Semantic Web of Things on Blockchain: a decentralized semantic IoT data platform

Piyasa Basak Navya Basavaraju Lodovico Giaretta Kosta Ivancevic

***Abstract*—Blockchains, distributed filesystems, semantic data and IoT are relatively young technologies that grew up mostly independently from each other. Although there have been efforts to bring them together, there are still a lot of opportunities open. In this project, we show how to integrate these technologies to develop a scalable, decentralized semantic IoT data platform, able to connect producers to consumers in an efficient way and manage the entire lifecycle of the data.**

***Index Terms*—internet of things, blockchain, semantic web, distributed filesystem, data platform**

I. Introduction

In recent years, blockchains and IoT have been the topics on the hype. These two technologies are en-route to revolutionize and reshape the Internet.

The former employs a distributed, decentralized consensus to global order and validate any kind of transaction. It started out as an alternative to centralized payment systems, but it proved flexible enough to be used in many different fields, from identity management to supply chain traceability. This technology is marking the shift from centralized, closed databases to decentralized, open and trusted ledgers.

The latter, IoT, is a paradigm shift, where the cheap price and low power requirements of connected micro-controllers have led to their widespread installation, to monitor any kind of system, from street intersections to ship containers. These devices, in turn, produce a huge amount of data, which can only be exploited and thus rendered valuable when a distributed, scalable infrastructure is used to gather, process, aggregate, store and report these data across system boundaries.

The semantic web and distributed filesystems are two other technologies, somewhat older than the previous ones, that are going to play a significant role in the ongoing technology shift that blockchain and IoT are causing.

The semantic web is a paradigm where all available data on the Internet is encoded in a machine-readable format, enriched with taxonomies, ontologies and reasoning rules that allow computers to relate different pieces of data, answer elaborate questions, infer additional information and draw conclusions automatically. Despite the vast research on this topic and the creation of languages and tools to operate on semantic data, this paradigm has failed to reach widespread adoption. But the languages, tools and, most of all, general philosophy of the semantic web are finding new life in the world of IoT and blockchains, where the huge amounts of information, generated in a decentralized way by many independent actors, spanning many different domains, calls for advanced automation techniques.

IoT imposes very high requirements on our storage architectures: high throughput, high availability, low latency, fault tolerance, worldwide deployment, infinite scalability. This needs marking the flourishing of distributed, decentralized filesystems, that solve them by providing a transparent and efficient way to globally share any amount of data without any central coordination. They do this by drawing on well-tested technologies like content-addressing and peer-to-peer sharing.

These technologies are the building blocks of the Semantic Web of Things on Blockchain.

A. Concept and Design

Given the topic, we decided to focus our project on prototyping a scalable, decentralized semantic IoT data platform that could serve as the aggregation point to build a truly global Semantic Web of Things.

The platform should allow the seamless integration of any data producer with any data consumer through existing standardized technologies, allowing uniform queries on all the data available, so that consumers can get all and only the data they are interested in, no matter its origin, as shown in fig. 1. It should be completely decentralized and it should be economically sustainable, i.e. it should allow data monetization and provide incentives for the users offering computing resources to the platform.

We used the technologies described in this introduction as the main building blocks of our platform.

B. Structure of this Report

This section introduced our project scope and tech environment that inspired it. Section II describes some works similar or related to this project, highlighting their differences and limitations. Section III discusses the overall architecture of our platform and then moves on to detailed descriptions of its core components. Section IV lastly contains discussions regarding our organization, what we learned, the challenges and the future works related to this project.

II. Related Work

A lot of work has been done in the Semantic Web of Things field and in related topics. Unfortunately, joining together all

Piyasa Basak, *386178*, piyasa.basak@campus.tu-berlin.de
Navya Basavaraju, *386099*, navya.basavaraju@campus.tu-berlin.de
Lodovico Giaretta, *396563*, giaretta@campus.tu-berlin.de
Kosta Ivancevic, *397172*, kosta.ivancevic@campus.tu-berlin.de

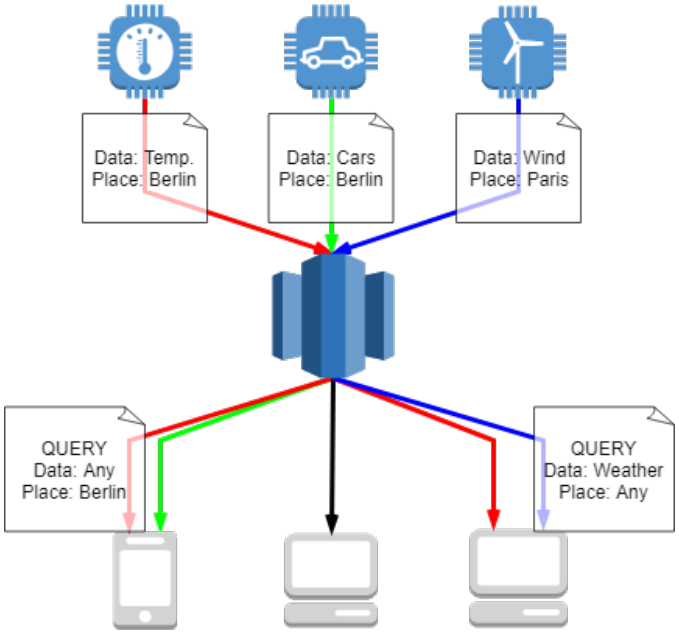

Fig. 1. Seamless integration of any producer with any consumer, as envisioned in our platform. Note how the central platform delivers the right data streams to the two consumers based on a semantic match between their query and the meta-information associated with each data stream. Please note that despite what this picture might suggest, our platform is completely distributed and decentralized.

the components needed to satisfy the requirements of a truly complete, scalable, decentralized IoT semantic data platform is a hard task. Indeed, most of these works focus on a smaller set of requirements, providing only a partial solution.

An overview of all the technologies on which the Semantic Web of Things can be built is available in [1], which introduces all required components: semantic representation, queries, storage, identifiers and so on.

