 2009. Less than 15 years later, the term blockchain has encapsulated many underlying sub-categories of distributed ledger usage. Smart contracts unlocked the possibility to run any application on top of the distributed ledger, which unlocked the possibility to create many projects with a common goal: the creation of trust, decentralization, automation, and of course data transparency. With the appearance of smart contracts, the IoT-blockchain combination started to become an obvious choice.
The state of the art is setting key findings in the BIoT domain by indicating that certain specific BIoT use cases are feasible **but require significantly more research for adoption** by mainstream companies or individuals. Utilizing blockchain alone, without additional layers or scalability designs, is infeasible due to inadequate throughput performance. However, with the advent of layer 2 solutions, scalability enhancements, and specialized layers, a new hope has emerged for realizing BIoT applications. Furthermore, with the recent appearance of blockchain-of-blockchain technology, a research gap is observed in the combination of multi-chain and IoT.

Usage of an automotive use case: The vehicle accident scenario established in early 2018 aimed to use blockchain for creating a new paradigm for information exchange. Based on the state of art, the thesis presented here lays out a new approach to this objective by advancing the paradigm one step further towards novel multi-chain solutions.

The **exploration of different blockchain platforms** (in a solo-chain configuration) provided insights into their relevance and validity for an IoT context, based on performance properties like transaction processing speed, scalability, and network throughput. These metrics offer valuable guidance for the application of blockchain in automotive and other IoT contexts. The results show and confirm that blockchain is usable in an IoT context, but lead to a number of challenges that could be resolved in a multi-chain configuration.

Consequently, one of the cornerstones of the research is the exploration of a multi-chain ecosystem within a BIoT context. By leveraging Polkadot's network paradigm and the Substrate Framework, we have established an innovative approach to achieving a unified multi-blockchain environment specifically designed for the automotive sector. This new setup enables more seamless and interoperable interactions between blockchain platforms, and potentially IoT devices operating on various networks.
Moreover, we have also studied the performances of the underlying interconnected blockchains of the ecosystem. Through our specific implementation, we have demonstrated that the configuration of the network is crucial in providing sufficient performance. The intercommunication between two blockchains was also tested to achieve communication without the need for implementing specific bridging layers.

Another contribution is the interdisciplinary research proposal and development of multi-chain technology, following a study conducted by Marta Ballatore on digital servitization in organizational and interorganizational environments. By combining the domains of management and computer science, we were able to address the issue of data-sharing in companies in a two phase approach. The "design science approach" method in management allowed Marta to extract managerial requirements, which were then translated (in collaboration with us) into blockchain specifications. The initial phase involved utilizing a solo-chain configuration, whereas in the second phase, after further interviews with IT experts (in the relevant sector), it was determined that a multi-chain configuration meets better the use case organizational and inter-organizational constraints.
This collaborative effort enhanced the investigation of multi-chain technology within our context, particularly at the IoT data sharing stage following an accident. Securing and managing IoT

data is crucial for BIoT, but for utilizing this data with other services (i.e. blockchain-based services), the system needs to seamlessly exchange information. This is achieved through cross-transactions utilizing the ”XCM” cross-transaction format of multi-chain in our implementation.

The next key contribution in this research is the **development and practical implementation of blockchain clients on IoT devices**. This work highlighted the complexities involved in a blockchain client program from a constrained device perspective due to varied cryptographic algorithms, network, and RPC protocols required by different blockchain platforms. Multiple integration of blockchain client application are made and tested to extract performance metrics (blockchain and IoT hardware related). This exploration led us to improve some of the BIoT demonstrations we created.

With the collaboration of fellow researchers we determined one of the blockchain client component that limited BIoT integration for very constrained devices. The work of Roland Kromes made an IoT architecture adaptation specifically for blockchain by creating functional cryptographic IP models and simulated their improvement. Eliptic curve cryptography and hashing function optimization proved to have a great impact in blockchain client application. The global idea was to improve the IoT hardware, but in this thesis we modified the client application entirely as an attempt to make BIoT a reality.

The proposed new design enabling constrained IoT devices to interact with blockchains is outlined as follows. The Light Transaction Protocol (LTP) abstracts the transaction format to avoid blockchain state dependency in the IoT. A combination of smart contracts and the LTP format enables the protocol to work. Its implementation has been made in collaboration with TU Delft University. We demonstrated the feasibility of transmitting IoT data over a limited communication medium (LoRaWAN) to two blockchain platforms (Substrate-based or Hyperledger Fabric). We utilize the continuous power connection of LoRaWAN gateways to implement a proxy module API that encapsulates LTP transactions into valid blockchain transactions, which are then sent to a blockchain node. The potential of **LTP lies in its ability to communicate with any blockchain platform with minimum hardware requirements** (provided it supports smart contracts to integrate LTP logic) and to transmit transactions using highly constrained IoT devices.

6.2 Perspectives

Perspectives in this thesis primarily concern four contributions: the LTP protocol, multi-chain usage with an IoT use case, applications implementations, and data improvement.

The Light Transaction Protocol proposal only defines the data format and the smart contract requirements for using this data. Security analysis needs improvement as it is not addressed in the research on LTP. A primary issue for discussion in implementation is the assumption that IoT devices are not malicious regarding DDoS[1] attacks. LoRaWAN gateways can receive malicious traffic flows by replaying LoRa communication packets due to the stateless design (no presence of transaction nonce). However, in practice, the LoRaWAN protocol itself includes a packet counter that can mitigate this security issue.

Additionally, LTP implementation requires keeping private keys in both the IoT device and the LoRaWAN gateway. In future work, IoT devices should always utilize secure cryptographic IP modules to store and sign messages. The main concern lies with the gateway, which also contains private keys to sign the valid transaction containing the LTP packet. This is likely the

[1] A Distributed Denial of Service (DDoS) attack is a malicious attempt to disrupt the normal traffic of a targeted server, service, or network by overwhelming it with a flood of internet traffic originating from multiple sources.

weakest point of failure in the current design, as these private keys are essential for communication at the blockchain level. In public blockchains, these private keys are required to own value (coin or token) to pay for network fees. In consortium blockchains, they do not necessarily own financial value but still ”own” control access to the network to send transactions. Future work should likely involve modifying the proxy module API to store or handle private keys differently (e.g., using cryptographic IP or establishing a secure handshake with a service to facilitate transmission).

Overall, despite some weak spots, the protocol addresses a significant issue in enabling BIoT for small devices with limited memory and computational capabilities. The main perspective that lies in LTP is to **facilitate access to any blockchain platforms for various types of devices**. An interesting future work would involve implementing LTP with different communication mediums and comparing the resulting solutions. It is conceivable that certain communication protocols will pose significant challenges due to their lack of intermediaries. The gateway aids LTP functionality by allowing the addition of a proxy module API, unlike Wi-Fi. Conversely, if an IoT device has sufficient computational power (e.g. with Wi-Fi), it is likely capable of directly implementing a blockchain client owing to its greater memory and computing resources.

The next perspective discusses multi-chain implementation. Substrate-based blockchains are built using the Substrate Framework, allowing us to arbitrarily code many aspects of the blockchain configuration. The demonstration utilizes a template that may not be optimized for our specific use case. We have created modules (i.e., smart contracts) directly included in the core logic of the blockchain, but some configuration options have not been sufficiently explored to obtain the best results. However, this does not negate the fact that our proof of concept has shown that by utilizing this technology, we can **create a much more friendly ecosystem for organizations to implement blockchain with sufficient modularity while still enabling native cross-communication with other networks**.

As more and more businesses get interested in blockchain for its intrinsic benefits, the major drawback is the existing blockchain network heterogeneity. The key perspective in multi-chain lies in the ecosystem of blockchains which allows to create interconnected services. As we started to suggest directly in the introduction, a multi-chain paradigm could bring existing organizations together, leading to the emergence of new data exchange models, and therefore also business models.

The **proof of concept utilizing Android Automotive** would benefit from direct testing within the vehicle’s In-Vehicle Infotainment (IVI) system. This necessitates improving the application by converting it from an Android application to an Android service, thereby enabling it to run in the background and continuously collect information until an accident is detected. Such an improvement would not only validate the application’s functionality in a real-world setting but also enhance its utility by making it a more integral part of the vehicle’s operating system.

Furthermore, the Android application and other specific IoT client programs do not currently address privacy or confidentiality concerns. As the integration of IoT and blockchain technologies continues to evolve, it is imperative to incorporate privacy and data protection mechanisms. This is critical for any applications handling personal data, where sensitive personal (and vehicle) data may be collected and shared across multiple platforms and stakeholders. Future work should, therefore, include a thorough analysis of privacy requirements and the development of secure data handling protocols that comply with relevant regulations and standards, such as the General Data Protection Regulation (GDPR).

This thesis contributes to the ongoing discussion on integrating blockchain and IoT technolo-

gies, with a focus on the automotive sector. While it demonstrates the possibility of using these technologies to improve vehicle safety and enable new ways of data sharing among organizations, there's an acknowledgment that this area is still developing. Future research and practical applications will be essential in addressing the challenges that arise and in further understanding the capabilities and limitations of combining blockchain and IoT technologies.

7 Bibliography

[1] A. Gaur, B. Scotney, G. Parr, and S. McClean, "Smart City Architecture and its Applications Based on IoT," *Procedia Computer Science*, vol. 52, pp. 1089–1094, 2015. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015).

[2] H. Treiblmaier, A. Rejeb, and A. Strebinger, "Blockchain as a Driver for Smart City Development: Application Fields and a Comprehensive Research Agenda," *Smart Cities*, vol. 3, no. 3, pp. 853–872, 2020.

[3] S. Li, L. D. Xu, and S. Zhao, "5G Internet of Things: A survey," *Journal of Industrial Information Integration*, vol. 10, pp. 1–9, 2018.

[4] M. Khuntia, D. Singh, and S. Sahoo, "Impact of Internet of Things (IoT) on 5G," in *Intelligent and Cloud Computing*, (Singapore), pp. 125–136, Springer Singapore, 2021.

[5] J. Laufs, H. Borrion, and B. Bradford, "Security and the smart city: A systematic review," *Sustainable Cities and Society*, vol. 55, p. 102023, 2020.

[6] C. N. Samue, G. Severine, B. David, F. Verdier, and G.-O. Patricia, "Automotive Data Certification Problem: A View on Effective Blockchain Architectural Solutions," in *2020 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 0167–0173, 2020.

[7] R. Jabbar, N. Fetais, M. Kharbeche, M. Krichen, K. Barkaoui, and M. Shinoy, "Blockchain for the Internet of Vehicles: How to Use Blockchain to Secure Vehicle-to-Everything (V2X) Communication and Payment?," *IEEE Sensors Journal*, vol. 21, no. 14, pp. 15807–15823, 2021.

[8] M. Cebe, E. Erdin, K. Akkaya, H. Aksu, and S. Uluagac, "Block4Forensic: An Integrated Lightweight Blockchain Framework for Forensics Applications of Connected Vehicles," *IEEE Communications Magazine*, vol. 56, no. 10, 2018.

[9] K. L. Brousmiche, T. Heno, C. Poulain, A. Dalmieres, and E. Ben Hamida, "Digitizing, Securing and Sharing Vehicles Life-cycle over a Consortium Blockchain: Lessons Learned," in *9th IFIP Conference on NTMS*, 2018.

[10] "Software-Defined Vehicles – A Forthcoming Industrial Evolution." https://www2.deloitte.com/cn/en/pages/consumer-business/articles/software-defined-cars-industrial-revolution-on-the-arrow.html. Accessed: 2023-03-30.

[11] "Outlook on the automotive software and electronics market through 2030." https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/mapping-the-automotive-software-and-electronics-landscape-through-2030. Accessed: 2023-03-30.

[12] "Growing opportunities in the Internet of Things." https://www.mckinsey.com/industries/private-equity-and-principal-investors/our-insights/growing-opportunities-in-the-internet-of-things. Accessed: 2023-03-30.

[13] "By 2025, Internet of things applications could have $11 trillion impact." https://www.mckinsey.com/mgi/overview/in-the-news/by-2025-internet-of-things-applications-could-have-11-trillion-impact. Accessed: 2023-03-30.

[14] R. Jabbar, M. Kharbeche, K. Al-Khalifa, M. Krichen, and K. Barkaoui, "Blockchain for the Internet of Vehicles: A Decentralized IoT Solution for Vehicles Communication Using Ethereum," *Sensors*, vol. 20, no. 14, 2020.

