Master Thesis : Designing a Community-Driven Decentralized Storage Network for IoT Data

Auteur : Schommers, Philippe
Promoteur(s) : Leduc, Guy; 12788
Faculté : Faculté des Sciences appliquées
Diplôme : Master en sciences informatiques, à finalité spécialisée en "computer systems security"
Année académique : 2020-2021
URI/URL : https://github.com/filoozom/tfe-uliege-code; http://hdl.handle.net/2268.2/11657

University of Liège - Faculty of Applied Sciences

Academic year 2020 - 2021

Designing a Community-Driven Decentralized Storage Network for IoT Data

Supervisors:
Prof. Guy LEDUC (ULiège)
Emmanuel TYCHON (Cisco)

Author:
Philippe SCHOMMERS

A master's thesis presented in partial fulfillment of the requirements for the "Computer Sciences" master degree.

Acknowledgements

First and foremost, I would like to express my special thanks of gratitude to my supervisors, Prof. Guy Leduc and Emmanuel Tychon, for their continued support and feedback. Thank you for keeping me in check and refocusing me when necessary, as well as for your valuable knowledge. I also appreciate the continued support despite changing the subject half way through the year.

Then, I wish to thank Thomas Vasbinder and Ralf Schommers for their precious insight on the technical and structural aspects of this work, as well as on the completeness and missing references.

Moreover, I would like to thank Constance Wielick, Artium Belet, Grégory Decker and Nicolas Mazy for proofreading this work, as well as for their invaluable help in offering me alternative points of view on a very specialized and technical subject matter.

Finally, my thoughts go to François Leduc and Loïc Champagne, as well as to all my other friends and family, for their continued support throughout my master's degree.

Abstract

Title: Designing a Community-Driven Decentralized Storage Network for IoT Data
Author: Philippe Schommers
Supervisors: Prof. Guy Leduc (ULiège), Emmanuel Tychon (Cisco)

Master in Computer Science
Faculty of Applied Sciences
University of Liège
Academic year 2020 - 2021

Abstract: Internet of Things (IoT) is an increasingly popular topic in information technology. It comes with a significant amount of challenges, such as the capability of simultaneously processing millions of data points per minute as well as storing petabytes of data. The aim of this work is to create a community-driven decentralized storage platform to process the aforementioned data. To illustrate the concept, a platform collecting air quality data from around the world is taken as an example. Because of censorship and scaling issues, the use of a distributed network is required. To tackle those challenges, blockchain technology is considered. However, current production grade blockchains are not able to process such large amounts of data. A few improvements are further analyzed, but were not retained because of intrinsic issues related to this technology. Finally, an alternative peer-to-peer network, that only relies on blockchain for its security aspect, is proposed instead. It is then implemented and compared to a benchmark example based on Ethereum. The proposed solution has two main assets that make it scalable. First, the data is not stored indefinitely as it is in standard blockchain implementations, and disappears when it is not useful to anyone anymore. Secondly, data can be published without requiring a consensus protocol. As a consequence, this work shows that the proof of concept is decentralized, cheap, community-driven, easy to use, open and fast.

Keywords: IoT, Blockchain, Decentralization, Peer-to-Peer, Storage, Open, Community-Driven, Open Data

Contents

Acronyms

CID Content Identifier. 9

CRDT Conflict-free Replicated Data Type. 11, 17

DAO Decentralized Autonomous Organization. 5

dApp decentralized application. 5, 11, 19, 49

DHT Distributed Hash Table. 9

DoS Denial of Service. 30, 46

DPoS Delegated Proof-of-Stake. 13, 15

EVM Ethereum Virtual Machine. 12

GDPR General Data Protection Regulation. 3, 48

HIP Helium Improvement Proposals. 9

HSM Hardware Security Module. 6

IOPS Input/output operations per second. 23

IoT Internet of Things. 1, 2, 4–6, 8–11, 15–17, 19, 20, 22, 23, 25, 27–29, 42, 47, 49

IPFS InterPlanetary File System. 9, 10

IPLD InterPlanetary Linked Data. 9

LoRa Long Range. vii, 8

LoRaWAN LoRa Wide Area Networks. 8, 9

P2P peer-to-peer. ii, 4, 9–11, 27–29, 33, 43, 49

PoA Proof-of-Authority. 13–15

PoS Proof-of-Stake. 13, 15

PoW Proof-of-Work. 13, 15

Protobuf Protocol Buffers. 31

TCP Transmission Control Protocol. 36

TPS transactions per second. 15, 22, 23, 25, 43

TTN The Things Network. 8

UDP User Datagram Protocol. 36

WS WebSocket. 36

Chapter 1

Introduction

IoT (Internet of Things) is constantly gaining in popularity, and comes with significant challenges. In particular, there is an emerging market for large scale community-driven applications. A decentralized platform that can accommodate these is studied throughout this work. It should be able to process millions of data points per minute and to store multiple terabytes of data per year. Moreover, this platform is expected to have the following features: being decentralized, cheap, community-driven, easy to use, open and fast.

It has to be emphasized that decentralized IoT platforms and storage networks are already widely used. However, no combination of both of them exists in one coherent product. There is thus a need for an efficient platform where IoT data could be published, stored, queried and analyzed.

Similarly to IoT, blockchain technology is also steadily gaining in popularity. Its decentralized ledger has a lot of interesting properties. Most popular blockchains do not belong to anyone, and are developed and maintained by a community. The stored data is also replicated around the world and can be validated by anyone. This means that the data does not belong to any company or any government. Thus, it cannot be censored by any entity, making blockchain technology the perfect candidate for a global or a community-driven IoT deployment.

This thesis studies the feasibility of large-scale community-driven decentralized IoT projects. In Chapter 2, current ongoing projects with similar features will be described. Chapter 3 will detail main features of blockchain technology. In Chapter 4, this technology will be confronted with the requirements of this work. Chapter 5 will then specify a practical protocol that aims to deal with the specifications of this platform. Chapter 6 implements the latter protocol and analyzes its characteristics. Finally, Chapter 7 explores potential improvements towards a production-ready platform.

1.1 Current state of IoT deployments

IoT devices are getting more popular every day. At the time of writing, there are more than 10 billion IoT devices out there, with more than twice the amount expected by the end of 2025 [15]. Those devices can be found pretty much anywhere: in parking places, buses, metro stations, etc. However, they present a significant challenge and cost to deploy and maintain.

At this point, most IoT devices are registered on centralized platforms managed by bigger companies. Deployments are also still fairly complicated, and thus restricted to enterprise usage or hobbyists. There is however consumer grade hardware that can be installed in houses, but it generally requires monthly subscriptions, and thus remains costly. The most popular consumer IoT devices are smart, voice-controlled speakers and security hardware like cameras and movement or smoke detectors. Again, the data of these devices is generally stored on private platforms belonging to Amazon, Google or Microsoft.

In general IoT devices are private, meaning that they belong to specific individuals or companies, and consequently the data they generate does too. Nobody wants their private security systems to be accessible by anyone other than themselves. There is however an emerging market for open data and community-driven IoT projects. For example, public parking lots could fit each spot with a device that could check whether or not the place is occupied. Said data could then be made available on a public website, so that everyone could always check if a spot is available.

Likewise, one could create a platform that aggregates air quality data from everywhere around the globe. Such a project would be difficult to achieve, if not impossible. Not only would it be extremely expensive, but it would also require good relationships with other countries. Data storage would also be an issue, as some countries would probably want the data to be stored locally, making the infrastructure more complicated and costly. However, this is exactly the kind of project that can be achieved with a global community in which anyone can participate freely.

1.2 Example: Global air quality platform

Collecting air quality data is straightforward when only aiming to cover a small location. For example, a company could measure the air quality inside all of its offices easily, even if it owned thousands of buildings. The entire process could be handled in-house, so there could be a unique management and data collection platform and all sensors could be similar, making management easier. The company would not need partnerships or access to unauthorized locations. The scale of the deployment would also be related to the size of the company, meaning that financing such a project would not be an issue.

This type of project becomes more complicated when it starts to target larger locations, like an entire city or even a country. This could probably not be achieved by a single company, or at least not without a partnership with the local government. At this point, it basically needs to become a government project, with different partners and authorization to place sensors in various locations around the country. Essentially, only a government could authorize such an initiative, but would not necessarily want to finance it.

Such an initiative could even work at the size of the European Union, as relationships between included countries should be good, and the European Commission could initiate such a project if it wanted to. However, this would likely take years and would still require a lot of coordination.

In order for this platform to achieve even larger scale, say the entire globe, everything becomes exponentially more complicated. This would be a really ambitious project with its own share of issues. The sheer amount of sensors required would make it impossible for any individual company or government to afford such a project. It would also require access to very diverse locations, which requires good relations between governments, as the platform could not be considered global if some countries are excluded. There is also the possibility that some countries would not allow that type of data to be stored in foreign countries, which adds complexity. A global project of such magnitude is essentially utopian.

However, this project is still interesting and is a good basis to keep in mind for this work. If it were possible to solve such a challenge, it would also be possible to apply a similar solution to smaller, more local projects. Also, even though it is ambitious, it could still potentially be achieved by a community-driven project, at least to some extent.

Take for example a scenario where anyone could buy a device that measures air quality and make the collected data accessible to everyone else. If enough people participate, there should at some point be data from basically anywhere. Sure, it is unlikely that every single city around the world would be covered, and there would probably be huge areas without any sensor at all. However, it is likely that populated parts of developed countries would at some point be covered by a local citizen interested in the project. Arguably, the most important aspect that would allow a community to succeed in such a project is the fact that the costs are spread out. Every individual who wants to participate pays for their own sensor only and makes it available to everyone else.

A community by itself is not sufficient for such a project. There are also significant technological challenges. It is not straightforward to just make data available to anyone on the internet, especially securely. The data cannot simply be stored on a cloud provider, as it would be extremely expensive and too centralized. In fact, it would be very problematic if the provider closed or blocked that type of activity. There is also the issue of data location, and laws like GDPR (General Data Protection Regulation) for example. Such a platform would also need to make sure that the data reported by sensors is accurate, which is not straightforward.

It would also be interesting to study the economy of such a project, and to potentially add a system to reward people who paid for sensors and provide accurate air quality data. Alternatively, the data could be kept private and be made available by payment, but that particular case will not be covered in this work.

1.3 Objectives

In short, the objective of this work is to design a decentralized storage platform for IoT data. This section goes through more specific requirements.

1.3.1 Decentralized

The platform should be decentralized. As explained in 1.2 Example: Global air quality platform, it is important that the data is not owned by any sole individual, company or government.

This is important for censorship resistance. If one company or government controlled the platform, it could easily block access from other countries, companies or individuals, which can never be an option for this project.

1.3.2 Cheap

The platform should be cheap to get into for any particular individual. There are a few different parts to consider in this type of community-driven project. First, there is the hardware, that contains the sensors and all the hardware that allows it to communicate with the platform. The cost of this hardware is highly dependent on its features and is thus not taken into account in the cost of this platform.

The second part is the access layer, so the cost of accessing the platform's network. In this case, it is basically the cost of the internet access, which mostly depends on the location of the subscription and thus also cannot really be taken into account. However, the bandwidth usage should be kept low, and the device should be able to limit its usage. In less developed countries, where 1GB of mobile data can cost up to $50 [39], it would probably be better to have less up-to-date or complete data instead of having none at all because it is too expensive to run. The owner should be able to configure the interval at which data is published to the platform and what data they want to synchronize locally.

Lastly, there is also the cost of storing data and interacting with the decentralized system. This part is the one this work focuses the most on, and for which blockchain and a custom P2P (peer-to-peer) solutions are analyzed.

Data storage has a cost, and as the data is made openly available for free by community members, they also carry the weight of it. Thereby it would not be reasonable for the platform to enforce a minimum time during which the data is stored. It would be best if the data was mostly stored by users that actually make use of it. For example, the data could only be made accessible for one week, after which it is deleted by the publisher. Then, end-users of said data would store it at their own expense. Basically, this dissociates the burdens of data generation and storage, and allows the system to balance itself based on demand.

Finally, it is unlikely that costs could be kept down by enforcing the fact that data must remain accessible on the network forever. As such, this work targets a platform where data that is not actively stored by anyone simply disappears. This assumes that if data is not useful at a certain point, it likely never will be, and thus does not need to fill hard drives. This is not necessarily true, but in any case this platform should never force anyone to keep data they do not want or need.

1.3.3 Community-driven

The objective of this entire work is to build a community-driven project. This has multiple advantages.

Firstly, the network is paid for by many different people, which allows it to scale way beyond any other type of platform.

Secondly, the data comes from a lot of different people, meaning that it is a lot more complicated to tamper with. A single entity could falsify data pretty easily, but it is way more complicated to falsify data originating from different sources.

This work is going to assume that the development and all decisions regarding the platform are done by a single entity, because the primary objective is to have decentralized data and not development. It would however be interesting to study the development by–or the transfer to–a DAO (Decentralized Autonomous Organization). This would allow the platform to be maintained by the community itself, thanks to features proposed by vote. The code could also be developed by volunteers, whose forks could be integrated based on a governance. However, completely decentralized blockchain development is an entire problem in its own right, and cannot be studied over the course of this work.

1.3.4 Easy to use

In order for something to become mainstream, it generally must be easy to use. For this project, there is not a whole lot to do from a user's perspective. In fact, they really only have to on-board the device. For the access part, it should be mostly done through a dApp (decentralized application), which could run in a browser for example. This would make it easy for anyone to access the data on the decentralized platform.