Most of the work published in this field focus on either building semantic infrastructures on top of blockchain technology (e.g. [1], [2]) or bundling semantic information in IoT data (e.g. [3]). The former provides a semantic platform but leave aside its usage with IoT data, while the latter provide semantically rich data but do not focus on how to distribute and discover them through a platform.

A more holistic approach has been taken in [4], where smart contracts deployed on a custom blockchain, enhanced with a semantic reasoning engine, are used for registering, discovering and fetching semantically annotated data. This system takes into consideration the entire workflow needed to publish and search semantic data in a decentralized fashion, including monetisation, but leaves the actual data storage and data transmission unspecified. Also, this system uses non-standard semantic frameworks.

Another holistic approach, completely different from the previous one, has been presented in [5]. Here, the system manages the entire process: data acquisition and annotation, storage, search, retrieval and even subscription to new data matching a search. The system uses well known, standardized frameworks for all components. Unfortunately, this system is not decentralized, consisting of either a single node or a closed cluster (thus providing scalability).

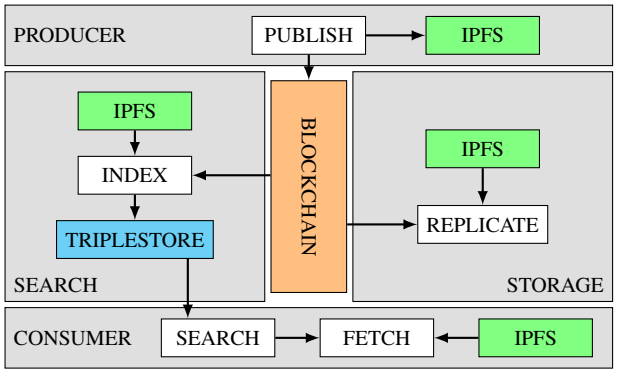

Fig. 2. Overall architecture of the platform. The gray boxes represent roles, the white ones operations and the colored ones are storage systems. Note that each node has its own IPFS agent, but all agents are part of the same global distributed filesystem, thus behaving like a unique entity.

III. Our Approach

A. Overall Architecture

The overall architecture of our platform is depicted in fig. 2. We classify the nodes participating in the platform in four roles, based on the subsystems they are running and the responsibilities they take:

- **producer nodes:** they publish new data and its semantic information to the platform via IPFS[1] and the blockchain; they also interact with smart contracts on the blockchain to achieve data replication;
- **storage nodes:** they interact with smart contracts on the blockchain to perform replication of the IPFS data; they are incentivized through crypto-currency rewards;
- **search nodes:** they monitor the blockchain to intercept the publishing of new data; they can then fetch from IPFS the semantic metadata associated with the newly-published data, index it into their triplestore and thus make it available to consumers through their semantic search interface;
- **consumer nodes:** they search the platform for data that matches their semantic requirements through SPARQL queries; when they find relevant data, they can fetch it from IPFS.

Note that a single machine might fulfil more roles at the same time. Note also that the diagram does not specify which nodes are blockchain miners, as the platform poses no requirement on this. Any role might mine, and mining could be even performed by nodes that take no other role.

In our tests, the blockchain and producer roles were performed by Raspberry PIs, while the search, consumer and storage roles by laptops.

B. Storage and Metadata Model

The core of our platform is its storage model. We developed it with the following requirements:

[1]InterPlanetary FileSystem – a distributed file system – https://ipfs.io/

1) **open to any type of data:** no assumption shall be made and no limitation imposed on the size or format of the data published;
2) **machine-readable semantic data:** the platform shall explicitly support a standardized, yet very flexible, way for machines to make sense out of new types of data, to be able to interact with the entire contents of the platform and not only with those published in a specific format;
3) **fast, lightweight data:** the fulfilment of the previous requirements shall not slow down or hinder in any way the most common scenario, i.e. the interaction with well-known data, where the format is known and semantic information is not needed;
4) **advanced semantic search:** it shall be possible to provide an advanced full-platform semantic search service without having to index the entire contents of the platform, but only relevant metadata; the aim of the semantic search shall not be to return the data entries matching a request, but rather pointers to the blocks that potentially contain those entries;
5) **flexible, multi-purpose metadata:** the platform shall provide a standardized way to express many kinds of meta-information associated with the published data, ranging from publisher details to licensing information;
6) **decentralized and fault tolerant:** information should be stored in such a way that the failure or malicious behaviour of a small subset of nodes causes as little harm and data unavailability/loss as possible;
7) **scalable and fast:** IoT data comes in huge amounts; the storage system must be able to handle it on a global scale.

To satisfy requirements 1, 6 and 7, we opted to use IPFS as our storage layer. IPFS is decentralized, distributed filesystem built on top of established protocols like BitTorrent and Merkle DAGs, and it can provide scalable, fast, decentralized, world-wide access to any kind of data out of the box. Unfortunately, it cannot truly provide fault tolerance. The reasons behind this issue and our solution to it are explained in section III-C.

In order to meet requirements 1 and 3, our storage model specifies that data can be published in any format, be it CSV, JSON, XML or any other kind, textual or binary. We do not require the data to be semantically annotated, as this would be against requirement 3, as semantic annotation increase the size of data and require more logic in the consumer, which is forced to handle a semantic format.

To handle requirement 2, we encourage (but do not force) to use well-known machine-readable data formats, of which dozens exists, starting from the most common, already cited, CSV, JSON and XML, to the more recent Apache Avro[2], Google ProtocolBuffers[3] and so on. As we already stated, we don't require data to be semantically annotated, but we strongly encourage to publish on IPFS, along with non-annotated data, a template file that consumers can use to make sense (i.e. to semantically annotate) the raw data. This template file itself can be written in a variety of languages, for example, XSLT[4] and Mustache. Each of these formats is more suited for a different raw data format (e.g. XSLT for XML, Mustache for JSON), but it is also possible to perform conversion of the raw data between its format and the format accepted by the template, as transformation rules between simple XML and JSON and from CSV to any of the others are well known. The advantage of using a format like XSLT over a format like Mustache is that the first is itself a structured, machine-readable language, while the latter performs simple string interpolation and is thus not machine-readable.