[15] "About Smart IoT for Mobility (SIM) project." https://univ-cotedazur.eu/sim/about. Accessed: 2023-12-29.

[16] F. Verdier, P. de Filippi, F. Mallet, P. Collet, L. Arena, A. Attour, M. Ballatore, M. Chessa, A. Festré, P. Guitton-Ouhamou, R. Bernhard, and B. Miramond, "Smart IoT for Mobility: Automating of Mobility Value Chain through the Adoption of Smart Contracts within IoT Platforms." 17th Driving Simulation & Virtual Reality Conference (DSC 2018), Sept. 2018. Poster.

[17] M. Ballatore, L. Gerrits, R. Kromes, L. Arena, and F. Verdier, "Toward the Conception of a Multichain to Meet Users' Future Needs: A Design Science Research Approach to Digital Servitization in the Automotive Industry," *IEEE Transactions on Engineering Management*, pp. 1–15, 2023.

[18] L. Arena, A. Attour, and M. Ballatore, "Bricoler une stratégie d'affaires numérique via les smart contrats. le cas du véhicule connecté de renault," pp. 1–12.

[19] R. Kromes, L. Gerrits, and F. Verdier, "Adaptation of an embedded architecture to run Hyperledger Sawtooth Application," in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 0409–0415, 2019.

[20] R. Kumar, F. Khan, S. Kadry, and S. Rho, "A Survey on blockchain for industrial Internet of Things," *Alexandria Engineering Journal*, vol. 61, no. 8, pp. 6001–6022, 2022.

[21] G. Moore, "The future of integrated electronics," *Understanding Moore's Law: Four Decades of Innovation*, pp. 37–55, 1964.

[22] IoT Analytics GmbH, "State of IoT 2022: Number of connected IoT devices growing 18% to 14.4 billion globally." https://iot-analytics.com/number-connected-iot-devices. [Accessed 08-May-2023].

[23] Statista, "IoT connected devices worldwide 2019-2030." https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide. [Accessed 08-May-2023].

[24] International Energy Agency 4E, "Total Energy Model for Connected Devices." https://www.iea-4e.org/wp-content/uploads/2020/11/A2b_-_EDNA_TEM_Report_V1.0.pdf. [Accessed 08-May-2023].

[25] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-Time Video Analytics: The Killer App for Edge Computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[26] P. Fraga-Lamas, M. Celaya-Echarri, P. Lopez-Iturri, L. Castedo, L. Azpilicueta, E. Aguirre, M. Suárez-Albela, F. Falcone, and T. M. Fernández-Caramés, "Design and Experimental Validation of a LoRaWAN Fog Computing Based Architecture for IoT Enabled Smart Campus Applications," *Sensors*, vol. 19, p. 3287, Jan. 2019.

[27] N. V. R. Kumar, C. Bhuvana, and S. Anushya, "Comparison of ZigBee and Bluetooth wireless technologies-survey," in *2017 International Conference on Information Communication and Embedded Systems (ICICES)*, pp. 1–4, 2017.

[28] M. Jouhari, N. Saeed, M.-S. Alouini, and E. M. Amhoud, "A Survey on Scalable LoRaWAN for Massive IoT: Recent Advances, Potentials, and Challenges," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 3, pp. 1841–1876, 2023.

[29] H. Jiang, C. Cai, X. Ma, Y. Yang, and J. Liu, "Smart Home Based on WiFi Sensing: A Survey," *IEEE Access*, vol. 6, pp. 13317–13325, 2018.

[30] Sigfox™, "Sigfox overview." https://www.st.com/content/ccc/resource/training/technical/product_training/group1/1f/ec/ff/ce/c1/c1/40/3e/STM32WL-Communication-Sigfox_SIGFOX/files/STM32WL-Communication-Sigfox_SIGFOX.pdf/_jcr_content/translations/en.STM32WL-Communication-Sigfox_SIGFOX.pdf. [Accessed 11-May-2023].

[31] Thread Group, "Thread Network Fundamentals White Paper." https://www.threadgroup.org/Portals/0/documents/support/Thread%20Network%20Fundamentals_v3.pdf. [Accessed 11-May-2023].

[32] R. Kenaza, A. Khemane, H. Bendjenna, A. Meraoumia, and L. Laimeche, "Internet of Things (IoT): Architecture, Applications, and Security Challenges," in *2022 4th International Conference on Pattern Analysis and Intelligent Systems (PAIS)*, (Oum El Bouaghi, Algeria), pp. 1–5, IEEE, Oct 2022.

[33] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125–1142, 2017.

[34] P. Sivagami, P. Illavarason, R. Harikrishnan, and G. V. S. R. Reddy, "IoT Ecosystem- A survey on Classification of IoT," in *2020 International Conference on Advanced Scientific Innovation in Science, Engineering and Technology (ICASISET)*, EAI, 1 2021.

[35] C. Bormann, M. Ersue, and A. Keränen, "Terminology for Constrained-Node Networks." RFC 7228, May 2014.

[36] C. Sabri, L. Kriaa, and S. L. Azzouz, "Comparison of IoT Constrained Devices Operating Systems: A Survey," in *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, pp. 369–375, 2017.

[37] B. Merabtane and N. Benabadji, "Optimized design of an extreme low power datalogger for photovoltaic panels," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 13, pp. 2734–2742, June 2023.

[38] C. Sisavath and L. Yu, "Design and implementation of security system for smart home based on IOT technology," *Procedia Computer Science*, vol. 183, pp. 4–13, 2021. Proceedings of the 10th International Conference of Information and Communication Technology.

[39] M. Anandhalli and V. P. Baligar, "A novel approach in real-time vehicle detection and tracking using Raspberry Pi," *Alexandria Engineering Journal*, vol. 57, no. 3, pp. 1597–1607, 2018.

[40] M. M. Al-Kofahi, M. Y. Al-Shorman, and O. M. Al-Kofahi, "Toward energy efficient microcontrollers and Internet-of-Things systems," *Computers & Electrical Engineering*, vol. 79, p. 106457, Oct. 2019.

[41] W. A. Jabbar, T. K. Kian, R. M. Ramli, S. N. Zubir, N. S. M. Zamrizaman, M. Balfaqih, V. Shepelev, and S. Alharbi, "Design and Fabrication of Smart Home With Internet of Things Enabled Automation System," *IEEE Access*, vol. 7, pp. 144059–144074, 2019.

[42] O. O. Akintade, T. K. Yesufu, and L. O. Kehinde, "Development of Power Consumption Models for ESP8266-Enabled Low-Cost IoT Monitoring Nodes," *Advances in Internet of Things*, vol. 9, pp. 1–14, Jan. 2019.

[43] P. Deane, *The first industrial revolution*. Cambridge University Press, 1979.

[44] J. Mokyr and R. H. Strotz, "The second industrial revolution, 1870-1914," *Storia dell'economia Mondiale*, vol. 21945, no. 1, 1998.

[45] A. Charlesworth, *The digital revolution*. Dorling Kindersley Ltd, 2009.

[46] M. Castells, "An introduction to the information age," *City*, vol. 2, no. 7, pp. 6–16, 1997.

[47] McKinsey, "IoT value set to accelerate through 2030: Where and how to capture it." https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/iot-value-set-to-accelerate-through-2030-where-and-how-to-capture-it. [Accessed 08-May-2023].

[48] T. Shenkoya, "Social change: A comparative analysis of the impact of the IoT in Japan, Germany and Australia," *Internet of Things*, vol. 11, p. 100250, 2020.

[49] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019.

[50] M. Merenda, C. Porcaro, and D. Iero, "Edge Machine Learning for AI-Enabled IoT Devices: A Review," *Sensors*, vol. 20, no. 9, 2020.

[51] P.-E. Novac, G. Boukli Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, "Quantization and Deployment of Deep Neural Networks on Microcontrollers," *Sensors*, vol. 21, no. 9, 2021.

[52] N. Abderrahmane, E. Lemaire, and B. Miramond, "Design Space Exploration of Hardware Spiking Neurons for Embedded Artificial Intelligence," *Neural Networks*, vol. 121, pp. 366–386, 2020.

[53] N. Schizas, A. Karras, C. Karras, and S. Sioutas, "TinyML for Ultra-Low Power AI and Large Scale IoT Deployments: A Systematic Review," *Future Internet*, vol. 14, no. 12, 2022.

[54] M. M. Dhanvijay and S. C. Patil, "Internet of Things: A survey of enabling technologies in healthcare and its applications," *Computer Networks*, vol. 153, pp. 113–131, 2019.

[55] U. Albalawi and S. Joshi, "Secure and trusted telemedicine in Internet of Things IoT," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pp. 30–34, 2018.

[56] INRIA, "TousAntiCovid." https://gitlab.inria.fr/stopcovid19. Accessed: 2023-08-29.

[57] A. Karmakar, N. Dey, T. Baral, M. Chowdhury, and M. Rehan, "Industrial Internet of Things: A Review," in *2019 International Conference on Opto-Electronics and Applied Optics (Optronix)*, pp. 1–6, 2019.

[58] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, Opportunities, and Directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[59] S. M. A. A. Abir, A. Anwar, J. Choi, and A. S. M. Kayes, "IoT-Enabled Smart Energy Grid: Applications and Challenges," *IEEE Access*, vol. 9, pp. 50961–50981, 2021.

[60] M. binti Mohamad Noor and W. H. Hassan, "Current research on Internet of Things (IoT) security: A survey," *Computer Networks*, vol. 148, pp. 283–294, 2019.

[61] K. Sgantzos and I. Grigg, "Artificial Intelligence Implementations on the Blockchain. Use Cases and Future Applications," *Future Internet*, vol. 11, no. 8, 2019.

[62] IBM, "IBM System/360 Dates and characteristics." https://web.archive.org/web/20230207025054mp_/https://www.ibm.com/ibm/history/exhibits/mainframe/mainframe_FS360B.html, 2023.

[63] T. N. Y. Times, *The New York Times Guide to Essential Knowledge, Second Edition: A Desk Reference for the Curious Mind*, p. 448. St. Martin's Press, 2007.

[64] InfoWorld Media Group, *InfoWorld - Personal Computers: The Global Marketplace*, p. 1. InfoWorld, 02 1982.

[65] Amazon Web Services, "Announcing Amazon Elastic Compute Cloud (Amazon EC2)." https://web.archive.org/web/20140813195808/http://aws.amazon.com/about-aws/whats-new/2006/08/24/announcing-amazon-elastic-compute-cloud-amazon-ec2---beta/, 2006.

[66] Microsoft Corporation, "Windows Azure General Availability." https://web.archive.org/web/20140511230956/http://blogs.technet.com/b/microsoft_blog/archive/2010/02/01/windows-azure-general-availability.aspx, 2010.

[67] Bram Cohen, "The BitTorrent Protocol Specification." https://web.archive.org/web/20140208002821/http://bittorrent.org/beps/bep_0003.html, 2012. (BitTorrent was first released in 2001).

[68] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[69] M. S. Alfiad, D. van den Borne, S. L. Jansen, T. Wuth, M. Kuschnerov, G. Grosso, A. Napoli, and H. de Waardt, "111-Gb/s POLMUX-RZ-DQPSK transmission over LEAF: Optical versus electrical dispersion compensation," in *2009 Conference on Optical Fiber Communication*, pp. 1–3, 2009.

[70] M. Fourman, G. Boulton, P. Buneman, P. Clarke, S. McLaughlin, A. Milne, I. Ritchie, M. Schaffer, and P. Purvis, *Digital Scotland*. Royal Society of Edinburgh, Oct. 2010.

[71] J. Li and M. Forzati, "The Social-Economic Impact of Fiber Broadband: A Hype or a Reality?," in *2018 20th International Conference on Transparent Optical Networks (ICTON)*, pp. 1–7, 2018.

[72] W. Gao, W. G. Hatcher, and W. Yu, "A Survey of Blockchain: Techniques, Applications, and Challenges," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–11, 2018.

[73] D. Chaum, "Blind signatures for untraceable payments," in *Advances in Cryptology*, (Boston, MA), pp. 199–203, Springer US, 1983.

[74] L. Law, S. Sabett, and J. Solinas, "How to Make a Mint: The Cryptography of Anonymous Electronic Cash," *American University Law Review*, vol. 46, p. 1131, 1996.

[75] V. Buterin *et al.*, "Ethereum white paper," *https://github.com/ethereum/wiki/wiki/White-Paper*, 2014.

