Assessing Blockchain Nodes on IoT Devices

Can Ethereum nodes perform on IoT embedded devices?

Master Thesis

Ryan SIOW

University of Fribourg

October 2021

Kill the middleman of necessity,
push power to the edges and build systems
that are equally fair for the least amongst us.
— *Charles Hoskinson*

Acknowledgements

I would like to thank Prof. Hans-Georg Fill for giving me an interesting and exciting project to work on, as I am interested in blockchain technology. I would also like to thank Dr. Härer Felix and assistant Simon Curty for their time and valuable help throughout my thesis. Finally, I would also like to express my gratitude to my family and friends for their support.

Fribourg, 27 October 2021 R. S.

Abstract

At the time of writing, blockchain technologies, also known as a distributed ledger, has gained massive momentum thanks to Bitcoin's cryptocurrency success these past few years. Unlike traditional database systems where transactions happen in a trusted environment, blockchain does not require third-party control of data. Moreover, it does not allow the update or deletion of data stored on the blockchain, thus preventing data tampering. Transactions are time-stamped, validated, kept, and synchronized in a decentralized manner among the participants of the blockchain network. With the incorporation of smart contracts, blockchains like Ethereum can go beyond the definition of just a digital distributed ledger, thus enabling decentralized apps.

Its wide range of applications has led many people to adopt this novel technology in various fields of industry such as finance, supply chain, healthcare, or the Internet of Things, to address mainly trust and security issues. In IoT, for example, which can be pictured as a network of connected embedded machines, devices come from various manufacturers worldwide with different law regulations regarding privacy and security, making it hard for IoT adopters to trust these partners. Although IoT helps contribute to automation and connectivity, the increase of IoT capabilities can lead to security issues.

In the context of the rise of blockchain and its promising technology, this work focuses on blockchain integration in IoT systems. More specifically, assessing blockchain nodes in permissioned or public Ethereum blockchain on low-powered IoT embedded devices like the Raspberry Pi and the EV3 Mindstorm. To this end, we collected performance-related metrics to analyze how well these nodes can run on such devices. Additionally, we designed a public blockchain-integrated IoT system using the RPI as an Ethereum node and the EV3 as a sensing device to demonstrate the feasibility of such a use case.

Key words: blockchain, private/permissioned blockchain, public blockchain, Ethereum, smart contract, dApp, IoT, RPI, EV3, performance.

Supervised by:
Dr. Hans-Georg Fill, Digitalization and Information Systems Group, University of Fribourg
Assisted by:
Dr. Felix Härer, Digitalization and Information Systems Group, University of Fribourg
Simon Curty, Digitalization and Information Systems Group, University of Fribourg

Contents

List of Figures

List of Tables

Listings

1 Introduction

In the first chapter, the research topic and motivation of the thesis are explained, and the related problems are revealed. Based on the motivation and problems, the research questions are then derived and formulated. Subsequently, the research methodologies used to answer the research questions and the structure of the work are described.

1.1 Motivation

On the one hand, blockchain has become increasingly popular over the past decade. It is a technology that connects individuals or entities such as companies directly, commonly known as peer-to-peer communication. Since the creation of the internet, people have shared information such as sending emails, posting on social media, and sharing documents. Blockchain as technology takes the connectivity of the internet beyond its original application. It offers users the internet of value, a concept originally proposed by Ripple [Chase and MacBrough, 2018]. Participants of a blockchain can now exchange value on a peer-to-peer basis in contrast to exchanging only information. Although it is often associated with Bitcoin and other "crypto assets," this technology enables both complex programming (such as smart contracts) and applications (such as decentralized apps or dApps [Antonopoulos and Wood, 2018]). The former gives a great potential to quickly and efficiently undertake various transactions and transference of rights and property. For example, the ownership of a physical object like a car or a digital object like art, music, or collectibles can be easily and cost-effectively transferred without the need of a third party. This trusted peer-to-peer exchange and the removal of the "middle man" help drive what some have come to refer to as "Web 3.0" [Werner Vermaak, 2021].
The "Internet of Things" (or IoT), on the other hand, is gaining in popularity as well, as the ever-growing network of "smart devices" helps to contribute to automation and connectivity of the latter. The Internet of Things must employ various technologies stacks to ensure automatic data transfer, analysis, and response between multiple devices. For example, without Big Data, Machine Learning, and Artificial Intelligence, automation would be impossible, while wireless communication technologies and cloud computing greatly facilitate connectivity.
Adopting IoT may offer a wide variety of benefits for organizations, and the resulting data

of such systems can provide them with valuable insights. For example, it can improve staff productivity and reduce human labor by doing mundane tasks automatically. It can enable efficient operation management thanks to automated control over multiple operation areas such as inventory management or shipping tracking, or help to use intelligently resources and assets thanks to automated scheduling and monitoring with the assistance of interconnected sensors.
However, the intention of using IoT as part of an organization's automation may expose some risks related to security as having multiple connected devices exposes several backdoors issues. Today, IoT includes buildings, cars, and even coffee machines embedded with sensors, software, and network connectivity. Because communications of these devices are operated in a centralized manner, hackers can easily gain access to them. According to this paper about securing IoT communication using Blockchain technology [Fakhri and Mutijarsa, 2018], blockchain has the potential to address these critical security concerns as all of the information and data are decentralized. As IoT capabilities increase, security issues regarding IoT become more critical.
Various implementations of Blockchain technology exist that are integrated into IoT systems to help secure its data, and network [Atasen and Ustunel, 2019, Özyılmaz and Yurdakul, 2017]. IoT systems often imply using low-powered devices such as Raspberry Pis or Arduino microcontrollers. Some research explored IoT systems with Blockchain nodes directly implemented in those low-powered devices instead of a standard computer [Choi et al., 2018, Gong et al., 2020, Atasen and Ustunel, 2019].
With this merging of Blockchain and IoT and security concerns regarding data transfer within an IoT system, the question of using blockchain to protect IoT endpoint devices becomes more apparent. As this technology is being adopted, one could ask whether using blockchain on a Raspberry Pi or any other low-powered device is more efficient than using it on a PC, for example. Several comparisons exist, but none of them address the difference between hardware. For example, some studies [Sanju et al., 2018] looked at the energy consumption of Blockchain platforms [Sankaran et al., 2018] or tried to realize a realistic energy profiling of Blochains for securing IoT. Other studies questioned the performance of Blockchain platforms in varying workloads [Pongnumkul et al., 2017], and even a framework (Blockbench) was built to analyze private Blockchain performance [Dinh et al., 2017].

Following these considerations, three research questions can be formulated:

- **RQ1:** How can software for public and permissioned blockchain nodes be run on Raspberry Pi and EV3 hardware?

- **RQ2:** How can performance tests for blockchains be established using synthetic data?

- **RQ3:** How does the performance of blockchain nodes differ on the investigated hardware devices?

1.2 Methodology (and Structure)

The formulated research questions above seek to see if blockchain nodes can be run in IoT edge devices like Raspberry Pi or EV3 and investigate how well they perform. This can be translated as an assessment of a novel distributed technology into IoT and studying its performance in various devices. As blockchain technology is still in its infancy stage, assessing its feasibility and performance when integrated into IoT is an important matter as many see blockchain as a solution to IoT's security problems.
In an attempt to provide an answer to our research questions, we will use a technique for experimental design and measurement for computer system performance analysis as described in [Jain, 1991]. The performance evaluation steps (marked with the related research question) are as follows:

1. State the goals of the study and define system boundaries *(Ch 1, 2, 3)*
2. List (software) system services and possible outcomes *(Chapter 2, 3, 4, RQ1)*
3. List (hardware) system and workload parameters *(Chapter 4, RQ1)*
4. Select performance metrics *(Chapter 4, RQ2)*
5. Select the workload *(Chapter 4, RQ2)*
6. Design the experiments *(Chapter 4)*
7. Analyze and interpret the data *(Chapter 5, RQ3)*
8. Present the results *(Chapter 5, RQ3)*

In our case, the system refers to blockchain IoT. In the first step, the goal of the study is enunciated by the research questions. System boundaries are elaborated through the following two chapters, which introduce blockchain technology as well as its integration in IoT. To address these two subjects, we will do a literature review to expose in the chapter 2 the foundations of blockchain and its basic principles. Then, related works in blockchain IoT integration are mentioned in chapter 3 to describe the benefits and challenges of integrating such a resource-demanding technology in IoT. The state of the art of existing architectures is then represented to show what is feasible.
Once the architecture for our use case is identified, we will try to experiment with various software implementations on a Raspberry Pi and EV3 in chapter 4 as an attempt to answer RQ1 and, respectively, steps 2 and 3. To address RQ2, we will select the metrics as well as a structure of blockchain (custom private blockchain & public blockchain) that will consist of the workload to run on our selected IoT edge devices and design a scenario to test the performance of our implemented blockchain IoT system.
Chapter 5 will be dedicated to interpreting and analyzing the collected data and consequently will answer RQ3.

2 Background

In the first instance, this chapter will cover the foundation and basics of blockchain, as the growing interest in this new and promising technology is set as a context. Secondly, we dive to some extent into a particular blockchain called Ethereum. Concepts such as dApp, EVM, nodes, and consensus mechanism are explained.

2.1 The Rise of Blockchain and its Foundation

At the time of writing, Bitcoin's cryptocurrency [Nakamoto, 2008] success these past few years helped blockchain technologies to gain massive momentum. Although initially developed as part of Bitcoin, the blockchain arouses popular interest as many sectors, and industries represent significant areas for blockchain application such as finance [Jaoude and Saade, 2019], healthcare [McGhin et al., 2019, Mettler, 2016], governmental voting [Hjalmarsson et al., 2018] and Internet of Things [Jaoude and Saade, 2019, Choi et al., 2018, Atasen and Ustunel, 2019]. This innovative technology is much anticipated to influence future industries by creating an opportunity to enhance business processes as well as building trust in records management and data sharing.

As opposed to traditional database systems where transactions happen in a trusted environment [Zubi, 2009], blockchain does not require third-party control of data. It does not assume that participants, also known as nodes, trust each other as trusted intermediaries can be hard to find. Furthermore, it does not have the ability to update or delete data. Transactions are time-stamped, validated, kept, and synchronized in a decentralized manner in the sense that all nodes of a blockchain network agree on an ordered set of blocks, each containing multiple transactions and replicated across multiple users. This distributed ledger can be viewed as a log of ordered transactions that ensures security and trust. As promising as it might sound, this technology is already big and powerful enough to bring significant changes, emphasizing the paradigm, among others, of the Internet of Things and artificial intelligence.

2.1.1 What is Blockchain and its characteristics

One of the very first reasons to use blockchain was to avoid the double-spending problem that Bitcoin solved. Nowadays, it is used for various purposes, such as IoT in this study. Blockchain happens in a peer-to-peer (P2P) network and utilizes the principle of asymmetric cryptography to sign transactions. A detailed description of its inner working is out of the scope of this paper, but the interested reader can find further details in [Schneier and Sutherland, 1995]. In this network, a community of participants, also known as nodes, validate tamper-resistant transactions with consistency in a decentralized way by recording the latter's data in a block. The new block is linked to a chain of ordered blocks, hence the term "blockchain". Similar to a linked list in data structures where the previous element of the list can be accessible thanks to a pointer, each block is connected to the previous block with a hash from the latter. As shown in Figure 2.1, a block is composed of a header and a body in which the transaction data is stored. The header contains several indispensable components:

- **a hash of the previous block:** allows to detect of any tampering on the previous block,
- **a timestamp:** is used to record the time when a new block is created,
- **a nonce:** is used during the creation and verification of a block,
- **a Merkle tree root** to verify all the transactions in the block at once.

The Merkle tree is a binary tree with each leaf node labeled with the hash of one transaction stored in the block body and the non-leaf nodes labeled with the concatenation of the hash of its child nodes [Liang, 2020]. Instead of verifying all the transactions in the block separately, the Merkle root simplifies the verification process by comparing the Merkle root only.

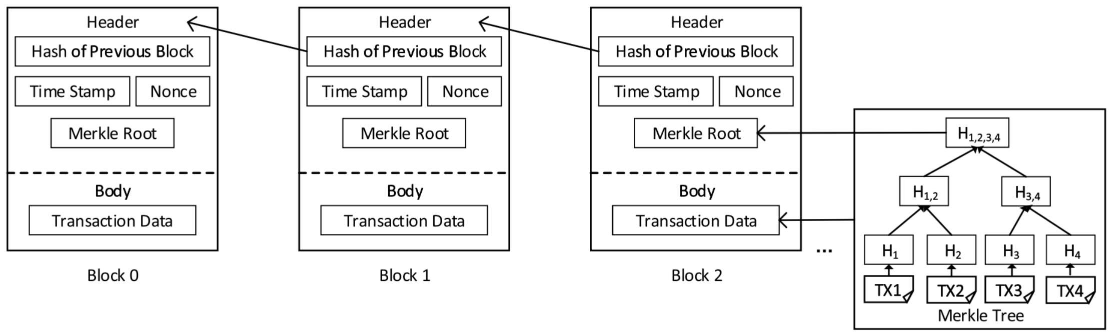


Figure 2.1 – The structure of a Blockchain [Liang, 2020].

Regarding the nodes, they usually come in three types: light nodes, full nodes, and miners/-validators. The latter store the entire blockchain data and compete with great computational power for block validation in order to earn token rewards. Full nodes also store the complete ledger and are mainly responsible for the broadcast and verification of the transactions. Light nodes, on the other hand, synchronize only the block header and request everything else.

Depending on the authorization and availability of such data, blockchains are categorized into public/permissionless and private/permissioned blockchains.
In public/permissionless, all the nodes in the network can join and verify the blockchain without the approval of third parties. Participants maintain a local replica of the blockchain, and depending on the type of node (miners/validators), they can publish a new block. Public blockchains are considered fully distributed as all the nodes in the network are granted the authority to maintain a ledger. Such a system is economic incentive like Bitcoin, Ethereum [Ethereum, 2021], or Litecoin [Litecoin, 2021].
In private/Permissioned, authorized nodes are given access to perform transactions, carry out smart contracts (explained in subsection 2.2.1) or participate in working on publishing new blocks. A single owner/organization usually maintains the network (in contrast with consortium blockchain, where there are several owners/organizations). Hyperledger-Fabric [Hyperledger Fabric's About, 2021] and Ripple [Ripple's About, 2021] are good examples of permissioned blockchains.
It can also be classified between ledgers that are aimed exclusively at tracking digital assets (e.g., Bitcoin or Litecoin) and those that run certain logic enabled thanks to smart contracts (e.g., Ethereum or Avalanche). The different types of blockchains are illustrated in Figure 2.2, inspired by a paper reviewing the use of blockchain for the internet of things [Fernandez-Carames and Fraga-Lamas, 2018].

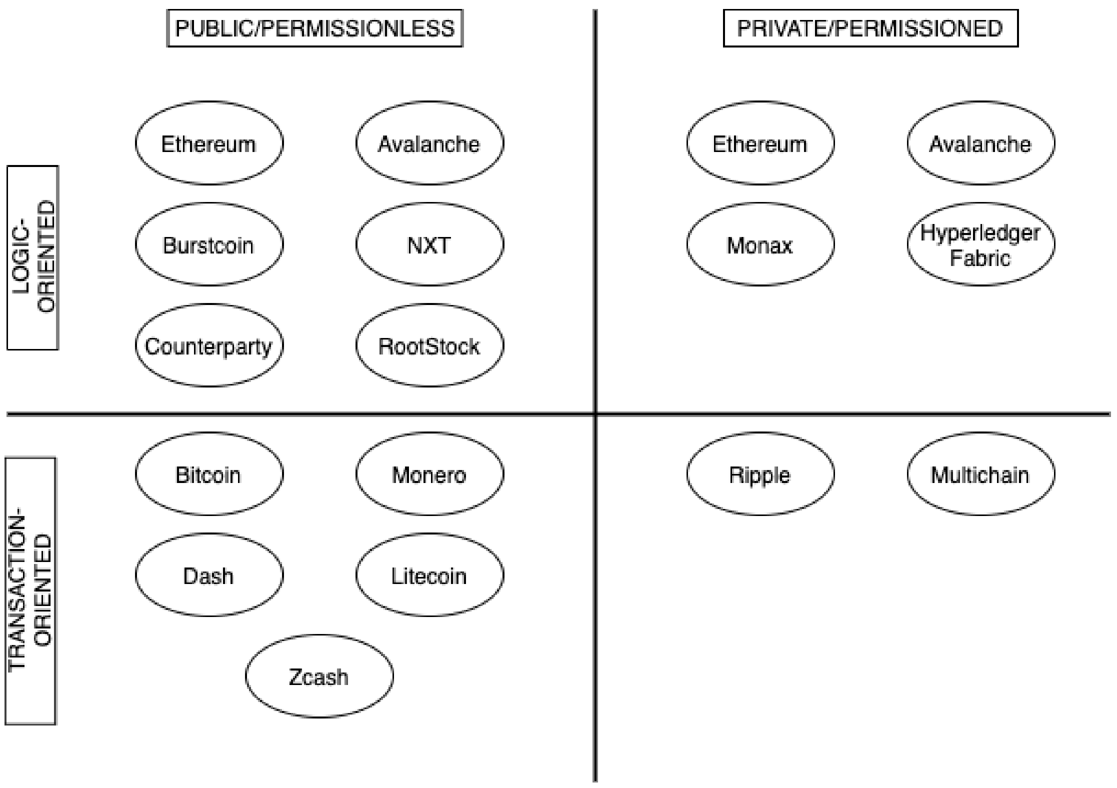


Figure 2.2 – Blockchain taxonomy and practical examples.

Despite the differences between blockchains, they all aim to share information with security and transparency amongst all parties that can access it. In this, Bitcoin and Ethereum are similar. The latter goes beyond by enabling smart contract, a concept introduced by Nick Szabo [Szabo, 1997]. With this innovation, developers can write programs that will be automatically executed as part of transaction validation on blockchains. In other words, a system with smart contracts implemented can be used as a computing platform. Ethereum is the most well-known and wildly used ledger platform in this field. Smart contracts enable the creation and use of distributed applications, more commonly known as dApps (explained in subsection 2.2.2) that run without a central server in contrast to today's classical application.

