Implementation of smart contracts for blockchain based IoT applications

Georgios Papadodimas, Georgios Palaiokrasas, Antonios Litke, Theodora Varvarigou
Electrical and Computer Engineering Department
National Technical University of Athens
Athens, Greece
george.papadodimas@gmail.com, geopal@mail.ntua.gr, litke@mail.ntua.gr, dora@telecom.ntua.gr

***Abstract*—An increasing number of people, organizations and corporations are expressing their interest in the decentralization technology of the blockchain. The creation of the blockchain marks the time when we start building distributed peer-to-peer networks consisting of non-trusting members that interact with each other without a trusted intermediary but in a verifiable manner. In this paper, we propose a decentralized application (DApp) based on blockchain technology for sharing Internet of Things (IoT) sensors' data, and demonstrate various challenges addressed during the development process. This application combines blockchain technology with IoT and operates through smart contracts that are executed on the Ethereum blockchain. More specifically the application is a platform for sharing (buying and selling) measurements of IoT weather sensors and operates on the Ethereum blockchain, acting as a marketplace for IoT sensor data. This application applies the Sensing-as-a-Service (S^2aaS) business model combined with blockchain.**

Keywords—blockchain, ethereum, smart contracts, internet of things, sensing as a service

I. Introduction

The IoT industry is continuously growing, leading to more IoT sensors being manufactured and placed. As the number of IoT sensors is growing, it is increasingly important to find ways to maximize the value of the data generated by those sensors. The future of the IoT depends heavily not only on the technological solutions that will affect security, costs and other issues, but also on the value that data will have for people too. One way to significantly increase the value of the data and the efficiency of the operations as well, whilst at the same time achieve economies of scale and reduced costs, is to share the available data [1], [2]. In this paper we will present a blockchain based decentralized Sensing-as-a-Service (S^2aaS) application [3]. The source code of all system's components is provided in [4], while an application demo video is available at [5].

S^2aaS is a relatively new business model, which brings new data monetization opportunities to operators by enabling them to easily sell sensor data. S^2aaS makes IoT sensor data more easily accessible to customers, because it allows them to simply access (buy) data from sensors that are already in operation, so they are not obliged to maintain and operate numerous sensors. S^2aaS is a promising candidate for an IOT-enabled business model pattern, the same way E-Commerce was a business model for the first wave of the Internet [6].

Blockchain was originally developed by a person, or group of people called Satoshi Nakamoto, as the accounting method for Bitcoin cryptocurrency [7]. The technology and the ideas behind blockchain evolved since then and we now realize that its potential goes beyond cryptocurrencies [8], such as IoT applications [9], applications for the government sector [10], [11], funding mechanisms [12] and many more. A milestone for the course of blockchain technology was the development of Ethereum project, offering new solutions by enabling smart contracts' implementation and execution. The Ethereum blockchain is a Turing complete platform for executing smart contracts [13], [14], and not just a ledger serving financial transactions. It is a suite of tools and protocols for the creation and operation of Decentralized Applications (DApps), "applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third-party interference". It also supports a contract-oriented, high-level, Turing-complete programming language [15], allowing anyone to write smart contracts and create DApps.

There are two types of accounts in Ethereum [16], Externally Owned Accounts (EOAs), which are controlled by private keys and Contract Accounts, which are controlled by their contract code and can only be "activated" by an EOA. Ethereum DApps rely on smart contracts for their operation, namely a code of arbitrary algorithmic complexity that is executed on what is called Ethereum Virtual Machine ("EVM"). Smart contracts in Ethereum are (mainly) written in the programming language Solidity [17].