[76] N. Szabo, "Formalizing and Securing Relationships on Public Networks," *First Monday*, vol. 6, 1997.

[77] G. Wood and J. Steiner, *Trustless Computing—The What Not the How*, pp. 133–144. Cham: Springer International Publishing, 2016.

[78] S. K. Singh and V. R. Vadi, "Evolutionary Transformation of Blockchain Technology," *International Journal of Engineering Research & Technology*, vol. 10, Jan. 2022.

[79] P. Mukherjee and C. Pradhan, *Blockchain 1.0 to Blockchain 4.0—The Evolutionary Transformation of Blockchain Technology*, pp. 29–49. Cham: Springer International Publishing, 2021.

[80] D. Yaga, P. Mell, N. Roby, and K. Scarfone, *Blockchain Technology Overview*. NIST, Oct 2018.

[81] S. Popov, "The Tangle," *White paper*, vol. 1, no. 3, p. 30, 2018.

[82] D. L. Baird, M. Harmon, and P. Madsen, "Hedera: A Public Hashgraph Network & Governing Council,"

[83] Hyperledger Foundation, "Hyperledger Fabric." https://www.hyperledger.org/projects/fabric. Accessed: 2023-12-10.

[84] Hyperledger Foundation, "Hyperledger Sawtooth." https://www.hyperledger.org/projects/sawtooth. Accessed: 2023-12-10.

[85] Ethereum, "Official Go implementation of the Ethereum protocol." https://geth.ethereum.org/. Accessed: 2023-12-10.

[86] Hyperledger Foundation, "Hyperledger Besu." https://www.hyperledger.org/projects/besu. Accessed: 2023-12-10.

[87] D. Shakhbulatov, J. Medina, Z. Dong, and R. Rojas-Cessa, "How Blockchain Enhances Supply Chain Management: A Survey," *IEEE Open Journal of the Computer Society*, vol. 1, pp. 230–249, 2020.

[88] XDC Network, "XinFin Docs." https://docs.xinfin.org/. Accessed: 2023-12-11.

[89] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, pp. 382–401, July 1982.

[90] M. Barborak, A. Dahbura, and M. Malek, "The consensus problem in fault-tolerant computing," *ACM Computing Surveys*, vol. 25, pp. 171–220, June 1993.

[91] J. S. Søgaard, P. W. Eklund, L. Herskind, and J. Spasovski, "Improving Trust in a (Trans)National Invoicing System: The Performance of Crash vs. Byzantine Fault Tolerance at Scale," *Applied Sciences*, vol. 13, no. 12, 2023.

[92] OpenEthereum, "Aura." https://openethereum.github.io/Aura. Accessed: 2023-12-11.

[93] H. Moniz, "The Istanbul BFT Consensus Algorithm," *CoRR*, vol. abs/2002.03613, 2020.

[94] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, (USA), pp. 305–320, USENIX Association, 2014.

[95] Tendermint, "Tendermint." https://tendermint.com. Accessed: 2023-12-10.

[96] J. Nijsse and A. Litchfield, "A Taxonomy of Blockchain Consensus Methods," *Cryptography*, vol. 4, no. 4, 2020.

[97] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *CoRR*, vol. abs/1710.09437, 2017.

[98] A. Back, "Hashcash - A Denial of Service Counter-Measure," 2002.

[99] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," vol. 19, 2012.

[100] Hyperledger, "Hyperledger Sawtooth Documentation." https://sawtooth.hyperledger.org/docs/1.2/. Accessed: 2023-12-13.

[101] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, (USA), pp. 173–186, USENIX Association, 1999.

[102] L. Leslie *et al.*, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.

[103] Yugabyte, "What is the Raft Consensus Algorithm?." https://web.archive.org/web/20230601221742/https://www.yugabyte.com/tech/raft-consensus-algorithm/. Accessed: 2023-12-15.

[104] A. Stewart and E. Kokoris-Kogia, "GRANDPA: a Byzantine Finality Gadget," 2020.

[105] Péter Szilágyi, "Clique." https://eips.ethereum.org/EIPS/eip-225, 2017. Accessed: 2023-12-11.

[106] C. N. Samuel, *Connected car communication by DLT technologies : mobility service implementation by adaptation of consortium blockchain consensus algorithms*. Theses, Université Côte d'Azur, Dec. 2023.

[107] Alph Network, "Delegated Proof of Authority." https://docs.alph.network/get-started/hdpwork/. Accessed: 2023-12-10.

[108] BitShares Blockchain Foundation, "Delegated Proof of Stake (DPOS)." https://how.bitshares.works/en/master/technology/dpos.html. Accessed: 2023-12-11.

[109] Neo blockchain, "NEO White Paper." https://docs.neo.org/. Accessed: 2023-12-11.

[110] S. Zhang and J.-H. Lee, "Analysis of the main consensus protocols of blockchain," *ICT Express*, vol. 6, no. 2, pp. 93–97, 2020.

[111] K. Wüst and A. Gervais, "Do you Need a Blockchain?," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pp. 45–54, 2018.

[112] A. O. Bada, A. Damianou, C. M. Angelopoulos, and V. Katos, "Towards a Green Blockchain: A Review of Consensus Mechanisms and their Energy Consumption," in *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pp. 503–511, 2021.

[113] J. Li, M. Irfan, S. Samad, B. Ali, Y. Zhang, D. Badulescu, and A. Badulescu, "The Relationship between Energy Consumption, CO2 Emissions, Economic Growth, and Health Indicators," *International Journal of Environmental Research and Public Health*, vol. 20, no. 3, p. 2325, 2023.

[114] M. Platt, J. Sedlmeir, D. Platt, J. Xu, P. Tasca, N. Vadgama, and J. I. Ibañez, "The Energy Footprint of Blockchain Consensus Mechanisms Beyond Proof-of-Work," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 1135–1144, 2021.

[115] Visa, "Visa: The technology behind Visa." https://usa.visa.com/dam/VCOM/download/corporate/media/visanet-technology/visa-net-booklet.pdf, 2022. Accessed: 2023-12-19.

[116] D. A. . A. F. Rachel Rybarczyk, "On Bitcoin's Energy Consumption: A Quantitative Approach to a Subjective Question."

[117] M. Giancaspro, "Is a 'smart contract' really a smart idea? Insights from a legal perspective," *Computer Law & Security Review*, vol. 33, no. 6, pp. 825–835, 2017.

[118] G. Ciatto, R. Calegari, S. Mariani, E. Denti, and A. Omicini, "From the Blockchain to Logic Programming and Back: Research Perspectives," in *WOA 2018 – 19th Workshop "From Objects to Agents"* (M. Cossentino, L. Sabatucci, and V. Seidita, eds.), vol. 2215 of *CEUR Workshop Proceedings*, pp. 69–74, Sun SITE Central Europe, RWTH Aachen University, June 2018.

[119] S. Tikhomirov, "Ethereum: State of Knowledge and Research Perspectives," in *Foundations and Practice of Security*, (Cham), pp. 206–221, Springer International Publishing, 2018.

[120] J. Shorish, "Blockchain State Machine Representation." SocArXiv, Jan 2018.

[121] E. F. Moore, "A Simplified Universal Turing Machine," in *Proceedings of the 1952 ACM National Meeting (Toronto)*, ACM '52, (New York, NY, USA), pp. 50–54, Association for Computing Machinery, 1952.

[122] E. Stojmenova Duh, A. Duh, U. Droftina, T. Kos, U. Duh, T. Simonič Korošak, and D. Korošak, "Publish-and-Flourish: Using Blockchain Platform to Enable Cooperative Scholarly Communication," *Publications*, vol. 7, no. 2, 2019.

[123] A. Mavridou and A. Laszka, "Designing secure ethereum smart contracts: A finite state machine based approach," 2017.

[124] C. Saraf and S. Sabadra, "Blockchain platforms: A compendium," in *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*, (Bangkok), pp. 1–6, IEEE, May 2018.

[125] M. P. Caro, M. S. Ali, M. Vecchio, and R. Giaffreda, "Blockchain-based traceability in Agri-Food supply chain management: A practical implementation," in *2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IOT Tuscany)*, pp. 1–4, May 2018.

[126] T. A. Almeshal and A. A. Alhogail, "Blockchain for Businesses: A Scoping Review of Suitability Evaluations Frameworks," *IEEE Access*, vol. 9, pp. 155425–155442, 2021.

[127] 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, Feb 2019.

[128] M. Nour, J. P. Chaves-Avila, and A. Sanchez-Miralles, "Review of Blockchain Potential Applications in the Electricity Sector and Challenges for Large Scale Adoption," *IEEE Access*, vol. 10, pp. 47384–47418, 2022.

[129] I. Karamitsos, M. Papadaki, and N. B. A. Barghuthi, "Design of the Blockchain Smart Contract: A Use Case for Real Estate," *Journal of Information Security*, vol. 9, pp. 177–190, June 2018.

[130] 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.

[131] U. W. Chohan, "The Decentralized Autonomous Organization and Governance Issues," *SSRN Electronic Journal*, Dec. 2017.

[132] Ethereum, "Solidity Documentation v0.8.22." https://docs.soliditylang.org/en/v0.8.22/. Accessed: 2023-12-13.

[133] Parity Technologies, "Ink! Documentation." https://use.ink/. Accessed: 2023-12-13.

[134] Parity Technologies, "Wasmi." https://github.com/paritytech/Wasmi. Accessed: 2023-12-13.

[135] Tezos, "Tezos Michelson Documentation." https://docs.tezos.com/smart-contracts/languages/michelson. Accessed: 2023-12-13.

[136] Tezos, "Tezos Ligo Documentation." https://docs.tezos.com/smart-contracts/languages/ligo. Accessed: 2023-12-13.

[137] Hyperledger, "Hyperledger Fabric Smart Contracts and Chaincode." https://hyperledger-fabric.readthedocs.io/en/latest/smartcontract/smartcontract.html. Accessed: 2023-12-13.

[138] Parity Technologies, "Substrate Custom pallets." https://docs.substrate.io/build/custom-pallets/. Accessed: 2023-12-13.

[139] Ethereum, "Ethereum Virtual Machine (EVM)." https://ethereum.org/en/developers/docs/evm/. Accessed: 2023-12-13.

[140] I. Grigg, "The Ricardian contract," in *Proceedings. First IEEE International Workshop on Electronic Contracting, 2004.*, pp. 25–31, 2004.

[141] I. Grigg, "Financial Cryptography in 7 Layers." https://iang.org/papers/fc7.html, 1999. Accessed: 2023-12-13.

[142] OpenBazaar, "OpenBazaar Ricardian contracts." https://github.com/OpenBazaar/docs/blob/86e8a30da26e9ead48cb1e97241cd28a8cd336f8/docs/overview.md. Accessed: 2023-06-23.

[143] EOS.IO, "Ricardian Template Toolkit by EOS.IO." https://eos.io/for-developers/build/ricardian-template-toolkit/. Accessed: 2023-06-23.

[144] Ethereum, "Introduction to dApps." https://ethereum.org/en/developers/docs/dapps/, 2022. Accessed: 2023-12-19.

[145] Napster, Inc., "Napster Home Page." https://web.archive.org/web/20010302033202/https://www.napster.com/, 2001. (Napster was shutdown in July 11, 2001).

[146] Tor Project, "Tor Project Home Page." https://www.torproject.org/, 2023. Accessed: 2023-12-10.

[147] Ethereum, "Introduction to Web3." https://ethereum.org/en/web3/, 2022. Accessed: 2023-12-19.

[148] S. Green, "Smart contracts: Interpretation and rectification," *Lloyd's Maritime and Commercial Law Quarterly*, vol. 2018, pp. 234–251, May 2018.

[149] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, 2014.

[150] M. Salehi, J. Clark, and M. Mannan, "Not so Immutable: Upgradeability of Smart Contracts on Ethereum," in *Financial Cryptography and Data Security. FC 2022 International Workshops*, (Cham), pp. 539–554, Springer International Publishing, 2023.

[151] Michael del Castillo , "The DAO Attacked: Code Issue Leads to $60 Million Ether Theft." https://web.archive.org/web/20231115064947/https://www.coindesk.com/markets/2016/06/17/the-dao-attacked-code-issue-leads-to-60-million-ether-theft/, 2016. Accessed: 2023-12-14.

[152] JD Alois, "Ethereum Parity Hack May Impact ETH 500,000 or $146 Million." https://web.archive.org/web/20230330143200/https://www.crowdfundinsider.com/2017/11/124200-ethereum-parity-hack-may-impact-eth-500000-146-million/, 2017. Accessed: 2023-12-14.