The onboarding step could be complicated, for example if the user had to understand and learn about blockchain, IoT, networking and everything related to the device. The objective is to make this as simple as possible. The user should be able to plug the device in, configure the Wi-Fi network, register the device on a web application and that is it. Optionally, some functionalities of the device could be configured. For example, in relation to the air quality example (1.2), it could be possible to set a precise location of the device (for example if the built-in GPS did not manage to get a fix), set the data publication interval, set a data cap, or configure more complex networking for specific requirements. Those options should be easily accessible, for example through a mobile application that connects to the device via Wi-Fi or Bluetooth.

1.3.5 Open

In order for the platform to be entirely community-driven, the entire stack must be open. This includes the hardware, the software and all protocols. Different markets have different needs and manufacturing capabilities, and this platform thus cannot be too rigid in order to stay inclusive. One can not design a community-driven platform that solely relies on proprietary hardware or security by obscurity. This means that all protocols need to be documented clearly, and that the platform should be flexible enough to allow different types of devices to be manufactured and deployed.

For example, there could be high-end devices that include Wi-Fi, wired networking, Bluetooth and an HSM (Hardware Security Module) in addition to a very accurate primary sensor. On the other hand, other devices could only have wired networking and a less accurate sensor. They should still both be able to publish data to the platform, as long as they respect the specified protocols. With regards to the accuracy of the sensors, it might be wise to include information about margins of error on the platform, so that people relying on the data can make educated decisions.

1.3.6 Fast

The last missing ingredients for a successful IoT platform is speed. A web interface that relies on the platform would not make too much sense if users had to wait for a few minutes each time they open the web application before being able to read the data. In general, fast access to anything is an important factor for conversion rate [19].

Ideally, most data should be available for display in less than five seconds. As the platform should be decentralized, there will be some inevitable delays, and it is thus unlikely that data could be made accessible in under a second. However, caching and more centralized approaches could be implemented when very fast access is required.

1.4 Nice-to-haves

1.4.1 Accuracy

This particular topic depends on the goal of the platform, and is not always necessary. However, it is still interesting to explore for a lot of use-cases. In order to measure accuracy of data, it needs to have a specific format, and all the data needs to be comparable. This is the case for a platform that measures air quality, but is not for a more generalized platform that allows anyone to publish arbitrary data. Measuring accuracy also needs multiple data points from as many different data sources as possible.

On such a platform, it is not possible to make sure that the data sent by sensors is accurate. As this is an open network, people could tweak their devices to send arbitrary data to the network using the provided protocols. Depending on the application, this could have a bad impact, and should be avoided at all costs. This would for example be the case on a community-driven platform used by aviators that makes sure that the weather is good enough for flying. If people publish malicious data on purpose, it could have terrible consequences. In addition to bad actors, there could also be broken devices on the network that report incorrect data.

It is not possible to design a device or protocol that could make absolutely sure that the data is correct. One possible solution would be to aggregate data from different geographically close locations, ideally operated by different people, and to remove data that is outside of some deviation margins.

An improvement of the previous proposition would be to implement a reputation system, where geographically close data is constantly compared, and the devices that publish information are assigned a reputation score. The better the score, the more accurate the data is assumed to be.

Sadly, both of these methods rely at least on multiple devices in a vicinity, which is not always the case. Also, the closeness of devices depends on the application. For example, a daylight sensor could have comparable data on a range of 100km+, whereas temperatures can be way more disparate on such distances.

1.4.2 Economy

Initially, this work is going to assume that no economy is involved in the project. This means two things:

- the data is made openly available to anyone without the need to pay
- device owners are not compensated for their service

Basically, in this scenario people help the network because they want to or believe that they should, and not to seek personal gain.

Economy is always interesting to study, as it can heavily incentivize the network in its entirety. If people are getting compensated for their purchase and the data they make available, it is very likely that they will invest more heavily in the network. This in turn makes the network more robust, as more devices means more data, and thus better accuracy and better coverage. However, this work cannot focus on this aspect, even though it would likely be essential for a production grade platform.

Chapter 2

State of the art

This chapter covers some projects that share characteristics with the targeted platform. Some of these were used as inspiration for the Objectives (1.3), and others are used in the Proposed protocol (5). Most of the following projects are related to either the community-driven aspect or the data storage one. However, none of them does specifically what this work aims to achieve.

2.1 The Things Network

TTN (The Things Network) [34] is a community-driven project that aims to make a LoRaWAN (LoRa Wide Area Networks) network freely available around the world. Gateways are hosted by individuals who are interested in IoT, and are not compensated for it. All public gateways must be registered in TTN, which is a centralized platform. It is also possible to run a private network [4], although in that case the gateway does not take part in the community aspect.

On TTN, the data is stored on arbitrary services using integrations [20], so it is possible to basically store or process the data anywhere. Those services are configured by the end user, and gateways that receive packets route them directly to those services.

This project is interesting for this work because of its community-driven aspect and its relation to IoT.

2.2 Helium

Helium's objective [43] is comparable to TTN's: provide a LoRaWAN network across the globe. In fact, they call themselves The People's Network [16]. It is also a community-based project. However, it is not free to use: each packet (of maximum 24 bytes) costs \$0.00001 [35]. The fact that Helium is not free is actually an advantage in most cases, as it allows for a more robust network. Hotspot owners are paid for making their gateways available [28]. This gives a financial incentive to add hotspots to the network, and should allow the network to grow organically. For people wanting to experiment with the network, it is also possible to receive 10 000 Data Credits for free by signing up on Helium Console [17].

Helium makes use of a custom blockchain [1] to handle everything related to hotspots (onboarding, rewards,...) and data transfer. This blockchain is mostly used for the economical aspects, and does not store any data forwarded by the hotspots. The entire system is thus pretty much decentralized. Helium's development is mostly centralized and made by hired developers. However, the community can write HIPs (Helium Improvement Proposalss) in order to participate in the network and propose changes [3]. Those are then voted on and implemented if the community agrees. This is a collaborative process, and opinions are shared and taken into account in the process. The latter adds a crucial component to the community-driven aspect of the network.

This project is comparable to 1.2 Example: Global air quality platform. In fact, hotspots can be deployed by anyone, and are made accessible to other people. In comparison to this work's objectives, there is an additional economical aspect to Helium. Their forwarded data is encrypted too, in accordance with LoRaWAN's specification [38], meaning that it is not open data. Lastly, Helium does not concern itself with data storage, and only routes packets to configured services.

Helium is actually quite complementary to this work's objectives, as it already provides the access layer for IoT devices. However, it is missing the storage aspect, which this work targets.

2.3 InterPlanetary File System

IPFS (InterPlanetary File System) is a "peer-to-peer hypermedia protocol designed to make the web faster, safer, and more open" [22]. Its goal is to create the web of tomorrow in a decentralized fashion.

This project makes it possible for anyone to host files on the decentralized web. Those are referenced by CIDs (Content Identifiers), and nodes hosting those files are registered in a DHT (Distributed Hash Table). When content is accessed, it is downloaded from multiple peers at a time, in a P2P manner. Then, it is optionally made available to other nodes, if so chosen by the node operator. It is also possible to pin [27] certain CIDs in order to make sure to keep them up to date on a local node, and thus to guarantee its availability. This basically acts like an integrated caching or archiving mechanism. The pinned content is not deleted when the original data is not made available by the original source anymore.

A great use case for IPFS is decentralized websites. However, IPFS is actually able to do quite a bit more than just handling basic files. It now comes with IPLD (InterPlanetary Linked Data), an "ecosystem of formats and data structures for building applications that can be fully decentralized [21].” This ecosystem allows one to store data in arbitrary formats on IPFS. This would be required for IoT data, as storing each data point in a standard file would be inefficient and unpractical.

This could be interesting for this work’s IoT platform, as it would allow each sensor to make its data available. Furthermore, people that want access to the data for a longer period could pin it on their local nodes. This way, if the sensor or its owner decides to delete the data, it can still be accessed for archival reasons or further processing. Basically, anyone stores only what they are interested in. This decouples the burdens of data generation and data storage as planned, and would make for a healthy ecosystem.

2.4 Swarm

Quoting Swarm’s website [31]:

> Swarm is a system of peer-to-peer networked nodes that create a decentralized storage and communication service. The system is economically self-sustaining due to a built-in incentive system enforced through smart contracts on the Ethereum blockchain.
>
> Redundancy makes the system resilient to connectivity issues, node churn or targeted DDoS attacks and enables a zero-downtime service. Users remain sovereign owners of their personal data in alignment with fair data principles.
>
> Shifting the cost of access, hosting and execution to users removes the last obstacle to truly agile and adaptive application development.

Swarm’s objectives are quite similar to IPFS’, but the technology is based on different principles. On Swarm, it is necessary to pay for data upload. This stems from the fact that said data is automatically pushed to other peers in order to make sure that it stays available. This has only been the case since a short time ago when version 0.6.0 was released [29].

As the technology was likely to change when this work was started, it was not studied as an underlying base for this platform, but rather for ideas. The author has also had the pleasure of talking with multiple members of the Swarm Team about this work, being a contractor for them himself.

This project is very interesting for its decentralized storage aspect and for its built-in economy. However, it is not a perfect fit for this platform’s objectives, as people making data available would have to pay to do so. Swarm is also still in its infancy and has not reached mainnet yet [33].

2.5 OrbitDB

As defined by the authors of OrbitDB [26]:

> OrbitDB is a serverless, distributed, peer-to-peer database. OrbitDB uses IPFS as its data storage and IPFS PubSub to automatically sync databases with peers. It's an eventually consistent database that uses CRDTs (Conflict-free Replicated Data Types) for conflict-free database merges making OrbitDB an excellent choice for dApps, blockchain applications and offline-first web applications.

This database implements various types of data models that can be used for different use cases. For example, it provides a log type [25], which is an append-only, immutable log with traversable history. This is basically perfect for IoT metrics, as they are time-series based and thus do not change and have a specific order. This type of database makes it straightforward to append data to the log and to retrieve an arbitrary amount of last values, or to traverse the entire structure.

While this could be a great technology to use for this work's IoT platform, it is still lacking a few important properties. For example, it is not possible to search for specific elements. If one is interested in data from January 1st 2010, they would have to download and traverse the entire log until that date. It also comes with a series of features that are not necessary for this work's platform and cause a large overhead in storage. Fetching data from multiple devices would also be extremely slow, as each IoT object on the platform would have to maintain its own log.

Chapter 3

Blockchain

Blockchain [46] is a natural technology choice for this project. It is decentralized and some blockchains are programmable using their smart contracts. This makes it possible to store data and to implement functionality on a decentralized computer. Basically, this technology could serve as an underlying building block for the targeted platform.

3.1 Summary

To briefly summarize, a blockchain is a data structure in which each block references the previous one. This forms a chain of blocks, hence the name. References are hashes of entire blocks, and they make it possible to validate the entire chain. This is used to make sure that no data was corrupted or tampered with on the system. Blocks are mined with a variety of consensus algorithms (3.3.2).

3.2 Ethereum

For this platform, it makes sense to use a popular blockchain, as it would already have proven itself to be stable and useful. In this case, Ethereum was chosen as the most popular blockchain that supports smart contracts [12][9], and will be used as a basis for comparisons and testing.

Ethereum implements the EVM (Ethereum Virtual Machine) [13], which can run arbitrary code. This code is deployed on the chain using smart contracts, which are basically a set of functions. The most popular language to develop smart contracts in on Ethereum is Solidity [30][48]. This is also the language chosen for this work.

3.3 Characteristics

This section studies some characteristics of blockchains, which is important to understand what they bring to the table and if they are a good fit for this project.

3.3.1 Immutable

First things first, blockchains are immutable. This means that once something is recorded on it, it can never be altered or be deleted.

This lies in the fact that blockchains, as their name suggests, use a data structure that forms a chain of blocks. Each block contains all transactions that happened in that block, as well as additional metadata and a reference to the previous block. That reference is actually a hash of the entire previous block, which makes it possible to verify that a block was not tampered with.

This basic hashing technique makes it possible to verify an entire blockchain from its most current block back to its genesis, while making sure that each block is accounted for and valid, thus making the chain immutable. The current state of the entire system can also be rebuilt from scratch by “executing” each block sequentially from the beginning.

This is interesting for systems where one needs to be sure that all the data is genuine and was not tampered with. For example, systems that facilitate monetary transactions need to be secure so that no currency can be stolen or misplaced.

However, in order to be able to verify the entire chain, one needs to have all blocks generated since the inception of the blockchain at their disposal. If blocks are missing, it is no longer possible for the system to agree on a common global state. Simply put, this means that storage requirements to keep a blockchain up to date only go up, and never down. Emitting a transaction or writing anything on a blockchain means that in theory, every node in the entire system will have to store that data for eternity.

3.3.2 Consensus

Blockchains use a variety of consensus algorithms. Those are particularly popular in decentralized computing, as they are required to coordinate distributed systems. They ensure that each node in a system can agree on a common global state, even if some of them fail or act in nefarious ways. The most popular ones in blockchain are:

- PoW (Proof-of-Work)
- PoS (Proof-of-Stake)
- PoA (Proof-of-Authority)
- DPoS (Delegated Proof-of-Stake)

3.3.2.1 Objectives

Achieving a common global state is essential for some applications. For example, in financial systems, it is mandatory that two different nodes agree on the balance of a certain user. If not, there would be a split between both systems, and a user could spend currency they do not own anymore. Consensus is also required in other fields, like distributed locks or databases.