The DApp proposed in this paper is based on smart contracts and leverages security characteristics of the blockchain technology. We chose to utilize the blockchain technology for our S^2aaS application, in order to let users transact on a P2P level using cryptocurrencies rather than transacting with a central authority using fiat currency, as would happen if a typical client-server architecture had been chosen. Ethereum blockchain platform was preferred for the development of this DApp because of the security and flexibility it offers, the completeness and maturity of the available development tools and the fact that it is the most popular blockchain platform supporting smart contracts. Additionally, it is supported by a vibrant community that helps it constantly improve and its applications are very easily accessible by potential users [18]. These characteristics provide to Ethereum, in the opinion of the authors, an advantage over other blockchain platforms, such as EOS, Cardano, Stellar, NEO, that support smart contracts.

The developed application is a platform for sharing IoT weather sensor data, a marketplace, easily accessible to everyone. The currency used in this marketplace is a custom token, developed for this purpose.

II. Related Work

A presentation of how blockchain technology could disrupt Sensing-as-a-Service (S^2aaS) concepts and IoT in general is given in [6]. Besides the introduction to the concept of S^2aaS, a description of the characteristics of Bitcoin that are relevant for S^2aaS applications is provided, such as

decentralization, openness, pseudonymous identification, low fees and friction, scriptability and cryptographic verifiability. Afterwards, the basic concept of the combination among Bitcoin and S^{2}aaS is presented, by connecting every sensor to the Bitcoin's blockchain and having a unique Bitcoin address. A simple transaction includes three parts, the requester, who has a Bitcoin address, the Bitcoin blockchain and the sensor which also has a Bitcoin address. The requester sends a payment to the Bitcoin address of the sensor through a Bitcoin transaction. When the sensor notices the receipt of the payment, it creates a transaction to the Bitcoin address of the requester, including its most current datum encrypted with the requester's public key. Then the requester decrypts the datum with his private key. However, in most cases (e.g. real-time data feed etc.) S^{2}aaS applications will probably involve the transfer of more than one datum, which makes the process more complex. This problem is merely solvable now, with the technology of smart contracts that run on blockchains.

An extension of this work is presented in [19] . While in the former paper the concept of how blockchain technology could disrupt Sensing-as-a-Service was mainly presented on a theoretical basis, in the current paper a working prototype is also presented. The implementation of this prototype consists of:

1) Sensor client. A websocket client which registers with a websocket API was implemented so that the client will be able to be informed of a payment, in order to send a Bitcoin transaction with the relevant data and store them in the blockchain.

2) Requester client. This module retrieves sensors from the sensor repository, then selects the desired sensor and performs a payment to the sensor's Bitcoin address.

3) Sensor repository. It is a database with a RESTful HTTP API and a web front end.

As stated in the paper, this concept has some limitations. Firstly, the purchased data is publicly available in the blockchain which would stimulate free riding. This could be addressed by encrypting the data with the requester's public key which will then be decrypted by the requester client using its private key. Secondly, in order to provide the measurement data immediately, a 0-confirmation payment procedure is followed and a requester may be able to exploit it by attempting to double spend. Additionally, there are scaling issues. The blockchain would be bloated with transactions, exchanged data would be stored forever on Bitcoin nodes and sensors would have a large number of tiny unspent transaction outputs. The limitations of the former study have been addressed in the application of our study thanks to different design choices and the implementation of smart contracts deployed on Ethereum blockchain platform. Since we used smart contracts, we were able to provide to the user the option to buy measurements of a sensor taken during a time interval of his choice (e.g. he can buy data from a week from last month), thus much more flexibility is given to the users. In order to provide the former option and to address the issues mentioned before, we decided that the owners would store their sensors' data and expose them through an API. In this way, the sensor is not in charge of keeping all historical data and run an Ethereum node (which is technically challenging and would greatly increase its cost).