[153] R. Jia and S. Yin, "To EVM or Not to EVM: Blockchain Compatibility and Network Effects," in *Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security*, DeFi'22, (New York, NY, USA), pp. 23–29, Association for Computing Machinery, 2022.

[154] V. Buterin, "The Limits to Blockchain Scalability." https://vitalik.ca/general/2021/05/23/scaling.html.

[155] H. Guo and X. Yu, "A survey on blockchain technology and its security," *Blockchain: Research and Applications*, vol. 3, no. 2, p. 100067, 2022.

[156] D. Dasgupta, J. M. Shrein, and K. D. Gupta, "A survey of blockchain from security perspective," *Journal of Banking and Financial Technology*, vol. 3, pp. 1–17, Apr. 2019.

[157] C. Wirth and M. Kolain, "Privacy by BlockChain Design: A BlockChain-enabled GDPR-compliant Approach for Handling Personal Data," 2018.

[158] I. Kang, A. Gupta, and O. Seneviratne, "Blockchain Interoperability Landscape," in *2022 IEEE International Conference on Big Data (Big Data)*, (Los Alamitos, CA, USA), pp. 3191–3200, IEEE Computer Society, dec 2022.

[159] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A Survey on Blockchain Interoperability: Past, Present, and Future Trends," *ACM Comput. Surv.*, vol. 54, oct 2021.

[160] P. Lafourcade and M. Lombard-Platet, "About blockchain interoperability," *Information Processing Letters*, vol. 161, p. 105976, 2020.

[161] T. Hardjono, A. Lipton, and A. Pentland, "Towards a Design Philosophy for Interoperable Blockchain Systems," 2018.

[162] S. Schulte, M. Sigwart, P. Frauenthaler, and M. Borkowski, "Towards Blockchain Interoperability," in *Business Process Management: Blockchain and Central and Eastern Europe Forum*, (Cham), pp. 3–10, Springer International Publishing, 2019.

[163] V. Buterin, "Chain interoperability," *R3 research paper*, vol. 9, pp. 1–25, 2016.

[164] M. Herlihy, "Atomic Cross-Chain Swaps," in *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, (New York, NY, USA), p. 245–254, Association for Computing Machinery, 2018.

[165] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," *White paper*, vol. 21, no. 2327, p. 4662, 2016.

[166] J. Kwon and E. Buchman, "Cosmos whitepaper," *A Netw. Distrib. Ledgers*, vol. 27, 2019.

[167] Avalanche, "Avalanche Whitepapers," 2023. Accessed: 2023-12-22.

[168] Composable Finance, "Picasso infrastructure from Composable Finance." https://docs.composable.finance/networks/picasso-parachain-overview, 2018. Accessed: 2023-12-16.

[169] H. Abbas, M. Caprolu, and R. D. Pietro, "Analysis of Polkadot: Architecture, Internals, and Contradictions," in *2022 IEEE International Conference on Blockchain (Blockchain)*, (Los Alamitos, CA, USA), pp. 61–70, IEEE Computer Society, aug 2022.

[170] B. A. Scriber, "A Framework for Determining Blockchain Applicability," *IEEE Software*, vol. 35, pp. 70–77, Jul 2018.

[171] M. Staderini, E. Schiavone, and A. Bondavalli, "A Requirements-Driven Methodology for the Proper Selection and Configuration of Blockchains," in *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, (Salvador, Brazil), pp. 201–206, IEEE, Oct 2018.

[172] G. Büyüközkan and G. Tüfekçi, "A decision-making framework for evaluating appropriate business blockchain platforms using multiple preference formats and VIKOR," *Information Sciences*, vol. 571, pp. 337–357, Sep 2021.

[173] S. Farshidi, S. Jansen, S. España, and J. Verkleij, "Decision Support for Blockchain Platform Selection: Three Industry Case Studies," *IEEE Transactions on Engineering Management*, vol. 67, pp. 1109–1128, Nov 2020.

[174] S. Khatri, F. A. Alzahrani, M. T. J. Ansari, A. Agrawal, R. Kumar, and R. A. Khan, "A Systematic Analysis on Blockchain Integration With Healthcare Domain: Scope and Challenges," *IEEE Access*, vol. 9, pp. 84666–84687, 2021.

[175] I. J. Scott, M. de Castro Neto, and F. L. Pinheiro, "Bringing trust and transparency to the opaque world of waste management with blockchain: A Polkadot parathread application," *Computers & Industrial Engineering*, vol. 182, p. 109347, Aug 2023.

[176] S. Punathumkandi, V. M. Sundaram, and P. Panneer, "Interoperable Permissioned-Blockchain with Sustainable Performance," *Sustainability*, vol. 13, p. 11132, Jan. 2021.

[177] S. D. Kotey, E. T. Tchao, A.-R. Ahmed, A. S. Agbemenu, H. Nunoo-Mensah, A. Sikora, D. Welte, and E. Keelson, "Blockchain interoperability: the state of heterogenous blockchain-to-blockchain communication," *IET Communications*, vol. 17, no. 8, pp. 891–914, 2023.

[178] M. Salimitari, M. Chatterjee, and Y. P. Fallah, "A survey on consensus methods in blockchain for resource-constrained IoT networks," *Internet of Things*, vol. 11, p. 100212, Sept. 2020.

[179] M. Ballatore, L. Gerrits, R. Kromes, L. Arena, and F. Verdier, "Toward the Conception of a Multichain to Meet Users' Future Needs: A Design Science Research Approach to Digital Servitization in the Automotive Industry," *IEEE Transactions on Engineering Management*, pp. 1–15, 2023.

[180] IBM Corporation, "Device democracy." https://www.ibm.com/thought-leadership/institute-business-value/report/device-democracy. Accessed: 2023-08-07.

[181] P. Danzi, A. E. Kalor, R. B. Sorensen, A. K. Hagelskjaer, L. D. Nguyen, C. Stefanovic, and P. Popovski, "Communication Aspects of the Integration of Wireless IoT Devices with Distributed Ledger Technology," *IEEE Network*, vol. 34, no. 1, pp. 47–53, 2020.

[182] M. Pincheira, M. Vecchio, R. Giaffreda, and S. S. Kanhere, "Cost-effective IoT devices as trustworthy data sources for a blockchain-based water management system in precision agriculture," *Computers and Electronics in Agriculture*, vol. 180, p. 105889, 2021.

[183] B. Farahani, F. Firouzi, and M. Luecking, "The convergence of IoT and distributed ledger technologies (DLT): Opportunities, challenges, and solutions," *Journal of Network and Computer Applications*, vol. 177, p. 102936, Mar 2021.

[184] O. Alfandi, S. Khanji, L. Ahmad, and A. Khattak, "A survey on boosting IoT security and privacy through blockchain: Exploration, requirements, and open issues," *Cluster Computing*, vol. 24, p. 37–55, Mar. 2021.

[185] S. Kumar, A. Singh, A. Benslimane, P. Chithaluru, M. A. Albahar, R. S. Rathore, and R. M. Álvarez, "An Optimized Intelligent Computational Security Model for Interconnected Blockchain-IoT System & Cities," *Ad Hoc Networks*, vol. 151, p. 103299, Dec. 2023.

[186] M. A. Kandi, D. E. Kouicem, M. Doudou, H. Lakhlef, A. Bouabdallah, and Y. Challal, "A decentralized blockchain-based key management protocol for heterogeneous and dynamic IoT devices," *Computer Communications*, vol. 191, pp. 11–25, July 2022.

[187] International Organization for Standardization (ISO), "What is cryptography?." https://www.iso.org/information-security/what-is-cryptography. [Accessed 25-Sept-2023].

[188] M. Gill, "Preventing Money Laundering or Obstructing Business?: Financial Companies' Perspectives on "Know Your Customer" Procedures," *British Journal of Criminology*, vol. 44, pp. 582–594, July 2004.

[189] U. W. Chohan, "The Problems of Cryptocurrency Thefts and Exchange Shutdowns," *SSRN Electronic Journal*, Feb. 2018.

[190] L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, *Report on Post-Quantum Cryptography*. NIST, Apr 2016.

[191] I. T. L. Computer Security Division, "Selected Algorithms 2022 - Post-Quantum Cryptography | CSRC | CSRC," Jan. 2017.

[192] M. Allende, D. L. León, S. Cerón, A. Pareja, E. Pacheco, A. Leal, M. Da Silva, A. Pardo, D. Jones, D. J. Worrall, B. Merriman, J. Gilmore, N. Kitchener, and S. E. Venegas-Andraca, "Quantum-resistance in blockchain networks," *Scientific Reports*, vol. 13, p. 5664, Apr. 2023.

[193] Y. Harbi, Z. Aliouat, A. Refoufi, and S. Harous, "Recent Security Trends in Internet of Things: A Comprehensive Survey," *IEEE Access*, vol. 9, pp. 113292–113314, 2021.

[194] V. A. Thakor, M. A. Razzaque, and M. R. A. Khandaker, "Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities," *IEEE Access*, vol. 9, pp. 28177–28193, 2021.

[195] J. Arcenegui, R. Arjona, R. Román, and I. Baturone, "Secure Combination of IoT and Blockchain by Physically Binding IoT Devices to Smart Non-Fungible Tokens Using PUFs," *Sensors*, vol. 21, p. 3119, Jan. 2021.

[196] R. Kromes and F. Verdier, "Accelerating Blockchain Applications on IoT Architecture Models - Solutions and Drawbacks," *Distributed Ledger Technologies: Research and Practice*, Sept. 2023. Just Accepted.

[197] "Android Automotive - Documentation." https://source.android.com/docs/automotive. Accessed: 2023-12-29.

[198] ONISR, "Bilan 2019 de la sécurité routière." https://www.onisr.securite-routiere.gouv.fr/.

[199] Statista, "Total number of registered passenger cars in France from 2011 to 2022." https://www.statista.com/statistics/455887/passenger-cars-registered-in-france/.

[200] J. Kalajdjieski, M. Raikwar, N. Arsov, G. Velinov, and D. Gligoroski, "Databases fit for blockchain technology: A complete overview," *Blockchain: Research and Applications*, vol. 4, no. 1, p. 100116, 2023.

[201] S. Bergman, M. Asplund, and S. Nadjm-Tehrani, "Permissioned blockchains and distributed databases: A performance study," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 12, p. e5227, 2020. e5227 cpe.5227.

[202] "MySQL - Estimating Query Performance." https://dev.mysql.com/doc/refman/8.0/en/estimating-performance.html. Accessed: 2023-12-29.

[203] R. Kromes, *Designing a specific Low Power architecture for blockchain and Smart Contracts operations in IoT platform*. Theses, Université Côte d'Azur, Dec. 2021.

[204] L. Gerrits, R. Kromes, and F. Verdier, "A True Decentralized Implementation Based on IoT and Blockchain: a Vehicle Accident Use Case," in *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, pp. 1–6, 2020.

[205] L. Gerrits, T. Mabrut, and F. Verdier, "Communicate Vehicle Accident Data to Blockchain for Secure and Reliable Record-Keeping Using Android Automotive Application," in *2023 Fifth International Conference on Blockchain Computing and Applications (BCCA)*, pp. 22–27, 2023.

[206] H. Wang, G. Chen, Y. Zhang, and Z. Lin, "Multi-Certificate Attacks against Proof-of-Elapsed-Time and Their Countermeasures," in *Proceedings 2022 Network and Distributed System Security Symposium*, (San Diego, CA, USA), Internet Society, 2022.

[207] Parity Technologies, "Substrate Documentation." https://docs.substrate.io/. Accessed: 2023-12-13.

[208] Parity Technologies, "Substrate Runtimes." https://marketplace.substrate.io/runtimes/. Accessed: 2023-12-15.

[209] Block.one, "EOSIO." https://eos.io/. Accessed: 2023-12-15.

[210] Antelope, "Antelope Website." https://antelope.io/. Accessed: 2023-12-15.

[211] Nodle, "Nodle Project Website." https://www.nodle.com/network. Accessed: 2024-01-05.

[212] L. Gerrits, E. Kilimou, R. Kromes, L. Faure, and F. Verdier, "A Blockchain cloud architecture deployment for an industrial IoT use case," in *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, pp. 1–6, 2021.

[213] Y. Khacef and U. Cote d'Azur, "Smart IoT For Mobility," research report, UCA, 2020. Tutorat de Master 1 encadré François Verdier et Roland Kromes.