2.1.2 A Sens of Trust

The innovation of blockchain would not have been able to grow to such an extent without two major features: trust and decentralization.
With the removal of any third parties in such systems, members of a blockchain network have to trust that a majority of the power held in the network belongs to participants who share similar values. As third parties are not needed to verify transactions, the amount of trust required from any single actor in the system is minimized. Trust is, in a sense, distributed among participants [Preethi Kasireddy, 2021] thanks to consensus protocols such as proof-of-work (PoW) or proof-of-stake (PoS) that incentivize them to cooperate. In the former, a miner will have to provide proof to a specific mathematical challenge to have the right to update the blockchain. This proof usually requires enormous computing power to produce but is easily validated.
Such a mechanism helps all parties in the system to maintain data consistency and to reach a consensus on what the truth is. Integrity is then guaranteed if the collaborations between the network's stakeholders are executed as intended. Blockchain is also responsible as roles and responsibilities are determined in advance and reliable when contracts are executed as coded. Mainly used for secure and transparent transactions, the distributed ledger technology might reduce our dependency on a central entity and help to enforce decentralized transactions in an approach similar to the way the internet has reshaped social relationships.
Such a system that executes transactions reliably and its integrity by following a consensus protocol can allow tamper-proof coordination of activities giving the possibility to distributed ledgers to be the backbone of the Internet of Things.

2.2 Intro to Ethereum

Created by Vitalik Buterin, Ethereum is the most widely used blockchain in the field of smart contract. The use of the latter helps Ethereum to go beyond Bitcoin's limitations [Ethereum White Paper, 2021], and allows custom business logic to be used for new applications. For ease of understanding, let us see Ethereum as a distributed state machine instead of a distributed ledger [Ethereum Concepts, 2021]. This sophisticated analogy helps us un-

derstand that Ethereum's state is like a large data structure that holds accounts and balances and a state machine controlled by the Ethereum virtual machine (EVM), which can change depending on the block we look at, and which can process machine code. The main network is public, and anyone can download the software to run on their machines. Participants will need Ether, a digital currency, to run the software. Although the main platform is a public/permissionless blockchain network, developers can download and configure a private version of the ledger as the software is open-source.
Essential to this study, main concepts and features [Ethereum Concepts, 2021] of Ethereum such as decentralized application (dApp), the essential elements (accounts, transactions, consensus mechanism) and their implication in the blockchain, the incentive mechanism, and the Ethereum virtual machine (EVM) are explained.

2.2.1 What is a Smart Contract?

A smart contract is a program that self-executes when conditions are met. These defined terms can be seen as agreements between parties. As an analogy, vending machines specify and enforce agreements to give various items for certain payments and do not require humans to handle the operation. The concept of smart contract was first introduced by [Szabo, 1997] and predates blockchain technology:

> "A smart contract is a computerized transaction protocol that executes the terms of a contract. The general objectives of smart contract design are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitration and enforcement costs, and other transaction costs."
>
> - Nick Szabo, 1994

In the case of decentralized ledger technology such as Ethereum, smart contracts reside in the blockchain, which allows a decentralized use of it and enables the creation of dApps. Their primary goal is to reduce the chances of any fraudulent activity, the need for intermediates, or any other type of arbitration.
Smart contracts are written in high-level languages like Solidity [solidity, 2021], or Vyper [vyper, 2021]. Before deploying on the Ethereum blockchain, the corresponding source code is compiled to the equivalent EVM virtual machine instructions. The most used language is Solidity, a Javascript-like language designed to be easily adopted by new developers. Listing 2.1 is an example of a source code for a simple program. It maintains a counter variable that anyone can increment, but only the author of the smart contract can update the counter to a random value.

Listing 2.1 – Example of a Solidity source code.

```
pragma solidity >=0.4.22 <0.9.0;

contract Counter {
        uint    counter;
        address author;

        function Counter() public {
                counter = 0;
                author  = msg.sender;
        }
        function increment() public {
                counter += 1;
        }
        function set(uint new_value) public {
                if (msg.sender == author) {
                        counter = new_value;
                }
        }
        function get_counter() public constant returns (uint) {
                return counter;
        }
}
```

2.2.2 What is a dApp?

We have seen that smart contracts are self-executed programs stored on the blockchain. DApps, on the other hand, do not exist on the blockchain but rather interact with it. They are applications that are used to communicate with smart contracts, thus with blockchain itself. DApp is the abbreviation of decentralized application. The distinction between dApp and website or mobile app can be hard to determine regarding their interfaces, but there is a fundamental difference. As its name suggests, it is an application that has its back-end code built on a decentralized peer-to-peer network in contrast to traditional web applications where it is built on centralized servers [Ethereum Concepts, 2021]. It combines a smart contract representing the core logic of the application interacting with the blockchain and a front-end written in standard languages (HTML, CSS, and javascript) to render the page and make calls to its back-end. Furthermore, the front-end and assets like photos or videos can be hosted on decentralized storage such as Swarm [Swarm, 2021], and IPFS [ipfs, 2021]. Instead of an API that connects to a database like in conventional web applications, dApps use "wallets" that communicate with the blockchain by triggering functions of a smart contract. These wallets manage cryptographic keys and blockchain addresses.
A dApp can also be characterized as a deterministic, turing complete, and isolated application. Deterministic because they perform the same instructions regardless of the environment they run on, turing complete as it can solve any computational problem given the required resources, and isolated since it is executed in the Ethereum virtual machine (EVM explained later on), which means that even if the dApp is malfunctioning, it will not harm the blockchain network.
Developing a dApp comes with several advantages [Ethereum Concepts, 2021]. Since the

smart contract of a dApp lives on the blockchain, it will always be online, so there is zero downtime. It also makes it harder for hackers to launch DoS attacks towards specific apps. Users of such applications can benefit from data privacy as they do not need to provide real-world identity to interact. Because of the inherent trustless characteristic of distributed ledger, dApp can take advantage of data integrity as these are immutable and indisputable, and predictability as smart contracts can be verifiable in predictable ways without the need to trust a central authority.

2.2.3 Concepts (Accounts, Transactions and Consensus Mechanism)

Before we synthesize the overall operation of the Ethereum blockchain, a brief explanation of its important constituent elements is necessary [Ethereum Concepts, 2021].

- **Currency:** Ether or "ETH", Ethereum's crypto-currency is used to pay transaction fees and computational services.
- **Accounts:** There exist two types of accounts that are externally owned accounts (EOA) that are managed by humans and contract accounts. Each of them has a 20-bytes address and is composed of a nonce, codehash/contract code, ether balance, and a storage hash. EOA is controlled by a private key and contract account by smart contract. Furthermore, contract account can only be activated by EOA.
- **Gas:** Ethereum's operation requires a certain amount of computational effort that is measured in Gas that can itself define the required fee to process the transaction.
- **Transactions:** Only EOA can initiate a transaction and not a contract. For example, if Alice sends Bob 1 ETH, Alice's account must be debited, and Bob's must be credited. This state-changing action takes place within a transaction like in Listing 2.2 that contains the address of the sender and the recipient, the sender's signature, gas limit and gas price, a nonce, the number of ETH, and the data to be sent.

Listing 2.2 – Example of a signed transaction in a Geth client. [Ethereum Concepts, 2021]