3.3.2.2 Security

Except PoA, which is centralized, all of the above algorithms are entirely decentralized and byzantine fault tolerant. In most cases, this means that an attacker would need to control more than 50% of the network (be it hashing power or staking weight) in order to attack it. Still, this would not allow the attacker to execute arbitrary transactions, as they still need to be signed. With this type of resources, a nefarious actor could reorganize the blockchain, so basically rewrite the history. This allows double-spends when mixed with an external component. For example, assume the attacker owns 1 Bitcoin, which they sell for $60,000 on an exchange. Typically, the latter would wait for a few confirmations (for example 4 blocks, so 40 minutes in the case of Bitcoin) before crediting the attacker's account with currency. Now, if the attacker is able to rewrite the last 5 blocks (the one with their transaction to the exchange and the 4 confirmations), then they could send their Bitcoin to another exchange and get $60,000 there too. In that case, they would have to withdraw their money from the first exchange first at the risk of seeing their transaction reverted. At this point, the attacker effectively would have spent their Bitcoin twice, and gotten double the value out of it.

3.3.2.3 Speed

Reaching consensus takes time, as a majority of nodes need to agree on what happens to the distributed ledger. As this work targets a global platform distributed in the entire world, network latency is already a limiting factor with regards to speed. Nodes participating in consensus also need to be powerful enough in order to handle all the data that needs to be ingested. This includes fast networking, fast CPUs and fast storage.

For the platform targeted by this work, there would be an enormous amount of data to agree on, and consensus algorithms used in current blockchains would not be able to keep up. In fact, no single machine would likely be able to keep up with the influx of millions of data points per second. When adding consensus to the mix, this becomes exponentially more difficult.

3.3.3 Decentralized

Blockchains can be either permissioned or permissionless. Permissioned blockchains are private in the sense that they require permission to join and participate in consensus. They generally use governance structures, and are thus not completely decentralized. A popular

Blockchain	TPS	Consensus algorithm	Release year
Bitcoin	7 [41]	PoW [44]	2009 [42]
Ethereum	15 [11]	PoW [13]	2015 [13]
EOS	4,000 [6]	DPoS [5]	2018 [5]
Ethereum 2.0	100,000 [36]	PoS [14]	2020 [8]

Figure 3.1: Comparison of different blockchains and their TPS

example is Hyperledged Fabric, but popular permissionless blockchains like Ethereum can also be made permissioned. As a matter of fact, popular testnets like Görli[1], Kovan[2] or Rinkeby[3] use PoA (Proof-of-Authority), which uses a permissioned consensus algorithm. This type of blockchain is of no interest to this work, as the latter aims at being as open as possible.

Permissionless blockchains are entirely decentralized and rely on consensus algorithms like PoW (Proof-of-Work) or more recently PoS (Proof-of-Stake). They are open to anyone willing to participate while still offering high security. However, as consensus needs to be reached amongst thousands of nodes, it is generally slower and harder to scale. This means that permissionless blockchains generally have a low TPS (transactions per second) throughput, as seen in Figure 3.1.

The speed of a blockchain mostly depends on the consensus algorithm. For this work, transactions per second is a very important metric, as it defines an upper bound on the supported IoT devices. If a blockchain like Bitcoin can only handle 7 TPS, that means that the entire platform could only support 1500 IoT devices publishing one data point every 5 minutes. That's not enough by a long shot. Once Ethereum v2 entirely replaces Ethereum v1, and assuming the fact that it actually manages to handle 100k TPS, the upper bound is still of only 2.5 million IoT devices. This could be enough for some use cases, but is not for a global IoT platform. Remember that the market already accounts for tens of billions of devices right now, so even the most cutting edge blockchains are still several orders of magnitude too slow.

3.3.4 Economy

In order to write data onto a blockchain, it is necessary to pay a transaction fee. This fee gets distributed to nodes participating in consensus for their hard work. The economy makes blockchains more robust, as they give an incentive to secure the network.

[1]`https://goerli.net/`
[2]`https://kovan-testnet.github.io/website/`
[3]`https://www.rinkeby.io/`

Chapter 4

Blockchain for this work

This chapter builds upon the Blockchain chapter (3) and puts the characteristics in relation with this work's objectives. Then, a basic concrete example is given on how an IoT platform could be built upon blockchain using smart contract. The latter is then used as a base for a cost analysis. Finally, a few ideas are explored to make blockchain more suitable for large scale data storage.

4.1 Characteristics

This section puts the characteristics of blockchain (3.3) in relation with this work's objectives.

4.1.1 Immutable

While this might be required for financial systems, it raises the question of whether or not it also is for an IoT platform. Does it make sense to force all nodes on the platform to keep an archive of every single data point ever recorded on it? Does it even make sense to be able to prove that the entire chain of data has not been tampered with?

Regarding the data storage aspect, it makes sense to do a bit of calculation to figure out how much storage would be required. With a node address, timestamp and temperature and humidity floats, one data point would take at the very least 150 bytes. Considering one device that publishes one data point every five minutes for one year, the storage cost would be of at least 15 768 kB. This is the bare minimum of required storage. With indexes and queryable data structures, this would quickly increase. Now, considering that IoT is all about scale, it is also necessary to consider more devices. For one million of them, publishing at the same frequency, the storage cost is already 16 Terabytes per year. Remember, all of this data needs to be stored forever and by every node on the network. Considering that the IoT market targets a few billion more devices, it becomes clear that standard blockchains cannot accommodate that type of data.

Regarding the security aspect of being able to validate the entire chain: for this platform, data is generated by many different people for free. They can join the network, leave it and publish whatever data they want. Ideally it would be accurate data, but it is not necessarily the case. Thus, one must wonder if being able to validate arbitrary sensor data really makes sense in such a scenario. Does one really need to know that a device `x` owned by an individual `y` published the data point `j` precisely at time `t`, which is after data point `i` and before `k`? This work assumes that there is no real reason for the targeted platform. Being able to validate that an individual `y` published a data point `x` is sufficient to be able to grant trust to said individual, and doing so only requires digital signatures.

4.1.2 Consensus

Regarding the IoT platform this work targets, is it really necessary that all nodes agree on the fact that some node published a data point before including it in some type of ledger? The main benefit would be that every node in the system would agree on everything that happened in the entire network. Every node would also agree on how many data points each node published, when that happened and what each of them contained.

As much as this might be nice, it is not at all required for this platform. There is no transaction between nodes, and in fact, there is not even a global state to maintain, as each data point is independent. The entire system is actually a basic append-only log. So once a data point is published, it is added to the log, but there is no need to compute a new state or to move currencies around. A CRDT would arguably be more useful for this type of data.

According to Wikipedia [47]:

> In distributed computing, a conflict-free replicated data type (CRDT) is a data structure which can be replicated across multiple computers in a network, where the replicas can be updated independently and concurrently without coordination between the replicas, and where it is always mathematically possible to resolve inconsistencies that might come up.

For this platform, consensus could be useful for only two things:

1. Agree on the time at which a data point was generated.
2. Keep track of all data points that were published by each node, making sure that the entire state can be replicated.

The first point cannot be solved without consensus. Typically in blockchain, it makes sense to distinguish between two dates in the lifespan of a transaction. First, the date at which a transaction was created and broadcasted, and then the time at which is was “mined”, or included in the ledger with consensus agreement. The first date cannot be trusted because anyone could forge a transaction that claims to have been created at an earlier date. This makes sense, because if a transaction is not included in a block, it can still get added to later ones, as long as it remains valid. So, a transaction could legitimately be included in a block months after its creation. However, the date at which a transaction was mined is agreed upon by every node participating in consensus. The latter process thus sets an upper bound on the creation date of a transaction. When combined with time-limited transactions, it is also possible to set a lower bound. For example, the blockchain could refuse transactions whose creation date is lower than one hour ago. With those two bounds, it is possible to make sure that a data point was created at around a given date, without the need to trust its creator. This could be enforced on public blockchains that support smart contracts and on custom blockchains.

In this work, we argue that the actual creation date of a data point is not that interesting. The information contained in a data point cannot be verified either way, so there already needs to be trust involved. What good does it make to know that a data point was created at a specific time, without knowing if its data is accurate? This work thus chooses to also trust the timestamp defined in a data point by its sender. Consensus is thus not required for this part.

For the second point, there are other partial alternatives. In order to make sure that one has the entire collection of data point from a specific node, it is enough to create a chain of data points from the most recent one to the first one. Every time a node publishes a data point, it includes a reference to the previous one, very much like blockchains work. Each data point is also signed by its sender to guarantee its origin. A common global state cannot be achieved without consensus, but like for the previous point, this work chooses to ignore that fact as it does not provide interesting information anyways.

4.1.3 Decentralized

As per this work's objectives (1.3), decentralization is a requirement for this work. This characteristic thus matches perfectly.

4.1.4 Speed

The speed aspect mostly depends on consensus (3.3.2), which directly comes from the decentralized aspect (3.3.3). Currently deployed blockchains are not fast enough, and thus do not fit this project. As stated by Ethereum's founder Vitalik Buterin, scaling blockchain turns out to be a very complicated problem when trying to stay completely decentralized [24]. The main bottlenecks being storage size and data capacity.

4.1.5 Economy

The economical aspect of blockchain does not matter for this work, as it is mostly useful to secure the underlying network. It is however interesting to note that an external economy can be built on top of it thanks to smart contracts, which is a "nice-to-have" (1.4) for the platform.

4.1.6 Conclusion

In conclusion, the characteristics match the following criteria of this work's target platform:

	Decentralized	Cheap	Community-driven	Easy to use	Open	Fast
Blockchain	yes	?	yes	yes	yes	no

The "easy to use" characteristic does not come from the fact that blockchains are easy to understand, or that contracts are easy to write. However, it is possible to build a dApp on top of a blockchain that is easy to use for the end user, which is one of the objectives of this work.

The "cheap" aspect was not sufficiently covered, and is studied in the next section.

4.2 Concrete example on Ethereum

In relation to the objective of being cheap, it is necessary to evaluate the cost and feasibility of running the desired IoT platform on an existing blockchain. In order to do this, this work includes a very basic example smart contract (A.1) for Ethereum, which is currently the most popular blockchain featuring smart contracts. This section goes into a lot of technical details.

The smart contract has two write functions and two read ones. Only the write functions require transaction fees on Ethereum, so retrieving data is always free. On Ethereum, the transaction fee is called gas, and it increases with the complexity of the code called by the transaction.

Events are emitted when a device is added to contract and when data points are published. These make it possible for subscribers to react instantly as soon as the transaction is mined by the network. For example, a dashboard could monitor these events and update the displayed data as soon as the monitored device publishes it.

The example contract, available on GitHub[1], was deployed on the Görli testnet at the following address: 0x66db634d31295701c3e7518a9ec5132e63f856ee. It can be explored on Etherscan (`https://goerli.etherscan.io`), which also makes it possible to interact with the contract. The remainder of this section assumes a gas price of 250 gwei[2] and a value of $1800 per ETH[3]. Both are volatile, so the actual values are prone to changes. The deployment of the sample contract costs 731,588 gas, which represents a cost of 0.182897 ETH or $329.21.

For the purposes of this example, three wallets are being used to create the contract and to interact with it:

[1]`https://github.com/filoozom/tfe-uliege-code/tree/main/ethereum`

[2]Gwei is a denomination of ether, commonly utilized to price gas. 10^9 gwei = 1 ETH [10]

[3]Ether, also written as ETH, is the native currency of Ethereum [10]

1. Deployer: 0x6893AE9Fbb9b17bb227Bc94870Cb5091f3d123b8
   Used to deploy the contract and as a funding source for other wallets.
2. Owner: 0x79b564e241d95dA1F1c8AaBF5CE502c60e261A7c
   Secure wallet that owns and registers devices.
3. Device: 0xEd4613aFb7AC61FA2760b61f044Ce6bE3d09930a
   Device owned by the above Owner, used to publish data.

4.2.1 Write functions

4.2.1.1 Register

The `register` function makes it possible to register IoT devices on the blockchain. Each device has three fields:

- Owner: address of the wallet used to register the device. The owner can modify or remove the device from the contract.
- ID: address of the device that will publish data points.
- Coordinates: latitude and longitude of the device.

There are two different Ethereum wallets: one for the owner and one per deployed device. This is done as a security measure, as devices are considered to be less secure than a standard private wallet. This way, if an attacker gets a hold of the private keys of the device, the owner can still remove it from the blockchain or rotate the keys out.

The smart contract could be supplemented with additional metadata, like device capabilities or publishing schedules, but the example was kept as simple as possible.

Registering a device consumes 171,012 gas, as illustrated in the transaction whose hash is 0xd24f9b668aa7598e708a343008ba96f2c78e225e047a6d7ea44f55e0c31b9f21. This results in a cost of 0.042753 ETH, or $76.96.

4.2.1.2 Publish

The `publish` function allows a device to publish a data point. For this basic example, a data point includes temperature and humidity readings, as well as a timestamp of when the transaction was mined and the address of the device that generated it.

As Solidity does not entirely support floats at this point, the data is currently stored in 2 byte long integers. The humidity is expressed in percents, so it is unsigned. The temperature is signed as it can be negative. Both metrics are expressed with a precision of one decimal. Using integers also has the added benefit of taking less space and thus costing less. The temperature can still go from -3276.8°C to 3276.7°C, which should be sufficient. A temperature of 3210.9°C is encoded as the 32109 integer.