[214] Ethereum, "RLP (Recursive Length Prefix)." https://ethereum.org/en/developers/docs/data-structures-and-encoding/rlp/. Accessed: 2023-10-20.

[215] Google, "Protobuf." https://developers.google.com/protocol-buffers. Accessed: 2023-10-20.

[216] Substrate, "SCALE Codec." https://docs.substrate.io/reference/scale-codec/. Accessed: 2023-10-20.

[217] Bitcoin Developers, "Library for EC operations on curve secp256k1." https://github.com/bitcoin-core/secp256k1. (accessed: 11.11.2023).

[218] L. Gerrits, "Comparative study of EOS and IOTA blockchains in the context of Smart IoT for Mobility," internship report, stage Master 2 Estel Université de Nice-Sophia Antipolis, Sept. 2020. Encadré par François Verdier équipe EDGE.

[219] Bitcoin, "Transaction malleability." https://en.bitcoin.it/wiki/Transaction_malleability. Accessed: 2023-10-20.

[220] N. Massiéra, "Projet Smart IoT for Mobility," internship report, stage Master 2 ESTEL universite de Nice Sophia Antipolis, Sept. 2018. Encadré par François Verdier.

[221] M. Arnaudo, L. Gerrits, I. Grishkov, R. Kromes, and F. Verdier, "Blockchains Accesses for Low-Power Embedded Devices Using LoRaWAN," in *Proceedings of the 12th International Conference on the Internet of Things*, IoT '22, (New York, NY, USA), pp. 119–126, Association for Computing Machinery, 2023.

[222] Orne Brocaar, "ChirpStack." https://www.chirpstack.io. Accessed: 2023-09-05.

[223] A. Banks and R. Gupta, "MQTT Version 3.1.1. OASIS Standard.." http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html, 2014.

[224] G. Falazi, U. Breitenbücher, F. Daniel, A. Lamparelli, F. Leymann, and V. Yussupov, "Smart Contract Invocation Protocol (SCIP): A Protocol for the Uniform Integration of Heterogeneous Blockchain Smart Contracts," in *Advanced Information Systems Engineering*, (Cham), pp. 134–149, Springer International Publishing, 2020.

[225] Nicky Montana, "Blockchain layers (L0, L1, L2, L3) in a Diagram." https://medium.com/@nick.5montana/blockchain-layers-l0-l1-l2-l3-in-a-diagram-569162398db. Accessed: 2024-02-15.

[226] Ethereum, "The Merge." https://ethereum.org/en/roadmap/merge/, 2022. Accessed: 2023-12-19.

8 Appendix

8.1 Consensuses

8.1.1 Block creation flow: PoW vs PoS

This section illustrates the fundamental differences between the Proof of Work (PoW) and Proof of Stake (PoS) blockchain consensus mechanisms, as depicted in Figures 8.1.1 and 8.1.2.

Figure 8.1.1: Proof of Work (PoW) Consensus Mechanism. PoW, utilized by blockchains such as Bitcoin and Ethereum, involves a computational competition among nodes to create a new block. Nodes must solve a cryptographic puzzle that demands significant computational power. The first node to find the solution is granted the right to add a new block to the blockchain. The process requires nodes to continuously adjust a nonce value to discover the correct solution, a task that is both energy-intensive and time-consuming. The complexity of PoW puzzles ensures the security of the blockchain, as altering a single block becomes increasingly difficult with each new block added. [68] [110]

Figure 8.1.2: Proof of Stake (PoS) Consensus Mechanism. PoS presents an more energy-efficient alternative to PoW, where the probability of a node being chosen to create a new block is based on its stake, i.e., the number of coins it holds, rather than computational power. This mechanism significantly reduces the energy required for block creation. PoS also considers the 'coin age', a factor that influences a node's chance of solving the puzzle and creating a new block [99]. The transition of Ethereum from PoW to PoS [226] underscores the growing importance and potential of this more sustainable consensus mechanism in blockchain technology. [99] [110]

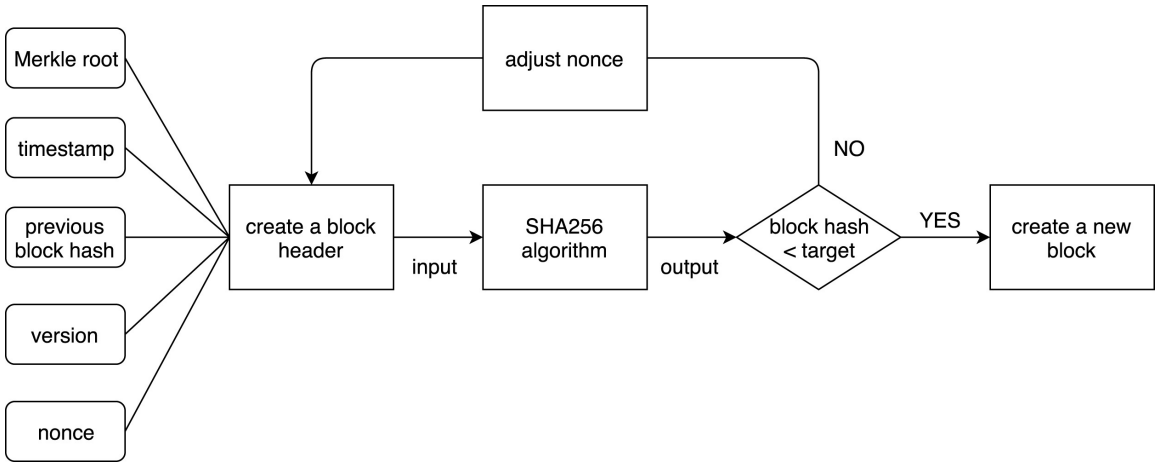


Figure 8.1.1: Flow of PoW [110]

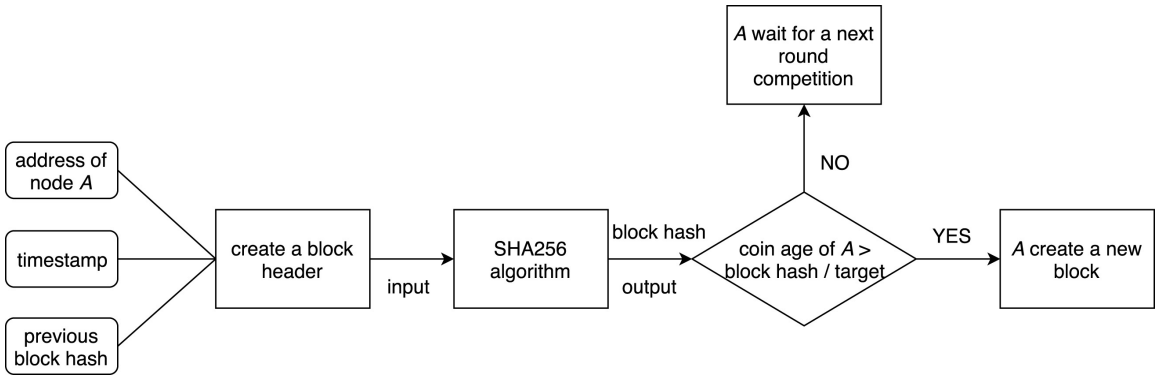


Figure 8.1.2: Flow of PoS [110]

8.2 Smart Contracts

8.2.1 Comparison of Solidity, Rust with Ink!, and Michelson/Ligo Code Styles

8.2.1.1 Solidity (Ethereum)

Language Characteristics: Solidity is a high-level language influenced by C++, Python, and JavaScript, designed to target the Ethereum Virtual Machine (EVM).
Syntax Example:

Code 8.1: Solidity Smart Contract Example

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SimpleIncrementer {
  uint256 private count;

  constructor() {
    count = 0;
  }

  function increment() public {
    count += 1;
  }

  function getCount() public view returns (uint256) {
    return count;
  }
}
```

Key Features: Statically typed, supports inheritance, libraries, and complex user-defined types.
Use Case: Predominantly used for Ethereum-based blockchain applications.

8.2.1.2 Rust with Ink! (Polkadot/Kusama)

Language Characteristics: Rust is a multi-paradigm system programming language focused on safety and performance. Ink! is a Rust-based eDSL[1] for writing Wasm smart contracts.
Syntax Example:

[1] eDSL (embedded Domain Specific Language) refers to a specialized set of tools and language features designed for a specific purpose, and is embedded within a broader programming language. In blockchain development, an eDSL like Ink! for Rust, offers specific functionalities for creating smart contracts, while utilizing the familiar environment of the Rust host language.

Code 8.2: Rust with Ink! Smart Contract Example

```rust
use ink_lang as ink;
#[ink::contract]
pub mod simple_incrementer {
  #[ink(storage)]
  pub struct SimpleIncrementer {
    count: u128,
  }
  impl SimpleIncrementer {
    #[ink(constructor)]
    pub fn new() -> Self {
      Self { count: 0 }
    }

    #[ink(message)]
    pub fn increment(&mut self) {
      self.count += 1;
    }

    #[ink(message)]
    pub fn get_count(&self) -> u128 {
      self.count
    }
  }
}
```

Key Features: Emphasizes safety and correctness, suitable for writing secure smart contracts.
Use Case: Mainly used for Polkadot/Kusama and other Substrate-based blockchains.

8.2.1.3 Michelson and Ligo (Tezos)

Language Characteristics: Michelson is a low-level stack-based language. Ligo is a high-level language that compiles down to Michelson.
Michelson Syntax Example:

Code 8.3: Michelson Smart Contract Example

```
parameter (or (unit %increment) (unit %getCount));
storage nat;
code { UNPAIR ;
  IF_LEFT
  { DROP ; PUSH nat 1 ; SWAP ; ADD ; NIL operation ; PAIR }
  { NIL operation ; DIP { DUP } ; SWAP ; PAIR } };
```

Ligo Syntax Example:

Code 8.4: Ligo Smart Contract Example

```
type action is
Increment
| GetCount

type storage is nat

(* Entry point *)
function main (const p : action; const s : storage) : (list(operation) * storage) is
block {
  var new_s : storage := s;
} with
case p of
Increment -> (list [], new_s + 1)
| GetCount -> (list [], new_s)
end
```

Key Features: Michelson is designed for formal verification. Ligo is more user-friendly and higher-level.
Use Case: Specifically tailored for Tezos blockchain applications.

8.2.2 Solidity Smart Contract Incrementer

We will use here the same smart contract code as Code 8.1, which provide a basic integer incrementer. The corresponding bytecode for Code 8.1 (using compiler soljson-v0.8.22+commit.4fc1097e) is:

0x608060405234801561000f575f80fd5b505f8081905550610149806100235f395ff3fe608060405234801561000f575f80fd5b506004361061003457 5f3560e01c8063a87d942c14610038578063d09de08a14610056575b5f80fd5b61004061006565b60405161004d919061009a565b60405180910390f35b61005e610068565b005b5f8054905090565b60015f8082825461007991906100e0565b925050819055505565b5f819050919050565b6100948161008256 5b82525050565b5f6020820190506100ad5f830184610085565b92915050565b5f7f4e487b7100000000000000000000000000000000000000000000000000000000 000000000000000000000000000000005f52601160045260245ffd5b5f6100ea82610082565b91506100f5836100825 6 5b9250828201905080821115610101576101005b5b9291505056fea2646970667358221220cd3119620df73d08d1093c5d65478a0c84b97b6278d223528b47e7fcee5ccc5964736f6c63430008160033

Corresponding ABI for Code 8.1:

Code 8.5: Solidity Smart Contract Incrementer ABI

```
[
  {
    "inputs": [],
    "name": "increment",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [],
    "stateMutability": "nonpayable",
    "type": "constructor"
  },
  {
    "inputs": [],
    "name": "getCount",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  }
]
```

8.2.3 Use Case Pallet in Rust language for Substrate-based blockchain

Two differentiate the vehicle management and the accident use case, in practice, two pallets are created called `sim_renault` (Code 8.6) and `sim_renault_accident` (Code 8.7).

Code 8.6: Use Case "sim_renault" Pallet in Rust Language for Substrate-based blockchains