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "result": {
    "raw": "0xf88380018203339407a565b7ed7d7a678680a4c162885bedbb695fe080
      a44401a6e
      40000000000000000000000000000000000000000000000000000000000001226
      a0223a7c9bcf5531c99be5ea7082183816eb20cfe0bbc322e97cc5c7f71ab8b20
      ea02aadee6b34b45bb15bc42d9c09de4a6754e7000908da72d48cc
      7704971491663",
    "tx": {
      "nonce": "0x0",
      "gasPrice": "0x1234",
      "gas": "0x55555",
      "to": "0x07a565b7ed7d7a678680a4c162885bedbb695fe0",
      "value": "0x1234",
                "input": "0xabcd",
                "v": "0x26",
                "r": "0x223a7c9bcf5531c99be5ea7082183816eb20cfe0bbc322e97cc5c7f71
                  ab8b20e",
                "s": "0x2aadee6b34b45bb15bc42d9c09de4a6754e7000908da72d48cc
                  7704971491663",
                "hash": "0xeba2df809e7a612a0a0d444ccfa5c839624bdc00dd29e3340d46df
                  3870f8a30e"
            }
        }
    }
```

- **Consensus mechanisms:** In short, a consensus mechanism ensures the viability of online transactions and eliminates the possibility of double-spending. It also makes sure participants do not cheat. Ethereum uses three types of consensus algorithms depending on the setup, namely Proof of Work (PoW), Proof of Stake (PoS), and Proof of Authority (PoA). As this study does not address the subject of consensus protocols, here are short descriptions of each of them. An in-depth study can be read in [Xiao et al., 2020].
  Proof of Work – tries to solve a very difficult mathematical puzzle, but its result is easy to check. The performance is low.
  Proof of Stake – encourages the validators that have a big stake of the crypto-currency, which shows the ability to create blocks. The larger the stake, the higher the probability that the network will allow the creation of a block. The performance is usually high in that case.
  Proof of Authority - delegate the production of new blocks to a small and fixed number of pre-selected miners. New blocks are created only when a majority is reached by the validators. PoA becomes semi-centralized as we have pre-selected authorities. It is suited for private and consortium blockchains. The performance is also high.

- **Nodes:** As mentioned in the previous section, a node refers to a participant of the peer-to-peer network that runs the software Ethereum client. Depending on the synchronization strategies, which means how fast we want the node to get the most up-to-date information on the blockchain's state, three distinct types of node exist, namely, full, light and archive.
  Full nodes store the entire blockchain data and participate in block validation as well as blocks and states verification. All states can be derived from a full node. They also serve the network and provide data on requests.
  Light nodes, on the other hand, will only store the header chain and request everything else. As they can not participate in block validation, they can only verify the validity of the data. It is useful for low-capacity devices.
  Archive nodes are the heavyweight as they not only store everything like a full node, but they build an archive of historical states as well. This type of node can be helpful in the case of chain analytic, for example.

- **Networks:** When setting up an Ethereum client, multiple choices regarding the type of network are available. As public networks, there is the "mainnet" which is the primary public Etherium blockchain, and several "testnets" where developers can test their smart contract in a production-like environment before deployment to the mainnet. There is also the possibility to develop locally in a private network.

2.2.4 Ethereum's Incentive Mechanism (London Upgrade)

Ether and gas are two important notions in Ethereum's incentive mechanism. Miners that contribute to the blockchain growth are rewarded with ETH. On the other hand, gas is more of an abstract notion that expresses the amount of effort needed to process a transaction or a smart contract.
A participant requests an operation to be executed by setting a Gas price it is willing to pay, which consists of a base fee and a tip (priority fee). Gas price can be described as the amount of Ether per unit of gas the transaction will pay to the miners that execute the operation. To make calculation simpler, Gas prices are denoted as Gwei, which is itself a denominator of ETH ($1Gwei = 10^{(-9)}ETH$). As an example, let us take a transaction of Ether between accounts where Alice sends Bob 1 ETH, and the gas price is (100+10) Gwei. The usual Gas limit for such a transaction is 21,000 Gas. The total fee will be:

$$\text{Gas limit} * (\text{Base fee} + \text{Tip}) = 21,000 * (100 + 10) = 2,310,000,000 \text{ Gwei} = 0.00231 \text{ ETH}$$

In this operation, Alice's account will be charged 1.00231 ETH, Bob will receive 1.0000 ETH, and the miner that executed the transaction will receive the tip of 0.00021 ETH as compensation for the computational effort. And 0.0021 ETH is burned.
This small example shows the importance of gas fees referred to as retribution to active participants like miners who operate the blockchain correctly. As the miners use their hardware and electricity, it is natural to compensate them and incentivize them to run the distributed ledger properly. Keeping the network secure is another important reason for gas fees as it helps to avoid DoS attacks by requiring a fee for every operation on the network.

2.2.5 Ethereum Virtual Machine (EVM)

Ethereum Virtual Machine (EVM) can be described as one single entity maintained by all the nodes in the blockchain network that run an Ethereum client. The latter contains the EVM in which every account and smart contract lives. It is a Turing complete machine with predefined instructions to compute a new valid state from block to block to keep the uninterrupted and immutable operations of the Ethereum blockchain that can be seen as a state machine. Figure 2.3 demonstrate how the EVM sits within an Ethereum node. There are three distinct software layers in a node: Smart contract code, EVM, and an Ethereum client. The latter is responsible for the communication within the network and the outside world. When deploying a smart contract, it is first encoded into bytecode thanks to a compiler such as the Solidity compiler (Solc) and then given a unique address. This smart contract runs within the EVM, which executes it every time it receives a message call that triggers it. All the functionalities of the

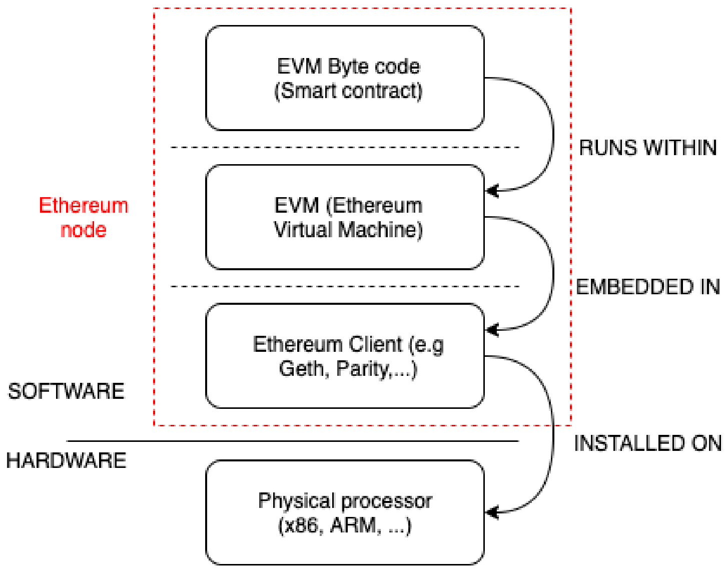

EVM Byte code (Smart contract)
RUNS WITHIN
Ethereum node
EVM (Ethereum Virtual Machine)
EMBEDDED IN
Ethereum Client (e.g Geth, Parity,...)
SOFTWARE
INSTALLED ON
HARDWARE
Physical processor (x86, ARM, ...)

Figure 2.3 – Ethereum architecture layer by cf. [Saini, 2021].

EVM are embedded in the software client, such as Geth or Parity. The very bottom layer depicts the hardware that runs the Ethereum client.
Regarding the model execution of the EVM, it is presented in Figure 2.4. There are four data registers in the EVM: a program counter (PC), a Gas counter, a stack, and a memory. The PC value points to which operational codes (opcode or instructions) are to be executed, and the amount of Gas needed for its execution is stored in the Gas counter register. Once a smart contract is compiled into bytecode, it is stored in a virtual ROM (read-only memory), which means that once on the blockchain, it can not be updated; therefore, it is immutable. It is then processed as a number of opcodes that perform stack operations like PUSH, SWAP, AND, ADD, SUB, XOR, etc. During the execution, the data is stored in the memory space.

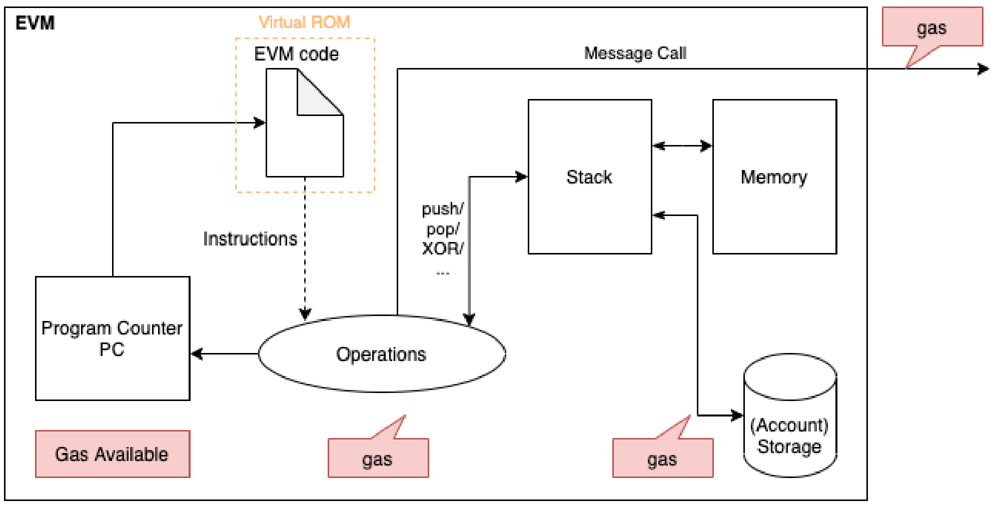


Figure 2.4 – The execution model of the EVM by cf. [evm illustrated, 2021].

3 IoT and Blockchains

In our work, we are interested to see how we can run different Ethereum types of nodes on low-resource IoT edge devices like a Raspberry Pi and investigate how well they perform. Throughout the chapter, related works are mentioned to show the state-of-the-art. We will therefore see in the first place the challenges and benefits of integrating blockchain in IoT as it can solve major security issues. We will then identify existing architectures and use cases and propose an architecture that will suit our experiment.

3.1 Blockchain and IoT Integration

The Internet of Things (IoT) can be pictured as a network of connected embedded devices interacting with miscellaneous everyday objects referred to as things. The constant growth from the last decade of this technology has seen its application expanding in a broad spectrum of domains such as home automation, industrial automation, transportation, healthcare, and so on. The IoT's rapid increase in its application and its diversity must be met with sustainability by building IoT stack, standardized protocols, and proper layers for an architecture [Fernandez-Carames and Fraga-Lamas, 2018].
At the moment, most IoT systems rely on the centralized client-server architecture, in which networks are organized in a hierarchical way where embedded devices communicate their data to gateways which are in turn connected to cloud servers for processing and analytics. However, the popularity of the IoT is prompt to privacy and security issues. With the emergence of blockchain technology and its properties described in the previous section, the use of a distributed ledger architecture can bring numerous benefits to the IoT world as it can solve those issues. As depicted in Figure 3.1, the current IoT cloud-centered architecture could evolve into a distributed system where the cloud's functionality would be distributed among the participant of the network.

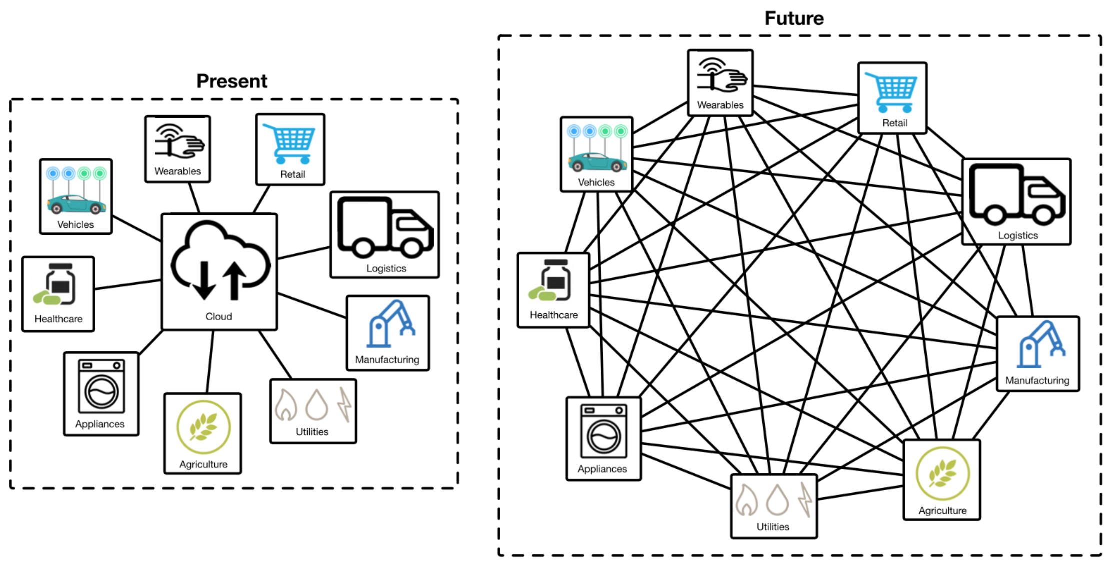


Figure 3.1 – Present & future of IoT architecture [Fernandez-Carames and Fraga-Lamas, 2018].

3.1.1 IoT and Security Problems

With IoT devices coming from various manufacturers worldwide that have different law regulations regarding privacy and security, it is hard for IoT adopters to trust these partners as some of them can give device access and control to certain authorities like governments, manufacturers, or services providers.
The embedded devices used in IoT like Raspberry Pi or Arduino are composed of microprocessors or microcontrollers with input-output pins where sensors are usually connected. Various protocols are used to communicate between these devices, such as MQTT, Wifi, Bluetooth, etc. This multitude of protocols [Sharma and Gondhi, 2018] as well as the heterogeneity of the devices coming from untrusted parties contribute to an unsecured network with possible compromised data. This lack of trust in manufacturers and hardware components enforces the need for privacy and security to be at the core of IoT solutions. By bringing blockchain properties into IoT, it is possible to address these issues.
In the case of privacy, identity certification can be a problem as an identity provider can have too much control over authorizing entities. To tackle this issue, the use of a permissioned blockchain is often proposed for securing and managing multiple IoT nodes [Kravitz and Cooper, 2017]. This type of blockchain can give a distributed identity management solution to increase security and protection against attacks by rotating asymmetric keys thanks to a mechanism called "Device Group Membership" (DGM). This system puts all the devices of a user in a group, and when this user carries out a transaction, it is seen as it was performed by a device that belonged to the user's group. Another interesting technique to increase privacy is to use zero-knowledge-proof algorithms often used in blockchain. A good example is Zcash [Zcash, 2021], in which we can prove certain information to someone without revealing them. This method can be used in IoT for authentication or transactions in

order to hide the identity of a user or device.
To guarantee security, the usual three requirements have to be fulfilled: confidentiality, integrity, and availability. The last two requirements are the most straightforward to be fulfilled by blockchains characteristics. Indeed, information stored on blockchain can not be altered, thus providing data integrity. Regarding availability, the distributed nature of blockchains allows the system to work without interruption, even if some nodes are shut down. However, availability and integrity can be compromised in a 51-percent attack in which a group of miners controls more than 50 percent of the network, thus performing transactions at wish. Blockchain technology does not enforce confidentiality as strongly as the other two security requirements due to its decentralized aspect, implying that data fed to the network can be seen by all participants without restriction. There exist methods to provide confidentiality, such as zero-knowledge proof mentioned above. Private or permissioned blockchains can ensure an extra layer of security as they have restricted access, and nodes must be specifically selected to view and participate in a network.

3.1.2 Energy Constraint of PoW

Implementing blockchain into IoT infrastructure to secure it in a distributed fashion can be challenging. Indeed, IoT is often composed of resource-constraining devices, and PoW requires a significant amount of energy. When developing an IoT application, energy efficiency should be at the center of attention. Therefore, there have been extensive researches to find energy-friendly consensus algorithms.
In their papers, [Puthal et al., 2019, Puthal and Mohanty, 2019] suggest an alternative consensus to PoW called Poof-of-Authentication (PoA) to make blockchain lightweight by authenticating blocks to gain trust values within each iteration of successful block authentication.
In the work of [Alrubei et al., 2021], they combine PoW and Proof-of-Authority (PoA) in an innovative consensus called Honesty-based Distributed Proof-of-Authority (HDPoA). By doing so, they enable strong security offered by PoW and solves its long confirmation time thanks to PoA.
A comparison of some popular consensus algorithms against PoW is shown in Table 3.1 [Zainuddin, 2021]. Indeed, PoW is effective when it comes to securing a decentralized network, but the process is energy-intensive and environmentally unfriendly, which can lead to scalability concerns.
An alternative solution to address this issue is to modify the IoT infrastructure as proposed in [Dittmann and Jelitto, 2019]. They introduce the concept of blockchain proxy to which an IoT device can offload a large part of this software footprint. Their approach assumes a blockchain with public key infrastructure (PKI) management. A PKI authority trusted by the blockchain network issues identity certificates to IoT sensors which use the certified private key to sign its blockchain transactions. The sensors then send the signed transactions along with their certificate to the blockchain proxy, which executes the smart contract.

Table 3.1 – Comparison of Consensus Algorithms.

	Proof-of-Work (PoW)	Proof-of-Stake (PoS)	Delegated Proof-of-Stake (DPOS)	Proof-of-Authority (PoA)
Energy Consumption	High	Low	Very Low	Low
Transaction Per Second	7-30	30-173	2.5-2500	50-2000
Transaction Fees	High	Low	Low	Low
Structure	Decentralized	Decentralized	Decentralized	Semi-Centralized
Examples	Bitcoin	Cardano, Ethereum	Bitshare, Cardano, Tron	Ethereum, VeChain

3.1.3 Challenges and Benefits

We have seen that blockchains can provide a solution to IoT security and privacy problems. However, the integration of such technology comes with numerous challenges and benefits, whether it is in IoT or any other domains, as observed in a comparative study about benefits, challenges and functionalities [Ali et al., 2021]. Here are some general and most relevant challenges and benefits of blockchain-based IoT that we could list:
Challenges

- **Scalability:** as of now, most PoW blockchains are not scalable due to low transactions per second. The response time can be a problem too, as all the recorded transactions in the network must go through a validation process. PoS and sharding techniques can remedy the problem. Permissioned-based blockchains are also proven to be faster in terms of transaction.

- **Energy efficiency:** IoT ecosystems are very diverse and are composed of devices that have different computing capabilities. This differences between them can lead to energy and time loss as some of the devices will not be able to run the same encryption algorithms at the desired speed.

- **Storage:** transactions and device IDs are stored on the blockchain, but the ledger itself has to be stored on the nodes themselves. As time passes, the blocks will be added, and the ledger will grow in size that is beyond the capabilities of a wide range of low storage capacity IoT embedded devices.

- **Lack of skills:** few people are specialized in blockchain technology and understand how it really works. Combined with IoT, this can be a hurdle.

- **Legal and compliance:** IoT has already a hard time with compliance issues since many device manufacturers abide by different regulations worldwide. With the introduction of blockchain, this will be a new territory in all aspects without any legal or compliance code to follow.

Benefits

- **Security:** integrity and availability are provided thanks to blockchains decentralized nature and its data immutability. Confidentiality can be achieved in light of zero-knowledge-proof algorithms. It also eliminates a single source of failure within the IoT ecosystem, protecting the IoT devices' data from tampering.
- **Anonymity:** participants use anonymous and unique address numbers to keep their identities private.
- **Trust:** trust among IoT devices can be established when they exchange data through a blockchain network instead of going through a third party.
- **Lower costs:** the processing of transactions and coordination between devices is easier as data are submitted on a peer-to-peer basis without centralized control, which can lower business expenses.

3.2 Conceptual Architectures for Blockchain-based IoT Use Cases

Given the constrained resources in IoT networks and their challenge, there exist several ongoing research regarding the architectural integration of blockchain in IoT. Indeed, when incorporating blockchain in IoT, we are faced with a scalability trilemma in which it is almost infeasible to simultaneously maximize the three imperative criteria of scalability, decentralization, and security. At most, two of the three criteria can be satisfied. Therefore, trade-offs are forced to be made depending upon the use-cases of the network, and the application-specific needs [Agarwal and Pal, 2020].
[Novo, 2018] proposed an architecture based on blockchain technology to manage access control for IoT devices. The author used a management hub to decouple the blockchain-related resource-demanding tasks on IoT devices. A single smart contract was introduced to facilitate the access control policies, and a proof-of-concept implementation was developed based on Ethereum to illustrate the feasibility of the scheme. In a similar fashion, [Alphand et al., 2018] proposed the use of smart contract to generate a temporary key to authorize access to IoT resources. [Rifi et al., 2017] proposed the use of an off-chain database to store the IoT and use blockchain to store the point in order to protect access to IoT data.
This shows that researchers are gaining interest in the potential use of blockchain technology in IoT. However, there are only a few studies focusing on the actual implementation in real IoT devices and even fewer that explore the scalability and performance of running a blockchain node in said devices. To propose a viable architecture for our study, we are first going through some blockchain IoT architecture solutions and show some use cases.

3.2.1 Architectures

Most of the existing solutions can be broken down to a more general architecture as illustrated in Figure 3.2. In the local network, groups of IoT devices collect raw data that is formatted and encrypted in a consistent format due to the heterogeneity of devices. This formatting operation will help for a proper transaction in the blockchain network. The prepared transaction is sent from the gateway to a node in the blockchain network. After obtaining the data, it is the responsibility of the nodes to create blocks, verify them and chain them together. These nodes, which often run a specific blockchain client (GETH in the case of Ethereum), serve as the interface between the IoT network and the blockchain network.
Depending on the IoT application, this general architecture uses a blockchain with specific requirements to facilitate the operation. When implementing such a structure, energy efficiency and performance are important, as seen in subsection 3.1.2. Therefore, one might use PoS or PoA instead of PoW consensus. Furthermore, the use of permissioned blockchain helps to impose more strict access control on known entities.

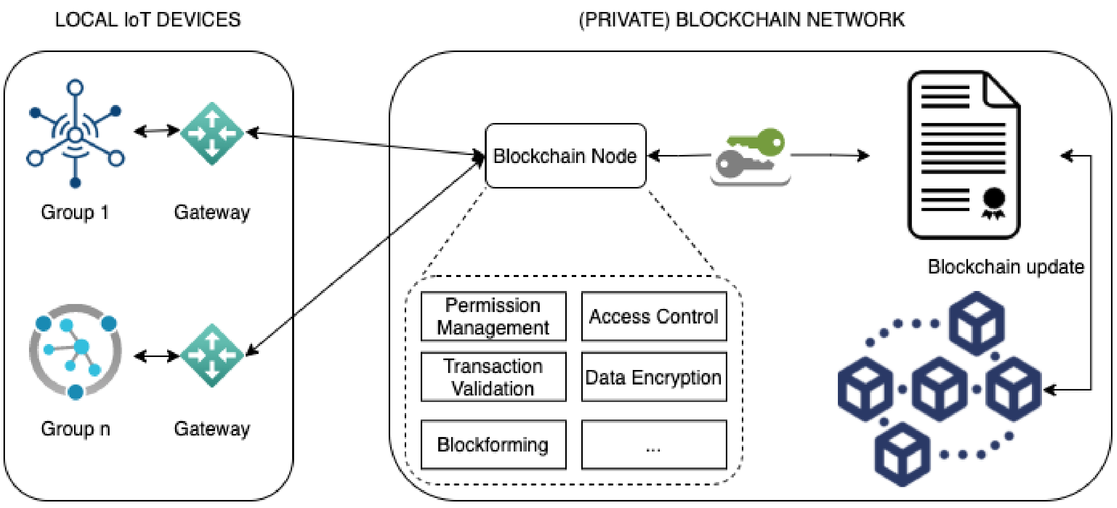


Figure 3.2 – A general blockchain IoT architecture.

One of the scalability issues mentioned in subsection 3.1.3 is the growing ledger as time passes and more devices are added. Full nodes are required to keep an exact copy of the blockchain to guarantee consistency. [Wang et al., 2019] propose a new method to scale out data storage. They propose a hierarchical storage structure to tackle this challenge. The majority of the blockchain is stored in a decentralized cloud such as [ipfs, 2021] to leverage the vast amount of data that IoT devices can generate, while the most most recent blocks are stored in the private network.
This thesis aims to investigate the feasibility of implementing different types of Ethereum nodes in IoT devices. Therefore, we need an architecture in which the blockchain node is present in an IoT device such as a Raspberry Pi, for example. Figure 3.3 pictures the possible architecture to be implemented in this study. Groups of devices send their sensed data to

a collector like a Raspberry Pi, responsible for formatting them. The latter also includes a running piece of client software, also referred to as "node," that will allow validating the transactions.
Besides, this configuration is perhaps more secure. It might prevent less tampering of devices data since the latter are collected and treated in a blockchain client before being updated in the network. Furthermore, it would also be advisable to implement hierarchical storage to enable extra storage like [Wang et al., 2019] proposed. Another solution to tackle the storage problem is to implement only light nodes in the data collectors and have full nodes implemented on other more resourceful machines.

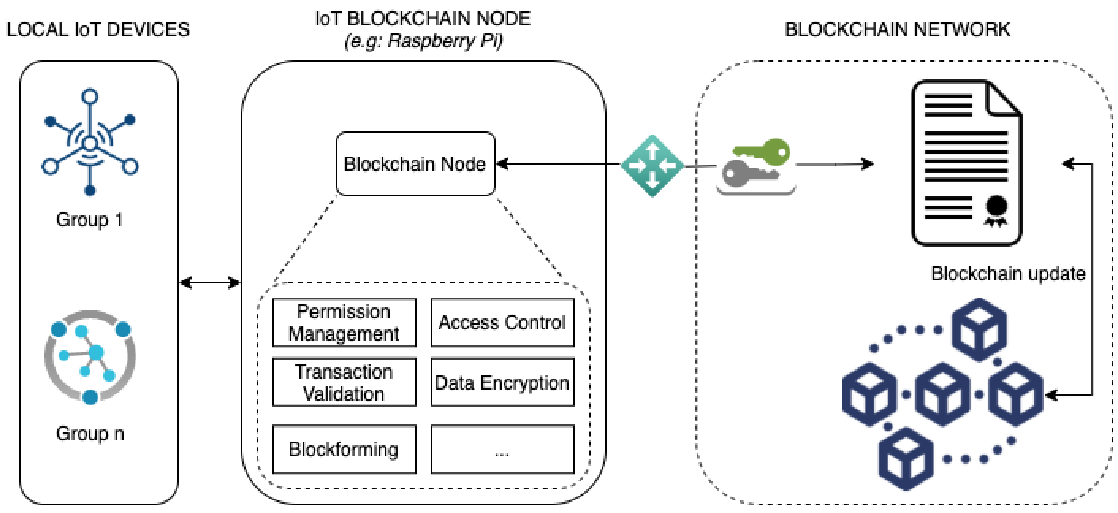


Figure 3.3 – A blockchain IoT architecture with a node in an IoT device.

3.2.2 Use Cases

The combination of IoT with Blockchain can have a significant impact across multiple industries. Here are three examples:

- **Supply Chain Management:** the immutable property of blockchain makes information continuity and traceability easier among the stakeholders, as shown in Figure 3.4. As the technology is also transparent, it helps to access the huge amount of data generated along the supply chain. In [Alkhader et al., 2020], they use Ethereum smart contracts to govern and trace transactions initiated by participants involved in the manufacturing process. IPS is used as a decentralized storage to store IoT devices records. Figure A1 in the Appendix section depicts the architecture of such a system thoroughly.
- **Healthcare:** a centralized controlled scheme for IoT devices in the healthcare industry can be problematic, as there is a single point of failure and a possibility of data manipulation and tampering as well as privacy evasion. With the help of blockchain, it can non only secure the network against manipulation attacks that target stored data, but it can

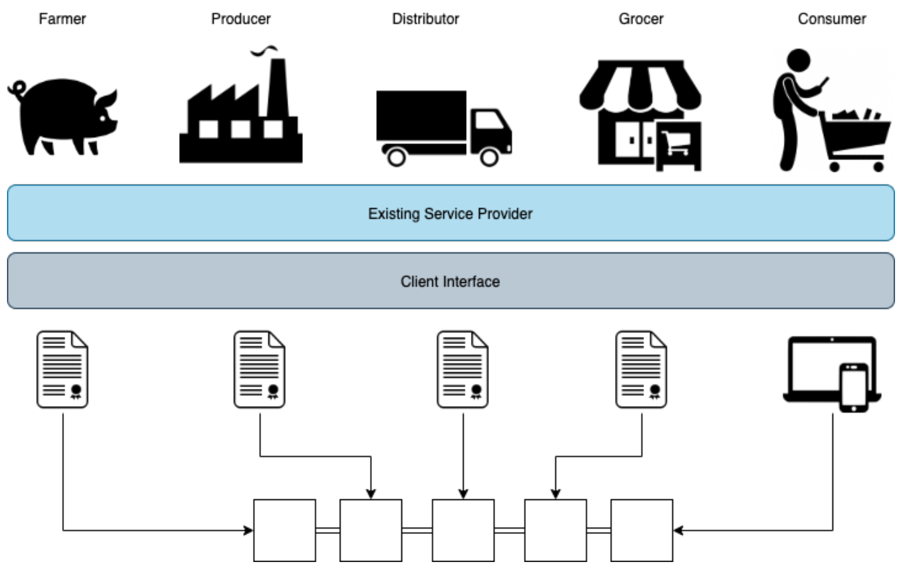


Figure 3.4 – Example of a global food supply chain.

also provide a secure platform for all devices in the network to communicate with each other. In [Ray et al., 2021], they propose several interesting use cases like medical record access and drug supply chain management using the following connectivity in Figure 3.5. A detailed architecture implementation and data flow can be found in their paper.

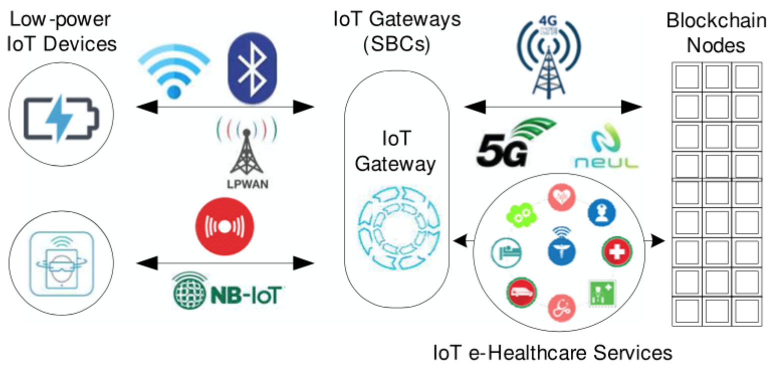


Figure 3.5 – IoT connectivity for blockchain e-healthcare [Ray et al., 2021].

- **Smart Homes:** Blockhain IoT solves security issues and provides ownership of data by removing the centralized infrastructure that is not appropriate in such an application. In [Arif et al., 2020], they instigate the adoption of blockchain and propose an architecture shown in Figure 3.6.

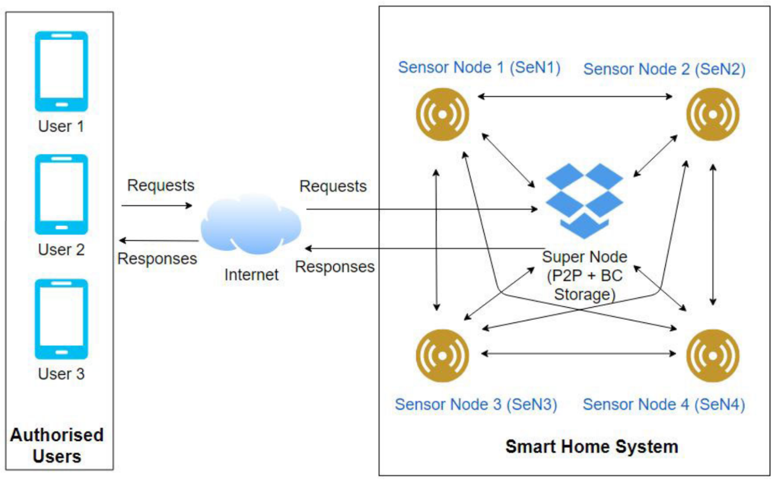


Figure 3.6 – Adopted smart home architecture using IoT-Blockchain [Arif et al., 2020].

These use cases inspired us to design our IoT blockchain-integrated architecture to emulate a real-world use case scenario. We will call this use case "BIoT use case" for simplicity. IoT devices embedded with sensors will interact with a smart contract deployed on the blockchain beforehand in this simple system. In this case, the IoT sensors will store their sensed values in the blockchain. Because these sensors are limited in computational resources, running a node on such a device will probably not be possible. Instead, we will use an intermediary IoT device with better capacity, like a Raspberry Pi, to run a node to interact with the smart contract.
In our work, we are interested to see how it is possible to run a node on such a low-powered device and investigate the performance of blockchain integration in such a system. We, therefore, decided to monitor the node performance during contract interaction. Consequently, we will use a database to store metrics that will help us to monitor the performance.

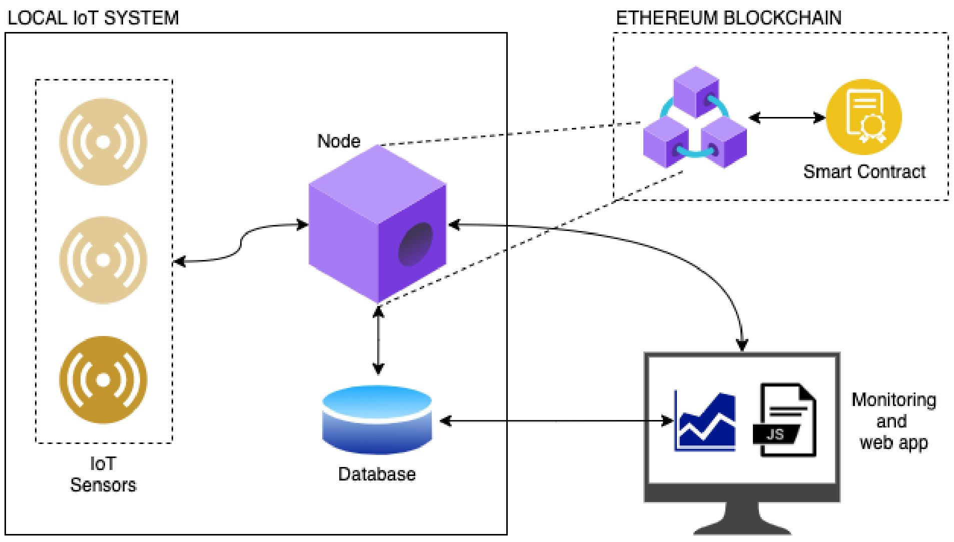


Figure 3.7 – BIoT use case possible architecture.

4 Implementation

As a reminder, this thesis aims to assess Ethereum blockchain nodes on IoT edge devices and evaluate their related performance.
For this purpose, we are in the first instance describing the selected IoT hardware briefly for our implementation.
We will then list the available software and frameworks for Ethereum blockchain development and explain how we implemented one of the Ethereum clients.
In the third phase, we will elaborate on which data we will collect and how to monitor it. These metrics will serve to analyze the performance of our implemented systems.
Finally, in the last part of this chapter, we will propose different environment implementation setups (RPI and EV3) in which we implement the same blockchain settings (types of nodes/network) for further comparisons.

4.1 Hardware

The IoT edge devices used in IoT are often single-board computers since their small size allows them to be embedded where space is limited. As energy consumption is important, these types of computers are also efficient regarding saving power. According to [Ethereum, 2021] documentation, hardware requirement in a public network setup are not excessively high since we only want our node to be synced and not mined in an IoT context. Nevertheless, syncing the Ethereum blockchain is very input/output intensive, and the minimum recommended requirements are:

- CPU with 2+ cores,
- Minimum 4GB of RAM with an SSD, otherwise 8GB+ if it is HDD,
- 8MBit/s of bandwidth.

Moreover, here are some possible alternatives listed in table 4.1 in contrast with an early 2015 Macbook Pro that will serve as our baseline for further comparison.

Table 4.1 – Different single-board devices in comparison with Macbook (early 2015).

Models	CPU	#Cores	RAM	Bandwidth
Banana Pi M3[1]	Allwinner A83T (1.8GHz)	8	2GB	up to 1000Mbit/s
NanoPi Neo4[2]	Dual Cortex-A72 (2.0GHz) Quad Cortex-A53 (1.5GHz)	6	1GB	up to 1000Mbit/s
Odroid-XU4[3]	Samsung Exynos 5422 (2GHz) Quad Cortex-A7 (1.4 GHz)	8	2GB	up to 1000Mbit/s
Rock Pi 4C[4]	Dual Cortex-A72 (1.8GHz) Quad Cortex-A53 (1.4GHz)	6	4GB	up to 1000Mbit/s
ASUS Tinker Board S[5]	Quad Cortex-A17 (1.8GHz)	4	2GB	up to 1000Mbit/s
Raspberry Pi 4 B[6]	Quad Cortex-A72 (1.5GHz)	4	1GB, 2GB, 4GB, 8GB	up to 1000Mbit/s
Raspberry Pi 3 B+[7]	Quad Cortex-A53 (1.4GHz)	4	1GB	300 Mbit/s
MacBook 13 inch (early 2015)	Dual Intel i7 (3.1GHz)	4	8GB	up to 1000Mbit/s

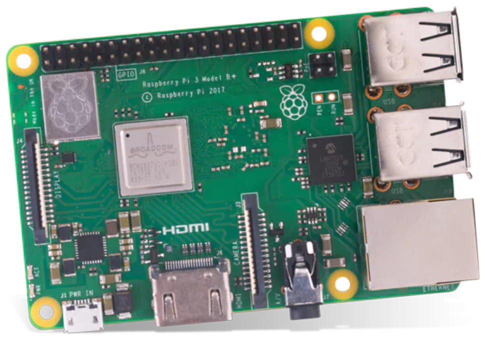
4.1.1 Investigated Devices

It is more interesting to see how the lower tier of single-board computers can implement an Ethereum node. Therefore here are the investigated devices shown in Figure 4.1:

Raspberry Pi (RPi) - RPI is a widely used edge device in IoT projects as it is relatively cheap and cost-effective as well as developer-friendly. Hence, the RPi can serve as the "internet gateway" for IoT devices. It can, in a sense, act as a web server for uploading and transiting sensor data on IoT platforms. Its low-cost aspect and ease of use make it an interesting device to test blockchain integration in IoT. In our implementation, we are using the Raspberry Pi 3 B+ version shown in Figure 4.1a.

EV3 Lego brick[8] - is a single-board computer with a Debian Linux-based operating system called "ev3dev". It is produced by Lego and is used for the development of programmable robots for educational purposes. It is composed of one main processor (TI Sitara AM1808 at 300MHz) and 64MB of RAM.

[1]https://www.banana-pi.org/m3.html
[2]https://www.friendlyarm.com/index.php?route=product/product&product_id=241
[3]https://www.hardkernel.com/
[4]https://rockpi.org/rockpi4
[5]https://tinker-board.asus.com/product/tinker-board-s.html
[6]https://www.raspberrypi.org/products/raspberry-pi-4-model-b/
[7]https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/
[8]https://www.lego.com/en-ch/product/ev3-intelligent-brick-45500



(a) Raspberry Pi 3 B+ [Mouser, 2018]

(b) EV3 brick [Lego, 2013]

Figure 4.1 – Investigated devices: Raspberry Pi 3 B+ and Lego EV3 brick.

4.2 Software Stack

4.2.1 Ethereum Client

A node refers to a running piece of client software which is an implementation of Ethereum that can verify all transactions in each block, keeping the network secure and the data accurate. Many Ethereum open-source clients exist in a variety of programming languages such as Go, Rust, JavaScript, Python, C# .NET, and Java. This diversity contributes to a stronger and more secure network. Table 4.2 summarizes the different clients.
When choosing a client, we should base on our preferences since each of them has unique use cases and advantages. Diversity allows implementations to be focused on different features and user audiences. In our case, we decided to go with Geth as it is one of the original implementations of the Ethereum protocol. Furthermore, it is also the most widespread client with a strong community and has a variety of tooling for users and developers. Written in Go language, it is fully open-source and licensed under the GNU LGPL v3.

The mentioned clients in Table 4.2 can run different types of nodes depending on the sync strategies, which enables faster synchronization time. This synchronization refers to how quickly it can get the most up-to-date information on Ethereum's state. The Geth client features:

- **Full** - downloads all the blocks (including headers, transactions and receipts) and generates the state of the blockchain incrementally by executing every block.

- **Fast** (Default) – downloads all blocks (including headers, transactions and receipts), verifies all headers, and downloads the state, and verifies it against the headers.

- **Light** – downloads all block headers, block data, and verifies some randomly.

Table 4.2 – Different Ethereum clients.

Client	Language	OS	Networks	Sync strategy
Geth[9]	Go	Linux, Windows, macOs	Mainnet, Görli, Rinkeby, Ropsten	Light, Fast, Full
OpenEthereum[10]	Rust	Linux, Windows, macOS	Mainnet, Görli, Rinkeby, Ropsten	Warp, Full
Nethermind[11]	C#, .NET	Linux, Windows, macOS	Mainnet, Görli, Rinkeby, Ropsten	Beam, Fast, Full
Besu[12]	Java	Linux, Windows, macOS	Mainnet, Görli, Rinkeby, Ropsten	Fast, Full
Erigon[13]	Go	Linux, Windows, macOS	Mainnet, Görli, Rinkeby, Ropsten	Fast, Full

4.2.2 Development Framework

Building a dApp requires different development tools. They come with a plethora of functionalities and features that will help automate development efforts. Smart contracts and dApp development require reliable debugging tools, libraries, API, and many other things to develop in a safe environment. The world of blockchain evolves rapidly, and new Ethereum development tools emerge every now and then while some get deprecated and obsolete. Table 4.3 shows the most recent and used (at the time of writing) Ethereum development tools.

[9] https://geth.ethereum.org/
[10] https://github.com/openethereum/openethereum
[11] https://nethermind.io/
[12] https://www.hyperledger.org/use/besu
[13] https://github.com/ledgerwatch/erigon
[14] https://www.trufflesuite.com/
[15] https://www.trufflesuite.com/ganache
[16] https://docs.ethers.io/v5/
[17] https://remix.ethereum.org/
[18] https://www.trufflesuite.com/drizzle
[19] https://infura.io/
[20] https://metamask.io/

Table 4.3 – Different Ethereum development tools.

Development Tools	What	Description
Truffle[14]	Back-end Development framework	It is used for developing and deploying Ethereum applications. It helps to create projects, code and compile smart contracts, run automated tests, migrate, and interact with the contracts.
Ganache[15]	Rapid Blockchain Development	It instantly create a personal blockchain to run on a local device throughout the development cycle. It helps to develop, test, and deploy Ethereum smart contracts in safer environment.
Ether.js[16]	Library	It is a complete and compact Ethereum library ecosystem. It offers rich features and usability.
Remix IDE [17]	Web & Desktop Application	It is an open-source JavaScript-based compiler that allows direct coding from a web browser. It offers a rich collection of plug-ins and libraries.
Drizzle[18]	Front-end Development framework	It offers a collection of front-end libraries. It makes dApp development more convenient and predictable as it is designed for smooth synchronisation between a smart contract state and user interface.
Infura[19]	Blockchain Development	It is and API access for Ethereum and IPFS networks. Is supports JSON-RPC over HTTPS and WebSocket with instant access.
Metamask[20]	Crypto Wallet	It is an Ethereum wallet and a blockchain application gateway that functions as a web browser extension. It offers key vault, login security, token wallet, and exchange facilities.

4.3 Metrics

When assessing node performance test, the choice of the right metric to collect is important. In this section, we list the collected data in order to establish a performance test (RQ2). We can group the metrics into three categories: blockchain specific metrics, P2P layer metrics, and system node metrics
In Geth implementation, various metrics can be queried out of the client node and dumped directly into an Influx[21] database for ease of manipulation such as monitoring. Listing 4.1 shows the options to append to enable metrics on the Geth CLI.

Listing 4.1 – Geth flags enabling metrics collection and dumping into Influxdb.