Consumers that do not have pre-existing knowledge of the data source can thus download, together with the raw data, this template file. They can then apply the template to the raw data to obtain semantically annotated data in a standard format, which might be XML/RDF, Turtle, Notation3 or any other semantic triples format. This allows the consumers to autonomously make sense of new kinds of data and relate them to each other. An advantage of this templating system is that an IoT data publisher might reuse the same template file for multiple data files, without having to duplicate the semantic boilerplate, which is mostly the same. In this way, storage, bandwidth and time are saved for both the producers and the consumers.

The problem that arises is where to store, and how to retrieve, the association between a data file and its template file. We envision this information as part of a bigger set of metadata that can be associated with a data file, as per requirement 5. This information is generated by the producer and published on IPFS as a sequence of semantic triples in any relevant format (XML/RDF, Turtle, ...). As stated, the meta-information might cover a wide range of aspects. It must satisfy a standard ontology defined by the platform, to provide a uniform access point to all data available, with detailed information presented in a domain-agnostic way, but it can also embed additional domain-specific information that might be useful to specific consumers. Some of the most important aspects covered by the standardized ontology shall be:

- **data and format:** the IPFS hash of the data file and the format of the data are the only mandatory information;
- **semantic template and format:** the IPFS hash of the template file and its format;
- **publisher details:** name, organization, link to a website or other further information, contact details;
- **general data info:** a general indication of what the data is about (e.g. weather, transportation, ...) and a brief human-readable description;
- **general search info:** time and location information expressed in a standard way, allowing simple domain-agnostic time and search queries;
- **domain-specific search info:** any additional fields that domain-aware consumers might want to use in their searches;
- **legal information:** what kind of license covers the data, whether a payment is needed for any use, only for commercial use or not at all, any additional restriction.

[2]https://avro.apache.org/ – retrieved 31.07.2018

[3]https://developers.google.com/protocol-buffers/ – retrieved 31.07.2018

[4]eXtensible Stylesheet Language Transformation

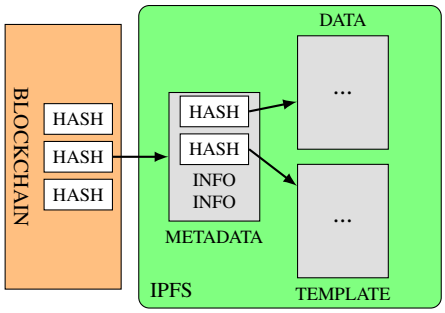

Fig. 3. Diagram of our storage model. All data files, templates and metadata files are stored on IPFS. The blockchain only stores small hashes that reference the IPFS contents. The data is decoupled from the template to allow its usage in lightweight applications that don't need the extra semantic information.

This meta-information poses the bases to satisfy requirement 4 and provide advanced search capabilities in the platform. In fact, all the information stored in the metadata file, and no other information, can be queried and used as a filter when using the search system as a data discovery mechanism. The publishing process enacted by the producer nodes includes the publishing of the IPFS hash of the metadata file to the Ethereum blockchain, where our search nodes are able to intercept it, fetch the metadata and make it available to the queries of the consumers. Detailed information on how this is achieved are exposed in section III-D.

C. Proof of Storage

A problem that arises when using IPFS to store data, regardless of what kind of data it is, is that it is stored only on the machine that published it unless it is being requested or has been explicitly stored on some other machine.

To be more precise, by default, the data is *pinned*, and thus stored indefinitely, only on the machine of the publisher. If any other machine requires the file, the IPFS agent on that machine will download it, serve it to the process that requested it, and cache it temporarily in its own store. Thus, consumers of a data file only provide temporary replication, unless he explicitly instructs its IPFS agent to *pin* the file.

An IPFS agent might also decide to request and cache a data file that is not actively requested by the machine, but that is being requested by many others, in order to have a stronger position in future exchanges and be able to obtain more quickly the files that are really needed. The details of this protocol are defined by the BitTorrent specification, on which IPFS is based [6].

While the mechanisms just described guarantee system *scalability*, as highly requested files are replicated in, and can be downloaded from, multiple locations, it does not guarantee *fault tolerance*, as less popular files end up being stored only by the producer, with the risk of unrecoverable data loss. This is a well-known issue of BitTorrent [7], but it is not acceptable for an IoT data platform.

For this reason, we decided to develop a smart-contract-based system that allows producers to offer ethers in order to have the contents of specific IPFS hashes duplicated on other machines. Other machines could then prove to the smart contract that they own a copy of the offered file in order to receive the offered ethers.

To achieve this, we needed a Proof-of-Storage algorithm with the following features:

- **non-interactive:** it should not require multiple steps, neither to issue the challenge nor to verify it; both actions should require a single transaction each;
- **minimal verifier storage:** the smart contract should only need to store a minimal amount of information from the issuer in order to verify the challenge solution;
- **minimal computational cost for verification:** the transaction to verify the challenge solution should not consume too much gas, otherwise claiming the reward would not be profitable;
- **verifier-issued challenges:** once the smart contract has received from the issuer the material needed to verify challenges, it should be able to, without any additional interaction with the user, generate different challenges for the same file, in order to guarantee that provers keep storing the file for the desired period of time.

There exists extensive research, with more of it ongoing right now, in the topic of PoS. Due to various factors, including the lack of cryptographic knowledge in the team, the lack of ready-to-use software libraries for PoS and the scarcity of time, we decided to implement our own PoS algorithm. This algorithm has not been audited by any means and can be considered neither secure nor production-ready. It is only suitable as a proof-of-concept for this project and as a starting point for future cryptographic efforts.