```
#![cfg_attr(not(feature = "std"), no_std)]
```

```
2
3  //! Pallet for Renault car manufacturer.
4  //! Aim: Manages Renault's vehicles.
5  pub use pallet::*;
6
7  #[cfg(test)]
8  mod mock;
9  #[cfg(test)]
10 mod tests;
11 #[cfg(feature = "runtime-benchmarks")]
12 mod benchmarking;
13
14 #[frame_support::pallet]
15 pub mod pallet {
16   use super::*;
17   use frame_support::{dispatch::DispatchResultWithPostInfo, pallet_prelude::*, weights::Pays};
18   use frame_system::pallet_prelude::*;
19
20   /// Configure the pallet by specifying the parameters and types on which it depends.
21   #[pallet::config]
22   pub trait Config: frame_system::Config {
23     /// Because this pallet emits events, it depends on the runtime's definition of an event.
24     type RuntimeEvent: From<Event<Self>> + IsType<<Self as frame_system::Config>::RuntimeEvent
       >;
25     // /// Weight information for extrinsics in this pallet.
26     // type WeightInfo: WeightInfo;
27   }
28
29   #[pallet::pallet]
30   #[pallet::generate_store(pub(super) trait Store)]
31   // #[pallet::without_storage_info]
32   pub struct Pallet<T>(_);
33
34   /// List of factory IDs added by the admin (sudo).
35   /// (
36   ///    actory ID => block nb
37   /// )
38   #[pallet::storage]
39   pub type Factories<T: Config> =
40   StorageMap<_, Blake2_128Concat, T::AccountId, T::BlockNumber, OptionQuery>;
41
42   /// List of vehicle ID added by the factories.
43   /// (
44   ///    vehicle ID => (factory ID, block nb)
45   /// )
46   #[pallet::storage]
47   pub type Vehicles<T: Config> =
48   StorageMap<_, Blake2_128Concat, T::AccountId, (T::AccountId, T::BlockNumber), OptionQuery>;
49
50   /// List of vehicle ID status.
51   /// (
52   ///    vehicle ID => is_initialized
53   /// )
54   #[pallet::storage]
55   pub type VehiclesStatus<T: Config> =
56   StorageMap<_, Blake2_128Concat, T::AccountId, bool, OptionQuery>;
57
58   #[pallet::event]
59   #[pallet::generate_deposit(pub(super) fn deposit_event)]
60   pub enum Event<T: Config> {
61     /// Event when a factory has been added to storage. FactoryStored(factory_id) [
       FactoryStored, AccountId]
62     FactoryStored(T::AccountId),
63     /// Event when a vehicle has been added to storage by a factory. VehicleStored(vehicle_id,
        origin) [VehicleStored, AccountId, AccountId]
```

```
64      VehicleStored(T::AccountId, T::AccountId),
65      /// Vehicle is now initialized.  VehicleInitialized(vehicle_id) [CrashStored, AccountId]
66      VehicleInitialized(T::AccountId),
67    }
68
69    // Errors inform users that something went wrong.
70    #[pallet::error]
71    pub enum Error<T> {
72      /// Factory is already in storage
73      FactoryAlreadyStored,
74      /// Factory is not in storage.
75      UnknownFactory,
76      /// Vehicle is not in storage.
77      UnknownVehicle,
78      /// Vehicle is already in storage
79      VehicleAlreadyStored,
80      /// Vehicle and origin aren't match
81      VehicleNotMatchingOrigin,
82    }
83
84    #[pallet::hooks]
85    impl<T: Config> Hooks<BlockNumberFor<T>> for Pallet<T> {}
86
87    #[pallet::call]
88    impl<T: Config> Pallet<T> {
89      /// Create a new factory.
90      /// Dispatchable that takes a singles value as a parameter (factory ID), writes the value
        to
91      /// storage (Factories) and emits an event. This function must be dispatched by a signed
        extrinsic.
92      #[pallet::weight((0, Pays::No))]
93      pub fn create_factory(
94      origin: OriginFor<T>,
95      factory_id: T::AccountId,
96      ) -> DispatchResultWithPostInfo {
97        ensure_root(origin)?;
98
99        // Verify that the specified factory_id has not already been stored.
100       ensure!(!Factories::<T>::contains_key(&factory_id), Error::<T>::FactoryAlreadyStored);
101
102       // Get the block number from the FRAME System module.
103       let current_block = <frame_system::Pallet<T>>::block_number();
104
105       // Store the factory_id with the sender and block number.
106       Factories::<T>::insert(&factory_id, current_block);
107
108       // Emit an event.
109       Self::deposit_event(Event::FactoryStored(factory_id));
110       Ok(().into())
111     }
112     /// Create a new vehicle.
113     /// Dispatchable that takes a singles value as a parameter (vehicle ID), writes the value
        to
114     /// storage (Vehicles) and emits an event. This function must be dispatched by a signed
        extrinsic.
115     #[pallet::weight((0, Pays::No))]
116     pub fn create_vehicle(
117     origin: OriginFor<T>,
118     vehicle_id: T::AccountId,
119     ) -> DispatchResultWithPostInfo {
120       let who = ensure_signed(origin)?;
121
122       // Verify that the specified factory_id exists.
123       ensure!(Factories::<T>::contains_key(&who), Error::<T>::UnknownFactory);
124
```

```
      // Verify that the specified car_id has not already been stored.
      ensure!(!Vehicles::<T>::contains_key(&vehicle_id), Error::<T>::VehicleAlreadyStored);

      // Get the block number from the FRAME System module.
      let current_block = <frame_system::Pallet<T>>::block_number();

      // Store the factory_id with the sender and block number.
      Vehicles::<T>::insert(&vehicle_id, (&who, current_block));

      // Emit an event.
      Self::deposit_event(Event::VehicleStored(vehicle_id, who));
      Ok(().into())
    }

    /// Init a vehicle.
    /// Dispatchable that takes a singles value as a parameter (vehicle ID), writes the value
    to
    /// storage (Factories) and emits an event. This function must be dispatched by a signed
    extrinsic.
    #[pallet::weight((0, Pays::No))]
    pub fn init_vehicle(
    origin: OriginFor<T>,
    vehicle_id: T::AccountId,
    ) -> DispatchResultWithPostInfo {
      let who = ensure_signed(origin)?;

      // Verify that the origin vehicle exists.
      ensure!(Vehicles::<T>::contains_key(&who), Error::<T>::UnknownVehicle);

      // Verify that the specified vehicle_id matches the origin.
      ensure!(who == vehicle_id, Error::<T>::VehicleNotMatchingOrigin);

      // Set the vehicle as initialized.
      VehiclesStatus::<T>::insert(&vehicle_id, true);

      // Emit an event.
      Self::deposit_event(Event::VehicleInitialized(vehicle_id));
      Ok(().into())
    }
  }
  impl<T: Config> Pallet<T> {
    /// Return status of a vehicle
    pub fn is_vehicle(vehicle_id: T::AccountId) -> bool {
      let status = VehiclesStatus::<T>::get(&vehicle_id).unwrap_or(false);
      if status == true {
        true
      } else {
        false
      }
    }
  }

  // Next is all the necessary to init the pallet with genesis info
  #[pallet::genesis_config]
  pub struct GenesisConfig<T: Config> {
    /// The `AccountId` of the sudo key.
    pub init_factory_and_vehicle: Option<T::AccountId>,
  }
  #[cfg(feature = "std")]
  impl<T: Config> Default for GenesisConfig<T> {
    fn default() -> Self {
      Self { init_factory_and_vehicle: None }
    }
  }
  #[pallet::genesis_build]
```

```
  impl<T: Config> GenesisBuild<T> for GenesisConfig<T> {
    fn build(&self) {
      if let Some(ref init_factory_and_vehicle) = self.init_factory_and_vehicle {
        Factories::<T>::insert(init_factory_and_vehicle.clone(), <frame_system::Pallet<T>>::block_number());
        Vehicles::<T>::insert(init_factory_and_vehicle.clone(), (init_factory_and_vehicle.clone(), <frame_system::Pallet<T>>::block_number()));
        VehiclesStatus::<T>::insert(init_factory_and_vehicle.clone(), true);
      }
    }
  }
}
```

Code 8.7: Use Case ”sim_renault_accident” Pallet in Rust Language for Substrate-based blockchains

```
#![cfg_attr(not(feature = "std"), no_std)]

//! Pallet to report an accident at Renault.
//! Aim: report an accident at Renault by sending a data hash. The raw data should be stored elsewhere.
//! NOTE: This pallet is tightly coupled with pallet-sim-renault.
pub use pallet::*;

#[cfg(feature = "runtime-benchmarks")]
mod benchmarking;

#[frame_support::pallet]
pub mod pallet {
  use super::*;
  use frame_support::{
    dispatch::{DispatchError, DispatchResult},
    inherent::Vec,
    pallet_prelude::*,
    weights::Pays,
  };
  use frame_system::{pallet_prelude::*, Config as SystemConfig};
  use log;
  use sha2::{Digest, Sha256};

  /// Custom error when retrieving accident data
  #[derive(Clone, Debug, Decode, Encode, Eq, PartialEq, TypeInfo)]
  pub enum GetAccidentDataError {
    /// bad count argument
    GetVehicleAccidentDataBadCount,
    /// accident data doesn't exist
    GetVehicleAccidentDataNotExist,
  }

  /// Configure the pallet by specifying the parameters and types on which it depends.
  #[pallet::config]
  pub trait Config: frame_system::Config + pallet_sim_renault::Config {
    /// Because this pallet emits events, it depends on the runtime's definition of an event.
    type RuntimeEvent: From<Event<Self>> + IsType<<Self as frame_system::Config>::RuntimeEvent>;
  }

  #[pallet::pallet]
  #[pallet::generate_store(pub(super) trait Store)]
  // #[pallet::without_storage_info]
  pub struct Pallet<T>(_);

  /// List of accidents.
  /// Accident ID = Hash(vehicle ID + AccidentCount(vehicle ID) )
  /// (
```

```
48    ///    Accident ID => data_hash
49    /// )
50    #[pallet::storage]
51    pub type Accidents<T: Config> =
52    StorageMap<_, Blake2_128Concat, [u8; 32], [u8; 36], OptionQuery>;
53
54    /// List of accident count.
55    /// (
56    ///    vehicle ID => accident_count
57    /// )
58    #[pallet::storage]
59    pub type AccidentCount<T: Config> =
60    StorageMap<_, Blake2_128Concat, T::AccountId, u32, OptionQuery>;
61
62    #[pallet::event]
63    #[pallet::generate_deposit(pub(super) fn deposit_event)]
64    pub enum Event<T: Config> {
65      /// Event when a accident has been added to storage. AccidentStored(vehicle_id, count,
      data_hash) [AccidentStored, AccountId, u32, [u8; 32]]
66      AccidentStored(T::AccountId, u32, [u8; 36]),
67    }
68
69    // Errors inform users that something went wrong.
70    #[pallet::error]
71    pub enum Error<T> {
72      /// Vehicle is not in storage.
73      UnknownVehicle,
74      /// Vehicle is already in storage
75      AccidentAlreadyStored,
76      /// Vehicle ID and origin aren't match
77      VehicleNotMatchingOrigin,
78      /// Error vehicle status is false
79      VehicleStatusError,
80    }
81
82    #[pallet::hooks]
83    impl<T: Config> Hooks<BlockNumberFor<T>> for Pallet<T> {}
84
85    #[allow(unused_variables)]
86    #[pallet::call]
87    impl<T: Config> Pallet<T> {
88      /// Report an accident to Renault.
89      /// Dispatchable that allows to report an accident and store a data hash associated to the
       accident.
90      #[pallet::weight((0, Pays::No))]
91      pub fn report_accident(
92      origin: OriginFor<T>,
93      // vehicle_id: T::AccountId,
94      data_hash: [u8; 36],
95      ) -> DispatchResultWithPostInfo {
96        // if_std! {
97          //  println!("{:02x?}", data_hash);
98          // }
99        let vehicle_id = ensure_signed(origin)?;
100
101       //check if vehicle exists in pallet sim_renault
102       ensure!(
103       pallet_sim_renault::Pallet::<T>::is_vehicle(vehicle_id.clone()),
104       Error::<T>::UnknownVehicle
105       );
106
107       //get vehicle accident count
108       // let count: u32 = AccidentCount::get(&vehicle_id)?;
109       let count: u32 = match <AccidentCount<T>>::get(&vehicle_id) {
110         // Return an error if the value has not been set.
```

```
        None => 0,
        Some(val) => val,
      };

      //create key from vehicle_id and count
      let mut parts = Vec::new();
      parts.push(vehicle_id.encode());
      parts.push(count.to_le_bytes().to_vec());
      let accident_key: [u8; 32] = Self::create_composite_key(parts);