```
geth --metrics --metrics.influxdb --metrics.influxdb.endpoint "localhost:8086"\
--metrics.influxdb.username "geth" --metrics.influxdb.password "chosenPassword"
```

[21] https://www.influxdata.com/

4.3.1 Data collected

Blockchain node metrics

- **Goroutines** - A Goroutine is a function or method which executes independently and simultaneously in connection with any other Goroutines present in a program. Similar to a light weighted thread, every concurrently executing activity in Go language is known as a Goroutines. This measure is in integer.
- **Data rates** - This is the level of database read/write, as well as compaction read/write measured in kB/s.
- **Session totals** - The total memory of a session of database read/write, as well as compaction read/write data, measured in MB.
- **Persistent size** - The persistent size of the data measured in MB. This means the data is stable and recoverable even if there was ever a power interruption.

P2P metrics

- **Peers** - The number of direct peers in integer.
- **Traffic** - Data ingress (or inbound) and data egress (or outbound), measured in kB/s.

System node metrics

- **CPU** - shows how many calculations the processor performs. If the CPU load is high, the node is calculating something such as checking electronic signature or processing transactions with strong cryptography. It is measured in percentage.
- **Memory** - shows the number of memory allocated (allocs), the amount of memory currently used (used), and the memory allocated on the heap (held). They are all measured in MB.
- **Storage** - The number of read/write on the system disk in kB/s. It also shows the compaction time in ms.
- **Network** - It regroup the number of sent/received packets per second and the inbound-/outbound data measured in kB/s.

4.3.2 Monitoring

For a better data visualization, we decided to use a monitoring solution like Grafana[22] which is a multi-platform open-source analytics and interactive visualization web application. It

[22] https://grafana.com/

groups data together in dashboards to help users better understand their data metrics through queries, informative visualization, and alerts in an efficient and organized manner. Figure 4.2 shows an example of a dashboard regrouping Geth metrics.

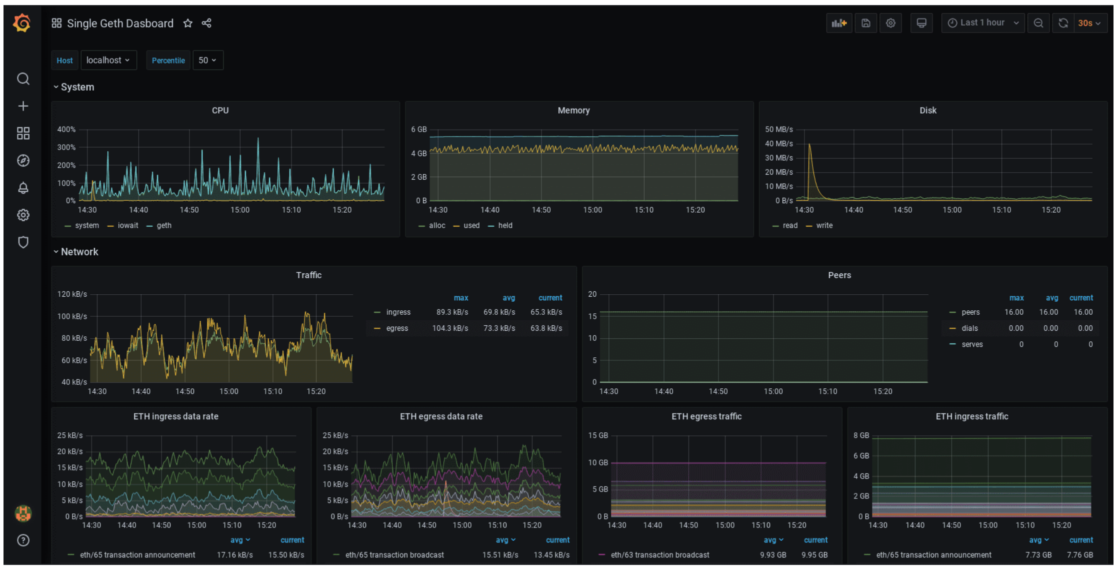


Figure 4.2 – Example of a Grafana dashboard with Geth metrics.

4.4 Realizing Blockchain-base IoT Architectures

This section will give two scenarios in which we will try to interact with an Ethereum blockchain in a public and permissioned fashion, given the hardware and software tools. The two implementations will follow the state-of-the-art architecture in blockchain IoT mentioned in the previous chapter. They will also give an explanation as to how public and permissioned blockchain nodes can be run on the selected hardware (RQ1). We will also show how we implemented our BIoT use case in the final part of this section.

4.4.1 Private Network

This first scenario will explore how we can build a permissioned blockchain and implement a private node into the selected hardware devices. We have seen that a private/permissioned blockchain can be a good option in IoT system as it can provide scalability and is less energy demanding than its public counterpart. Furthermore, this private blockchain uses PoA consensus in which approved accounts, known as validators, validate transactions and blocks.
There are several ways to build a private blockchain. We can either implement several private nodes within the same hardware device or build one private node on each device and connect them to the same local network. As the second option is closer to reality, we decided to opt for this solution. Once the blockchain network is up and running, we can append an Influx

database to a node in order to help collect performance data and visualize them on Grafana. An extra step in our work would be to actually see how nodes perform in the validation and mining process by deploying a smart contract into the blockchain network.

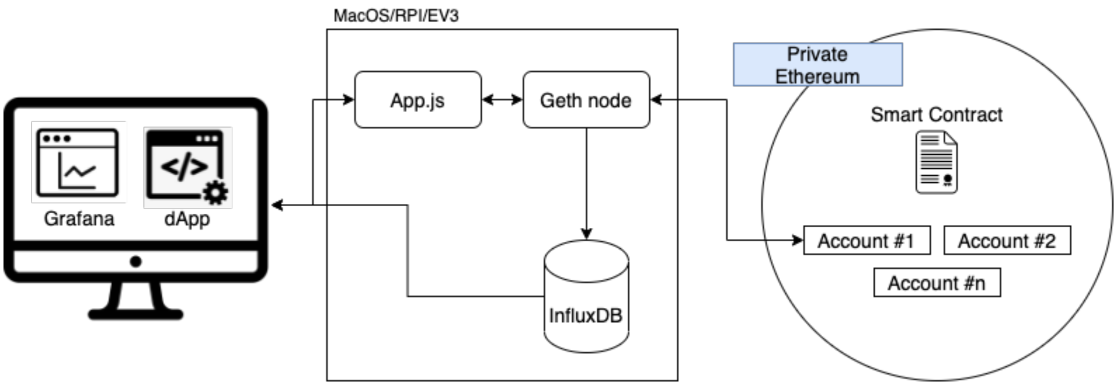


Figure 4.3 – Architecture of a permissioned Ethereum local network.

Figure 4.3 shows the implemented architecture in a local environment. We used Geth as a development tool to implement a private node. Figure 4.4 pictures how we structured node related files. The accounts.txt helps us to keep track of the created accounts. Before launching the private blockchain, we need a "block 0" that will be generated thanks to a genesis file that includes the configuration of the blockchain network as well as who starts out with how much ether. It is important to emphasize that every node in our private network has to be initialized with this genesis file. The initialization will create inside the node directory, a Geth directory where every persistent data will get written in. We can also find the public-private keys of the associated account in a Keystore file generated beforehand during the account creation, the credential of the account, a JSON file regrouping peer nodes, and a shell script shown in Listing 4.2 that will run the node.
Let us note some important flags in the latter. In line 1, we enable full chain sync. As we do not want to pay anything on a private network, the gas price is set to 0 in line 2. Line 3 enables HTTP-RPC server and exposes port 8545. Line 4 enables mining and allows insecure account unlocking when account-related RPCs are exposed by HTTP. Finally, line 5 enables the metrics by dumping them into an Influx database whose exposed port is 8086.

Listing 4.2 – Command to start a full node on a permissioned network.