The algorithm can be configured with two parameters: a block size B and a hashing function H. It comprises three functions:

- `issue`: performed by the user offering ethers in exchange for replication, it processes the target file and produces a single hash needed for verification.
  The file is split in blocks of B bytes (except the last, which might be smaller) and each of the blocks is hashed using H. After that, pairs of subsequent hashes are concatenated and hashed again; if the number of hashes is odd, the last hash is hashed again, without being concatenated with anything. This process is repeated until a single hash is obtained, thus creating a sort of Binary Merkle Tree. The root of the tree is the output of this function.
- `prove`: performed by the machine that is replicating the file, it processes the target file and a challenge index j and produces a block of data and a sequence of hashes.
  The file is split in chunks as describe in the `issue` function. The block of data output by this function is simply the one with index j. The sequence of hashes is constructed as follows. The same Binary Merkle Tree described in the `issue` function is constructed. The first entry is the hash that in the tree is the uncle of block j. The second entry is the hash that in the tree is the uncle of the first entry, and so on. The process stops when a direct child of the root is added to the sequence.
- `verify`: performed by the smart contract, it verifies the

block of data and the sequence of hashes submitted by the prover against the single hash submitted by the issuer and a challenge index.
The smart contract simply hashes the data block, and then iteratively concatenates the current result with the next item in the sequence. When the sequence is exhausted, the root hash should be obtained. The verifier uses the challenge index to derive, for each concatenation, whether to concatenate on the right or on the left, thus allowing to differentiate solutions for the correct challenge index from solutions for the wrong challenge index (i.e. a prover trying to use an old challenge solution).

This algorithm satisfies all our requirements previously stated. Moreover, a prover may easily cache the top parts of the tree in order to avoid traversing all of it every time a new challenge is issued, but the only way to be able to solve any challenge without owning the file would be to cache the entire tree, which is bigger than the file itself, thus discouraging cheating.

We implemented the described algorithm through a command line application written in Javascript on NodeJS. It uses the Web3JS[5] library to interact with an ethereum RPC node and yargs to expose an easy to use command line interface, with option parsing and help texts. The smart contract is written using the Solidity language.

Each of the commands available in the application interacts with a function in the smart contract. The most important are:

- the `deploy` command compiles and deploys the smart contract to the blockchain;
- the `init-challenges` command generates a cryptographically-secure random seed and sends it to the smart contract, which uses it to generate new challenge indexes for all existing offers; through an option, it is possible to keep running this command in a loop with a custom interval; we suggest an interval of one day, but this can be tuned or even randomized according to the needs;
- the `offer create` command performs the issue function on a given file and uploads the root hash to the smart contract, along with other parameters like the amount of ethers offered, the block size and the replication factor;
- the `fund deposit` command transfers a specified amount of ethers from the address of the issuer to the smart contract, that will use them to pay for the storage offers created by the issuer;
- the `solve-challenge` command fetches the challenge index from the smart contract, performs the prove function on the given file and uploads the hash list to the smart contract; after verification by the contract, the address of the command issuer will receive the offered amount of ethers.

D. Semantic Search

The role of search nodes is to implement one of the key aspects of our platform: the ability to perform a semantic search over the entire collection of data files stored on it. This is key to giving each consumer the chance to interact with all data relevant to it.

[5]https://github.com/ethereum/web3.js/ – retrieved 31.07.2018

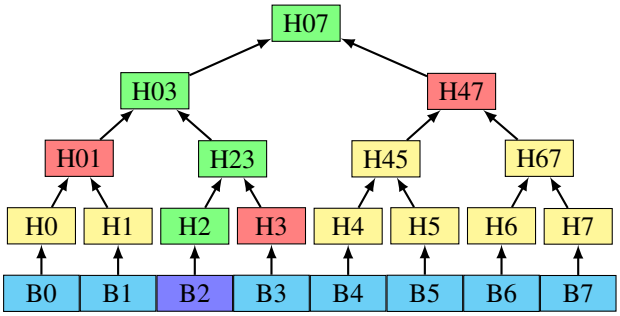

Fig. 4. Binary Merkle Tree used in the PoS algorithm. In this example the file is subdivided in eight chunks. The current challenge index is 2 and thus a prover needs to submit the sequence of dark blue data and red hashes: $[B2, H3, H01, H47]$, allowing then the verifier to compute the sequence of green hashes up to the root: $[H2, H23, H03, H07]$.

It must be stressed that this search system has no knowledge of the contents of any data file, so it is not possible to perform queries asking for files containing a specific entry, nor will the search system answer any query with actual data. Instead, the contents of queries and answers is limited to the metadata that publishers associate with their data files. So the fields in the queries and answers will match the metadata fields as briefly introduced in section III-B. Typically, the answer will contain at least the hash of the matched data file, allowing the consumer to then retrieve it from IPFS.

This limitation has been put in place to reduce the burden on search nodes, which would otherwise hold in their triplestores a bloated copy of all the data stored in IPFS via the platform. This would not be feasible, as search nodes cannot scale as well as IPFS. But it must be noted that this limitation is not a problem in practical scenarios, as the metadata ontology provides the time, location and subject information, which are sufficient for most queries, and allows for extra domain-specific information, which might be used for domain-specific filtering of data files. For example, the metadata file for a file containing a series of temperature measurements might declare the minimum and maximum temperature values that appear in the file. This, together with the standardized location information, would be enough to answer interesting questions like “Give me information about when Berlin had a temperature higher than 35°C”.

Figure 5 depicts the architecture of our prototype search node. The implementation consists of four main phases, detailed as follows:

- **Subscription to the Ethereum transactions:** in this step, the search node application contacts a blockchain node and subscribes to the transactions being published on the blockchain. Whenever a new transaction is published, the subscriber receives it, decides whether it is a valid metadata publish operation (correct target address and valid *additional data* field) and extracts the IPFS hash of the metadata (also known as *metahash*). When the search node is first started and the subscription created,

it immediately fetches all the pre-existing transactions, effectively catching-up its state with the platform.

- **Metadata fetch from IPFS:** this step is responsible for fetching the actual metadata file from the IPFS agent, using the metahash extracted in the previous step.
- **Metadata storage:** in our prototype, this is achieved by exploiting the transaction memory of the embedded SPARQL HTTP server. The metadata file fetched is parsed and stored in what is practically an in-memory triple-store, following the *subject-predicate-object* model. In a production deployment, this would be replaced by a full-fledged, standalone semantic database.
- **SPARQL query execution:** The SPARQL HTTP requests coming from the consumer nodes, which are POST requests holding a SPARQL query string in their bodies are directly executed by the server against its embedded in-memory triple-store. In a production environment, the server would unwrap the SPARQL query string and forward it to the standalone semantic database. The SPARQL query is typically expected to fetch the IPFS hash of the data block, which is then returned in the server response in JSON format, together with any other metadata requested by the query.