Publishing a data point consumes 107,580 gas, as illustrated in the following transaction: 0x0c1e90ca85961e683ec0e3d66ae1a012c12ce9a8cdef02ef1a2c0ca09e2c1609. This results in a cost of 0.026895 ETH, or $48.41, which is too expensive.

4.2.2 Read functions

Even though read-only functions are free, they are still subject to gas limits. Each operation executed on Ethereum consumes gas. If a read-only function is too compute-intensive, and thus exceeds the gas limit, it is aborted, and no result is returned.

As Solidity does not support returning dynamic arrays from functions, it is always necessary to return a fixed amount of elements. So if the `limit` parameter is set to 100, but there is only 1 result available, the function returns the first element as well as 99 other ones with zeroed out values. While this still does not cost anything, it is not exactly great for bandwidth, so the `limit` parameter should be chosen with care.

In order to allow the read functions to work even with millions of registered devices, both read functions include `limit` and `start` parameters. Basically, when listing nodes, one would query at a starting index of 0 with a chosen limit. For the second call, the start index is fixed at the index of the last returned value of the previous request. As the basic contract does not provide a way to remove nodes or data points, indexes can safely be used without causing duplicates or missing results.

The current implementation is too naive, and the read functions can run out of gas (read "run out of computing allowance") under certain conditions. For example, it would not be possible to locate a recent node in a small bounding box if say one million nodes were registered before hand. This is because the function call would run out of gas before getting the first result, and it would thus not be possible to start a new iteration. However, this does not matter too much for this work, as the main objective is to measure the cost of this solution.

4.2.2.1 Locate nodes

The `locateNodes` function makes it possible to list nodes in a certain latitude and longitude bounding box. The algorithm used in the example contract is very naive, and a more efficient data structure should be used for production use, like H3geo[4] for example.

However, as the objective of this smart contract is to measure cost, and read functions are free, no further thought is spent on this problem for this work.

4.2.2.2 Fetch data

The `fetchData` makes it possible to fetch data points for a specific device. The `from` and `to` parameters make it possible to filter by publishing date. This function is also inefficient and should use something like binary search to find elements corresponding to the filter.

[4] `https://h3geo.org/`

As it is more likely for this platform to fetch more current data points, it would also have been better to start the search from the end instead of the beginning.

4.2.3 Conclusion

As expected, it would be way too expensive to store IoT data on Ethereum. This is not only the case because of recent extremely high gas prices, as even with a price of 1 gwei, publishing a single data point would still cost 19.34 cents, and thus $55,77 a day per device.

Other blockchains might be less expensive, but most likely not enough. Also, other general purpose blockchains are less popular and probably less robust and stable.

This concludes the characteristics table of blockchain for this work (see 4.1.6):

	Decentralized	Cheap	Community-driven	Easy to use	Open	Fast
Blockchain	yes	no	yes	yes	yes	no

4.3 Potential solutions

This section explores a few potential solutions regarding the “cheap” and “fast” objectives of the targeted platform. These two characteristics go hand in hand when it comes to blockchains, as most of them make it possible to make transactions go through faster by paying higher fees. However, as the transactions per second are limited, it is not possible for every transaction to be cheap and fast at the same time. Thus, a blockchain that scales better should also be cheaper.

Similarly, ideas are explored for the storage aspect, as it is not possible to remain cheap when every user must store all historical data forever.

4.3.1 Transactions per second

This section covers techniques to improve the TPS metric, as it is clearly too low on current public blockchains for the use case targeted by this work, as seen in 3.3.3.

4.3.1.1 Sharding

Sharding is a popular technique to horizontally scale distributed systems. They are already heavily used in database systems, so it is not a strictly blockchain-related technology. The main gist of it is that the ledger gets split into different shards. Each of them is independent, and all of them combined form the global ledger. Each shard is responsible for specific parts of the global ledger.

Shards can be defined arbitrarily. For example, the system could be coded in order to put each contract or wallet on a shard based on the first hexadecimal character of their address. In this case, addresses from 0x0001 to 0x0fff would go to shard 0, and addresses from 0xf000 to 0xffff would go to shard 15, and there would be a total of 16 shards. This would be a generic way to distinguish shards, and might be a bit too naive. For example, it could be problematic for a big contract to be tied to one single shard. But distributing it on multiple ones could degrade performance. The choice of how data is sharded is thus important, as it also is in distributed databases.

Alternatively, shards could be distributed in a geographical way. For example, each country or state could have their own shard. This choice heavily depends on the use case. This second option could potentially fit the use case of an IoT platform better. It would be easier to locally sync the shard of a single country instead of all of them, and it would actually make a lot of sense if one only required data for that specific country, or only a subset of them.

For general public blockchains, it is necessary for shards to be able to communicate together. Users likely would need to transfer value from one shard to another, or have two contracts on different shards communicate together. For the IoT platform proposed by this work, this is not required, as there is no global state per se. Devices never communicate with other ones, and data points always stay in the same shard. The only case in which inter-shard communication could make sense is if a device's location were to change in such a way that it had to be moved to another shard. However, it does not make sense on this platform to move a device and its data from one place to another, as targeted IoT devices should not roam. If a device needs to move, it would make sense to generate an entirely new identity for it.

In practice, shards are planned to be used in Ethereum v2, which is the new version of Ethereum expected to go mainnet in 2021. This newer version is expected to have 64 distinct shards, and they should increase the TPS substantially. On Ethereum v2, each shard is a whole distinct blockchain, with its own wallets, account balances, smart contracts etc.

As a conclusion, sharding would make sense for an IoT platform.

4.3.2 Storage requirements

As explained in 4.1.1, storage requirements for IoT platfoms are huge. The minimal amount of data generated by a million IoT devices over one year was calculated at 16 TB. This is way too big for general consumer hardware, as 16TB+ hard drives go for more than €600 [37]. Not to mention the fact that hard drives would certainly be too slow for this type of workload given their low IOPS (Input/output operations per second). As the platform is supposed to be accessible to as many individuals as possible, it is necessary to find solutions to limit that requirement.

4.3.2.1 Pruning

Pruning is already heavily used in blockchain. In general, there is a distinction between three types of nodes:

- Light node: does not have the entire blockchain, and thus needs to trust external nodes.
- Full node: has all blocks and their transactions, and verifies the entire chain, but only keeps a few recent global states in case of reorganizations. The rest is pruned away. They are perfectly secure and users can query the latest global state. However, querying the state of a contract at a previous block would be extremely slow on a full node, as it would have to recalculate the entire chain up until that point.
- Archive node: has all the data like full nodes, but also keeps older intermediate global states in order to allow for fast queries of historical data. They are as secure as full nodes, but also allow fast queries to older states. This type of node is required if one needs to query for older intermediate states.

In between those three types, it is possible to prune any arbitrary amount of data in exchange for lesser speed or security. For example, a local node could download all blocks from a full node while syncing, update its global state, and then remove the raw blocks. This way, it would save on storage, as it would not need to keep track of the required data to rebuild the blockchain. Consider a counter contract, that only has one function to increment an integer, and one to read its value. Assuming the counter starts at 0, there could be an infinity of transactions that constantly increment the counter by one. A full node would need to store all of these transactions, which would take a lot of storage. With pruning enabled, the node could delete these transactions once they happened, and only keep the global state. In this very specific case, it would reduce the storage requirements to the size of an integer.

Sadly, it is not exactly that easy. The node would still need to keep differences between a few most current blocks in case of reorganizations, where the contract would need to be reset to an earlier state. However, chain splits are fairly rare on popular public blockchains, so reorganizations are also rare and small. Storing the difference of a few blocks should thus be sufficient.

A second disadvantage of this method is that this loses data. Once a node starts pruning raw blocks, it is no longer able to serve as a synchronization source for new nodes joining the network. A blockchain also needs to have at the very least one full node available permanently. Without any of them, it is not possible to sync a blockchain, as it is not possible to verify the entire chain. Basically, this allows smaller nodes to keep in sync with the global state by reducing the storage requirements, but does not solve that requirement for all nodes.

4.3.2.2 Ephemeral blockchains

Ephemeral blockchains, also called micro blockchains [7], are a theoretic concept in which blockchains are used for a short amount of time and then recycled or deleted. After that, a new chain is launched with its own genesis block. The final global state of the previous blockchain can be transferred to the new one, effectively creating a snapshot. To the authors' knowledge, an actual production blockchain or even implementation does not exist at the time of writing. Literature is also sparse in that space.

This approach has several issues. First off, blockchains work because it is possible to cryptographically verify that the entire chain is valid from its genesis block to its most current one. This is easy to do when there is only one, commonly agreed upon genesis block from which everything starts. With ephemeral blockchains, there would need to be a way to agree on the new genesis block, and someone would have to generate it. As this work focuses on completely decentralized systems, this implies the necessity of consensus. However, decentralized consensus is impossible to obtain without trusting connected peers or having a trusted initial state.

The process of moving from one ephemeral blockchain to another is very error prone, as once the previous chain is replaced and deleted, there is no way to make sure that the current blockchain is actually the right one. This opens up a lot of attack vectors. A potential solution for this would be to have one small persistent blockchain dedicated to the task of keeping a reference to all ephemeral genesis blocks. This way, if a split occurs or bad actors start to advertise new chains with invalid genesis blocks, there is always a single source of truth.

For this work, ephemeral chains could be used to store the data generated by IoT devices. This would allow the platform to use current technologies and to avoid the storage issue. However, older data would get increasingly difficult to come by. It also would not make any sense to transfer the final state of the previous blockchain to the new one. The platform is basically an append-only log, so copying the state over would basically mean copying all the data, and it would not solve the storage issue.

As a whole, this solution would make it a bit more difficult to search for data. One would need to query multiple different blockchains, some of which might not even exist anymore. Bootstrapping also becomes more difficult.

Now, while this may solve the storage problem, it still does not improve the TPS capacity provided by blockchains. It also still forces nodes to synchronize data of all IoT devices instead of just the ones they need, at least if the node operators want to stay in the ecosystem. Access to older data would also become harder.

4.3.2.3 Sharding

Sharding also indirectly reduces storage requirements, as nodes no longer need to keep the entire data set and only need the data from their shard. Roughly speaking, this divides the storage requirement by the number of shards on the network.

4.4 Potential custom blockchain

If someone really wanted to use blockchains to store data of millions of IoT objects, this works' authors' best guess would be to use a combination of geographical sharding as well as ephemeral blockchains whose genesis blocks are kept track of in a persistent blockchain. The latter could actually be a smart contract on any popular blockchain, although it would require some logic linked to the consensus algorithm used to generate new genesis blocks.

The advantages of this solution are that storage would not need to be allocated for eternity, while still allowing archival initiatives to store older ephemeral chains. Scaling would also be improved thanks to sharding, making it possible to reach arbitrarily high transactions per second based on the number of shards. This can be achieved thanks to the fact that all the data is strictly independent.

The disadvantages are that data retrieval for older data becomes complex and relies on people storing non-current chains. The granularity of data storage is also bad, as one would need to store the entire ephemeral chain for historical data to remain accessible, as well as the currently most recent chain. This does not scale at all and is unreliable at best.

In conclusion, a potential custom chain would not leverage any characteristics of blockchain, and it thus does not make any sense to explore this any further.

Such a solution would make sense for data that needs to be available forever and strictly verifiable, but this project does not meet these requirements. In fact, it already solely relies on non-trusted data from willing participants, and blockchain technology would not help make this any more trustful. There is also no financial transaction going on, and blockchains also do not scale to enough transactions per second. This is also the reason why ephemeral blockchains alone (4.3.2.2) would not be helpful in this case.

Chapter 5

Proposed protocol

Based on the Blockchain for this work chapter (4), it is fair to conclude that blockchains do not make sense to store the type of IoT data this work is studying. However, the data storage should still be entirely decentralized, and blockchain technology can still be used for some aspects of the platform, as long as it is not for storing a lot of ever-growing data.

The next logical technological choice is a decentralized P2P network. This type of network is already used for file sharing, and is used as a building block for all blockchain-related projects, as they rely on it to broadcast blocks and transactions [2].

A potential solution could take ideas from OrbitDB (2.5) and IPFS (2.3). However, OrbitDB has many features that do not make sense for this project, and both it and IPFS are a bit too rigid for this use case. Therefore this work proposes a solution based on IPFS' network layer only, called `libp2p` (5.2.1), and builds the rest on top of it.

5.1 Scope

The proposed protocol defines a best-effort network, on which data is not guaranteed to be available by any measure. Nodes can decide what they want to store by themselves. They can store all the data of the network, only store data related to some Sensors, or store nothing at all. They can also choose which data points they store. For example, a node can choose to store one data point a day, even if the related sensor produces one each minute. As such, there is no way of knowing if the network is currently holding all the data generated by a sensor or only parts of it. It is also impossible to know if more archived data will be available at some point, or if it disappeared for good.

The query language is also kept at a minimum. It is only possible to query data from a specific device, and bulk retrieval is not possible, although a few potential solutions are explored. The only allowed filter is a ranged query on publishing date. However, region-based bulk retrieval is studied as an intermediary solution.

5.2 Architecture

The proposed architecture relies on 4 main components that are represented on figure 5.1:

- Sensors: devices that collect data.
- Ingestors: some device that can pull data from Sensors, or that accepts data that is pushed by Sensors. They create and sign Data Points and then forward them to a Node.
- Nodes: software that runs the actual decentralized storage platform. They participate in the P2P network by storing and transfering data.
- A Smart Contract: it is used as a trusted ledger and contains information about all Ingestors on the network.