The applicability of blockchain technology in the Internet of Things sector is examined in [9]. Scalability and high costs issues of the IoT sensor networks are mentioned and blockchain is suggested as a solution. Advantages that come from the connection of IoT sensors to the blockchain include among others, the access to a convenient (existing) billing layer, which paves the way for a marketplace of services between devices. However, there are also some limitations and issues to be resolved. One issue is that a blockchain solution will generally underperform, resulting in lower transaction processing throughput and higher latencies, compared to a centralized database solution. Another issue that may come up, concerns privacy on the blockchain, since blockchains by design expose every information to everyone. Because of the complete transparency in the blockchain, transactional privacy (i.e. confidentiality) is hard to attain. The selection of the miner set is also a big issue. Although a miner cannot fake a transaction or rewrite history, he could prevent a new valid transaction being added to the blockchain. There is also the issue of the limited legal enforceability of smart contracts, though it is expected to change, as legislators have already started working on that matter [20]. Finally, more issues concerning expected value of tokenized assets and the autonomy of the network are presented, while the authors state that the combination of blockchains and IoT can be pretty powerful and will most probably cause significant transformations, bringing about new business models.

The management of IoT devices using the Ethereum blockchain is examined in [21]. Ethereum was used to configure IoT devices, in order to address issues concerning the synchronization of all connected devices, the integrity and correctness of the data. The team used a smartphone and three Raspberry Pis as meters to keep track of electricity usage of devices. The smartphone is used by the user to set up the configuration (energy usage policy) and then through a smart contract the policy is sent to the Ethereum network. The meters are connected to the Ethereum blockchain, retrieving values of policy periodically from it and storing in it the electricity usage. Because of the security characteristics of the Ethereum we are sure that the Raspberry Pis will have correct and intact data. The meters constantly monitor the policy (which is stored in Ethereum with the policy contract) and compare it with the energy usage. If the energy usage overcomes a specified threshold, the meter switches the devices from normal mode to energy saving mode and stores that to Ethereum with the help of the meter contract. Once the system was deployed and connected to the blockchain and some measures were recorded, some weaknesses regarding Ethereum were found. The transaction time is around 12 seconds, which is not fast enough for real time applications. Since a light client is not supported on Ethereum, we need either to use a proxy or have a large storage to save entire blockchain and to the time of the writing of the paper, a progress was noticed regarding this limitation [22].

III. System Overview And Requirements

The purpose of this study is the creation of an Ethereum blockchain DApp enabling users to easily buy and sell IoT sensor data, by using a custom token as the payment currency. For the purposes of this study we have created and experimented with a token named – NTUA Token. The proposed application is a marketplace for IoT weather sensors, but is developed in such way, that with very few changes it could be transformed to an application for any other kind of IoT sensor, information could be found in section IV.A.2.

A. Components

The system consists of two discrete parts:

1) Two smart contracts: They are deployed and executed on the Ethereum blockchain. There are two smart contracts, with the names NtuaToken and Broker. NtuaToken smart contract creates the NTUA Token (National Technical University of Athens token), a custom token created specifiaclly for reasearch and experimentation purposes, in order to explore as many characteristics of the Ethereum as possible. The economic impact of this custom token was not considered, as economic analysis exceeds the scope of this study. Broker smart contract is in charge of keeping all registered sensors, carrying out and recording transactions concerning IoT sensor data.

2) A web application: It as an interface between the users and the blockchain, helping users interact with (send transactions to and read data of) the smart contracts and access data they have bought. It also "protects" users from misreading or mistyping info when sending a transaction e.g. mistype an ethereum address, which is a common mistake and often results in loss of ether *[23]*.

B. User categories

There are three user categories:

1) The owner of the smart contracts. The person who owns the private key of the Externally Owned Account (EOA) from which the smart contracts were deployed. He is able to change the price (in ether) of NTUA Token and control the ether balance of the NtuaToken Contract.

2) The operators/owners of IoT sensors. They possess the private key of an EOA. They register new sensors in the system and change characteristics of sensors they have already registered (e.g. change of the price per measurement of a sensor). They provide the measurements of the registered IoT sensors. They can exchange NTUA Tokens for ether and vice versa. They receive NTUA Tokens in exchange for the data they sell.