To quickly and easily build a robust prototype of the search node, we leveraged many existing libraries and frameworks. The search interface is exposed as a RESTful API using the Spring Boot framework. The subscription to the Ethereum node is performed via the Web3J[6] library (which is the Java counterpart of Web3JS, the Javascript library used to implement the command-line Proof-of-Storage tool in section III-C). The IPFS fetch is performed by issuing an IPFS command line call through the Java Standard Library. The embedded HTTP server Fuseki, part of the Apache Jena project[7], is used to execute and serve the SPARQL requests, while the data is kept in its internal in-memory triplestore.

An example of a SPARQL query sent to a search node is presented below. It fetches the hashes of all data files whose content is related to the city of Berlin, ordering the results by their timestamp.

```
SELECT ?hash WHERE {
    ?hash ns0:hasCity 'Berlin'^^ns0:city .
    ?hash ns0:hasTime ?timestamp .
}
ORDER BY ?timestamp
```

An HTTP request containing that same query, sent to the search node, might appear as the following:

```
POST /api/sparql HTTP/1.1
Content-Length: 124
Content-Type: application/sparql

{"query":"SELECT ?hash WHERE { ?hash
 ↪  ns0:hasCity 'Berlin'^^ns0:city . ?hash
 ↪  ns0:hasTime ?timestamp . } ORDER BY
 ↪  ?timestamp"}
```

The body of a successful response would contain a JSON similar to the one below, where a single hash is presented.

[6]https://web3j.io/ – retrieved 31.07.2018
[7]https://jena.apache.org/ – retrieved 31.07.2018

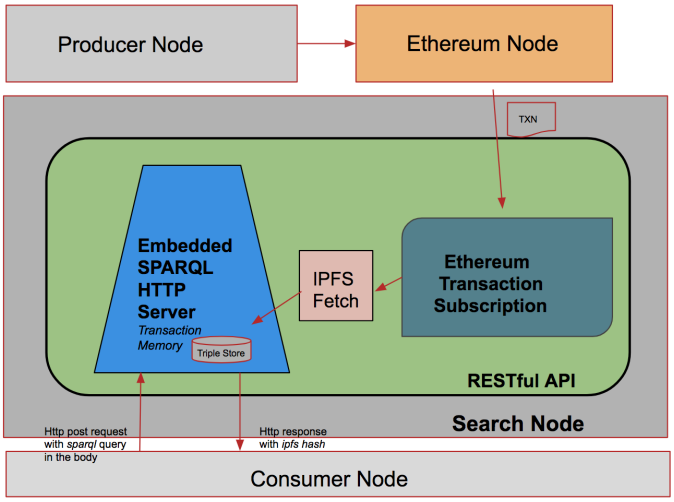

Fig. 5. Internal architecture of our prototype search node, and its interactions with other nodes

```
{
    "hash": {
        "type": "uri",
        "value": "https://ipfs.io/ipfs/
         ↪  QweZ112cvsbgtrz161bcavsuPnmN"
    }
}
```

E. Data Producers

In a typical IoT setup, the network of producers, that generate data through sensing, is implemented by low cost, low consumption and easy-to-maintain hardware platforms. Given the resemblance to that scenario, in our prototype, we implemented the data producers on a testbed consisting of multiple Raspberry Pi 3 model B Rev 1.2 instances. The underlying hardware architecture is the Broadcom BCM2835 System on Chip, equipped with ARMv7 quad-core processing unit, 1 GB RAM, Bluetooth and 4 USB2 ports, Ethernet, Wi-Fi and 40-pin GPIO[8]. The sensing is performed by the Sense HAT, an application-specific measurement add-on board that communicates with the system via its 40-pin GPIO. It fits a given application, as it is able to measure the following:

- Temperature
- Humidity
- Pressure
- Acceleration
- Orientation
- Magnetic Field

The chosen operating system is the Debian-based "Raspbian GNU/Linux 9 (Stretch)". At the given moment, it was the latest "Raspbian" release, having improved some shortcomings of previous releases (Jessie, Wheezy). The example node, consisting of the Raspberry Pi, equipped with the Sense HAT add-on board is shown in fig. 6.

The task of the producer node can be summarized by the steps below.

[8]General Purpose Input Output

Fig. 6. The single board computer Raspberry PI 3 model B, equipped with the Sense HAT add-on measurement board