Sensors, Ingestors and Nodes can be collocated on the same hardware if it is powerful enough. The architecture is designed to be flexible, and can accommodate all types of IoT devices.

The components are defined separately in order to allow for any type of device to be compatible. For example, some microcontrollers one could find on IoT devices are not capable of doing asymmetric encryption. Some do not have a full networking stack, and thus cannot participate in the peer to peer network. Therefore, this work assumes that a Sensor is the most basic device, and that the Ingestor can serve as a middleman between it and the Nodes.

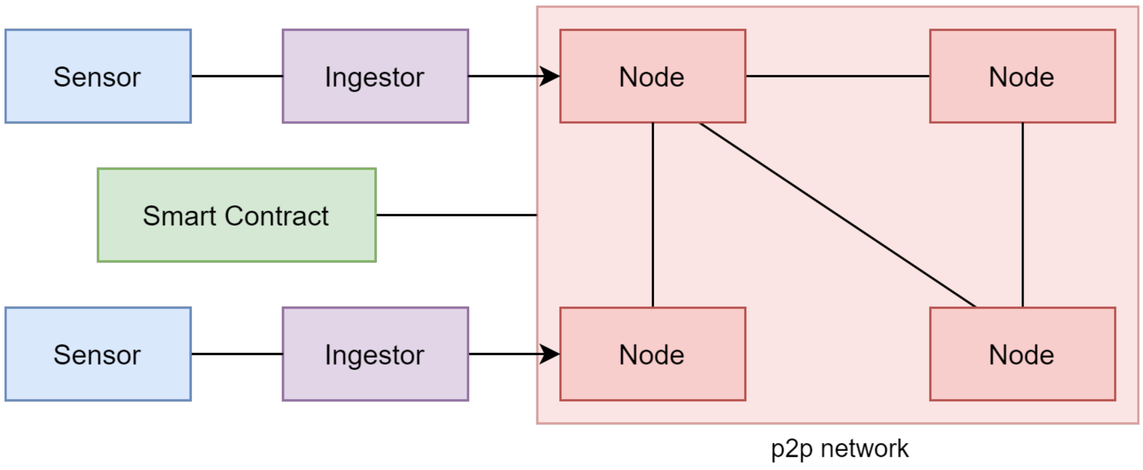

Figure 5.1: Each component in the architecture and how they communicate with each other.

5.2.1 Libp2p

`Libp2p` is the networking layer of IPFS, defined by its authors as [23]:

A modularized and extensible network stack to overcome the networking challenges faced when doing peer-to-peer applications.

It is used by many crypto-related projects like Helium [18] and Swarm [32]. It provides an easy way to define P2P networks in a flexible way, which is why it was chosen for this work. All communications between Nodes happen over the `libp2p` network.

5.3 Components

5.3.1 Sensors

The network requires IoT devices that collect and publish data. In this work, the software running on these devices is referred to as Sensors. These devices basically do not need any computing power. The target hardware is microcontrollers. They only need to be able to collect data and make it available or forward it to an Ingestor. The protocol used between Sensors and Ingestors is considered a black box and can be chosen arbitrarily based on requirements and preferences.

5.3.2 Ingestors

Ingestors are software that collect data from Sensors. They are assumed to have a full network stack, a way to tell the time and be powerful enough to sign messages using asynchronous cryptography. The target hardware is powerful microcontrollers with networking, or more fully-fledged devices. They can work in either a push or pull configuration. This means that either the Sensor contacts the Ingestor to send data to it, or the Ingestor contacts the Sensor to pull data from it. Ingestors have an identity, meaning a pair of private and public keys. They create and sign Data Points. A Data Point is data collected from a Sensor and represented in a specific format. Those points are then signed and forwarded to a Node.

Ingestors and their identity are specific to one Sensor, which are specific to one specific location. Before being used on the network, the identities need to be registered on the Smart Contract, as illustrated on figure 5.2. This link makes it possible to verify that a Data Point was published with a valid signature from a device registered on the platform.

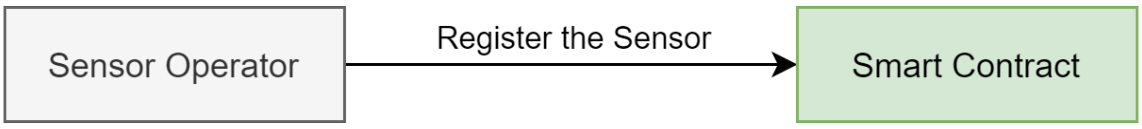

Figure 5.2: Identities are registered in the Smart Contract.

5.3.3 Nodes

Nodes are software that form the P2P network. They are responsible for storing data and transferring it. They use a specific protocol to request data and to make it available to other peers. As such, they are required to have a full network stack and be powerful enough to handle cryptography and store data. The target hardware is servers, but the software could run on something like a Raspberry Pi 4.

All communications happen on the `libp2p` stack. Nodes use either a Publish/Subscribe network[1] or direct connections to other Nodes. Those connections can happen over any protocol compatible with `libp2p`. In the proof of concept, both TCP and WebSocket are used. The specific protocol used for PubSub is Gossipsub[2], which has moderate amplification factors and good scaling properties.

5.3.4 Smart Contract

The Smart Contract, deployed on the Ethereum blockchain, is used to keep track of all registered Sensors. It also makes it possible to search for devices meeting particular requirements, like location or device capabilities. It also contains the schema of the data collected by the platform.

Although not strictly required, the Smart Contract provides some degree of security to the network, preventing DoS (Denial of Service) attacks. Paying for devices to be added to the platform hinders attackers by making it expensive to simulate many devices with non-compliant behavior.

In addition to the security aspect, the Smart Contract is also an elegant solution that allows this platform to use technologies and a blockchain that have had the chance to prove themselves without the need of reinventing the wheel. Without it the platform would have to implement some type of consensus algorithm amongst all nodes in order to maintain a distributed ledger of registered devices.

The Smart Contract also allows for future improvements, like adding an economy further down the line. It could also keep track of misbehaving nodes in order to ban them from the network.

The data stored by the Smart Contract can be represented in the following way:

```json
{
  "schema": {
    "temperature": "float",
    "humidity": "float"
  },
  "devices": [{
    "address": "0x0000000000000000000000000000000000000000",
    "location": [0, 0],
    "capabilities": ["temperature", "humidity"]
  }]
}
```

[1]`https://docs.libp2p.io/concepts/publish-subscribe/`

[2]`https://github.com/libp2p/specs/tree/master/pubsub/gossipsub`

5.4 Schemas

The protocols use a few different data schemas to transfer data. These are represented in Protobuf (Protocol Buffers), which is also used to serialize the data optimally in order to save on bandwidth requirements.

5.4.1 General

5.4.1.1 Data Point

Defines basic Data Point collected by a Sensor. The schema is defined in the Smart Contract, and all Data Points need to adhere to it. The following schema is an example in which the temperature is represented as a signed integer, and the humidity as an unsigned one. Both values are precise to one digit. Thus, a temperature of 34.5°C is encoded as the integer 345. The humidity is a value in percent, and can have values between 0 and 1000 given the one digit precision.

```
message DataPoint {
  sint32 temperature = 1;
  uint32 humidity = 2;
}
```

5.4.1.2 Timestamped Message

Each DataPoint needs to be encapsulated in a Timestamped Message. This is mostly used to know at what time a Data Point was created.

```
message TimestampedMessage {
  uint32 date = 1;
  oneof data {
    DataPoint dataPoint = 2;
  }
}
```

5.4.1.3 Signed Data

Finally, the Timestamped Messages are signed by the Ingestor.

```
message SignedData {
  bytes source = 1;
  bytes signature = 2;
  TimestampedMessage data = 3;
}
```

5.4.2 Requesting data

5.4.2.1 Request

```
message Request {
  string id = 1;
  repeated string multiAddresses = 2;
  Query query = 3;

  message Query {
    bytes device = 1;
    uint32 from = 2;
    uint32 to = 3;
  }
}
```

5.4.2.2 Request

```
message Reply {
  string id = 1;
  Results results = 2;

  message Results {
    uint32 count = 1;
    uint32 from = 2;
    uint32 to = 3;
  }
}
```

5.5 Protocols

5.5.1 Registering an Ingestor

An Ingestor is registered on the Ethereum Smart Contract, similarly to 4.2.1.1. This basically only involves a contract call transaction.

5.5.2 Publish a Data Point

Publishing a Data Point is done in a certain number of steps, as it must go through each component of the architecture. Figure 5.3 illustrates the interactions between components.

0. The Sensor collects data.

1. One of:

    - The Ingestor pulls the data from the Sensor.

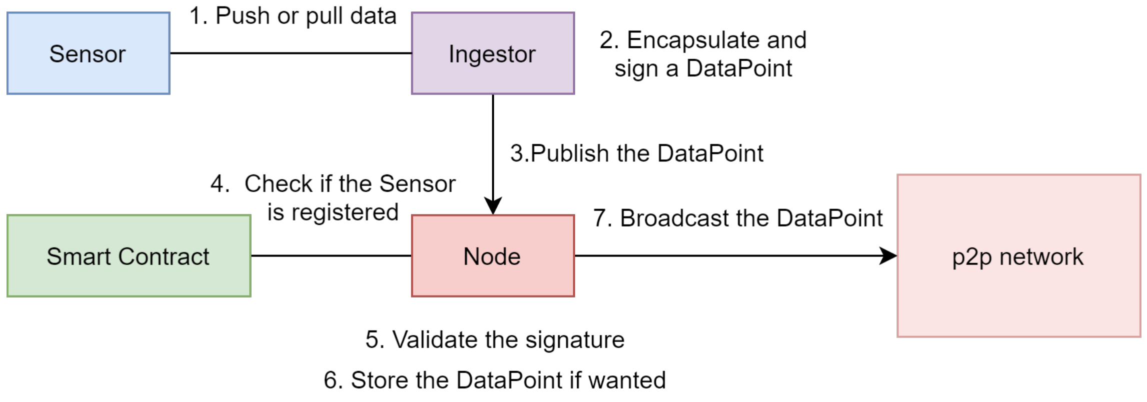
Sensor
1. Push or pull data
Ingestor
2. Encapsulate and sign a DataPoint
3.Publish the DataPoint
4. Check if the Sensor is registered
Smart Contract
Node
7. Broadcast the DataPoint
p2p network
5. Validate the signature
6. Store the DataPoint if wanted

Figure 5.3: Protocol used to publish a Data Point onto the network.

  - The Sensor pushes the data to the Ingestor.

2. The Ingestor creates a Data Point with the data from the Sensor, in accordance with the schema defined by the Smart Contract. It then timestamps and signs the Data Point with its identity.
3. The Ingestor sends the Data Point to a Node of its choosing, ideally one operated locally.
4. The Node queries the Smart Contract to fetch the identity of the Ingestor, and makes sure that the device is registered and the signature is valid.
   - If not, the Data Point is rejected and the connection with the Ingestor is closed.
5. The Node decides whether it wants to store the Data Point received, based on its local configuration.
6. The Node broadcasts the Data Point to the P2P network, which is a publish action on the PubSub topic linked to the Ingestor (`device-$ingestorId`).

Regarding step 4, the Node can either synchronously call the Smart Contract to make sure that the device is registered, or alternatively use local caching. It is also possible to keep the list of all registered devices synchronized locally, although this option does not scale as well if there are a lot of devices.

The last step allows every peer subscribed to the topic of this device to be immediately made aware that a new Data Point is available, without spamming peers that are not interested in the data of the publishing Ingestor.

5.5.3 Locating devices

In order to search for data in a specific location, one first must find devices that meet the criteria. This is done by querying the Smart Contract. It is similar to what was proposed in the blockchain-based solution. The relevant sections is 4.2.2.1.

5.5.4 Subscribe to devices

Subscribing to devices is basically just a subscription to the topic linked to the identity of the publishing Ingestor (`device-$ingestorId`). As soon as new Data Points are available, they are broadcasted on that topic and the subscribed nodes get them automatically. This allows nodes to keep in sync with the latest Data Points. For security, the signature of each Signed Data needs to be checked each time a Data Point is received. The source address should be compared with the one in the Smart Contract.

5.5.5 Requesting a device's data

Sometimes, having the latest Data Point is not enough. In fact, this does not allow people to make statistics or graphs of the evolution of metrics. It is not reasonable to expect anyone who needs archived data to have been around since the first required Data Point was published, so there needs to be a way of requesting data from other peers.

In order to do that, there needs to be a way to advertise and fetch what data is available on each node. This could be done by connecting to every peer and asking them what data they have. However, this would not scale, and is not practical at all.