3) The buyers of the data generated by the registered IoT sensors. They can also exchange NTUA Tokens for Ethers and vice versa. They can buy IoT sensor data and have access to the data they have already bought.

C. Use case

The operator of an IoT sensor opens the website of our application. He registers a new IoT sensor, by sending to the Broker smart contract a transaction with the related information (e.g. type, geographical position) of the sensor, signed with the private key of his EOA. He is required to maintain a RESTful API, which will provide the measurements of every sensor he has registered upon request. The sensor is now saved in the blockchain and its measurements are available to the buyers.

The use case includes the following steps: i) the potential buyer opens the website of our application; ii) he browses the available sensors; iii) he finds a sensor whose data is interested in buying; iv) he selects the sensor; v) he selects the time interval for which he wants to buy the sensor's measurements; vi) he sends a transaction to the blockchain in order to buy the selected data; vii) if his EOA has sufficient NTUA Tokens, the transaction is completed, tokens are transferred to the sensor's operator account and the buyer can now see the data he bought. Otherwise (if his NTUA Token balance in not sufficient) the transaction fails. He can buy NTUA Tokens anytime by giving ether.

Followingly, the buyer has access to the measurements he previously bought. The operator has received NTUA Tokens in return for the data he sold. He can either keep the tokens or sell them back to the NtuaToken smart contract. The token's price in ether is configurable and could be set by the contract's owner. We could consider that for this application by default, the exchange rate is set to 1 NTUA token for 1 ether. Users can sell NTUA tokens back to the contract at a price not greater than the price of the token. NtuaToken smart contract has already received ether when the buyer bought NTUA Tokens. The owner of the smart contracts can now retrieve

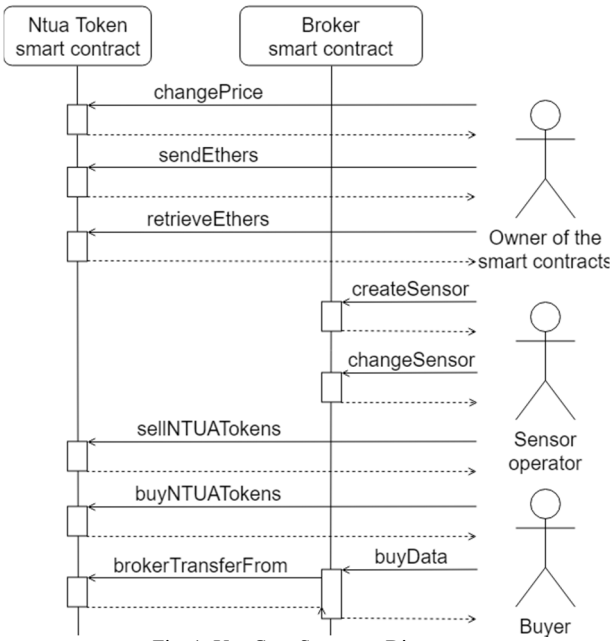

Fig. 1. Use Case Sequence Diagram

ethers from the NtuaToken contract's balance, or can send more ether to the contract. He can also change the price (in ether) of the token.

IV. Prototype Implementation

A. Smart Contracts

Once deployed to the blockchain, smart contracts' code is final and cannot be changed. So, during the development of the smart contracts extra caution was given in the testing process, so that we could identify and fix as many bugs as possible. For the development of the contracts in solidity web browser based IDE "Remix" was used. Extensive testing was carried out on private Ethereum blockchains, created with Ganache Cli, as well as on the Ropsten Test Net. The full code of the smart contracts can be found at [4]. Another important challenge we addressed, was the optimization of the code in order to minimize the gas requirement of each function, so that the fees someone pays to the blockchain, when calling a function of our smart contracts, are also minimized.