      // Verify that the specified data_hash has not already been stored.
      ensure!(
      !Accidents::<T>::contains_key(&accident_key),
      Error::<T>::AccidentAlreadyStored
      );

      // Store the data_hash.
      Accidents::<T>::insert(&accident_key, &data_hash);

      //inc vehicle accident count
      let next_count = count + 1;
      AccidentCount::<T>::insert(&vehicle_id, next_count);

      // Emit an event.
      Self::deposit_event(Event::AccidentStored(vehicle_id, count, data_hash));
      Ok(().into())
    }
  }

  impl<T: Config> Pallet<T> {
    pub fn create_composite_key(parts: Vec<Vec<u8>>) -> [u8; 32] {
      let concatenated = parts.iter().fold(Vec::new(), |mut res: Vec<u8>, new| {
        res.extend(new.as_slice());
        res
      });
      let mut hasher = Sha256::new();
      Digest::update(&mut hasher, concatenated.as_slice());
      hasher.finalize().into()
    }

    /// Return the data of vehicle
    // The existential deposit is not part of the pot so treasury account never gets deleted.
    pub fn get_accident_data(
    vehicle_id: T::AccountId,
    accident_count: u32,
    ) -> Result<[u8; 36], GetAccidentDataError> {
      if accident_count <= 0 {
        return Err(GetAccidentDataError::GetVehicleAccidentDataBadCount)
      } else {
        let new_accident_count = accident_count - 1;
        //create key from vehicle_id and count
        let mut parts = Vec::new();
        parts.push(vehicle_id.encode());
        parts.push(new_accident_count.to_le_bytes().to_vec());
        let accident_key: [u8; 32] = Self::create_composite_key(parts);

        match Accidents::<T>::get(accident_key) {
          Some(data) => Ok(data),
          None => Err(GetAccidentDataError::GetVehicleAccidentDataNotExist),
        }
      }
    }
  }

  /// Module representing pallet sim_renault_accident.
```

```
176   /// This is used to avoid importing the entire pallet or runtime
177   mod pallet_sim_insurance_accident {
178     use crate::*;
179     use codec::{Decode, Encode};
180     use frame_support::RuntimeDebug;
181
182     /// The encoded index correspondes to sim_renault_accident pallet configuration.
183     /// Ex: ReceiveData is the second pallet call
184     #[derive(Encode, Decode, RuntimeDebug)]
185     pub enum PalletSimInsuranceAccidentCall<T: Config> {
186       #[codec(index = 1)]
187       ReceiveData(T::AccountId, u32, [u8; 36]),
188     }
189
190     /// The encoded index correspondes to Insurance's Runtime module configuration.
191     /// Ex: PalletSimInsuranceAccident is fixed to the index 103. See the node
      construct_runtime! macro to view all indexes.
192     /// More about indexes here: https://substrate.stackexchange.com/a/1196/501
193     #[derive(Encode, Decode, RuntimeDebug)]
194     pub enum ParaChainCall<T: Config> {
195       #[codec(index = 103)]
196       PalletSimInsuranceAccident(PalletSimInsuranceAccidentCall<T>),
197     }
198   }
199 }
```

8.3 Client Applications

8.3.1 Ethereum Go Client Application

Code 8.8: WebSocket TCP Packets for the Go Client Application

```
reportfilename: ./report.xml
tcpflow: listening on lo
127.000.000.001.44142-127.000.000.001.08545: POST / HTTP/1.1
Host: localhost:8545
User-Agent: Go-http-client/1.1
Content-Length: 48
Accept: application/json
Content-Type: application/json
Accept-Encoding: gzip

{"jsonrpc":"2.0","id":1,"method":"eth_gasPrice"}
127.000.000.001.08545-127.000.000.001.44142: HTTP/1.1 200
Content-Type: application/json
Content-Length: 46

{"id":1,"jsonrpc":"2.0","result":"0x77359400"}
127.000.000.001.44142-127.000.000.001.08545: POST / HTTP/1.1
Host: localhost:8545
User-Agent: Go-http-client/1.1
Content-Length: 125
Accept: application/json
Content-Type: application/json
Accept-Encoding: gzip

{"jsonrpc":"2.0","id":2,"method":"eth_getTransactionCount","params":["0
    x0a461ecc38e6159eecc749da0c7c77416ccb073c", "pending"]}
127.000.000.001.08545-127.000.000.001.44142: HTTP/1.1 200
Content-Type: application/json
Content-Length: 41

{"id":2,"jsonrpc":"2.0","result":"0xfa2"}
127.000.000.001.44142-127.000.000.001.08545: POST / HTTP/1.1
Host: localhost:8545
User-Agent: Go-http-client/1.1
Content-Length: 300
Accept: application/json
Content-Type: application/json
Accept-Encoding: gzip

{"jsonrpc":"2.0","id":3,"method":"eth_sendRawTransaction","params":["
0xf86f820fa2847735940082520894d92373d4bb
00e93faf91fd127492763d289e948788016345785d8a000080820
a95a05a367b8e374ccd03adf4fa1ae49264b25d1e125da24dcb17
25196955e75173cda0340318430a3db765008e2245609c328908b
46b68ccc25edc72d366e560470924"]}
127.000.000.001.08545-127.000.000.001.44142: HTTP/1.1 200
Content-Type: application/json
Content-Length: 102

{"id":3,"jsonrpc":"2.0","result":"0
    xa425fca36225e4dc11b218fcac300a872a77439d347c63b2e83151741e2709d0"}
```

8.3.2 JavaScript Substrate-based Client Application

Code 8.9: JavaScript Substrate-based Client Application Code

```
const { ApiPromise, WsProvider, Keyring } = require('@polkadot/api');
async function sendReportAccidentExtrinsic() {
    // Connect to the local node
    //   const provider = new WsProvider('ws://localhost:9944');
    const provider = new WsProvider('ws://substrate-ws.unice.cust.tasfrance.com/');
    const api = await ApiPromise.create({ provider });

    // Create a keyring instance for sending transactions
    const keyring = new Keyring({ type: 'sr25519' });

    // Replace 'Alice' with the account that you want to use to send the extrinsic
    const alice = keyring.addFromUri('//Alice');

    // Define the dummy data hash (all "1"s)
    const dummyDataHash = "0
    x0101010101010101010101010101010101010101010101010101010101010101"

    try {
        // Send the extrinsic to report the accident
        const tx = api.tx.palletSimRenaultAccident.reportAccident(dummyDataHash);
        const txHash = await tx.signAndSend(alice);

        console.log(`Extrinsic sent with hash: ${txHash}`);
      } catch (error) {
        console.error('Error sending extrinsic:', error);
      }

    // Disconnect from the node
    await provider.disconnect();
  }
sendReportAccidentExtrinsic();
```

In the Javascript blockchain client targeting Substrate-based chains we observe that many messages are exchanged through the WebSocket:

1. Get current block hash
2. Get runtime version
3. Get system chain
4. Get system properties
5. Get RPC methods
6. Get state metadata
7. Subscribing to runtime version
8. Submit a transaction (extrinsic)

Code 8.10: WebSocket TCP Packets for the Substrate-based Client Application

```
0000   7b 22 69 64 22 3a 31 2c 22 6a 73 6f 6e 72 70 63   {"id":1,"jsonrpc
0010   22 3a 22 32 2e 30 22 2c 22 6d 65 74 68 6f 64 22   ":"2.0","method"
0020   3a 22 63 68 61 69 6e 5f 67 65 74 42 6c 6f 63 6b   :"chain_getBlock
0030   48 61 73 68 22 2c 22 70 61 72 61 6d 73 22 3a 5b   Hash","params":[
0040   30 5d 7d                                          0]}

0000   7b 22 69 64 22 3a 32 2c 22 6a 73 6f 6e 72 70 63   {"id":2,"jsonrpc
0010   22 3a 22 32 2e 30 22 2c 22 6d 65 74 68 6f 64 22   ":"2.0","method"
0020   3a 22 73 74 61 74 65 5f 67 65 74 52 75 6e 74 69   :"state_getRunti
0030   6d 65 56 65 72 73 69 6f 6e 22 2c 22 70 61 72 61   meVersion","para
0040   6d 73 22 3a 5b 5d 7d                              ms":[]}

0000   7b 22 69 64 22 3a 33 2c 22 6a 73 6f 6e 72 70 63   {"id":3,"jsonrpc
0010   22 3a 22 32 2e 30 22 2c 22 6d 65 74 68 6f 64 22   ":"2.0","method"
0020   3a 22 73 79 73 74 65 6d 5f 63 68 61 69 6e 22 2c   :"system_chain",
```

```
0030   22 70 61 72 61 6d 73 22 3a 5b 5d 7d                "params":[]}

0000   7b 22 69 64 22 3a 34 2c 22 6a 73 6f 6e 72 70 63   {"id":4,"jsonrpc
0010   22 3a 22 32 2e 30 22 2c 22 6d 65 74 68 6f 64 22   ":"2.0","method"
0020   3a 22 73 79 73 74 65 6d 5f 70 72 6f 70 65 72 74   :"system_propert
0030   69 65 73 22 2c 22 70 61 72 61 6d 73 22 3a 5b 5d   ies","params":[]
0040   7d                                                 }

0000   7b 22 69 64 22 3a 35 2c 22 6a 73 6f 6e 72 70 63   {"id":5,"jsonrpc
0010   22 3a 22 32 2e 30 22 2c 22 6d 65 74 68 6f 64 22   ":"2.0","method"
0020   3a 22 72 70 63 5f 6d 65 74 68 6f 64 73 22 2c 22   :"rpc_methods","
0030   70 61 72 61 6d 73 22 3a 5b 5d 7d                  params":[]}

0000   7b 22 69 64 22 3a 36 2c 22 6a 73 6f 6e 72 70 63   {"id":6,"jsonrpc
0010   22 3a 22 32 2e 30 22 2c 22 6d 65 74 68 6f 64 22   ":"2.0","method"
0020   3a 22 73 74 61 74 65 5f 67 65 74 4d 65 74 61 64   :"state_getMetad
0030   61 74 61 22 2c 22 70 61 72 61 6d 73 22 3a 5b 5d   ata","params":[]
0040   7d                                                 }

0000   00 42 38 f1 e6 d1 62 b3 f6 42 36 b6 08 00 45 28   .B8...b..B6...E(
0010   00 9c 71 be 40 00 fd 06 0a 72 b9 34 20 04 c0 a8   ..q.@....r.4 ...
0020   67 22 00 50 e0 aa d5 e6 e9 15 6a a6 f2 cb 80 18   g".P......j.....
0030   07 69 37 78 00 00 01 01 08 0a 60 68 a0 a5 2d 1d   .i7x......`h..-.
0040   64 a9 81 66 7b 22 6a 73 6f 6e 72 70 63 22 3a 22   d..f{"jsonrpc":"
0050   32 2e 30 22 2c 22 72 65 73 75 6c 74 22 3a 22 30   2.0","result":"0
0060   78 32 63 32 39 65 38 30 63 37 36 30 36 36 66 34   x2c29e80c76066f4
0070   30 37 63 30 38 33 35 65 61 65 37 62 63 31 31 31   07c0835eae7bc111
0080   61 32 62 37 62 35 37 31 62 31 63 62 31 33 38 34   a2b7b571b1cb1384
0090   38 65 65 64 38 61 39 61 64 62 37 38 34 63 32 32   8eed8a9adb784c22
00a0   63 22 2c 22 69 64 22 3a 31 7d                     c","id":1}

0000   81 7e 08 46 7b 22 6a 73 6f 6e 72 70 63 22 3a 22   .~.F{"jsonrpc":"
0010   32 2e 30 22 2c 22 72 65 73 75 6c 74 22 3a 7b 22   2.0","result":{"
0020   6d 65 74 68 6f 64 73 22 3a 5b 22 61 63 63 6f 75   methods":["accou
0030   6e 74 5f 6e 65 78 74 49 6e 64 65 78 22 2c 22 61   nt_nextIndex","a
0040   75 74 68 6f 72 5f 68 61 73 4b 65 79 22 2c 22 61   uthor_hasKey","a
0050   75 74 68 6f 72 5f 68 61 73 53 65 73 73 69 6f 6e   uthor_hasSession
0060   4b 65 79 73 22 2c 22 61 75 74 68 6f 72 5f 69 6e   Keys","author_in
                ...
                ...
                ...
0810   6e 73 61 63 74 69 6f 6e 5f 75 6e 73 74 61 62 6c   nsaction_unstabl
0820   65 5f 75 6e 77 61 74 63 68 22 2c 22 75 6e 73 75   e_unwatch","unsu
0830   62 73 63 72 69 62 65 5f 6e 65 77 48 65 61 64 22   bscribe_newHead"
0840   5d 7d 2c 22 69 64 22 3a 35 7d                     ]},"id":5}