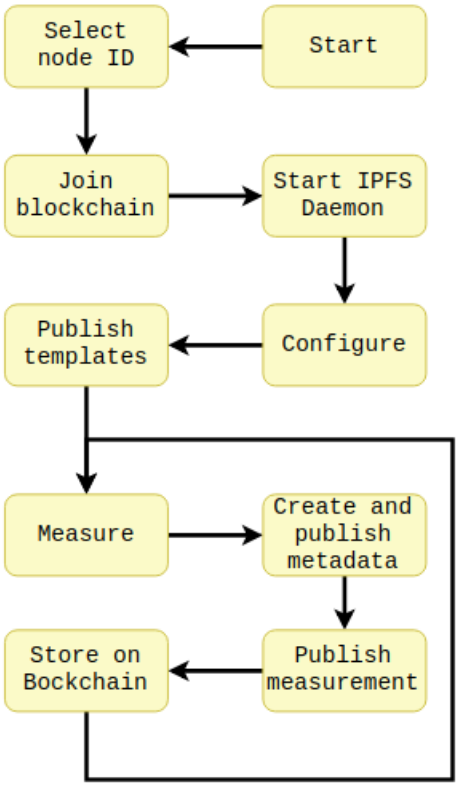

Fig. 7. Execution block diagram of a producer node

- **Perform and regulate a consistent real-time measurement process**, that is invoked with a certain time frequency. Before each node is started, the user has the possibility to configure the parameters that relate to the measurement process (specifying the measurement frequency and physical quantities, that are to be measured), geo-information (city, country, latitude, longitude, IP) and the information about the output data (address, usage permission, legal statement). These informations are used to fill the metadata file, as described in section III-B. After starting a node, measurements are created as CSV tabular files, both as single measurements and as a batch consisting of a set of successive measurements. With this double format, we aim to satisfy both the requirement of fine-granularity real-time data processing, as well as providing batch jobs the possibility to fetch a single file, that contains a full range of measurements made by a single sensor, instead of performing tens of requests to fetch the files holding each single data point. This is not only a sensible choice for real producers to satisfy their different consumers but also a useful way for us to test our system with different publishing patterns.
- **Publish all relevant files on IPFS**. After successful system initialization, each measurement collected as explained in the previous point is immediately published on IPFS, together with custom-created metadata files, that consist of the meta-information for each measurement. As detailed in section III-B the metadata, in a linked data manner, refers to the raw measurement files, as well as to XSLT template files, that are used to integrate the raw CSV information into a machine-readable RDF/XML[9] semantic file. The process of actual translation is done by the consumer in order to keep the amount of published data as low as possible. The result of this phase is an IPFS hash value that identifies the published metadata file, which holds all the relevant meta-information for the measurement.
- **Storing the IPFS hash of the metadata file on the blockchain**. Given its relatively small size, the IPFS hash of the metadata is used in the *additional data* parameter of each transaction, without affecting too much the overall gas cost. This implies that the transactions hold only platform information and no cryptocurrency value and as such, the receiving address is meaningless. In our prototype, they are addressed to the bootnode. However, this is only one of the design possibilities as in general any valid address can be used as the recipient for the transactions. The use of a single, standardized receiving address, whatever it is, is very useful, as it allows the search nodes to only subscribe to new transaction targeting that address, in order to monitor all newly published metadata.

For a given node, the above-mentioned steps are recursively iterated, which yields to a set of transactions, each having a unique IPFS hash for the metadata, of a single measurement. The content of the metadata points to other files (such as raw measurement and template that is used), as well as the measurement-specific information, for example, the geographical location of the sensor which made the measurement.

The set of execution actions of a producer node can be seen

[9]Resource Description Framework/eXtensible Markup Language

as a chain of successive actions, depicted on Figure 7. Initially, the user specifies the ID of the node from the set of pre-funded nodes. From the standpoint of a blockchain, this node now corresponds to that given account. Before the commencement of the measurement process, the node joins the blockchain via the bootnode. Furthermore, an IPFS daemon is started as a prerequisite for publishing data on IPFS. After both steps are successfully executed, the user is required to perform the initial configuration or to use a pre-existing one. Based on the configuration, the needed templates are published on IPFS once and for all (they never change, and multiple metadata files are allowed to point to the same template). Finally, the execution exhibits repetitive behaviour, which consists of performing measurements, followed by creating metadata, publishing both measurements and metadata and storing the hash of the metadata on IPFS.

F. Blockchain

A function of significant importance in our platform is the creation and maintenance of the blockchain network. For our platform, we chose to use Ethereum, thanks to its widespread adoption, which guarantees an easier learning curve, and its smart contact support, needed for our Proof-of-Storage component (see section III-C). This section will present an analysis of the architecture needed to perform this, and how we chose to implement it in our prototype.

We chose, in our implementation, to run the blockchain on our Raspberry Pi producer nodes. One drawback of using low-power hardware platforms like this is the inability to perform PoW[10] mining, due to low computational resources. This constrains our solution space to have a network that reaches a consensus through PoA[11], instead of the more traditional approach with PoW.

This is a choice that we performed in our prototype and that is not suitable for a public deployment. In a public context, either PoW or Proof-of-Stake should be used. We were able to perform this choice because it does not affect in any way the overall workflow or performance of our platform. The only noticeable difference is in the funding of the nodes, as explained in the next paragraph.

In Proof-of-Authority setup, mining rewards do not exist and transaction fees are usually kept very low (the minimum of 1 wei, in our prototype). The problem of acquiring the currency needed to perform transactions is solved by having a set of pre-funded accounts for the network, that can be created arbitrarily. With minor modifications to the genesis file, all nodes have sufficient amount of ether to perform their transactions. During the initialization phase, each node randomly picks an account from the set of pre-funded accounts and subsequently performs all transactions on behalf of it. Scalability is one of the most significant requirements. It is possible for multiple nodes to use the same set of credentials interchangeably. This ability is fundamental because the network is dynamic and it is expected that some of the nodes may be offline for some period of time.

[10]Proof of Work mechanism for reaching consensus.
[11]Proof of Authority mechanism for reaching consensus.

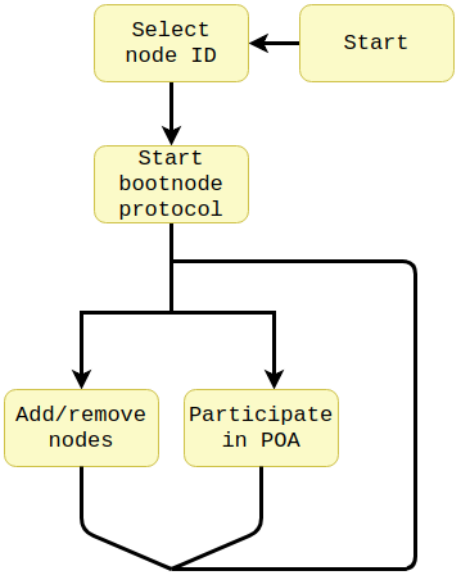

Fig. 8. Execution block diagram of a bootnode

The blockchain network is maintained by a specific node, called *bootnode*. The task of the bootnode is to start the network and to regulate the nodes that join or leave it. It is the only centralized subsystem in our entire platform, needed as a starting point to bootstrap the entire architecture. In our prototype, the bootnode protocol is started at one of the nodes that do not participate in the data collection and publishing (to avoid the extra burden on the Raspberry Pi that performs that task). This node, creates an *enode*[12] and shares it with its address and genesis file to all the nodes that may participate in the network at some point.

When a node joins the network, it must beforehand obtain the enode and address of the bootnode, as well as the same genesis file that is used by all other nodes in the private network. Only with that information, a node can successfully contact the bootnode and then join the network.

G. Monetization and incentives system

When designing our platform, we had to think not only about the technologies to use as building blocks but also at the economic dynamics that the structure of our system would generate, as these dynamics could lead to the flourishing of the platform or to its death.