1) NtuaToken

NtuaToken smart contract creates the custom token NTUA (National Technical University of Athens), which is used by

the application as the payment currency. The total NTUA tokens supply is fixed and equal to 100,000,000 units, as ether, it has 18 decimal digits, and its symbol is NTUATok.

```
NtuaToken()
function () payable
changePrice(uint _price)
function retrieveEthereum(uint256 amount)
function sellBack(uint256 amount, uint
_price)
function _transfer(address _from, address
_to, uint _value) internal
function transfer(address _to, uint256
_value)
function brokerTransferFrom(address _from,
address _to, uint256 _value) returns (bool
success)
```

Fig. 2. Functions of NtuaToken contract

This contract implements the fallback function, an unnamed function marked as payable [17]. The purpose of this function is to load NTUA tokens to the balance of the accounts that send Ethers to the address of the smart contract. This way, one can easily buy NTUA tokens and then buy IoT sensor data. Exploiting the function *sellBack,* the user is able to sell NTUA tokens back to the contract and receive ether in return. If the owner of the contract sends ether to its address, the balance of the contract in ether simply increases and he will not receive NTUA tokens.

We have specifically implemented the functions *changePrice*, *retrieveEthereum* which can only be called from the owner of the contract, enabling him to change the price of a single NTUA token in ether and retrieve ether from the contract's balance. Finally, three functions that implement the functionality of transferring NTUA tokens between accounts are created: i) the internal function *_transfer*, which transfers the specified amount of NTUA tokens from one account to another, if the balance of the sender has sufficient funds. Upon success, it emits an event called *Transfer*, notifying those who watch the blockchain's activity that the transaction is completed; ii) the function *transfer* which can be called outside the contract. It calls *_transfer* function in order to send

```
require(msg.sender == 'address of Broker
contract in the blockchain')
```

Fig. 3. Require the sender of the message to be the address of Broker contract

tokens from the sender of the transaction to the specified receiver; iii) the *brokerTransferFrom* function. It is called from the Broker contract and transfers NTUA tokens, when someone buys IoT sensor data. It only succeeds, if it is invoked from the Broker contract: we a priori know the address on which a smart contract will be deployed on the blockchain, because it is deterministically computed from the address of its creator (sender) and the number of transactions the creator has sent (nonce) [24].

2) Broker

Broker smart contract keeps record of all registered IoT sensors, and gives the possibility for the owners of the sensor to change information regarding their sensors, carries out and records all transactions concerning IoT weather sensor data.

Two structs were created so that we can store all information of the IoT sensors and information of the transactions regarding IoT sensor data.

```
struct sensor {
   address seller ;
   uint8 sensorType ;
   uint price ;
   uint32 startTime ;
   uint16 frequency ;
   int32 latitude ;
   int32 longitude ;
   string url ;
}
```

```
struct transaction {
   uint sensorID ;
   uint32 fromTime ;
   uint32 toTime ;
   uint amount ;
}
```

Fig. 4. Structs of Broker contract

The functions of Broker smart contract are:

```
Broker()
function createSensor(address seller1, uint8
type1, uint price1, uint32 startTime1, uint16
frequency1, int32 latitude1, int32
longitude1, string url1) external
function changeSensorSeller(uint32 sensorID1,
address seller1) external
function changeSensorPrice(uint32 sensorID1,
uint price1) external
function changeSensorUrl(uint32 sensorID1,
string url1) external
function buyData(uint32 sensorID, uint32
fromTime, uint32 toTime) external
```

Fig. 5. Functions of Broker contract

For each IoT weather sensor we store the seller (owner of the sensor), its type, the price per measurement, the time of the first measurement (startTime), the frequency (how many measurements the sensor takes per hour), its position (latitude and longitude) and finally the URL where the data will be available. It is worth noting that inside the smart contract, the type of the sensor is marked with an 8-bit integer (256 different values). So the same smart contract can be used for different sensors, e.g. values 1-20 for IoT weather sensors, values 21-40 for gas sensors. Currently only values between 1 and 10 are used, so in the future, 246 additional sensor types can be added to the system.