```
1 nohup geth --nousb --datadir=$pwd --syncmode ’full’ --port 30310\
2 --miner.gasprice 0 --miner.gastarget 470000000000\
3 --http --http.addr ’localhost’ --http.port 8545 --http.api admin,eth,miner,net,
    txpool,personal,web3\
4 --mine --allow-insecure-unlock --unlock "0
    xD7A4931DF5E52C7994A0DD42C1f131171ce5c2f1" --password pwd.txt\
5 --metrics --metrics.influxdb --metrics.influxdb.endpoint "http://0.0.0.0:8086" --
    metrics.influxdb.username "geth" --metrics.influxdb.password "blockchain"
```

At this point, our private Ethereum network should be up and running with all the nodes synchronized. We should also be able to collect performance-related data and monitor it on

```
blockchain
├── accounts.txt
├── genesis.json
└── node
    ├── geth
    ├── keystore
    ├── node1.sh
    ├── password.txt
    └── static-nodes.json
```

Figure 4.4 – Structure of files to create a private Ethereum Network.

Grafana.
We can go further by deploying and testing smart contract on the network to verify that the setup is correct by interacting with it. For this purpose, we used the Truffle framework, which helped us to code and compile a smart contract for a "Todo List" dApp. We can interact with the smart contract via a web interface coded in Node.js with Ether.js library imported, and thanks to the Metamask browser extension, which helps to authorize account transactions. As smart contract and dApps are not in the core subject of this thesis, we will not go into further details (see Appendix B1).

4.4.2 Test Network - Ropsten Testnet

The use of private blockchain networks in IoT is common as it can provide several perks. Nevertheless, we are curious to see how a public network such as the Ropsten Ethereum testnet can be integrated into IoT and how well it performs. As the name suggests, a public network is accessible to anyone in the world with an internet connection. In this case, everyone can read or create transactions as well as validate them. Public testing networks, also known as "testnets" are networks used by protocol developers or smart contract developers to test both protocol upgrades as well as potential smart contracts in a production-like environment before deployment to the main network, referred to as "Mainnet". The Ropsten testnet uses PoW, which means it is the best like-for-like representation of Ethereum. Figure 4.5 shows the implemented architecture in a public environment. Similar to the private implementation, we run a Geth node and attach to it InfluxDB, to which metrics are dumped for ease of monitoring on Grafana. Running a public Geth node is simpler than a private one as we do not need to specify the configuration of the blockchain network in a genesis file. All we need to do is connect to the testnet thanks to the command shown in Listing 4.3.

Listing 4.3 – Command to start a light node on ropsten testnet.

```
nohup geth --ropsten --cache=256 --syncmode "light"\
--http.addr 'localhost' --http.port 8545\
--metrics --metrics.influxdb --metrics.influxdb.endpoint "http://0.0.0.0:8086" --
  metrics.influxdb.username "geth" --metrics.influxdb.password "blockchain"
```

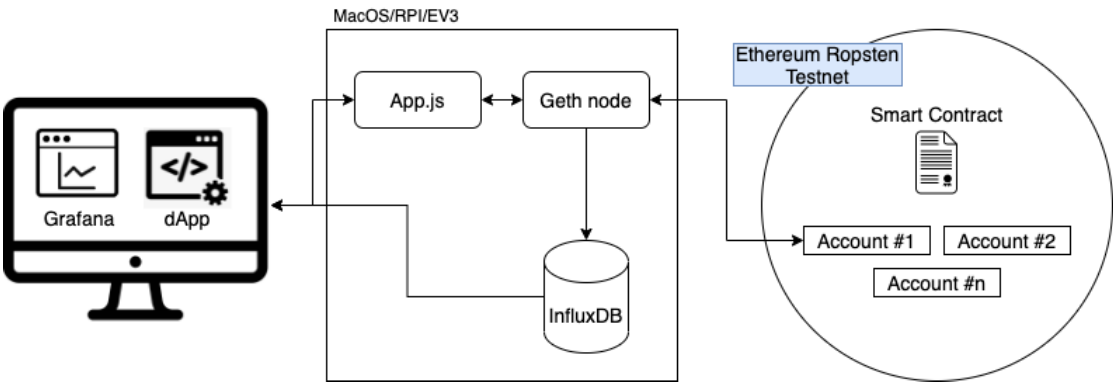


Figure 4.5 – Architecture of a public Ethereum test network (Ropsten).

Let us note some important options in this command line. We connect to the testnet thanks to the "ropsten" flag, and we specify a cache with specific memory depending on which machine we run the node to avoid fatal memory error. As previously mentioned, the synchronization modes are "fast", "full", and "light". Full synchronization is impossible on a low-resource device like the RPI as it requires Terabytes of disk space, and it must be an SSD. Regarding fast synchronization, it downloads the blocks and only verifies the associated PoW instead of starting from the genesis block and reprocessing all the transactions that ever occurred like in full synchronization (which could take weeks). However, fast synchronization on a low-powered device seems to be always behind as the node has trouble keeping synchronized with the ever-changing dataset. As mentioned in this Github issue[23], the node needs to constantly follow the network while trying to gather all the recent data. But until it actually does gather all the data, the node might not be usable since it could not cryptographically prove anything about any accounts in the network. As both "full" and "fast" synchronization are not possible on a low memory device, running a light node is the only option left. In this case, the node only downloads a few recent block headers, making it a very quick sync. The drawback to this method is that the node is unable to perform reliable validation, as it doesn't have complete records of the blockchain.
Once we run the command, a light node should be running and syncing with the Ropsten testnet. We should also be able to track performance data on Grafana.
Once we have a running node on the Ropsten testnet, we can further deploy a smart contract and show how we can easily interact with it to verify a correct setup. For this purpose, we first create an account on the node thanks to the Geth console. We can then import this account to Metamask since we have the RPC port open for our node.
For ease of implementation, we use a simple, smart contract called "CoolNumberContract" (see Appendix B2) that can set a number randomly and can print the current number stored on the blockchain. This contract was deployed through Remixe IDE framework. To interact with the tesnet, we will need testnet ETH that we can get from the Ropsten faucet[24].

[23]https://github.com/ethereum/go-ethereum/issues/16796
[24]https://faucet.ropsten.be/

4.4.3 Implementation of our BIoT Use Case

At the end of the previous chapter, we introduced our possible IoT blockchain-integrated design to emulate a real-world use case scenario. Let us precise the architecture from Figure 3.7 by including technical details as shown in Figure 4.6.
The idea is to store data from a low-powered IoT device embedded with sensors like the EV3 to a public blockchain like Ropsten testnet through the use of smart contract. The RPI is wired to the router by an ethernet RJ45 cable, whereas the EV3 is linked by WiFi to the 2.4Ghz bandwidth. We believe that the EV3 is not capable of running an Ethereum node, hence the use of a RPI. As mentioned in the previous section, we ran a light node on this latter and opened an HTTP-RPC endpoint on port `0.0.0.0:8485`. Now, because the EV3 is powerful enough to run scripts, it is possible to interact directly with the Geth client through the RPC endpoint. Otherwise, if we are using extremely low-powered devices that cannot run scripts, for example, light sensors that can only capture simple values and send them through lightweight protocols such as zig-bee, z-wave LoRaWAN, or Bluetooth. In that case, we can send these sensed data to the RPI that will act as a data collector. For the sake of simplicity, we will store the speed of the motor of the EV3 to a smart contract. To summarize, the EV3 acting as a sensor sends its speed of motor values to the RPI (acting as a collector) through the RPC port directly to interact with the smart contract in the blockchain.
To gather performance metrics, we used two services that are InfluxDB on port `0.0.0.0:8086` and Grafana on port `0.0.0.0:3000`. We can access the Grafana service from a web interface and analyze the performance data from there. In addition, we developed a web app in Node.js to interact with our smart contract, for example, printing values of the smart contract.

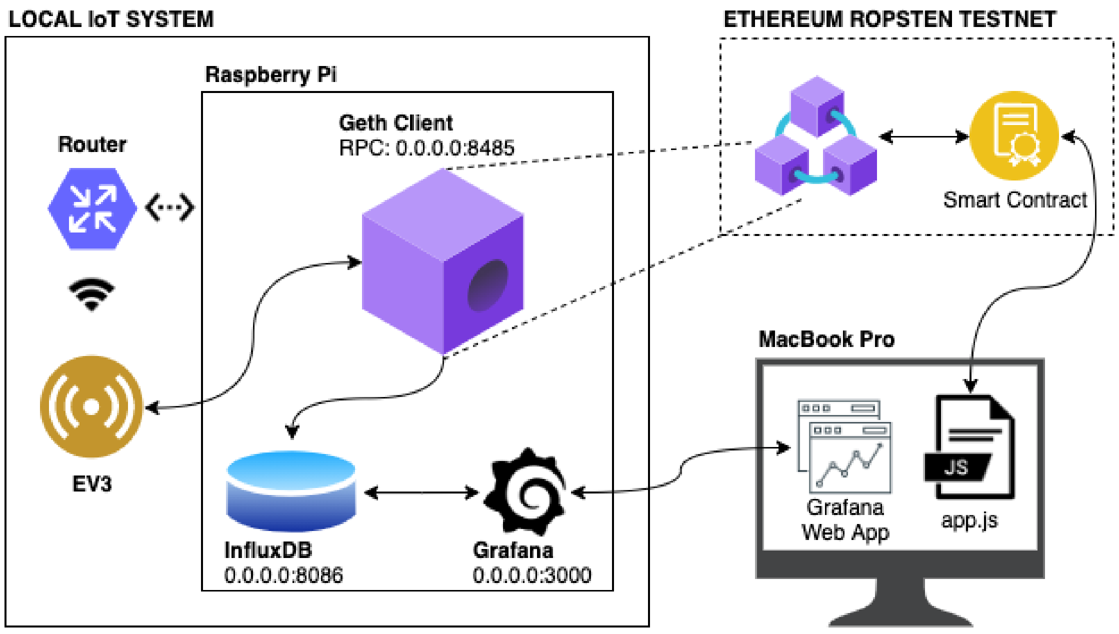


Figure 4.6 – Detailed BIoT use case architecture.

5 Result & Evaluation

The previous chapter showed how we could run public and permissioned Ethereum blockchain nodes on the selected hardware, as well as how we can establish performance tests. In this chapter, we will, in the first part, validate the feasibility of the two implemented scenarios. We will then demonstrate that we set up the network correctly by showing the interaction with smart contracts. In the last step, we will show how the performance differs on the investigated hardware devices (RQ3), and on the BIoT use case.

5.1 Feasibility

Thanks to existing tools and frameworks, we showed earlier in two scenarios how we can run public and permissioned Ethereum blockchain nodes on the EV3 and RPI, as well as what data to collect in order to study performance. It is now a matter of validation of the implementation and seeing if they were actually successful on the two selected hardware devices. Before we dive into the feasibility of each device, let us first recall the minimum recommended hardware requirements according to [Ethereum, 2021]:

- CPU with 2+ cores,
- minimum 4GB of RAM with an SSD, otherwise, 8GB+ if it is HDD,
- 8Mbit/s of bandwidth.

To validate that we can run an Ethereum node in both public and permissioned networks on the selected device, it is sufficient to show that we can run a light node on the Ropsten testnet as it is more energy costly than running a node in a private network. Otherwise, we need to demonstrate that it is only feasible in a permissioned network. In the worst case, if we cannot show the latter, running an Ethereum node, whether it is in a public or permissioned network, is not feasible on the device. To summarize, there are three outcomes: feasible in both public and permissioned network, feasible in permissioned network only, and not feasible at all.

5.1.1 Testbed

Before we proceed with the validation of the feasibility on the devices, Table 5.1 shows the hardware and software versions as well as the network settings we used on each device. It illustrates the testbed for our implementations and allows for future reproducibility. We can already see that we could not install some of the frameworks on the EV3 as it has memory limitations. Furthermore, the connectivity to the gateway is also different on each device. Because of their different bandwidth capacity, the MacBook Pro can connect to the 5GHz WiFi bandwidth, whereas the EV3 cannot. The RPI, on the other hand, is connected to the Ethernet by RJ45. This difference in network capacity is also reflected in the speed: for example, the machines' download and upload speeds are significantly different. Also, because the MacBook Pro was on WiFi and was not close to the router, we have a ping higher than the ethernet wired RPI. Subsequently, the EV3 has the worst request speed of them all.

Table 5.1 – Testbed - Hardware/Software version and network settings

TESTBED			**Mac darwin/amd64**	**RPI linux/arm7**	**EV3 linux/arm5**
OS	-		darwin Catalina 10.15.7	linux Raspbian buster 10	linux ev3-dev
Architecture	-		AMD64	ARM v7l	ARM v5l
Software/ Frameworks versions	Geth (cache)		1.10.8-stable (2048MB)	1.10.5-stable (256MB)	-
	Solc		0.6.12	0.6.12	-
	Truffle		5.4.8	-	-
	InfluxDB		1.8.9	1.8.9	-
	Grafana		8.1.4	8.1.2	-
	web3.js		1.5.2	1.5.2	-
Languages versions	Golang		go1.17.1	go1.17	go1.7.4
	Node		16.8.0	16.8.0	-
	Solidity		^0.6.6	^0.6.6	-
Network	Gateway		Sunrise 5GHz	Sunrise Ethernet	Sunrise 2.4GHz
	Connectivity		Wifi	RJ45	Wifi
	Speed	Download $Mbit/s$	300+	150+	0.8
		Upload $Mbit/s$	300+	100+	0.2
	Ping ms		3 - 10	0.5 - 1.5	100+

5.1.2 On MacOS

We need to make sure that we can run nodes on our baseline model for comparison, which is a MacBook Pro early 2015. Listing 5.1 shows three important pieces of information once we run the command to run a light node on the Ropsten testnet. Line 1 informs us when the command started, here at "`15:43:44`". Line 3 indicates that the block synchronization process has started (once it found a full node peer). Finally, line 7 shows that we are completely synchronized with the blockchain as it only imports the last new block header denoted with "`count=1`". If the node is not totally in sync, we would have a count greater than 1.
Let us note two interesting points: It took less than a second to find a full node peer to download the headers from. This can sometimes take several minutes or even hours if unlucky. The output also shows that it is trying to connect to other peers if possible; see line 5. The synchronization process itself lasted approximately 7 minutes, as the timestamp shows (lines 3 and 7).

Listing 5.1 – MacOS - Geth command line result.