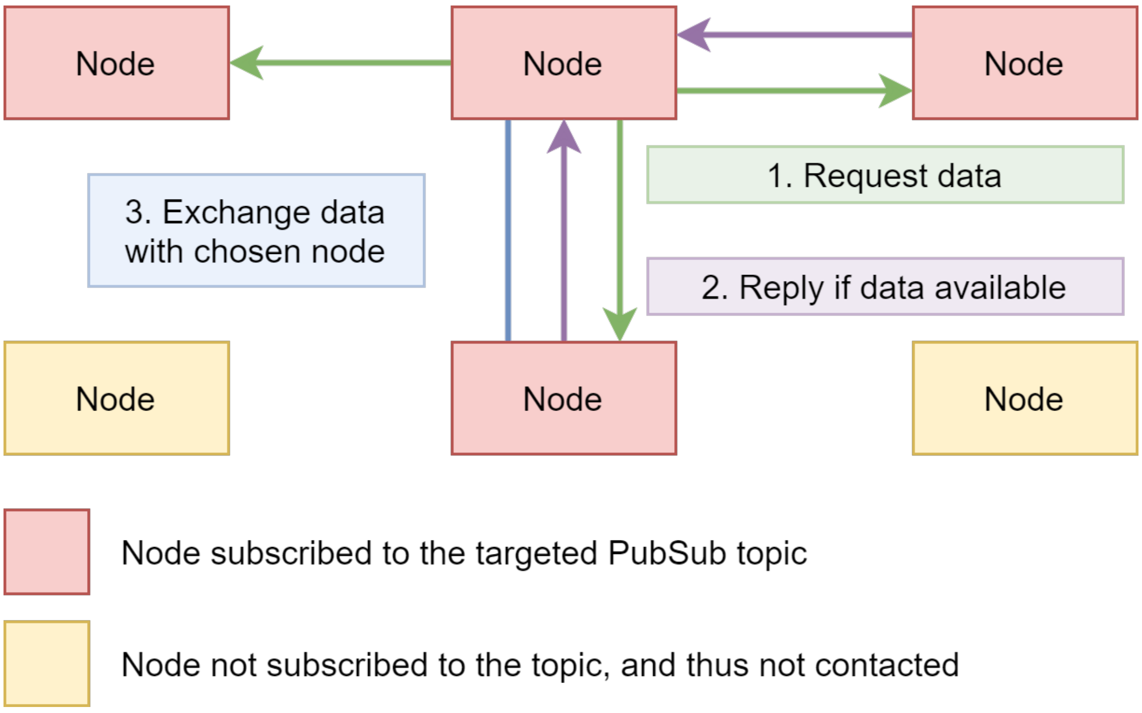
Node
Node
Node
1. Request data
3. Exchange data with chosen node
2. Reply if data available
Node
Node
Node
Node subscribed to the targeted PubSub topic
Node not subscribed to the topic, and thus not contacted

Figure 5.4: Protocol used to request data from other peers using PubSub exclusively.

Another solution, highlighted in figure 5.4, could be to broadcast a message on a topic specific to data requests, for example `request:device:$ingestorId`. Nodes that need data could then post a ranged data request, for example from January 1st to December

31^{st}, and nodes with the available data could then answer on that same channel. However, this would spam all nodes on said channel, even those that do not need the data. Thus, it would be better to open a one on one channel with the requester and then send them the data directly. Request messages could have the following schema:

```
message Request {
  string channel = 1;
  Query query = 2;

  message Query {
    bytes device = 1;
    uint32 from = 2;
    uint32 to = 3;
  }
}
```

Nodes with the data available (repliers) could then respond on the channel specified on the request with the following schema, where SignedData is the schema defined at 5.4.1.3 and contains DataPoints (5.4.1.1):

```
message Reply {
  repeated SignedData results = 1;
}
```

While the one to one channel solves to one to many spamming issue, it does not solve the many to one spamming issue. That is, if multiple nodes have the data and then start sending it to the requester together, the requester would end up being spammed again. In order to solve this, repliers publish their own random channel to the requester's channel, after which the requester can choose which replier to confirm the request to. Additional metadata is added to the reply so that the requester can make an educated decision on which replier to accept data from. Repliers that do not get a confirmation after some period of time (say 5 minutes) close the channel. All the data retrieved can be verified, as every DataPoint is individually signed.

The reply would contain the number of results as well as the first and last available dates, and the following schema could be used:

```
message Reply {
  string channel = 1;
  Results results = 2;

  message Results {
    uint32 count = 1;
    uint32 from = 2;
    uint32 to = 3;
  }
}
```

The confirmation would then be sent on the channel specified in the Reply, and would be an empty message:

```
message ReplyConfirm {}
```

At that point, the replier would send all the matched data with schema specified at the beginning:

```
message ReplyContent {
  repeated SignedData results = 1;
}
```

The last issue with this protocol is that it only relies on PubSub, which is made for many-to-many relations, and can be fairly slow. It is also nonsensical to route data across an entire network just so two peers can communicate. The only required change is to replace all channels in the messages by an array of multi addresses[3]. This way, nodes can send a "link" to a direct connection in a multitude of different protocols like TCP (Transmission Control Protocol), UDP (User Datagram Protocol), WS (WebSocket) or even through relays if direct connections are not possible.

In the end the final protocol, illustrated in figure 5.5 can be summarized with the following steps:

1. The requester broadcasts a request on the PubSub topic of the targeted Ingestor (on the `request:device:$ingestorId` topic).
2. Nodes that have data, called repliers, contact the requester directly using any of the compatible multi addresses, and specify what data they have available.
3. The requester chooses one or multiple Nodes and downloads data from it. Connections to other nodes are closed.

The reply and confirmation messages now happen over a direct connection. In addition to that, a unique identifier is added to each message exchanged during this protocol, in order to allow nodes to keep track of which reply is related to which request.

This protocol could be extended later on to include pricing information, once some type of economy is put in place.

Once the data was successfully retrieved, it can be made accessible to other nodes by subscribing to the appropriate topic and responding to requests.

The final schemas used by the data request protocol are the following:

```
message Request {
  string id = 1;
  repeated string multiAddresses = 2;
  Query query = 3;
```

[3]https://github.com/multiformats/multiaddr

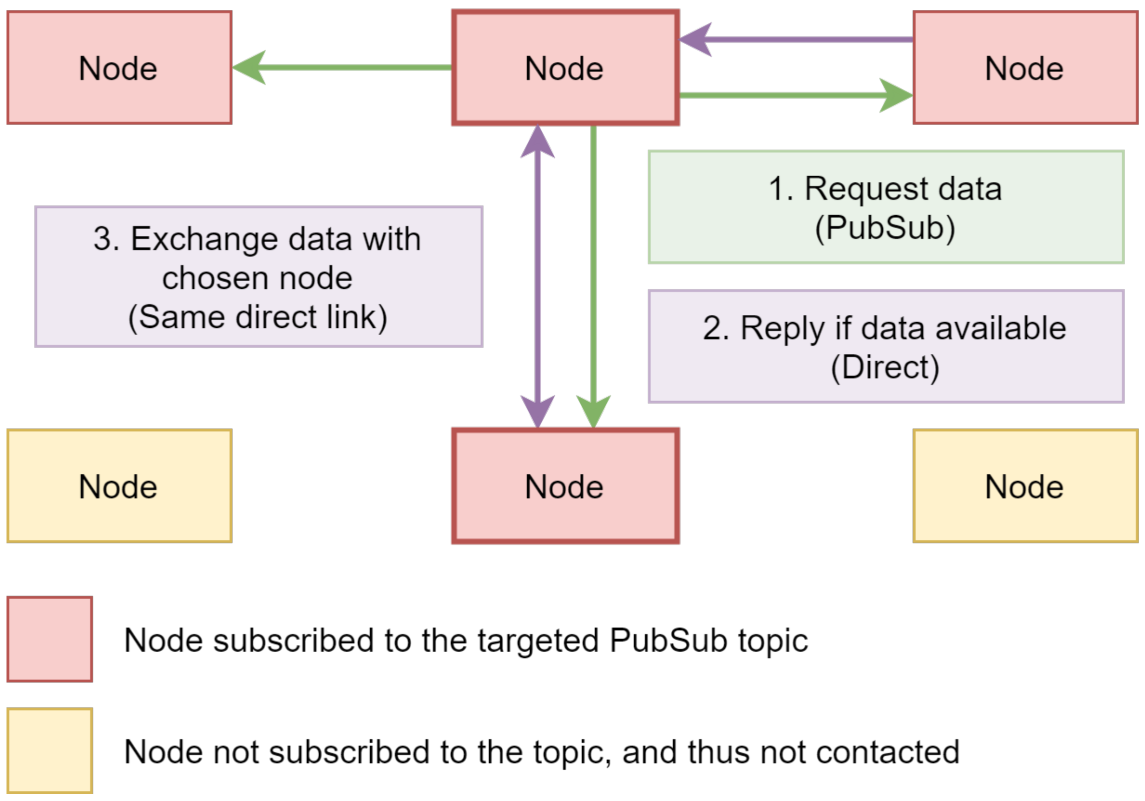
Node
Node
Node
1. Request data (PubSub)
3. Exchange data with chosen node (Same direct link)
2. Reply if data available (Direct)
Node
Node
Node
Node subscribed to the targeted PubSub topic
Node not subscribed to the topic, and thus not contacted

Figure 5.5: Protocol used to request data from other peers using PubSub and direct connections.

```

  message Query {
    bytes device = 1;
    uint32 from = 2;
    uint32 to = 3;
  }
}

message Reply {
  string id = 1;
  Results results = 2;

  message Results {
    uint32 count = 1;
    uint32 from = 2;
    uint32 to = 3;
  }
}

```

```
message ReplyConfirm {
  string id = 1;
}

message ReplyData {
  string id = 1;
  repeated SignedData results = 2;
}
```

5.5.6 Requesting additional data

The section 5.5.5 detailed the protocol used to fetch data about a certain device. However, this is punctual, and some data might be missing after that initial request. In order to fetch the Data Points published live from a device, it is sufficient to subscribe to the appropriate PubSub topic (`device:$ingestorId`). However, in order to fetch data that was not available during the initial request, it is necessary to find a new solution.

This is actually not too complicated, as a node can remember with which other ones it has exchanged data already. So, when a new node comes online, one can just check if it's a new one, and if so send it the request directly. That way, the newly online node can reply if it has more data, and the requester can fetch missing data.

The detection of nodes coming online is handle via PubSub. When data providers come online, they subscribe to the `request:device:$ingestorId` channel in case a requester publishes a request. When a provider subscribes, other nodes are made aware, and can thus directly connect. This is implicitly done by `libp2p` by broadcasting a silent message on the topic.

This solution does not require any new protocol or schema, and builds upon pre-existing ones.

5.6 Conclusion

The solution proposed in this chapter matches the following criteria:

	Decentralized	Cheap	Community-driven	Easy to use	Open	Fast
Proposal	yes	yes	yes	yes	yes	?

The "fast" aspect can only be studied theoretically at this point, as it was not possible to simulate a network with an accurate state. However, as no consensus is involved, all operations should be fast and happen in real time. Retrieval of data could be slow if only a badly connected peer owns the data, but should eventually become faster if the related device is popular.

Chapter 6

Proof of concept

A proof of concept was developed and the source code was published on GitHub[1]. Most of the code is written in JavaScript, either for Node.js or for the browser.

The Smart Contract was deployed at 0xcb20cb08475edc9b4b1652a879f20c8d13fc6483 on the Görli testnet.

6.1 Smart Contract

In order to be able to search for devices in the Smart Contract, an efficient data structure needed to be implemented. Although read operations are free on Ethereum, they are only allowed to use a specific amount of computing power. Without going into too much details, this is related to Ethereum's gas limit [13]. To be clear, implementing an efficient algorithm always makes sense either way.

The chosen data structure is the quadtree [40], which allows for efficient storage and retrieval of two dimensional data, while remaining relatively straightforward. The exact source code can be found in the `platform/ethereum/contracts/Platform.sol` folder. To the author's knowledge, this is the first time a quadtree has been implemented in Solidity and ran on Ethereum.

The contract only implements two functions, which implement the 5.5.1 Registering an Ingestor and 5.5.3 Locating devices protocols:

- register: takes an Ingestor ID as well as the coordinates (5.5.1 Registering an Ingestor)
- find: takes the center coordinates with height and width as well as the maximum number of elements to return (5.5.3 Locating devices)

[1]`https://github.com/filoozom/tfe-uliege-code/tree/main/platform`

It also exposes a constant that defines how latitude and longitude are transformed from floats to integers (by multiplying or dividing by $10^{LOCATION\_MULTIPLIER}$).

Initially, the Smart Contract was supposed to also contain a schema of the DataPoints used on this platform, and the devices also had metadata attached. For example, they could have included a name or a list of properties, like which sensors are available, what precision they have, and at which interval data should be published. However, this didn't make it into the proof of concept in order to focus on the more important aspects.

6.2 Sensor

The Sensor code is the simplest implementation of some code that could run on actual micro-controllers. It's written in Node.js too, but basically only opens a TCP socket and sends 8 bytes on it before closing it. As the protocol used between the Sensor and the Ingestor is considered to be a black box, there is no reason to detail this section further.

This is the first part of the 5.5.2 Publish a Data Point protocol.

6.3 Ingestor

The Ingestor implements one `libp2p` protocol as well as the black box protocol used to communicate with the Sensor. The latter is not too important. The `ingest` protocol encapsulates the data received by the Sensor, timestamps it and then signs it. The result is then sent over a direct channel to a specific Node.

This is done in order to encourage the use of a local Node, as opposed to relying on external ones. The Ingestor could broadcast the Data Point on the `device-$ingestorId` PubSub topic for other Nodes to pick up, but this would likely result in immediate loss. This stems from the fact that there could be no one syncing that device, and that therefore no one would get that Data Point and store it. Thus, it is more interesting for the network to run a local Node, which would act as a buffer to keep the data around for a few months before getting rid of it.

This makes for the phases 2 and 3 of the 5.5.2 Publish a Data Point protocol.