Broker contract has a function called *createSensor*. It is invoked when then owner of an IoT sensor wants to register it in the application. It emits the event *SensorCreated* to notify the clients.

The *changeSensorSeller*, *changeSensorPrice*, *changeSensorUrl* functions which change the sensors seller (transfers the ownership of a sensor), the price per measurement and the URL where the data are available respectively, by specifying then sensor ID and the new value. Each one of them emits an event upon success. These functions succeed only when the sender of the transaction is the owner of the sensor with the specified ID. The function *buyData* is invoked when someone wants to buy data for a specific time interval of a specific IoT sensor. This function invokes the *brokerTransferFrom* function of the NtuaToken contract. Upon success (e.g. correct sensor ID given, sender has sufficient NTUA tokens etc.), an event called *CompletedTransaction* is emitted and the buyer gains access to the data.

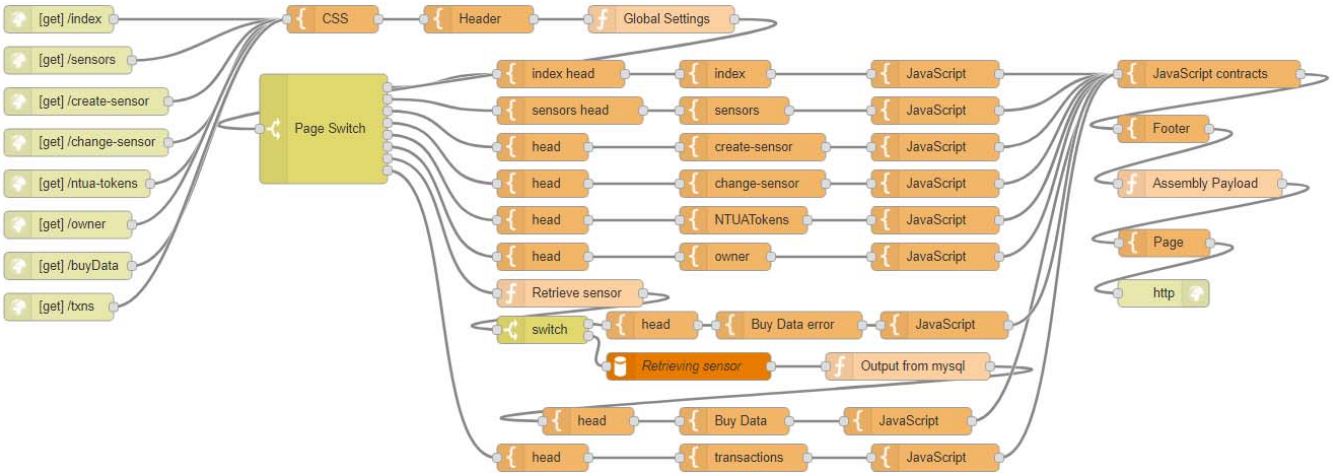

Fig. 6. Node-RED flow which handles HTTP requests and serves webpages

B. *Website and Database*

The website was designed to provide an interface between the users and the blockchain, so they can see information and send transactions fast and easy. If it is accessed with a browser like Ethereum Mist, or Chrome with Metamask plugin, the user can easily send transactions to the blockchain.

For the development of the website we used Node-RED [25], a flow-based development tool built on top of Node.js. So, JavaScript is used for the backend and HTML, jQuery, and Bootstrap for the front-end. The website's flow in Node-RED is shown in Figure 6. The Node.js server that has been deployed in our implementation, constantly listens the blockchain for the events of the smart contracts of the application. Once an event is detected, a record is stored in a Maria DB database, an SQL like relational database [26].