```
1  INFO [09-13|15:43:44.739] Starting Geth on Ropsten testnet...
2  ...
3  INFO [09-13|15:43:46.413] Block synchronisation started
4  ...
5  INFO [09-13|15:44:20.324] Looking for peers                 peercount=1 tried=2
     static=0
6  ...
7  INFO [09-13|15:50:39.925] Imported new block headers        count=1    elapsed
     =8.873ms   number=11,027,850 hash=369c83..63f314
```

Result: running a node is feasible on both public and permissioned network.

5.1.3 On Raspberry Pi

With a similar output as on the MacBook Pro, we have the following in Listing 5.2: it took less than a second to find a full node peer to download the headers from. The output also shows that it is trying to connect to other peers if possible, but the count is still 1; see line 5. The synchronization process itself lasted approximately 26 minutes, as the timestamp shows (lines 3 and 7).

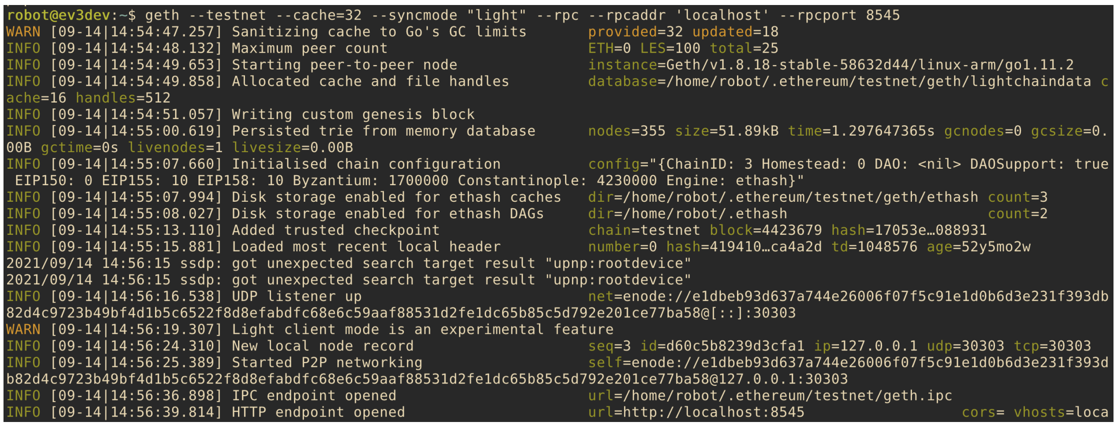
Listing 5.2 – RPI - Geth command line result.

```
1  INFO [09-13|15:46:59.825] Starting Geth on Ropsten testnet...
2  ...
3  INFO [09-13|15:47:02.569] Block synchronisation started
4  ...
5  INFO [09-13|15:47:22.366] Looking for peers                 peercount=1 tried=2
     static=0
6  ...
7  INFO [09-13|16:12:57.080] Imported new block headers        count=1    elapsed
     =113.695ms number=11,027,938 hash=84b909..ab35c3
```

Result: running a node is feasible on both public and permissioned network.

5.1.4 On EV3

With the minimum requirements to run a Geth client mentioned above, we strongly doubt that running a public or permissioned Ethereum node on the EV3 would be possible. Indeed, with its sole core and 64Mb of RAM, its computing resources fall far behind the recommended requirements. Nevertheless, we still conducted both implementations, and here are the results. After successfully installing Geth framework on the EV3, we tried to run a light node to connect to the Ropsten testnet, and we allocated 32Mb of caching memory since the hardware has a maximum of 64Mb. Unfortunately, and after several hours of waiting, the light node never tries to find a peer node, nor does it start synchronizing. The same happens when we run a light node in a private network setting. Figure 5.1 shows where it got stuck in the terminal.



```
robot@ev3dev:~$ geth --testnet --cache=32 --syncmode "light" --rpc --rpcaddr 'localhost' --rpcport 8545
WARN [09-14|14:54:47.257] Sanitizing cache to Go's GC limits       provided=32 updated=18
INFO [09-14|14:54:48.132] Maximum peer count                       ETH=0 LES=100 total=25
INFO [09-14|14:54:49.653] Starting peer-to-peer node               instance=Geth/v1.8.18-stable-58632d44/linux-arm/go1.11.2
INFO [09-14|14:54:49.858] Allocated cache and file handles         database=/home/robot/.ethereum/testnet/geth/lightchaindata c
ache=16 handles=512
INFO [09-14|14:54:51.057] Writing custom genesis block
INFO [09-14|14:55:00.619] Persisted trie from memory database      nodes=355 size=51.89kB time=1.297647365s gcnodes=0 gcsize=0.
00B gctime=0s livenodes=1 livesize=0.00B
INFO [09-14|14:55:07.660] Initialised chain configuration          config="{ChainID: 3 Homestead: 0 DAO: <nil> DAOSupport: true
 EIP150: 0 EIP155: 10 EIP158: 10 Byzantium: 1700000 Constantinople: 4230000 Engine: ethash}"
INFO [09-14|14:55:07.994] Disk storage enabled for ethash caches   dir=/home/robot/.ethereum/testnet/geth/ethash count=3
INFO [09-14|14:55:08.027] Disk storage enabled for ethash DAGs     dir=/home/robot/.ethash                       count=2
INFO [09-14|14:55:13.110] Added trusted checkpoint                 chain=testnet block=4423679 hash=17053e…088931
INFO [09-14|14:55:15.881] Loaded most recent local header          number=0 hash=419410…ca4a2d td=1048576 age=52y5mo2w
2021/09/14 14:56:15 ssdp: got unexpected search target result "upnp:rootdevice"
2021/09/14 14:56:15 ssdp: got unexpected search target result "upnp:rootdevice"
INFO [09-14|14:56:16.538] UDP listener up                          net=enode://e1dbeb93d637a744e26006f07f5c91e1d0b6d3e231f393db
82d4c9723b49bf4d1b5c6522f8d8efabdfc68e6c59aaf88531d2fe1dc65b85c5d792e201ce77ba58@[::]:30303
WARN [09-14|14:56:19.307] Light client mode is an experimental feature
INFO [09-14|14:56:24.310] New local node record                    seq=3 id=d60c5b8239d3cfa1 ip=127.0.0.1 udp=30303 tcp=30303
INFO [09-14|14:56:25.389] Started P2P networking                   self=enode://e1dbeb93d637a744e26006f07f5c91e1d0b6d3e231f393d
b82d4c9723b49bf4d1b5c6522f8d8efabdfc68e6c59aaf88531d2fe1dc65b85c5d792e201ce77ba58@127.0.0.1:30303
INFO [09-14|14:56:36.898] IPC endpoint opened                      url=/home/robot/.ethereum/testnet/geth.ipc
INFO [09-14|14:56:39.814] HTTP endpoint opened                     url=http://localhost:8545                  cors= vhosts=loca
```

Figure 5.1 – EV3 - Geth command line result.

Result: running a node is **NOT** feasible on both public and permissioned network.

We conclude that running a node is not feasible on this very low-powered device. Nevertheless, there is still a way to interact with the Ethereum blockchain through remote procedural calls (RPC). Indeed, we can target the RPC endpoints that we enabled on a running node by accessing through HTTP from the EV3, and interacting with a smart contract that lies on the blockchain thanks to Ethereum libraries.
For this purpose, we build an API in Golang since the most widely used implementation of the Ethereum protocol is implemented in this language. First, we need to bind our "CoolNumberContract" smart contract to our Go API into easy-to-use Go packages with the command `abigen --sol=CoolNumberContract.sol --pkg=main --out=coolNumber.go`. The output "`coolNumber.go`" gives us utility functions, types and variables that we can use in our Go API to interact with the smart contract.
Then, in our Go API called "`main.go`" in Listing 5.3, we create an RPC connection to a remote node (running on our RPI) and then connect to the already deployed smart contract thanks to its address on the blockchain. Our API prints the last "cool number" stored on the blockchain (line 30). Furthermore, we can also print the last block header at the instant we run this program (line 34) to give extra proof that we are indeed connected to the blockchain.

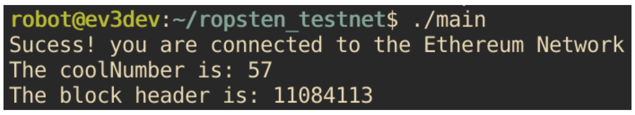
Listing 5.3 – Go API - main.go

```go
package main

import (
    "context"
    "fmt"
    "log"

    "github.com/ethereum/go-ethereum/accounts/abi/bind"
    "github.com/ethereum/go-ethereum/common"
    "github.com/ethereum/go-ethereum/ethclient"
)

func main() {
    // Create an IPC based RPC connection to a remote node (here our RPI)
    conn, err := ethclient.Dial("http://192.168.1.110:8545")
    if err != nil {
        log.Fatalf("Failed to connect to the Ethereum client: %v", err)
    } else {
        fmt.Println("Sucess! you are connected to the Ethereum Network")
    }

    //Get the contract address from hex string
    addr := common.HexToAddress("0xB202CCCd3b66F63f567D40d6e22C1C1BAB627824")

    // Bind to an already deployed contract and print coolNumber
    contract, err := NewCoolNumberContract(addr, conn)
    if err != nil {
        log.Fatalf("Failed to instantiate coolNumber contract: %v", err)
    } else {
        coolNumber, _ := contract.CoolNumber(&bind.CallOpts{})
        fmt.Println("The coolNumber is:", coolNumber)
    }

    //Print block header
    header, err := conn.HeaderByNumber(context.Background(), nil)
    if err != nil{
        log.Fatal(err)
    } else {
        fmt.Println("The block header is:",header.Number.String())
    }
}
```

Finally, we can compile our API and specify the Go environment with "`GOOS=linux GOARCH=arm GOARM=5 go build main.go coolNumber.go`" so that the binary can run on the EV3 device. The program is then run with the command "`./main`". Figure 5.2 shows a successful RPC connection on the EV3. We can verify that we are indeed interacting with the smart contract as it displays 57 as the latest number on the EV3 (Figure 5.2a) and the web3 interface (Figure 5.2b. In addition, the latest block header at the runtime was "11084113" which can be verified on the RPI node output:

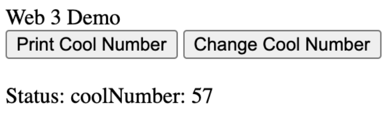
```
INFO [09-22|10:47:21.414] Imported new block headers      count=1 elapsed
=23.343ms number=11,084,113 hash=1ab7ce..c7e43e
```

We thus demonstrate that it is still possible to connect to the blockchain and interact with a smart contract despite the device constraints.



```
robot@ev3dev:~/ropsten_testnet$ ./main
Sucess! you are connected to the Ethereum Network
The coolNumber is: 57
The block header is: 11084113
```

(a) Execution of Go API on EV3.



Web 3 Demo

Status: coolNumber: 57

(b) Output of Cool Number dApp.

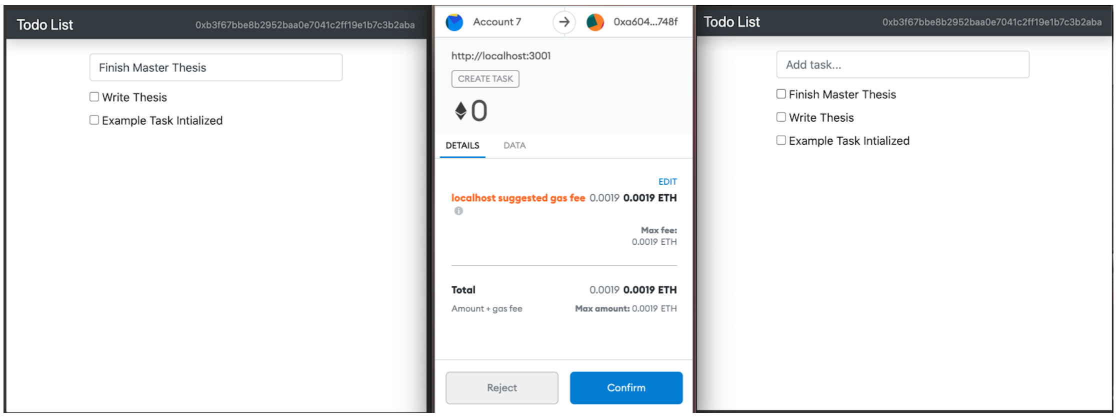
Figure 5.2 – RPC demonstration on EV3.

5.2 Smart Contract Deployment

As an additional testimony of a good setup in both implementations, we decided to interact with smart contracts that we deployed on both networks. We will briefly explain how we deploy and interact with the contracts in both scenarios through the use of web3.js libraries.

5.2.1 On a Private Network

We build a simple dApp called "ToDo List" that allows us to create and toggle tasks that are done through a web interface. Its related smart contract was deployed in the private network thanks to the Truffle framework. The web interface was implemented in Node.js and has web3 libraries imported to interact with our local Ethereum node. To handle transactions through the web interface, we linked our private account to Metamask. Figure 5.3 shows the flow of the dApp and proves that the private network setup is correct and works as intended.



Todo List 0xb3f67bbe8b2952baa0e7041c2ff19e1b7c3b2aba

Finish Master Thesis

☐ Write Thesis

☐ Example Task Intialized

Account 7 → 0xa604...748f

http://localhost:3001

CREATE TASK

0

localhost suggested gas fee 0.0019 **0.0019 ETH**

Max fee: 0.0019 ETH

Total 0.0019 **0.0019 ETH**

Amount + gas fee Max amount: 0.0019 ETH

Todo List 0xb3f67bbe8b2952baa0e7041c2ff19e1b7c3b2aba

Add task...

☐ Finish Master Thesis

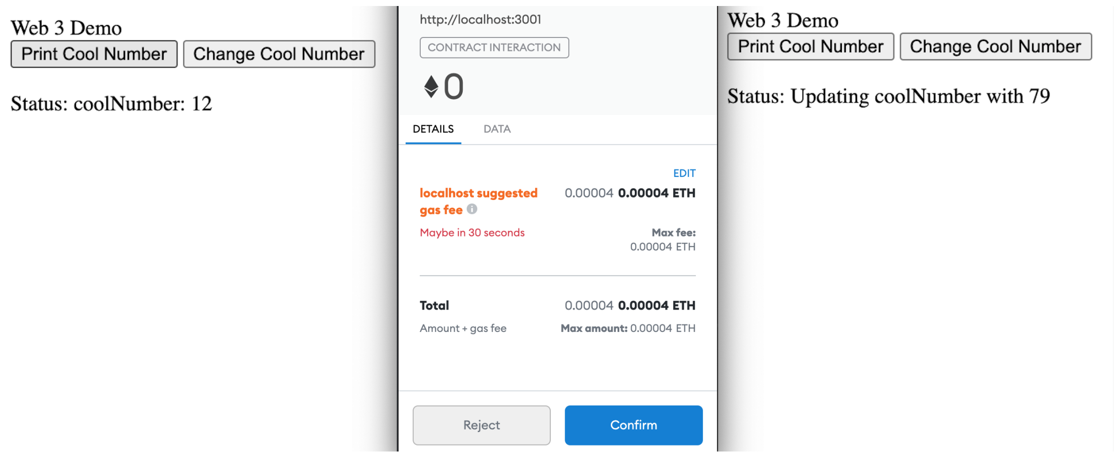
☐ Write Thesis

☐ Example Task Intialized

Figure 5.3 – ToDo List dApp flow.

5.2.2 On Ropsten Testnet

On the public testnet, we decided to build a simpler dApp for ease of use throughout the implementation. This dApp will print the current number stored on the blockchain and can also set a new random number. Its smart contract was deployed in the network through the Remixe IDE (in order to diversify the use of frameworks). The web interface was also built in Node.js, and we use web3 functionalities to interact with our smart contract. Again, we imported our public account to Metamask and acquired some fake ETH through the Ropsten faucet. Figure 5.4 shows the flow of the dApp, indicating that our light node setup works as intended.


Web 3 Demo
Print Cool Number | Change Cool Number
Status: coolNumber: 12

http://localhost:3001
CONTRACT INTERACTION
0
DETAILS DATA
EDIT
localhost suggested gas fee 0.00004 **0.00004 ETH**
Maybe in 30 seconds **Max fee:** 0.00004 ETH
Total 0.00004 **0.00004 ETH**
Amount + gas fee **Max amount:** 0.00004 ETH
Reject Confirm

Web 3 Demo
Print Cool Number | Change Cool Number
Status: Updating coolNumber with 79

Figure 5.4 – Cool number dApp flow.

5.3 Performance of Devices

In this section, we will see how the performance of blockchain nodes differs between the MackBook Pro that will serve as our baseline and the RPI. We will first analyze data in the private network scenario and then on the public Ropsten testnet scenario.

5.3.1 In Private Network

Table 5.2 shows preliminary results of a private blockchain with two full nodes—one on the MacBook Pro and the other one on the RPI. We decided to analyze the data from a window of 30 minutes since the blockchain initialization. Furthermore, the RPI was linked to the local network with an ethernet cable, whereas the MacBook Pro was connected through WiFi. As the collected data come from the beginning point of the private blockchain, the result can be significantly different from data that would have been collected after thousands of blocks. Therefore, this analysis will put an emphasis on the system node metrics. Nevertheless, let us have a quick look at blockchain-related metrics.
The number of Goroutines in both machines is relatively similar. Regarding data rates, the level of write in the blockchain is higher in the MacBook Pro, probably due to higher computing power. The level of reads, on the other hand, is lower than on the RPI. Maybe this is due to the difference in connection setups, as it is probably faster to read through an ethernet cable. Similarly, the same observation can be made with the total memory of sessions. Notice that the compaction read/write on both data rates and total sessions metrics are close to zero on both machines as we are at the starting of the blockchain. The persistent size of data is less on

the MacBook Pro, indicating that the RPI caches more data.

Table 5.2 – Private Full node performance comparison between MacOS and RPI.

Type of metrics	**Metrics**	**Sub metrics**	**MacOS (30min)**	**RPI (30min)**
Blockchain	Goroutines (min, max) $[int]$	-	94, 121	96, 127
	Data Rates (avg) $[B/s]$	leveldb read	105	270
		leveldb write	29.9	19.1
		compaction read	0	0
		compaction write	0	0
	Session Totals (avg) $[kB]$	leveldb read	22	40
		leveldb write	48.3	2.84
		compaction read	0	0
		compaction write	0	0
	Persistent Size (max) $[kB]$	-	64	97.8
P2P	#Peers $[int]$	-	1	1
	Traffic (avg) $[kB/s]$	Ingress	1.47	1.51
		Egress	2.25	2.29
System node	CPU (max) [%]	-	8	46
	Memory (avg) $[MB]$	allocated	0.003	0.003
		used	394	71.2
		held	424	75.6
	Network (avg)	ETH Ingress Data Rate $[B/s]$	3	5.12
		ETH Egress Data Rate $[B/s]$	4.42	6.01
		ETH Ingress Traffic $[kB]$	3.32	4.99
		ETH Egress Traffic $[kB]$	1.95	4.34

The most notable differences can be seen in the system node metrics. The CPU usage by Geth is as high as 46% on the RPI and 8% on the MacBook Pro. Again, because the RPI has fewer computing resources, it solicits more CPU power. Another significant distinction is memory usage. Since the MacBook Pro has 8GB RAM, it can use more memory than the RPI. Therefore, the memory used and held is lower on the RPI. This memory usage probably has an impact on the persistent size.
The blockchain private network serves more to the RPI as it has a higher ingress/ingress traffic and data rate. It is hard to conclude the exact reason behind the latter and the former as we only have two nodes in our private network and, moreover, two full nodes. It might be due to the fact that the MacBook Pro used more memory than the RPI.
Overall, the RPI is more than capable of running a full node on a private blockchain. Compared to the MacBook Pro, there is no big difference in the metrics apart from CPU and memory usage.

5.3.2 In Ropsten Testnet

Table 5.3 shows an overview of the metrics that we were able to collect during the synchronization process on the testnet. The duration of this process differs from the two hardware: around 7 minutes for the MacBook Pro and 26 minutes for the RPI. The metrics are categorized into the three types we defined in the previous chapter. Some data are averaged when it makes sense, whereas for others, only the extremum was taken. An important point to note is the different caching that was used before we ran the nodes. We used 256Mb of caching on the RPI (that has 1GB RAM) and 2048Mb on the MacBook Pro (with 8GB RAM). Therefore, we have a ratio of 0.256 between caching and RAM on both devices. Moreover, both machines were able to connect to one full node peer.
The synchronization process is much faster on the MacBook Pro as it has more computer power, and we allowed 1Gb of caching memory, thus helping the process. The number of Goroutines is similar in both machines. Because of the specified caching, metrics such as data rates, total session, and persistent size are significantly less on the MacBook Pro.
A noticeable difference lies in CPU and memory usage. As we are running a light node on multi-core devices, it is possible to execute multiple instructions in parallel, hence the CPU usage over 100%. As we have four cores on both devices, the maximum percentage of CPU is 400%. It went as high as 400% on the RPI, thus reaching the maximum and 362% on the MacBook Pro. Regarding the memory, since we set a higher caching on the MacBook Pro, more memory is used compared to the RPI. The compaction time is notably greater on the RPI as it has fewer computing resources. Regarding the network traffic, the MacBook Pro has greater bandwidth and therefore manage to have a better ingress/egress data rate.
In general, the MacBook Pro outperforms the RPI by far, as it has an SSD, and bigger RAM, thus allowing greater caching. These hardware specifications can significantly improve the speed of synchronization. Nevertheless, despite its hardware limitation, the RPI is capable of running a light node without issues, provided that we specify the right caching when running the light node.

Table 5.3 – Light node syncing performance comparison on Ropsten testnet between MacOS and RPI.

Type of metrics	**Metrics**	**Sub metrics**	**MacOS (7min)**	**RPI (26min)**
Blockchain	Goroutines (min, max) $[int]$	-	109, 178	109, 179
	Data Rates (avg) $[kB/s]$	leveldb read	1.56	95
		leveldb write	528	297
		compaction read	1.46	62.6
		compaction write	2.92	145
	Session Totals (avg) $[MB]$	leveldb read	0.058	151
		leveldb write	245	474
		compaction read	0.051	95
		compaction write	0.103	229
	Persistent Size (max) $[MB]$	-	0.051	134
P2P	#Peers $[int]$	-	1	1
	Traffic (avg) $[kB/s]$	Ingress	274	50.2
		Egress	0.833	0.205
System node	CPU (max) [%]	-	362	400
	Memory (avg) $[MB]$	allocated	0.130	0.027
		used	620	306
		held	727	429
	Storage (avg) $[ms]$	-	1.91	43
	Network (avg)	In Packet Tot $[p/s]$	9.86	2.34
		Out Packet Tot $[p/s]$	9.80	2.28
		In Traffic Tot $[kB/s]$	473	85.6
		Out Traffic Tot $[kB/s]$	238	57.8

5.4 BIoT Use Case

Now that we show that it is possible to interact with the blockchain despite the limitation of the EV3, we can implement our IoT use case involving the EV3 as a sensor device and the RPI as a data collector, which also has a light node running. We will validate the implementation in the first part and then show the performance of the RPI during repeated contract interaction on the blockchain.

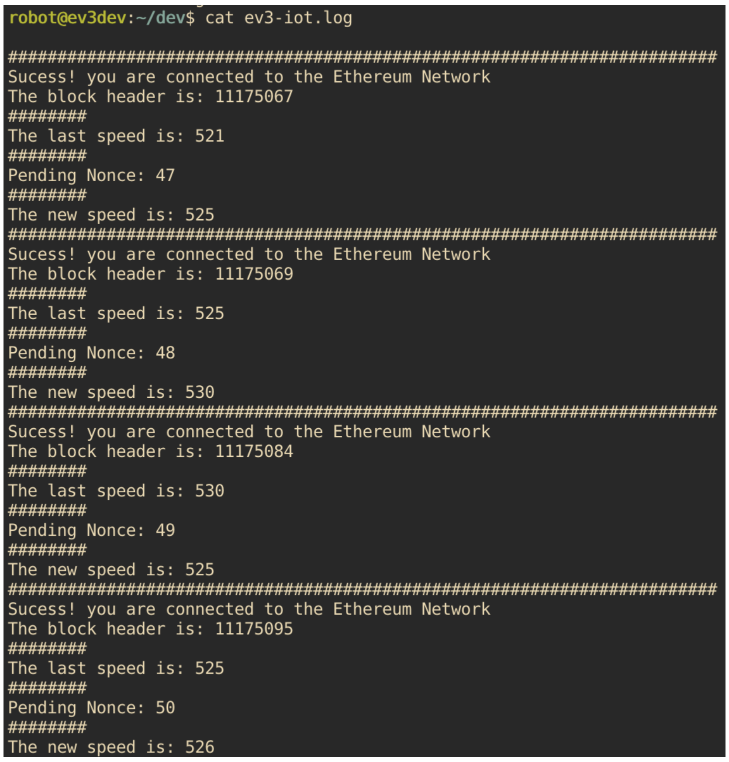
5.4.1 Implementation Validation

As we cannot run a light node on the EV3, we must interact with the blockchain through RPC calls. Similar to the way we interact with the "CoolNumber" smart contract, we build an API in Golang and use Geth libraries to connect our EV3 to a node that is running on the RPI and bind to a newly deployed "EV3SpeedMonitor" contract (see Appendix B3). However, as we are dealing with paid transactions, in this specific case storing data to the blockchain, we need to also bind to an account as shown in line 5 in Listing 5.4 (see the full script in Appendix B4). We can then store the collected data from the EV3 motor to the smart contract, line 17.

Listing 5.4 – BIoT use case API snippet - ev3-iot.go