The first question to answer is why users should dedicate their computing resources to our platform. For users acting as producers and consumers, the benefits are clear: the ability to exchange data on a huge worldwide open market. For storage nodes, the incentive is given by the rewards that producers offer to have their data replicated.

Regarding search nodes, we think that typically a single user would not run a search node alone. Rather, a few clusters of search nodes would operate in the platform. Each cluster would be independent of the others and internally operate as

[12]An *enode* is a unique identifier used to recognize the bootnode over the network.

a load-balanced replicated service. Groups of willing users might operate their clusters at their own expense, as a service to the community, given that the computational pressure (and thus the cost) on each single node would be low. Other groups might instead use advertisements to cover their costs or offer premium services under subscription fees. Big companies could enter the market of search services, operating their own private clusters, providing free search in exchange of advertisement and user profiling, in a way similar to current web search engines. These different options should guarantee a varied and efficient search ecosystem to all the users.

It was previously stated that blockchain mining could happen on any machine, and that, while our prototype uses Ethereum with PoA, any blockchain with support for smart contracts would equally work. In any case, the incentive to run blockchain nodes would be given by the transaction fees. It could be even envisioned to remove the transaction fees, as search nodes and storage nodes would still be incentivized to run their own local miners (may be shared with other nodes in the neighbourhood) in order to satisfy their need to continuously and efficiently interact with the blockchain, and especially to run their blockchain subscriptions, listening for new metadata publications and new replication offers respectively.

After settling why users would dedicate their resources to the platform, we can move to the actual monetization of the data by the publishers. Unfortunately, this is not an easy topic.

In a digital environment, protecting intellectual property becomes extremely hard. As soon as a consumer receives some data from a producer, there is almost no way to prevent him from freely sharing that data with other consumers, thus hurting the revenue of the producer. Furthermore, in an IPFS environment, data might silently reside even in machines of users that don't need it, thus rendering the philosophy "If it is in your machine, you have to pay for it" not applicable.

Our idea, which does not prevent all data theft, but provides a reasonable framework to identify profitable uses of data and verify that corresponding payments have been issued is described here.

The metadata information of each data block contains paying instructions (i.e. Ethereum address and additional transaction data) that consumers can use to pay for the data. The metadata also contains licensing information describing what uses are allowed free of charge and what uses require payment. For example, a piece of data might require payment for commercial usage, payment for any public usage (commercial, research and free public services) or payment for any usage (as before, but also for private, unpublished usage). For the latter licensing option, there is no effective way to verify or enforce it. For the first two options, on the other hand, verification is possible: as the data is used in a product that is exposed to others (a website, a software, a paper, ...), the license would require the consumer to insert in the product a reference to the transaction where the payment is performed. Any user of the product would then be able to verify that the product is using legally acquired data, by checking that the additional transaction data embedded in the payment state that that payment is being issued for the data that is actually being used and by the entity that is actually using it.

This could be used as legal information by the data producers to prosecute unlawful consumers, and by the general public to steer away from unlawful services.

IV. Evaluation and Discussion

A. Challenges and Limitations

After careful analysis and testing, we came to the conclusion that using hardware platforms with low computation capabilities introduces significant limitations in terms of the blockchain network.

In some scenarios, it can be difficult to predict the size of the private blockchain network before deployment, and the best solution is to estimate the expansion ratio of the network. Assuming that the network has overgrown its capacities in terms of the nodes, in order to further expand, the system needs to be stopped in order for the new nodes to be added to the genesis file. This will create a completely new network and all the nodes are forced to transfer from the old network to the new network.

The mining inability further reduces the performance with the usage of PoA implemented. By definition, the consensus is achieved when $50\% + 1$ nodes approve the transaction. Consider a scenario where, for some reason, a significant amount of nodes are offline (for example due to maintenance or network failure). If such situation occurs, all transactions in the network are stalled. Furthermore, assume that we aim in achieving a scalable network with the ability to expand with time. This implies that at the beginning, there is a set of inactive nodes, thus leading to a higher probability of being unable to allocate $50\% + 1$ active nodes for reaching the consensus.

With careful analysis of the aforementioned constraints, we conclude that they are in collision, thus making us unable to optimize for the dynamically expandable network that does not exhibit stalling due to the lack of consensus. If we aim to optimize the network for future expansion, we are increasing the probability of transaction stall. In contrast, if we aim to tightly bound the number of nodes so that the probability of a network not reaching consensus is reduced to a minimum, we are preventing network growth.

Both of the above-mentioned problems are easily solvable by using more powerful hardware and thus switching to PoW, but for the large-scale implementation, power consumption and price per unit surpass ones achieved by using Raspberry Pi or similar platforms. Another possible solution is to employ a different consensus model, e.g. Proof-of-Stake. A positive aspect of our architecture is that any blockchain platform and consensus algorithm can be plugged in with minimal changes, as long as support for smart contracts is provided.

B. Team and Project Organization

Our team, comprised of 4 members, was formed based on the common interest in the topic. As a team, we brainstormed on the topic itself and worked towards idea generation. The key member in the team, Lodovico Giaretta, helped the team to conclude on an idea to work on. After a successful first

discussion with the supervisors Dr Danh Le Phuoc and Qian Liu, we started with the project by creating the milestones viz., a) Mid Term b) Final.

Midterm Milestone: For the first half of the project, we divided the work into 4 parts so each one of the team members could self-assign one of the tasks and work on it. We created 4 tasks viz., i) IoT sensor simulation setup ii) Blockchain setup with automation iii) Platforms study for smart contract iv) Deep dive into RDF and semantic search.

Final Milestone: After finalising the platforms and libraries to use for each of the requirements in our prototype, we broke down the development following our architectural pattern viz., i) Producer Node ii) Boot Node iii) Storage Node and iv) Search Node.

After completing each one of the aforementioned Node implementations, we did the integration and tested end-to-end. We created the videos to showcase the end-to-end scenario. The team had a very good cooperation from the team member Kosta Ivancevic who was responsible for setup and always available with the setup for any team member who had dependency with his setup during development and integration.

The presentations for the Midterm and Final milestones were organized in such a way that each one of the team members would talk about the topic that he or she worked on.