0000   81 7f 00 00 00 00 00 01 33 e2 7b 22 6a 73 6f 6e   ........3.{"json
0010   72 70 63 22 3a 22 32 2e 30 22 2c 22 72 65 73 75   rpc":"2.0","resu
0020   6c 74 22 3a 22 30 78 36 64 36 35 37 34 36 31 30   lt":"0x6d6574610
0030   65 32 35 30 32 30 30 30 63 31 63 37 33 37 30 35   e2502000c1c73705
0040   66 36 33 36 66 37 32 36 35 31 38 36 33 37 32 37   f636f72651863727
0050   39 37 30 37 34 36 66 32 63 34 31 36 33 36 33 36   970746f2c4163636
                ...
                ...
                ...
133b0   32 36 37 36 35 35 34 37 32 36 31 36 65 37 33 36   267655472616e736
133c0   31 36 33 37 34 36 39 36 66 36 65 35 30 36 31 37   16374696f6e50617
133d0   39 36 64 36 35 36 65 37 34 31 64 30 32 39 63 32   96d656e741d029c2
133e0   31 30 32 22 2c 22 69 64 22 3a 36 7d               102","id":6}

0000   7b 22 69 64 22 3a 37 2c 22 6a 73 6f 6e 72 70 63   {"id":7,"jsonrpc
0010   22 3a 22 32 2e 30 22 2c 22 6d 65 74 68 6f 64 22   ":"2.0","method"
0020   3a 22 73 74 61 74 65 5f 73 75 62 73 63 72 69 62   :"state_subscrib
0030   65 52 75 6e 74 69 6d 65 56 65 72 73 69 6f 6e 22   eRuntimeVersion"
0040   2c 22 70 61 72 61 6d 73 22 3a 5b 5d 7d            ,"params":[]}
```

```
0000   7b 22 69 64 22 3a 38 2c 22 6a 73 6f 6e 72 70 63   {"id":8,"jsonrpc
0010   22 3a 22 32 2e 30 22 2c 22 6d 65 74 68 6f 64 22   ":"2.0","method"
0020   3a 22 63 68 61 69 6e 5f 67 65 74 48 65 61 64 65   :"chain_getHeade
0030   72 22 2c 22 70 61 72 61 6d 73 22 3a 5b 5d 7d      r","params":[]}

0000   7b 22 69 64 22 3a 31 33 2c 22 6a 73 6f 6e 72 70   {"id":13,"jsonrp
0010   63 22 3a 22 32 2e 30 22 2c 22 6d 65 74 68 6f 64   c":"2.0","method
0020   22 3a 22 73 74 61 74 65 5f 67 65 74 52 75 6e 74   ":"state_getRunt
0030   69 6d 65 56 65 72 73 69 6f 6e 22 2c 22 70 61 72   imeVersion","par
0040   61 6d 73 22 3a 5b 22 30 78 30 62 38 38 38 62 38   ams":["0x0b888b8
0050   62 38 62 35 63 35 62 64 30 62 30 30 32 32 63 36   b8b5c5bd0b0022c6
0060   37 32 37 30 65 37 36 37 37 34 64 30 63 38 39 64   7270e76774d0c89d
0070   66 33 31 65 30 39 62 31 38 38 35 63 30 38 61 31   f31e09b1885c08a1
0080   32 38 62 32 38 35 32 31 63 22 5d 7d               28b28521c"]}

0000   7b 22 69 64 22 3a 31 34 2c 22 6a 73 6f 6e 72 70   {"id":14,"jsonrp
0010   63 22 3a 22 32 2e 30 22 2c 22 6d 65 74 68 6f 64   c":"2.0","method
0020   22 3a 22 73 74 61 74 65 5f 67 65 74 52 75 6e 74   ":"state_getRunt
0030   69 6d 65 56 65 72 73 69 6f 6e 22 2c 22 70 61 72   imeVersion","par
0040   61 6d 73 22 3a 5b 22 30 78 39 66 61 36 62 65 31   ams":["0x9fa6be1
0050   39 31 36 34 66 31 31 38 36 63 30 38 33 62 30 39   9164f1186c083b09
0060   31 33 63 32 34 61 62 64 66 34 33 31 34 65 32 63   13c24abdf4314e2c
0070   65 65 64 32 37 38 62 36 38 37 31 32 64 66 63 61   eed278b68712dfca
0080   34 38 65 36 35 38 30 63 66 22 5d 7d               48e6580cf"]}

0000   7b 22 69 64 22 3a 31 37 2c 22 6a 73 6f 6e 72 70   {"id":17,"jsonrp
0010   63 22 3a 22 32 2e 30 22 2c 22 6d 65 74 68 6f 64   c":"2.0","method
0020   22 3a 22 61 75 74 68 6f 72 5f 73 75 62 6d 69 74   ":"author_submit
0030   45 78 74 72 69 6e 73 69 63 22 2c 22 70 61 72 61   Extrinsic","para
0040   6d 73 22 3a 5b 22 30 78 33 35 30 32 38 34 30 30   ms":["0x35028400
0050   64 34 33 35 39 33 63 37 31 35 66 64 64 33 31 63   d43593c715fdd31c
0060   36 31 31 34 31 61 62 64 30 34 61 39 39 66 64 36   61141abd04a99fd6
0070   38 32 32 63 38 35 35 38 38 35 34 63 63 64 65 33   822c8558854ccde3
0080   39 61 35 36 38 34 65 37 61 35 36 64 61 32 37 64   9a5684e7a56da27d
0090   30 31 61 30 63 65 35 63 37 36 32 35 32 36 33 34   01a0ce5c76252634
00a0   66 35 39 33 66 63 34 33 62 34 64 36 61 33 63 34   f593fc43b4d6a3c4
00b0   61 36 66 33 37 37 32 34 62 66 33 39 66 62 32 35   a6f37724bf39fb25
00c0   32 35 61 64 65 66 39 64 61 63 61 36 32 37 39 32   25adef9daca62792
00d0   34 64 36 37 34 35 33 66 31 32 64 32 35 66 62 32   4d67453f12d25fb2
00e0   35 64 30 63 62 32 30 63 35 65 38 36 33 61 64 35   5d0cb20c5e863ad5
00f0   65 33 35 38 32 37 38 36 66 31 61 34 66 64 39 38   e3582786f1a4fd98
0100   38 64 32 34 30 33 62 37 39 34 38 30 37 62 65 32   8d2403b794807be2
0110   38 39 38 35 30 31 31 63 30 30 30 61 30 30 30 31   8985011c000a0001
0120   30 31 30 31 30 31 30 31 30 31 30 31 30 31 30 31   0101010101010101
0130   30 31 30 31 30 31 30 31 30 31 30 31 30 31 30 31   0101010101010101
0140   30 31 30 31 30 31 30 31 30 31 30 31 30 31 30 31   0101010101010101
0150   30 31 30 31 30 31 30 31 30 31 30 31 30 31 30 31   0101010101010101
0160   30 31 30 31 30 31 22 5d 7d                        010101"]}

0000   00 42 38 f1 e6 d1 62 b3 f6 42 36 b6 08 00 45 28   .B8...b..B6...E(
0010   00 9d 9d 23 40 00 fd 06 df 0b b9 34 20 04 c0 a8   ...#@......4 ...
0020   67 22 00 50 e0 aa d5 e8 3a 78 6a a6 f9 6d 80 18   g".P....:xj..m..
0030   07 8f a9 7a 00 00 01 01 08 0a 60 68 a3 66 2d 1d   ...z......`h.f-.
0040   67 59 81 67 7b 22 6a 73 6f 6e 72 70 63 22 3a 22   gY.g{"jsonrpc":"
0050   32 2e 30 22 2c 22 72 65 73 75 6c 74 22 3a 22 30   2.0","result":"0
0060   78 61 30 36 38 63 32 32 39 33 37 32 39 31 33 62   xa068c229372913b
0070   33 39 30 38 63 61 62 65 33 36 63 63 63 37 37 30   3908cabe36ccc770
0080   63 33 32 65 34 33 37 32 39 36 64 66 33 64 39 64   c32e437296df3d9d
0090   39 63 62 62 32 31 32 64 39 39 34 63 38 33 31 35   9cbb212d994c8315
00a0   38 22 2c 22 69 64 22 3a 31 37 7d                  8","id":17}
```

8.3.3 Rust Substrate-based Client Application

Code 8.11: Rust Substrate-based Client Application Code

```rust
#[macro_use]
extern crate clap;

mod common;
use rppal::gpio::Gpio;
use rppal::gpio::OutputPin;
use clap::App;
use keyring::AccountKeyring;
use sp_core::crypto::Pair;
use sp_core::sr25519;
use sp_core::H256;
use substrate_api_client::rpc::WsRpcClient;
use substrate_api_client::{
  compose_extrinsic, compose_extrinsic_offline, Api, ApiResult, GenericAddress,
  UncheckedExtrinsicV4, XtStatus,
};

const PIN_1: u8 = 6;
const PIN_2: u8 = 5;

fn main() {
  println!("Start program");
  let mut pin1 = match Gpio::new() {
    Ok(val1) => match val1.get(PIN_1) {
      Ok(val2) => val2.into_output(),
      Err(e) => panic!("Error GPIO {}", e),
    },
    Err(e) => panic!("Error GPIO {}", e),
  };
  let mut pin2 = match Gpio::new() {
    Ok(val1) => match val1.get(PIN_2) {
      Ok(val2) => val2.into_output(),
      Err(e) => panic!("Error GPIO {}", e),
    },
    Err(e) => panic!("Error GPIO {}", e),
  };
  pin1.set_low();
  pin2.set_low();

  env_logger::init();
  let url = get_node_url_from_cli();

  println!("Start set_storage");

  let from = AccountKeyring::Alice.pair();
  let client = WsRpcClient::new(&url);
  let init_api = Api::<sr25519::Pair, _>::new(client).map(|api| api.set_signer(from));

  match init_api {
    Ok(_) => println!("API OK"),
    Err(e) => return eprintln!("API ERROR: {}", e),
  };
  let api = init_api.unwrap();

  println!("end init set_storage");

  pin1.set_low();
  pin2.set_high();

  let key = "foo"; //0x666F6F
  let val = "bar"; //0x626172
  println!(
  "Set key: {:X?} <=> {:X?}",
  key.clone(),
```

```
  key.clone().as_bytes().to_vec()
  );
  println!(
  "Set value: {:X?} <=> {:X?}",
  val.clone(),
  val.clone().as_bytes().to_vec()
  );

  let res = set_storage_value(
  api.clone(),
  key.clone().as_bytes().to_vec(),
  val.clone().as_bytes().to_vec(),
  &mut pin1,
  &mut pin2,
  );
  println!("End send Tx");
  match res {
    Err(e) => eprintln!("ERROR: {}", e),
    Ok(blockh) => println!("[+] Transaction got included in block {:?}", blockh),
  }

  pin1.set_high();
  pin2.set_high();
}

/// Set storage data value with a given key
/// # Arguments
/// * `api` - Api endpoint
/// * `key` - Storage key
/// * `value` - Storage value
fn set_storage_value(
api: Api<sr25519::Pair, WsRpcClient>,
key: Vec<u8>,
value: Vec<u8>,
pin1: &mut OutputPin,
pin2: &mut OutputPin,
) -> ApiResult<Option<H256>> {
  // set the storage
  // define the recipient
  pin1.set_high();
  pin2.set_low();
  println!("Create extrinsic");
  let xt: UncheckedExtrinsicV4<_> =
  compose_extrinsic!(api.clone(), "KeyvalueModule", "store", key, value);
  println!("End create extrinsic");
  pin1.set_low();
  pin2.set_high();
  println!("Send Tx");
  api.send_extrinsic(xt.hex_encode(), XtStatus::InBlock)
}

pub fn get_node_url_from_cli() -> String {
  let yml = load_yaml!("../src/cli.yml");
  let matches = App::from_yaml(yml).get_matches();

  let node_ip = matches
  .value_of("node-server")
  .unwrap_or("ws://10.212.115.235");
  let node_port = matches.value_of("node-port").unwrap_or("9944");
  let url = format!("{}:{}", node_ip, node_port);
  println!("Interacting with node on {}", url);
  url
}
```