```
1  func main() {
2      ...
3
4      //-->Bind to an account
5      privateKey, err := crypto.HexToECDSA("2
         c88557feb65fa683290492e7a91b67dae8876260dba0972f98b4a5578c942d9")
6      if err != nil {
7          log.Fatal(err)
8      }
9
10     ...
11     auth, _ := bind.NewKeyedTransactorWithChainID(privateKey, big.NewInt(3))
12
13     //-->Write in the smart contract
14     metric := make(chan uint32) //use metric channel for goroutines sync
15     go monitorDeviceRun(metric)
16     m1 := <- metric //receive value from metric channel
17     speedMonitor.Update(auth, m1)
18 }
19
20 func monitorDeviceRun(metric chan uint32) {
21     ...
22
23     // while #duration seconds have not passed
24     for i := 0; i < duration; i += 5 {
25         // get some metric from motor, e.g current speed
26         value, _ := outA.Speed()
27         result = uint32(value)
28         time.Sleep(5 * time.Second)
29     }
30     metric <- result //send value to metric channel
31 }
```

Once the data is correctly stored on the blockchain, we can automate this process of data storing by running the script regularly, for example, every two minutes. This can be done thanks to a cron job: `*/2 * * * * /home/robot/ev3-iot >>/home/robot/ev3-iot.log`. We also store the logs of the API to see what data are stored on the blockchain as seen in Figure 5.5. After each time the script is run, we see that the block header and the pending nonce are incremented, indicating that we are indeed interacting with the blockchain and the smart contract correctly. In addition, we also see that the "last speed" fetch from the blockchain is always the "new speed" that is to be stored in the previous script execution. This proves that our API is working as intended.