We started with the documentation right from the beginning with *readme.md* on our Gitlab repository. For the final report, we divided the sections for each one of us to work. Lodovico Giaretta, being the team leader, made sure the content is reviewed and required correction/additions made.

Overall, it was a good teamwork with full coordination and well driven by the team leader.

C. Learning Outcomes

One of the goals of this course and project was to give us the opportunity to experiment and research on these new technologies, learning how they work and how they can be integrated together to build something unique.

We personally believe that this goal was achieved and surpassed. Not only we learned how blockchains, distributed filesystems and semantic data technologies work. We also learned how to think outside of the box to produce prototypes that, according to our research, have never been produced before.

We also gained valuable experience in teamwork and project organization and in the tools that can be used to perform these tasks, as detailed in the previous section.

D. Future Work

There is always room for improvement in a project, especially if it has been developed during a very short time frame. Indeed, due to the time constraints, we only managed to produce a simple working proof-of-concept prototype of our platform, and we plan to extend it in many aspects.

The most important and most challenging topic to tackle is the scalability of the search system. Our prototype supports multiple search nodes, but each of them currently has to maintain a complete in-memory triplestore to be able to handle the incoming queries. This provides good support for replication and fault tolerance, but not for scalability. To build a truly scalable solution, we need to make partitions of the triplestore and forward the queries to all partitions that might contain relevant triples. While the partitioning itself is easy, performing it in such a way that it is possible to easily and efficiently forward each query to a small set of partitions, without missing any relevant results, is a challenging task.

An approach that looks promising and that we will investigate is the use of Content Addressable Networks (CANs) [8]. In particular, given that we are storing triples, a 3-dimensional CAN, where each coordinate is the hash of a field of the triple, might be appropriate. Then a search node would have to perform custom parsing of the SPARQL query and produce an execution plan that minimizes the portion of the search space that must be queried. This is akin to "normal" relational databases, where the query engine needs to come up with an execution plan that minimizes the number of rows to visit.

We will investigate other approaches, like delegating the storage of triples and the execution of partitioned queries to an existing relational database, thus leveraging years of experience in the field. We might also come up with completely new ideas.

Another important task is to produce a formal OWL specification of our metadata ontology, to serve as the specification for writing tools, documentation for developers and validation reference for the search nodes, before indexing the metadata.

We will also invest time in writing additional sample producers and consumers, fetching data from available online sources like OpenWeatherMap[13] and the VBB LiveKarte[14]. This will allow us to test our platform in larger environments, to better test two aspects: the scalability of our system, i.e. its ability to handle larger amounts of data, and its flexibility, i.e. its ability to handle different kinds of data, while providing a smooth experience to producers and consumers.

Talking about scalability and flexibility, we would like to further investigate into the topic of real-time data and how we can evolve our platform to better support them. In fact, currently, our platform is perfect for uploading large batch data with low frequency and small almost-real-time data with high frequency (e.g. temperature readings or even pictures every 15-30 seconds), but it does not support out of the box the delivery of high-bandwidth continuous streams of data, e.g. from live cameras.

The last point, but not less important, is to further research into the topic of Proof-of-Storage, in order to enhance or replace the algorithm used in our prototype, which will probably prove not secure enough to withstand malicious attackers, as we are already investigating some potential weaknesses in it. To perform this task we might look for a collaboration with someone with a cryptography background, as the team lacks the advanced knowledge needed to develop effective cryptographic algorithms.

[13]https://openweathermap.org/api – retrieved 31.07.2018
[14]https://vbb.de/livekarte – retrieved 31.07.2018

V. Conclusion

In this project report, we showed how existing technologies, like distributed filesystems, blockchains and semantic web can be integrated to offer a novel solution to the challenges posed by the IoT revolution.

More precisely, we built a working prototype of a decentralized, scalable semantic IoT data platform, that significantly differs from the solutions offered in the literature that we analyzed.

Although our prototype is a long way from being production ready, as many components were implemented in a way that is not suitable for a wide public deployment, it serves as a proof-of-concept and proof-of-feasibility for our vision.

There is still a lot of work to be done before this prototype can be turned into a real platform, but the first results are very encouraging and we believe that our architecture, if not our implementation, will stand the challenges ahead of us and prove successful.

Acknowledgment

We would like to thank our project supervisors, Dr Danh Le Phuoc and Qian Liu, for their support and feedback. We would also like to thank the Open Distributed Systems group of TU Berlin and Fraunhofer FOKUS for organizing this project.

References

[1] H. Ugarte, “A more pragmatic Web 3.0: Linked Blockchain Data”, 2017.
[2] A. Third and J. Domingue, “Linked Data Indexing of Distributed Ledgers.” in Proceedings of the 26th International Conference on World Wide Web Companion, 2017.
[3] M. Al-Osta, B. Ahmed and G. Abdelouahed, “A Lightweight Semantic Web-based Approach for Data Annotation on IoT Gateways” in Procedia Computer Science, vol. 113, pp. 186–193, 2017.
[4] M. Ruta, F. Scioscia, S. Ieva, G. Capurso and E. Di Sciascio, “Semantic Blockchain to Improve Scalability in the Internet of Things”, in Open Journal of Internet Of Things, vol. 3, issue 1, pp. 46–61, 2017.
[5] D. Le-Phuoc, H. Nguyen Mau Quoc, H. Ngo Quoc, T. Tran Nhat and M. Hauswirth, “The Graph of Things: A step towards the Live Knowledge Graph of connected things” in Web Semantics: Science, Services and Agents on the World Wide Web, vols. 37-38, pp. 25–35, 2016.
[6] B. Cohen, “The BitTorrent Protocol Specification”, online at http://www.bittorrent.org/beps/bep_0003.html, created 2008, last modified 2017, retrieved 31.07.2018.
[7] S. Kaune, R. Cuevas Rumin, G. Tyson, A. Mauthe, C. Guerrero, R. Steinmetz, “Unraveling BitTorrent’s File Unavailability: Measurements, Analysis and Solution Exploration”, arXiv:0912.0625 [cs.NI], 2009.
[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker, “A scalable content-addressable network”, in Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM ’01), 2001.