6.4 Node

The Node is the most important part of the platform has it does all the heavy lifting. It implements a series of protocols and an private API that can be used to interact with it. Just to be clear: that API is never used by any other Node in a centralized fashion, and is only used for configuration purposes. An interface could also be built upon that API.

6.4.1 Libp2p

6.4.1.1 Ingest

This is an implementation of the receiving end of the protocol implemented in 6.3 Ingestor. First, it validates the signature of the Data Point. Then, it checks if the Ingestor is registered on the Smart Contract, in which case it broadcasts it on the related PubSub topic (`device-$ingestorId`).

The Node can be configured with the API in order to choose from which devices it should sync data.

These are parts 3 to 7 of the 5.5.2 Publish a Data Point protocol.

6.4.1.2 Request

This is the most complicated protocol, as it uses PubSub and a variety of messages on direct connections. However, it strictly does as described in 5.5.5 Requesting a device's data, and does not need additional details.

6.4.1.3 Sync

This protocol is straightforward and mostly consists of publishing and subscribing to PubSub topics related to devices, as well as storing data. It is the implementation of 5.5.4 Subscribe to devices.

6.4.2 Ethereum

6.4.2.1 Sync devices

This protocol is used as a caching layer between the Smart Contract stored on Ethereum and the Node. On first start up, and until the local state is synchronized to the state of Ethereum's current block, the Node pulls all `NewDevice` events from the Smart Contract. This is done from the latest local block up to the most current Ethereum one. Once the initial sync is done, the Node listens for more `NewDevice` events in real time. All those events are decoded, and devices are stored in a local database.

This allows the Node to always know which devices are registered in order to validate incoming Data Points as fast as possible.

This is a caching layer related to step 4 of the 5.5.2 Publish a Data Point protocol.

6.4.3 API

The API is used to interact with the node. Without going into too many details, it makes it possible to fetch and request data, as well as to manage device subscriptions. It can also be used to fetch the cached devices from 6.4.2.1 Sync devices.

6.5 Web interface

The web interface has three main features:

- Register devices on the Smart Contract
- Search for devices and display them on a map
- Subscribe to updates

It runs in a completely decentralized fashion, meaning that it only uses Ethereum and the `libp2p` network, without the need to deploy a specific gateway or to use centralized APIs. As such, it basically functions like a fully-featured Node in the browser, except that it cannot store data. It thus implements the same protocols as described in 5 Proposed protocol.

6.5.1 Device registration

The registration is a transaction that calls the `register` function on the Smart Contract. This is handled by MetaMask[2], a browser extension that acts like an Ethereum wallet and makes it possible to sign transactions.

Basically, the user can choose the location on a map where their device is. Then, they insert the Ingestor's identifier. The web interfaces finally generates the transaction, after which the user must sign it using MetaMask. Once the transaction is mined, the device automatically appears on the web interface.

6.5.2 Device search

The web interface is mostly composed of a map that displays devices. It automatically calls the `find` function on the Smart Contract and only displays the results. This way, the user can choose what location they are interested in and only query those IoT objects instead of having to fetch all the data.

6.5.3 Subscribing to devices

In order to demonstrate the Publish/Subscribe and Relay networks, the web interface implements the device synchronization protocols. When selecting an IoT object on the web interface, it automatically subscribes to it over the `libp2p` network. Data is then automatically kept up to date with the latest Data Points as soon as they are published.

This directly uses the 5.5.4 Subscribe to devices protocol.

[2]`https://metamask.io/`

6.6 Takeaways

The proof of concept that implements the proposed protocol made it possible to reach a few conclusions regarding this work. Testing was mostly done manually, except for a few unit tests for the Smart Contract.

First, the implemented platform proves that it is possible to design a decentralized system integrated with blockchain that is easy to use. The web application gives an easy access to all aspects of the platform, and abstracts the interactions with the Smart Contract away. The P2P network is also naturally integrated and completely hidden away from the user. For all intents and purposes, the user experience is exactly the same as when browsing a bog-standard website, despite communicating with many remote Nodes.

Secondly, the platform is feature complete, and every aspect of the proposed protocol was implemented. This is the case for both the browser and native experiences, although the cutting edge technologies used in the browser require the latest version of Chrome at this point. Other browsers could also work but were not tested. The MetaMask plugin is also required to access the Ethereum network in the browser, although alternatives exist.

Lastly, some insight was gained as to the performance aspect of this platform. In theory, it should be way faster than blockchain, as no consensus is required and data can be ingested all over the world in an instant. Basic testing showed that to be the case on a containerized deployment of a few hundred nodes. However, this data needs to be taken with a grain of salt, as testing the performance aspect of this platform requires more in-depth work. In particular, network latency is certainly a huge factor for a network deployed worldwide. It is also complicated to simulate an accurate network, not to mention the fact that "transactions per second" are not an appropriate metric for this platform. As this solution can run in network splits without any issues, it would be trivial to spin up a few hundred nodes independent of each other and then post millions of data points per second on each of them, reaching hundreds of millions of "transactions per second". Calculating the actual synchronization capacity of an entire network is basically impossible without a testnet with a variety of users in different locations, having different networking, capacity and access characteristics. However, as synchronization mostly relies on GossipSub, an already proven and benchmarked protocol [45], it is safe to assume that the technology can scale.

Chapter 7

Future work

The 5 Proposed protocol chapter only includes protocols required to achieve a minimal viable product. However it would need a few extensions in order to scale. This chapter discusses a few ideas of avenues for improvement without going into too much detail.

Less technical aspects like real-world testing, economical and legal aspects are also addressed. These are required for an actual production grade network.

7.1 Bulk retrieval of data

The described protocols work fine for a few devices, or for data consumers that only need data from a few devices. It becomes a bit more problematic when one wants to display data from tens or hundreds of thousands of devices. The current protocol requires Nodes to subscribe to Data Points on a per-Sensor basis. This means that if a Node is interested in the Data Points of 10 000 Sensors, it needs to subscribe to 10 000 PubSub topics, which does not scale at all.

In order to achieve a platform with such functionality, there would need to be nodes that aggregate data from thousands of devices at a time. This could be done globally for every device on the network, geographically or based on some discriminator. Again, this would need to be done on a voluntary basis.

An idea based on Sensor groups is worth exploring. Basically, groups of Sensors are defined, and instead of requesting data from one Sensor, one can request data from the entire group at once.

7.1.1 Arbitrary Sensor groups

The easiest solution would be to define arbitrary fixed groups of Sensors. For example, each country, state or city could become a group. In this case, each time a Sensor publishes data, it does so on its own PubSub topic as well as the one for each corresponding group.

A Sensor in Liège would thus, for example, publish on the `Liège`, `Belgium`, `Europe` and `Earth` topics. However, this is too rigid.

Also, having an `Earth` group would only really work if giant Nodes were able to store every single Data Point ever published on the entire platform, which is not possible. The size of these group is thus an important consideration.

7.1.2 Mutable Sensor groups

Having mutable Sensor groups is a lot harder than having fixed ones. Consensus is required, as well as voting. We would not want individual Nodes to create new topics as they please, as there would quickly be a huge amount of them, and it would be unsustainable.

To achieve mutable groups, the author believes that it would be best to deploy a smart contract on a public blockchain, where Node operators can propose new groups. These get added once enough people voted for it. Voting would require some currency, that would be locked for the duration of the vote in order to avoid double-voting. Once a group gets added, an event is published on the smart contract, and Ingestors should start publishing on the new topic if they match the group criteria. Votes could be done by locking up Ethereum or a new token specific to this platform. This choice depends on the economy aspects that could be defined in the future.

7.2 Keeping track of all published data

Consider a scenario where a Sensor publishes data points every 5 minutes. That data is kept for one week by the generator's Node. Two other nodes keep one data point per hour for one month, and a last one keeps one data point per day for a year.

After one week, $1/12^{th}$ of the data is gone, and never to be seen again. How does a new Node interested in the data of that Sensor know if it was able to fetch all the data currently available on the network? How to know if it's not temporary, or when the requester should stop asking for that data?

Making sure that one owns *all* the data would be straightforward if Data Points were published as a chain, where each point has a pointer to the previous one. However, this does no longer work when Nodes arbitrarily prune data, which this network heavily relies on.

The easy solution would be to only synchronize data from Nodes that have the entire chain. But this goes against the principles of the platform. It would also prohibit the retrieval of any data if all nodes prune it in some way or another, despite having some data points available. This is thus a no go.

Another solution would be to send a list of missing Data Points along with the requested ones. However, this does not scale very well, and to the author's knowledge, there is no way to prove that these points actually existed at some point.

Alternatively, the network could define specific fixed retention policies. For example, at the end of each day, week or month, Sensors could publish packets that contain the average measurements for each fixed period of time. But this solution is too rigid. What if one is interested in the maxima or minima? What if someone wants bi-weekly data points?

The proposed protocol (5) was designed to be a best-effort protocol. It just cannot guarantee if all data points are available or not. If Sensors send data every 5 minutes, but aggregators only keep one data point per day, at some point the granular data will disappear. When that happens, there is no way to retrieve deleted data points, and no way of knowing how many are missing. It this case, it does not make sense to publish Data Points with a pointer to the previous one, as context will be lost at some point either way.

7.3 Querying for specific data

The proposed protocol for requesting data only allows for a ranged time query. This is problematic if a user is only interested in the maximum temperature recorded each week for the last year. This is currently not possible, or at least it would require the retrieval of each Data Point during said period to do the calculation locally.

A device that collects data every 5 minutes generates 105,120 Data Points per year. Querying one year worth of weekly averages of such a device would thus require the retrieval of all that data, after which the averages can be calculated. This is extremely inefficient, as 52 Data Points should have sufficed.

A very basic solution would be to implement a fixed query language that is a bit more advanced than the current `Request` one. However, this would be complicated, as the result would not be the same depending on what node answered. The signature and validity check of original Data Points would also be lost, resulting in a security issue.

7.4 Security

The proposed protocol does not address security aspects of the `libp2p` network. For example, the network should be cautious with DoS attacks, as they can get amplified a lot over PubSub.

If one Sensor publishes loads of data points, each individual Node can blacklist it, unsubscribe from the related PubSub topics, and call it a day.

If an attacker simulates lots of different Sensors, it becomes a bit harder to defend from this kind of flooding behavior, especially if the publication of Data Points happens across many different Nodes. One potential solution could be to force users who want to add a Sensor to the network to pay some amount to register the device. This is already kind of the case with the Smart Contract, although it does not actually enforce a specific cost. The fees required to register an Ingestor are currently only dependent on Ethereum's gas price, and could be as cheap as they could be expensive depending on that single criteria.

With a fee in place, even if the registration only costs 5€, spinning up a large scale attack would quickly cost a lot.

7.5 Benchmarking

As explained in the Takeaways section (6.6) of the Proof of concept chapter (6), there is a need for further benchmarking and performance testing. This comes from the fact that this work aims at a platform that operates at global scale. As it is also decentralized and open, it means that there would be a wide variety of hardware, networking and software configurations. Nodes would have different latency patterns, which could make the entire platform slower than expected. This is quite complex to test locally, as simulating an accurate environment heavily depends on usage patterns.

As this platform does not require consensus or any global synchronization, it should be able to operate as efficiently as possible when hardware and networking are provisioned correctly. However, the author was able to experience firsthand that this is rarely the case on a variety of different projects, including Swarm. Globally, the network latency and capacity is likely the biggest issue.

In order to test the performance, a realistic network is required, which can only be achieved by having actual people test it out at large scale on a public testnet.

7.6 Economy

An economical aspect is interesting to think about for any project. For example, this one could profit from a market or some type of incentive system that rewards people who buy IoT devices and make their data available to the broader audience. This would make the platform more robust and complete by motivating more people to deploy devices.

Different systems can be imagined. The most straightforward one would be to rely on donations. As the platform uses a smart contract where devices are registered, it could also be used to facilitate payments or donations to device owners. This can already be achieved in the current state of the platform, as device owners are specified in the contract, and are in fact Ethereum wallets. It is thus possible to send cryptocurrencies directly to those addresses.

This system can become arbitrarily complex, by forcing people who need access to large quantities of data to pay for it beforehand. However, this type of economy does not fit too well with the values of this project, which aims at being as open as possible, for both the technical aspects and the data. External markets where companies sponsor devices in specific locations could also emerge naturally.

7.7 Legality

This work only studies the technical aspects of the platform. However, as censorship-resistance is part of the argument, it would make sense to study the legal aspects as well. For example, the fact that some data could end up being stored in a different country than the one it originated from could be problematic, but is a requirement for this platform. Laws like GDPR also need to be considered. However, this is not a subject the author can comment on, and would need to be done by an external entity.

Conclusion

This work started by defining precise and realistic objectives of the targeted IoT platform: it had to be decentralized, cheap, community-driven, easy to use, open and fast. Some nice-to-haves, like being accurate and having an economy for more sustained growth, were also investigated. A platform that collects air quality data from all over the world was used as an example.

Then, existing projects with similar characteristics were studied in order to gain better insights on working principles. Helium was particularly interesting for its community-driven aspects while also leveraging blockchain. IPFS was also inspiring, as it provides a fully decentralized P2P network for data storage, while also being entirely free to use. Similarly, Swarm offered decentralized storage with built-in economy and redundancy.

In the third chapter, the characteristics of blockchain were explored, as this technology offered interesting assets, such as its decentralized and programmable aspects. The immutability aspect was also explained, as well as different consensus algorithms. The limitations of popular blockchains were highlighted, mostly in relation to their low transactions per second, and indirectly the cost related to that limit. The conclusion was that blockchain is decentralized, community-driven and open. It can also be used to make dApps that are easy to use for the end user.