```
robot@ev3dev:~/dev$ cat ev3-iot.log

########################################################################
Sucess! you are connected to the Ethereum Network
The block header is: 11175067
########
The last speed is: 521
########
Pending Nonce: 47
########
The new speed is: 525
########################################################################
Sucess! you are connected to the Ethereum Network
The block header is: 11175069
########
The last speed is: 525
########
Pending Nonce: 48
########
The new speed is: 530
########################################################################
Sucess! you are connected to the Ethereum Network
The block header is: 11175084
########
The last speed is: 530
########
Pending Nonce: 49
########
The new speed is: 525
########################################################################
Sucess! you are connected to the Ethereum Network
The block header is: 11175095
########
The last speed is: 525
########
Pending Nonce: 50
########
The new speed is: 526
```

Figure 5.5 – BIoT use case API log output.

5.4.2 Performance of BIoT Use Case

The following Table 5.4 shows the performance of the RPI in which a light node Geth client is running. This performance shows how the device is handled under repeated smart contract interaction. The collected metrics are average over a period of one hour, during which the data are uploaded from the EV3 every two minutes. Hence, there was a total of 30 contract interactions. Interacting with a smart contract through a light node is not as computer-intensive as synchronizing. Furthermore, the light node is not responsible for creating new blocks; therefore, the blockchain and P2P metrics are not particularly important. On the other hand, the CPU usage indicates that the RPI can handle the interactions well, at 78.5%. On the

EV3, the average CPU usage over this one hour period is 35.25%.

Table 5.4 – BIoT use case RPI performance on Ropsten testnet.

Type of metrics	Metrics	Sub metrics	RPI (1hour)	EV3 (1hour)
Blockchain	Goroutines (min, max) $[int]$	-	115, 133	-
	Data Rates (avg)	leveldb read $[B/s]$	10.8	-
		leveldb write $[kB/s]$	1.16	-
		compaction read	0	-
		compaction write	0	-
	Session Totals (avg) $[kB]$	leveldb read	94	-
		leveldb write	62.4	-
		compaction read	0	-
		compaction write	0	-
	Persistent Size (max) $[kB]$	-	54.6	-
P2P	#Peers $[int]$	-	2	-
	Traffic (avg) $[B/s]$	Ingress	944	-
		Egress	54.8	-
System node	CPU (avg) [%]	System	78.5	35.25
		I/O Wait	400	-
		Geth	10.9	-
	Memory (avg) $[MB]$	allocated	0.003	-
		used	148	-
		held	181	-
	Disk (avg) $[kB/s]$	read	2.67	-
		write	9.82	-
	Network (avg)	In Packet Tot $[p/s]$	0.468	-
		Out Packet Tot $[p/s]$	0.303	-
		In Traffic Tot $[kB/s]$	1.38	-
		Out Traffic Tot $[B/s]$	22	-

6 Conclusion

6.1 Summary

The popularity of blockchain has become increasingly popular over the past decade. This novel technology enabled an immutable and data tamper-proof distributed system. It can be seen as a digital ledger of duplicated transactions and distributed among the participants of the blockchain network. Some blockchains go beyond this simple definition of the digital ledger; for example, Ethereum also stores the state machine in a distributed manner, thus enabling smart contracts, a program that self-executes when conditions are met. With a wide range of applications, many have taken an interest in integrating blockchain in existing technology, such as IoT, to alleviate trust and security concerns.
IoT uses mainly low-powered devices with low storage capacity, thus making it a challenge to integrate blockchain. Given the constrained resources in IoT and its challenge, there exists several ongoing research regarding the architectural integration of blockchain in IoT. Nevertheless, few of them address the actual implementation in real IoT devices. Therefore, we focused our work on implementing a blockchain in real IoT scenarios, particularly Ethereum, and investigated the performance of different nodes on low-powered devices such as the RPI or the EV3.

6.1.1 Findings and Implications

All the scripts and synthetic data sets used in the thesis can be found on Zenodo repository [Siow, 2021a, Siow, 2021b]. We designed two scenarios with Geth framework to address the first question in which we interact with an Ethereum blockchain in public and permissioned ways. In the latter, which uses PoA consensus, running a full node on the RPI is possible as it is less energy demanding than its public counterpart. The use of private blockchain is a good option in IoT systems as it can provide scalability. Nevertheless, we wanted to show how we can run a node on a public blockchain to be closer to a real situation. In this scenario, we ran a light node on the public testnet called Ropsten, which uses PoW consensus. Therefore, it is the best like-for-like representation of Ethereum. Only the light node could be implemented

on the RPI as both "full" and "fast" synchronization are not possible on a low-memory device. Furthermore, the EV3 could not run any node on both scenarios, as it does not meet the minimum requirements to run a Geth client. In addition to the two scenarios, we designed a blockchain-integrated IoT system to emulate a real-world use case entitled BIoT. The latter shows how it is possible to store data from a low-powered IoT device such as the EV3 to a public blockchain through the use of smart contract. In this use case, the EV3 communicates through RPC with a RPI in which a light node is implemented.
Regarding the second research question, we listed the most important metrics to gather to analyze performance properly. We grouped the metrics into three categories: blockchain-specific, P2P layer, and system node. To help understand and visualize the metrics, we dumped the data from the two scenarios and the BIoT use case into a time-series database called InfluxDB and attached a monitoring platform called Grafana.
Finally, we analyzed the performance of the RPI in both private and public networks. A MacBook Pro early 2015 served as a baseline. In the private scenario, the RPI can run a full node on a private blockchain. Compared to the MacBook Pro, there is no significant difference in the metrics apart from CPU and memory usage. However, in the public scenario, the MacBook pro generally outperforms the RPI because of hardware limitations. Nevertheless, the RPI is capable of running a light node without issues, provided that we specify the right caching.

6.1.2 Final Conclusion

With blockchain becoming increasingly popular, we decided to implement and assess blockchain nodes' performance in IoT. To this end, we have successfully integrated a private and public Ethereum blockchain node on a RPI. We pushed the study further with the BIoT use case, which emulates a real-world use case scenario involving low-powered IoT devices. In this use case, we showed that although the EV3 cannot run any node, it can still interact with a public blockchain through RPC connection. In all of our implementation, we collected important performance-related metrics and analyzed them thanks to visualization tools. We learned from the metrics collected that the RPI can run a full node well in a private blockchain. However, only a light node can be implemented in a public blockchain because of its hardware limitation. Overall, blockchain is a suitable technology to be integrated into IoT, as it provides security and trust among the network participants.

References

[Agarwal and Pal, 2020] Agarwal, V. and Pal, S. (2020). Blockchain meets iot: A scalable architecture for security and maintenance. *2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS).*

[Aldweesh et al., 2021] Aldweesh, A., Alharby, M., Mehrnezhad, M., and van Moorsel, A. (2021). The opbench ethereum opcode benchmark framework: Design, implementation, validation and experiments. *Performance Evaluation*, 146.

[Ali et al., 2021] Ali, O., Jaradat, A., Kulakli, A., and Abuhalimeh, A. (2021). A comparative study: Blockchain technology utilization benefits, challenges and functionalities. *IEEE Access*, 9.

[Alkhader et al., 2020] Alkhader, W., Alkaabi, N., Salah, K., Jayaraman, R., Arshad, J., and Omar, M. (2020). Blockchain-based traceability and management for additive manufacturing. *IEEE Access*, 8.

[Alphand et al., 2018] Alphand, O., Amoretti, M., Claeys, T., Dall'Asta, S., Duda, A., Ferrari, G., Rousseau, F., Tourancheau, B., Veltri, L., and Zanichelli, F. (2018). Iotchain: A blockchain security architecture for the internet of things. *2018 IEEE Wireless Communications and Networking Conference (WCNC).*

[Alrubei et al., 2021] Alrubei, S., Ball, E., and Rigelsford, J. (2021). Securing iot-blockchain applications through honesty-based distributed proof of authority consensus algorithm. *2021 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA).*

[Antonopoulos and Wood, 2018] Antonopoulos, A. M. and Wood, G. (2018). *Mastering Ethereum : building smart contracts and DApps.* O'Reilly Media, Inc., first edition.

[Arif et al., 2020] Arif, S., Khan, M. A., Rehman, S. U., Kabir, M. A., and Imran, M. (2020). Investigating smart home security: Is blockchain the answer? *IEEE Access*, 8.

[Atasen and Ustunel, 2019] Atasen, K. and Ustunel, H. (2019). Designing a secure iot network by using blockchain. *2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT).*

[Chase and MacBrough, 2018] Chase, B. and MacBrough, E. (2018). Analysis of the xrp ledger consensus protocol. *ArXiv*, abs/1802.07242.

[Choi et al., 2018] Choi, S. S., Burm, J. W., Sung, W., Jang, J. W., and Reo, Y. J. (2018). A blockchain-based secure iot control scheme. *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*.

[Dinh et al., 2017] Dinh, T. T. A., Wang, J., Chen, G., Liu, R., Ooi, B. C., and Tan, K.-L. (2017). Blockbench. *Proceedings of the 2017 ACM International Conference on Management of Data*.

[Dittmann and Jelitto, 2019] Dittmann, G. and Jelitto, J. (2019). A blockchain proxy for lightweight iot devices. *2019 Crypto Valley Conference on Blockchain Technology (CVCBT)*.

[Ethereum, 2021] Ethereum (2021). Ethereum's. Available: https://ethereum.org/en/. Accessed on Jul. 28, 2021. [Online].

[Ethereum Concepts, 2021] Ethereum Concepts (2021). Ethereum's Concepts. Available: https://ethereum.org/en/developers/docs/intro-to-ethereum/. Accessed on Jul. 29, 2021. [Online].

[Ethereum White Paper, 2021] Ethereum White Paper (2021). Ethereum White Paper. Available: https://ethereum.org/en/whitepaper/. Accessed on Jul. 29, 2021. [Online].

[evm illustrated, 2021] evm illustrated (2021). Ethereum EVM Illustrated. Available: https://takenobu-hs.github.io/downloads/ethereum_evm_illustrated.pdf. Accessed on Aug. 2, 2021. [Online].

[Fakhri and Mutijarsa, 2018] Fakhri, D. and Mutijarsa, K. (2018). Secure iot communication using blockchain technology. *2018 International Symposium on Electronics and Smart Devices (ISESD)*.

[Fernandez-Carames and Fraga-Lamas, 2018] Fernandez-Carames, T. M. and Fraga-Lamas, P. (2018). A review on the use of blockchain for the internet of things. *IEEE Access*, 6.

[Gong et al., 2020] Gong, X., Liu, E., and Wang, R. (2020). Blockchain-based iot application using smart contracts: Case study of m2m autonomous trading. *2020 5th International Conference on Computer and Communication Systems (ICCCS)*.

[Hjalmarsson et al., 2018] Hjalmarsson, F. P., Hreioarsson, G. K., Hamdaqa, M., and Hjalmtysson, G. (2018). Blockchain-based e-voting system. *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*.

[Huh et al., 2017] Huh, S., Cho, S., and Kim, S. (2017). Managing iot devices using blockchain platform. *2017 19th International Conference on Advanced Communication Technology (ICACT)*.

[Hyperledger Fabric's About, 2021] Hyperledger Fabric's About (2021). Hyperledger Fabric. Available: Available:https://www.hyperledger.org/use/fabric. Accessed on Jul. 28, 2021. [Online].

[ipfs, 2021] ipfs (2021). IPFS. Available: https://ipfs.io/. Accessed on Jul. 30, 2021. [Online].

[Jain, 1991] Jain, R. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* John Wiley & Sons, Inc., USA, Michigan, illustrated edition.

[Jaoude and Saade, 2019] Jaoude, J. A. and Saade, R. G. (2019). Blockchain applications – usage in different domains. *IEEE Access*, 7.

[Kravitz and Cooper, 2017] Kravitz, D. W. and Cooper, J. (2017). Securing user identity and transactions symbiotically: Iot meets blockchain. *2017 Global Internet of Things Summit (GIoTS).*

[Lego, 2013] Lego (2013). Ev3 mindstorm. Available: https://www.lego.com/de-ch/product/ev3-intelligent-brick-45500. Accessed on Aug. 28, 2021. [Online].

[Liang, 2020] Liang, Y.-C. (2020). *Blockchain for Dynamic Spectrum Management.* Springer Singapore, Singapore.

[Litecoin, 2021] Litecoin (2021). Litecoin's. Available: https://litecoin.com/en/. Accessed on Jul. 28, 2021. [Online].

[McGhin et al., 2019] McGhin, T., Choo, K.-K. R., Liu, C. Z., and He, D. (2019). Blockchain in healthcare applications: Research challenges and opportunities. *Journal of Network and Computer Applications*, 135.

[Mettler, 2016] Mettler, M. (2016). Blockchain technology in healthcare: The revolution starts here. *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom).*

[Mouser, 2018] Mouser (2018). Rpi 3b+. Available: https://www.mouser.ch/new/raspberry-pi/raspberry-pi-3-bplus/. Accessed on Aug. 28, 2021. [Online].

[Nakamoto, 2008] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Bitcoin.*

[Novo, 2018] Novo, O. (2018). Blockchain meets iot: An architecture for scalable access management in iot. *IEEE Internet of Things Journal*, 5.

[Pongnumkul et al., 2017] Pongnumkul, S., Siripanpornchana, C., and Thajchayapong, S. (2017). Performance analysis of private blockchain platforms in varying workloads. *2017 26th International Conference on Computer Communication and Networks (ICCCN).*

[Preethi Kasireddy, 2021] Preethi Kasireddy (2021). What do we mean by blockchains are trustless. Available: https://www.preethikasireddy.com/post/what-do-we-mean-by-blockchains-are-trustless. Accessed on Jul. 25, 2021. [Online].

[Puthal and Mohanty, 2019] Puthal, D. and Mohanty, S. P. (2019). Proof of authentication: Iot-friendly blockchains. *IEEE Potentials*, 38.

[Puthal et al., 2019] Puthal, D., Mohanty, S. P., Nanda, P., Kougianos, E., and Das, G. (2019). Proof-of-authentication for scalable blockchain in resource-constrained distributed systems. *2019 IEEE International Conference on Consumer Electronics (ICCE)*.

[Ray et al., 2021] Ray, P. P., Dash, D., Salah, K., and Kumar, N. (2021). Blockchain for iot-based healthcare: Background, consensus, platforms, and use cases. *IEEE Systems Journal*, 15.

[Rifi et al., 2017] Rifi, N., Rachkidi, E., Agoulmine, N., and Taher, N. C. (2017). Towards using blockchain technology for iot data access protection. *2017 IEEE 17th International Conference on Ubiquitous Wireless Broadband (ICUWB)*.

[Ripple's About, 2021] Ripple's About (2021). Ripple's. Available: https://ripple.com/. Accessed on Jul. 28, 2021. [Online].

[Saini, 2021] Saini, V. (2021). Getting Deep Into EVM: How Ethereum Works Backstage. Available: https://hackernoon.com/getting-deep-into-evm-how-ethereum-works-backstage-ac7efa1f0015. Accessed on Aug. 2, 2021. [Online].

[Sanju et al., 2018] Sanju, S., Sankaran, S., and Achuthan, K. (2018). Energy comparison of blockchain platforms for internet of things. *2018 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS)*.

[Sankaran et al., 2018] Sankaran, S., Sanju, S., and Achuthan, K. (2018). Towards realistic energy profiling of blockchains for securing internet of things. *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*.

[Schneier and Sutherland, 1995] Schneier, B. and Sutherland, P. (1995). *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, Inc., USA, 2nd edition.

[Sharma and Gondhi, 2018] Sharma, C. and Gondhi, N. K. (2018). Communication protocol stack for constrained iot systems. *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*.

[Siow, 2021a] Siow, R. (2021a). Biot synthetic data set. Available: https://doi.org/10.5281/zenodo.5603282. [Online].

[Siow, 2021b] Siow, R. (2021b). ryansiow/mt_biot: Mt script referencing. Available: https://doi.org/10.5281/zenodo.5595105. [Online].

[solidity, 2021] solidity (2021). Solidity. Available: https://docs.soliditylang.org/en/v0.8.7/. Accessed on Aug. 28, 2021. [Online].

[Swarm, 2021] Swarm (2021). Swarm. Available: https://www.ethswarm.org/. Accessed on Jul. 30, 2021. [Online].

[Szabo, 1997] Szabo, N. (1997). Formalizing and securing relationships on public networks. *First Monday*, 2.

[vyper, 2021] vyper (2021). Vyper. Available: https://vyper.readthedocs.io/en/stable/. Accessed on Aug. 28, 2021. [Online].

[Wang et al., 2019] Wang, G., Shi, Z., Nixon, M., and Han, S. (2019). Chainsplitter: Towards blockchain-based industrial iot architecture for supporting hierarchical storage. *2019 IEEE International Conference on Blockchain (Blockchain)*.

[Werner Vermaak, 2021] Werner Vermaak (2021). What Is Web 3.0? Available: https://coinmarketcap.com/alexandria/article/what-is-web-3-0. Accessed on Jun. 16, 2021. [Online].

[Woo et al., 2020] Woo, S., Song, J., and Park, S. (2020). A distributed oracle using intel sgx for blockchain-based iot applications. *Sensors*, 20.

[Xiao et al., 2020] Xiao, Y., Zhang, N., Lou, W., and Hou, Y. T. (2020). A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys Tutorials*, 22(2):1432–1465.

[Zainuddin, 2021] Zainuddin, A. (2021). Guide to consensus algorithms: What is consensus mechanism? Available: https://masterthecrypto.com/guide-to-consensus-algorithms-what-is-consensus-mechanism/. Accessed on Aug. 25, 2021. [Online].

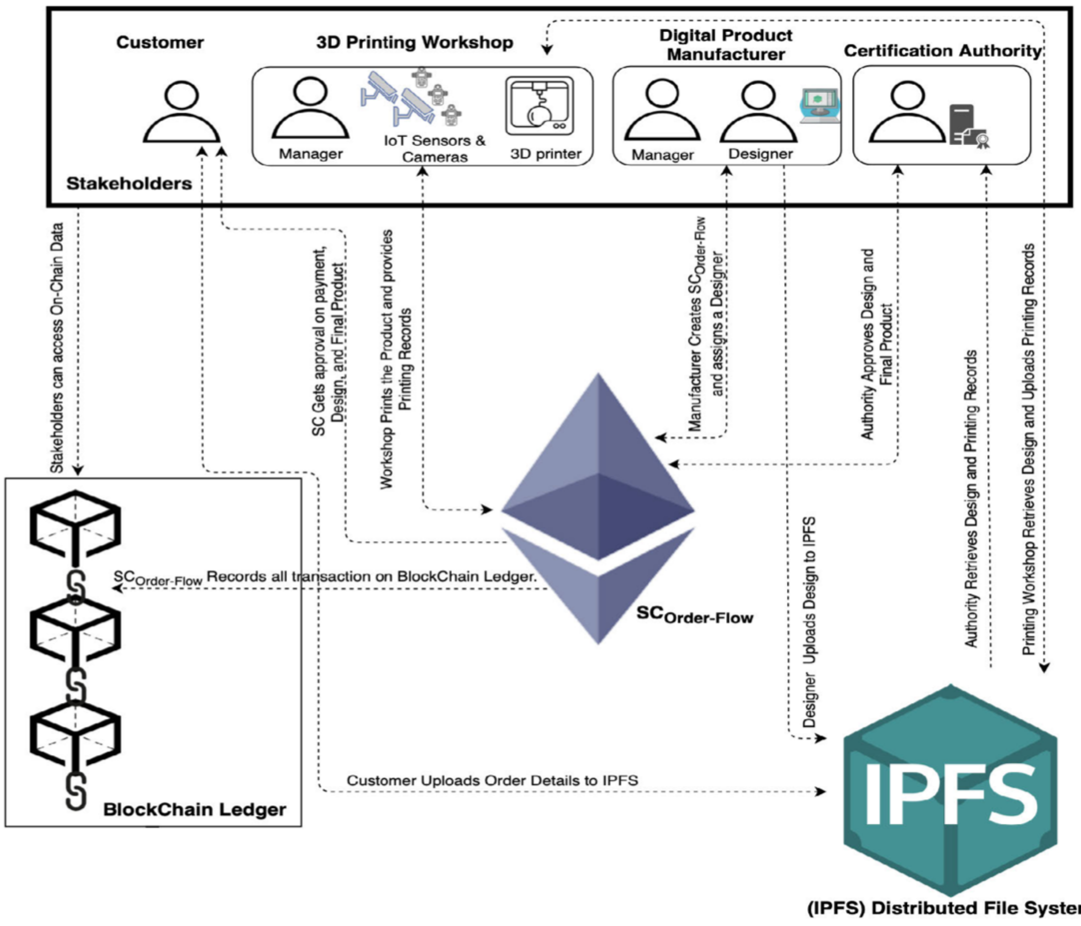
[Zcash, 2021] Zcash (2021). Zcash. Available: https://z.cash/. Accessed on Aug. 5, 2021. [Online].

[Zubi, 2009] Zubi, Z. S. (2009). On distributed database security aspects. *2009 International Conference on Multimedia Computing and Systems*.

[Özyılmaz and Yurdakul, 2017] Özyılmaz, K. R. and Yurdakul, A. (2017). Integrating low-power iot devices to a blockchain-based infrastructure. *Proceedings of the Thirteenth ACM International Conference on Embedded Software 2017 Companion - EMSOFT '17*.

Appendices

A Figures



Customer
3D Printing Workshop
Digital Product Manufacturer
Certification Authority
Manager
IoT Sensors & Cameras
3D printer
Manager
Designer
Stakeholders
Stakeholders can access On-Chain Data
SC Gets approval on payment, Design, and Final Product
Workshop Prints the Product and provides Printing Records
Manufacturer Creates SCOrder-Flow and assigns a Designer
Authority Approves Design and Final Product
Authority Retrieves Design and Printing Records
Printing Workshop Retrieves Design and Uploads Printing Records
Designer Uploads Design to IPFS
SCOrder-Flow Records all transaction on BlockChain Ledger.
SCOrder-Flow
Customer Uploads Order Details to IPFS
BlockChain Ledger
IPFS
(IPFS) Distributed File System

Figure A1 – System architecture, major components, key stakeholders and the interactions among them [Alkhader et al., 2020].

B Listings

Listing B1 – TodoList smart contract.

```
pragma solidity >=0.4.22 <0.9.0;

contract TodoList {
  uint public taskCount = 0;

  struct Task {
    uint id;
    string content;
    bool completed;
  }

  constructor() public{
    createTask('Example Task Intialized');
  }

  mapping(uint => Task) public tasks;

  event TaskCreated(
    uint id,
    string content,
    bool completed
  );

  event TaskCompleted(
    uint id,
    bool completed
  );

  function createTask(string memory _content) public {
    taskCount++;
    tasks[taskCount] = Task(taskCount, _content, false);
    emit TaskCreated(taskCount, _content, false);
  }

  function toggleCompleted(uint _id) public {
    Task memory _task = tasks[_id];
    _task.completed = !_task.completed;
    tasks[_id] = _task;
    emit TaskCompleted(_id, _task.completed);
  }
}
```

Listing B2 – CoolNumber smart contract.

```
pragma solidity ^0.6.6;

contract CoolNumberContract {
    uint public coolNumber = 10;

    function setCoolNumber(uint _coolNumber) public {
        coolNumber = _coolNumber;
    }
}
```

Listing B3 – EV3 speed monitor smart contract.

```
pragma solidity ^0.6.6;

contract EV3SpeedMonitor {
    string public deviceId;
    bool public running = false;
    uint32 public speed = 0;

    constructor(string memory _deviceId) public {
        deviceId = _deviceId;
    }

    function start(string calldata _deviceId) public {
        deviceId = _deviceId;
        running = true;
    }

    function stop() public {
        running = false;
    }

    function update(uint32 _speed) public {
        speed = _speed;
    }
}
```

Listing B4 – BIoT use case API full code - ev3-iot.go

```
package main

import (
    "context"
    "fmt"
    "log"
    "time"
    "math/big"
    "crypto/ecdsa"

    "github.com/ev3go/ev3dev"
    "github.com/ethereum/go-ethereum/accounts/abi/bind"
    "github.com/ethereum/go-ethereum/common"
    "github.com/ethereum/go-ethereum/crypto"
    "github.com/ethereum/go-ethereum/ethclient"
)

func main() {
    fmt.Println("
      ######################################################################")
    //-->Create an IPC based RPC connection to a remote node (here our RPI)
    conn, err := ethclient.Dial("http://192.168.1.38:8545")
    if err != nil {
        log.Fatalf("Failed to connect to the Ethereum client: %v", err)
    } else {
        fmt.Println("Sucess! you are connected to the Ethereum Network")
    }

    //-->Print block header
    header, err := conn.HeaderByNumber(context.Background(), nil)
    if err != nil{
        log.Fatal(err)
    } else {
        fmt.Println("The block header is:",header.Number.String())
    }

    //-->Get the contract address from hex string
    addr := common.HexToAddress("0x72f8a3ad080ed0101af41e30b381d7b004b7adea")

    //-->Bind to an already deployed contract and print coolNumber
    speedMonitor, err := NewEV3SpeedMonitor(addr, conn)
    if err != nil {
        log.Fatalf("Failed to instantiate coolNumber contract: %v", err)
    } else {
        speed, _ := speedMonitor.Speed(&bind.CallOpts{})
        //device, _ := monitor.DeviceId(&bind.CallOpts{})
        fmt.Println("########")
        fmt.Println("The last speed is:", speed)
        //fmt.Println("The EV3 device id is:", device)
    }

    //-->Bind to an account
    privateKey, err := crypto.HexToECDSA("2
      c88557feb65fa683290492e7a91b67dae8876260dba0972f98b4a5578c942d9")
    if err != nil {
        log.Fatal(err)
    }

    publicKey := privateKey.Public()
    publicKeyECDSA, ok := publicKey.(*ecdsa.PublicKey)
    if !ok {
        log.Fatal("error casting public key to ECDSA")
    }

    fromAddress := crypto.PubkeyToAddress(*publicKeyECDSA)
    fmt.Println("########")
    //fmt.Println("Address of account:", fromAddress)

    nonce, err := conn.PendingNonceAt(context.Background(), fromAddress)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println("Pending Nonce:", nonce)

    gasPrice, err := conn.SuggestGasPrice(context.Background())
    if err != nil {
        log.Fatal(err)
    }
    //fmt.Println("SuggestGasPrice:", gasPrice)

    auth, _ := bind.NewKeyedTransactorWithChainID(privateKey, big.NewInt(3))
    auth.Nonce = big.NewInt(int64(nonce))
    auth.Value = big.NewInt(0)      // in wei
    auth.GasLimit = uint64(3000000) // in units
    auth.GasPrice = gasPrice

    //-->Write in the smart contract
    metric := make(chan uint32) //create metric channel for concurrent goroutines
    go monitorDeviceRun(metric)
    m1 := <- metric //receive value from metric channel
    fmt.Println("########")
    fmt.Println("The new speed is:", m1)
```

```go
    speedMonitor.Update(auth, m1)
}

func monitorDeviceRun(metric chan uint32) {
    var result uint32
    // Get the handle for the large motor on outA.
    outA, err := ev3dev.TachoMotorFor("ev3-ports:outA", "lego-ev3-l-motor")
    if err != nil {
        log.Fatalf("failed to find motor on outA: %v", err)
    }
    // get max speed
    maxSpeed := outA.MaxSpeed()
    // set speed
    outA.SetSpeedSetpoint(50 * maxSpeed / 100)
    // set time to run, here 10seconds
    outA.SetTimeSetpoint(time.Duration(10) * time.Second)
    // start
    outA.Command("run-timed")
    // while #duration seconds have not passed
    for i := 0; i < 10; i += 5 {
        // get some metric from motor, e.g current speed
        value, _ := outA.Speed()
        result = uint32(value)
        time.Sleep(5 * time.Second)
    }
    metric <- result //send value to metric channel
}
```