The next chapter put the conclusions of the previous one in perspective with the goal of this work, and demonstrated that blockchain is neither cheap nor fast. This was shown with a concrete example of an implementation of a basic IoT platform built on top of Ethereum. Finally, potential solutions to make blockchain more suitable were explored. These included pruning, ephemeral blockchains and sharding. However, because of its intrinsic performance issues, blockchain technology was not retained for this work.

In the fifth chapter, a protocol for the targeted platform was proposed. A scope was defined and an architecture containing four different elements was established. Then, all those components (sensors, ingestors, nodes and a smart contract) and schemas were specified. Finally, protocols for registering devices, publishing data points, requesting data and synchronizing devices were specified. All of those rely on `libp2p` specifications. The resulting platform matched the features of the blockchain technology, with the addition of being cheaper and faster.

The proposed protocol was then implemented in NodeJS and Solidity, and deployed on the Görli testnet. The developed smart contract implements a quadtree for efficient 2D geographical lookups. The platform contains a web interface that allows users to easily register devices, to locate them on a map, as well as to subscribe to them for live data updates. A private API was also developed in order to interact with the nodes that implement the proposed protocols.

Finally, in order to make the platform more scalable and production ready, protocol extensions were discussed. One extension dealt with bulk retrieval of data, which was missing from the main protocol. A more advanced querying strategy was also discussed, as well as a way to improve the lifespan of data. Finally, security, legal and economical aspects were considered.

In conclusion, a peer-to-peer decentralized platform was built to accommodate community-driven projects. It has all desired features: cheap, easy to use, open and fast. However, some adjustments had to be considered, resulting in a more rigid platform (e.g. the query language limitation). As perspectives, further testing and benchmarking in a realistic testnet would be necessary in order to correctly evaluate the properties of this platform, such as overall speed and efficiency.

Appendices

Appendix A

Blockchain

A.1 Ethereum contract

```
// SPDX-License-Identifier: UNLICENSED

pragma solidity ^0.8.0;

contract Platform {
  // Configuration
  uint256 public immutable LOCATION_MULTIPLIER;
  uint256 public immutable DATA_PRECISION;

  // 6 decimal points precision
  struct Coordinates {
    int32 latitude;
    int32 longitude;
  }

  struct Device {
    address owner;
    address id;
    Coordinates coordinates;
  }

  // Temperature and humidity are precise to one decimal point
  struct DataPoint {
    address id;
    uint256 timestamp;
    int16 temperature; // From -3276.8°C to 3276.7°C
    uint16 humidity; // From 0 to 100%
  }

  event NewDevice(address owner, address id);
event NewDataPoint(DataPoint data);

Device[] devices;
mapping(address => Device) deviceMap;
mapping(address => DataPoint[]) dataPoints;

constructor(uint256 locationMultiplier, uint256 dataPrecision) {
  LOCATION_MULTIPLIER = locationMultiplier;
  DATA_PRECISION = dataPrecision;
}

function register(address id, Coordinates memory coordinates) public {
  require(
    deviceMap[msg.sender].owner == address(0),
    "This device is already registered"
  );

  Device memory device =
    Device({
      owner: msg.sender,
      id: id,
      coordinates: coordinates
    });

  devices.push(device);
  deviceMap[id] = device;

  emit NewDevice(msg.sender, id);
}

function publish(int16 temperature, uint16 humidity) public {
  require(
    deviceMap[msg.sender].owner != address(0),
    "This device is not registered"
  );

  DataPoint memory dataPoint =
    DataPoint({
      id: msg.sender,
      timestamp: block.timestamp,
      temperature: temperature,
      humidity: humidity
    });

  dataPoints[msg.sender].push(dataPoint);
  emit NewDataPoint(dataPoint);
}

function fetchData(
    address id,
    uint256 from,
    uint256 to,
    uint16 limit,
    uint256 start
  ) public view returns (DataPoint[] memory result, uint256 lastIndex) {
    result = new DataPoint[](limit);
    lastIndex = start;

    DataPoint[] memory points = dataPoints[id];
    uint16 j = 0;

    // This is extremely naïve and should be replaced with
    // something more efficient like binary search.
    for (; lastIndex < points.length && j < limit; lastIndex++) {
      DataPoint memory point = points[lastIndex];
      if (point.timestamp >= from && point.timestamp < to) {
        result[j++] = points[lastIndex];
      }
    }

    return (result, lastIndex);
  }

  // Call with "from" on upper left and "to" on bottom right
  function locateDevices(
    Coordinates memory from,
    Coordinates memory to,
    uint16 limit,
    uint256 start
  ) public view returns (Device[] memory result, uint256 lastIndex) {
    result = new Device[](limit);
    lastIndex = start;

    uint16 j = 0;

    // This is extremely naïve too, and should be
    // replaced with something more efficient.
    // H3?
    for (; lastIndex < devices.length && j < limit; lastIndex++) {
      Device memory device = devices[lastIndex];

      if (
        from.latitude <= device.coordinates.latitude &&
        from.longitude <= device.coordinates.longitude &&
        to.latitude >= device.coordinates.latitude &&
        to.longitude >= device.coordinates.longitude
      ) {
        result[j++] = device;
      }
    }

    return (result, lastIndex);
  }
}
```

Bibliography

[1] blockchain | Helium documentation. https://docs.helium.com/blockchain. (Accessed on 05/18/2021).

[2] Blockchain explained - intro - beginners guide to blockchain. https://blockchainhub.net/blockchain-intro/. (Accessed on 06/08/2021).

[3] community-governance | Helium documentation. https://docs.helium.com/community-governance. (Accessed on 05/18/2021).

[4] Connect to a private back-end | The Things Network. https://www.thethingsnetwork.org/docs/gateways/multitech/connect-private.html. (Accessed on 01/25/2021).

[5] Documentation/technicalwhitepaper.md at master · EOSIO/documentation. https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md. (Accessed on 05/18/2021).

[6] EOS network monitor - by CryptoLions. https://eosnetworkmonitor.io/. (Accessed on 05/18/2021).

[7] Ephemeral µblockchains: the mayflies tackling data storage limitations on the blockchain | Hacker Noon. https://hackernoon.com/ephemeral-%C2%B5blockchains-the-mayflies-tackling-data-storage-limitations-on-the-blockchain-f38bfdad51b. (Accessed on 05/27/2021).

[8] Ethereum 2.0 is go: Genesis block of beacon chain winks into existence. https://cointelegraph.com/news/ethereum-2-0-is-go-genesis-block-of-beacon-chain-winks-into-existence. (Accessed on 05/18/2021).

[9] Ethereum becoming more than crypto coder darling, Grayscale says - Bloomberg. https://www.bloomberg.com/news/articles/2020-12-04/ethereum-becoming-more-than-crypto-coder-darling-grayscale-says. (Accessed on 05/18/2021).

[10] Ethereum glossary | ethereum.org. https://ethereum.org/en/glossary/#gwei. (Accessed on 06/08/2021).

[11] Ethereum in 2020: Vitalik telegraphs high hopes in Denver | Fortune. https://fortune.com/2020/02/19/ethereum-bitcoin-closest-rival-cryptocurrency/. (Accessed on 05/18/2021).

[12] Ethereum races clock to collect enough coins for big upgrade - Bloomberg. https://www.bloomberg.com/news/articles/2020-11-23/ethereum-races-clock-to-collect-enough-coins-for-huge-upgrade. (Accessed on 05/18/2021).

[13] Ethereum whitepaper | ethereum.org. https://ethereum.org/en/whitepaper/. (Accessed on 05/16/2021).

[14] ethereum/eth2.0-specs: Ethereum 2.0 specifications. https://github.com/ethereum/eth2.0-specs. (Accessed on 05/18/2021).

[15] Global number of connected IoT devices 2015-2025 | Statista. https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/. (Accessed on 01/05/2021).

[16] Helium – introducing the people's network. https://www.helium.com/. (Accessed on 05/18/2021).

[17] Helium Console. https://www.helium.com/console. (Accessed on 05/18/2021).

[18] helium/erlang-libp2p: An erlang implementation of libp2p swarms. https://github.com/helium/erlang-libp2p. (Accessed on 05/26/2021).

[19] How website performance affects conversion rates | Cloudflare. https://www.cloudflare.com/learning/performance/more/website-performance-conversion-rates/. (Accessed on 05/24/2021).

[20] Integrations - Marketplace – The Things Network. https://www.thethingsnetwork.org/marketplace/products/integrations. (Accessed on 05/18/2021).

[21] InterPlanetary Linked Data | IPLD documentation. https://docs.ipld.io/#what-is-ipld. (Accessed on 02/10/2021).

[22] IPFS powers the distributed web. https://ipfs.io/. (Accessed on 02/10/2021).

[23] libp2p/readme.md at 1f03ef60cfcd96ddf20c59960a8189f86aabd0b1 · libp2p/libp2p. https://github.com/libp2p/libp2p/blob/1f03ef60cfcd96ddf20c59960a8189f86aabd0b1/README.md. (Accessed on 05/18/2021).

[24] The limits to blockchain scalability. https://vitalik.ca/general/2021/05/23/scaling.html. (Accessed on 05/26/2021).

[25] orbitdb/ipfs-log: Append-only log CRDT on IPFS. https://github.com/orbitdb/ipfs-log. (Accessed on 05/18/2021).

[26] orbitdb/orbit-db: Peer-to-peer databases for the decentralized web. https://github.com/orbitdb/orbit-db. (Accessed on 02/10/2021).

[27] Pin files | IPFS docs. https://docs.ipfs.io/how-to/pin-files/. (Accessed on 05/18/2021).

[28] proof-of-coverage | Helium documentation. https://docs.helium.com/blockchain/proof-of-coverage. (Accessed on 05/18/2021).

[29] Release v0.6.0 · ethersphere/bee. https://github.com/ethersphere/bee/releases/tag/v0.6.0. (Accessed on 05/27/2021).

[30] solidity/index.rst at v0.8.4 · ethereum/solidity. https://github.com/ethereum/solidity/blob/v0.8.4/docs/index.rst. (Accessed on 05/18/2021).

[31] Swarm. https://swarm.ethereum.org/. (Accessed on 05/26/2021).

[32] swarm-docs/introduction.rst at master · ethersphere/swarm-docs. https://github.com/ethersphere/swarm-docs/blob/master/contents/introduction.rst. (Accessed on 05/26/2021).

[33] Swarm secures funds for mainnet release | by Ethereum Swarm | Ethereum Swarm | Medium. https://medium.com/ethereum-swarm/swarm-secures-funds-for-mainnet-release-2992d453fa09. (Accessed on 05/27/2021).

[34] The Things Network. https://www.thethingsnetwork.org/. (Accessed on 05/18/2021).

[35] transaction-fees | Helium documentation. https://docs.helium.com/blockchain/transaction-fees/. (Accessed on 05/18/2021).

[36] vitalik.eth on Twitter: "ETH2 scaling for data will be available *before* ETH2 scaling for general computation. this implies that rollups will be the dominant scaling paradigm for at least a couple of years: first ~2-3k TPS with eth1 as data layer, then ~100k TPS with eth2 (phase 1). adjust accordingly." / twitter. https://twitter.com/VitalikButerin/status/1277961594958471168. (Accessed on 05/18/2021).

[37] WD Red Pro NAS hard drive | Western Digital Store. https://shop.westerndigital.com/en-ie/products/internal-drives/wd-red-pro-sata-hdd#WD161KFGX. (Accessed on 05/26/2021).

[38] What is LoRaWAN® specification - LoRa Alliance®. https://lora-alliance.org/about-lorawan/. (Accessed on 05/18/2021).

[39] Worldwide mobile data pricing 2021 | 1GB data cost in 230 countries - Cable.co.uk. https://www.cable.co.uk/mobiles/worldwide-data-pricing/#pricing. (Accessed on 06/08/2021).

[40] contributors. Quadtree — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Quadtree&oldid=1021622388, 2021. (Accessed on 05/16/2021).

[41] Bitcoin Wiki contributors. Scalability - bitcoin wiki. https://en.bitcoin.it/w/index.php?title=Scalability&oldid=68329, 2020. (Accessed on 05/18/2021).

[42] Bitcoin Wiki contributors. Genesis block - bitcoin wiki. https://en.bitcoin.it/w/index.php?title=Genesis_block&oldid=68529, 2021. (Accessed on 05/18/2021).

[43] Amir Haleem, Andrew Allen, Andrew Thompson, Marc Nijdam, and Rahul Garg. Helium: A decentralized wireless network. http://whitepaper.helium.com/. (Accessed on 05/18/2021).

[44] Satoshi Nakamoto. bitcoin.pdf. https://bitcoin.org/bitcoin.pdf. (Accessed on 05/18/2021).

[45] Dimitris Vyzovitis, Yusef Napora, Dirk McCormick, David Dias, and Yiannis Psaras. GossipSub: Attack-resilient message propagation in the Filecoin and ETH2.0 networks, 2020.

[46] Wikipedia contributors. Blockchain — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Blockchain&oldid=1023702717, 2021. (Accessed on 05/18/2021).

[47] Wikipedia contributors. Conflict-free replicated data type — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Conflict-free_replicated_data_type&oldid=1014197458, 2021. (Accessed on 05/18/2021).

[48] Wikipedia contributors. Solidity — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Solidity&oldid=1022515369, 2021. [Online; accessed 18-May-2021].