The application could operate without a database, since all information is stored in the blockchain. The database is only used to address performance issues. Retrieving data from the blockchain is very inefficient and slow, but retrieving data from an SQL like database is more efficient and faster. So, a Maria DB was used. This database only stores an exact copy of the information on the Ethereum blockchain that are relevant to the application. This way, when a user requests the available sensors, we retrieve them very fast from the relational database instead of looking them up in the Ethereum blockchain. We should highlight that by both ways exactly the same information will be served back to the user. The way the database is used does not alter the decentralized character of the application.

The website's main goal is to provide an interface between the users and the blockchain. This means users can send transactions to the blockchain without the interference of the server. The Ethereum blockchain guarantees the security of the transactions. After a webpage is served to the user, a connection is established between the browser and the Ethereum node, which runs locally on the user's computer. This connection is established only if a special browser is used. Then, a web3 object is injected by the browser. Using web3.js, a JavaScript library for interacting with a local or remote ethereum node [27], we can use the injected object to interact with the blockchain, as show in Figure 7. Transactions can now be sent to the blockchain. We can create the object of a contract, provided that we have its address and its Application Binary Interface (ABI). An example is shown in Figure 8.

```
window.addEventListener('load', function() {
  if (typeof web3 !== 'undefined') {
   window.web3=new Web3(web3.currentProvider)
   startApp()
  }});
```

Fig. 7. How to use Web3 object injected by the browser

```
const BrokerContract =
window.web3.eth.contract(BrokerContractAbi).
at(BrokerContractAddress, (err, ctr) => {})
```

Fig. 8. Creation of broker contract object

Now we can invoke functions of BrokerContract, or any other contract whose object we created before. For example, if the user presses the button to buy data of a sensor the code in Figure 9 is executed, which sends a transaction to the blockchain:

```
BrokerContract.buyData(sensorID,fromTime,
toTime,(err,res) => {}
```

Fig. 9. Invoke Broker contract's buyData function

In the event that the user uses Metamask plugin, when the above code is executed, a popup similar to that shown in Figure 10 will open. Through this popup, the user can set the

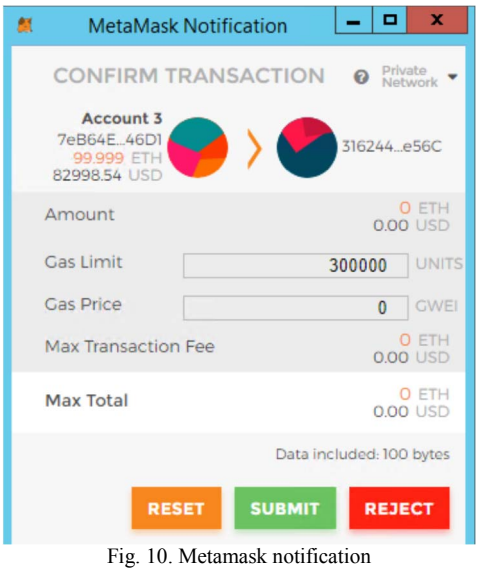

Fig. 10. Metamask notification

gas limit and the gas price offered for this transaction to the miners of the blockchain. He can also reject the transaction. If he presses reject no transaction will be sent to the blockchain. If he presses submit, the transaction will be signed with his private key and will then be sent to the blockchain. During this process, the server of the application never interferes, everything is done between the client and the Ethereum blockchain, thus the name Decentralized Application (DApp).

Our website provides eight main functionalities/screens:

1) A screen where the user can see all of his purchases and select a purchase to see its corresponding data.
2) A screen where the user can see, search and filter all the available sensors.
3) A screen where the user can see details of a sensor and buy its measurements taken in a time interval of his choice, as shown in Figure 11.
4) A screen where the user can check his balance and manage his NTUA tokens.
5) A screen where the owner/operator of a sensor can register it in the system.
6) A screen where the owner/operator of a registered sensor can change information concerning it.
7) A screen where the user can see, search and filter all transactions concerning the two smart contracts of the applications.
8) A screen where the owner of the smart contracts can find information concerning the contracts, change the price of the NTUA token, manage the ether balance of the NtuaToken contract.

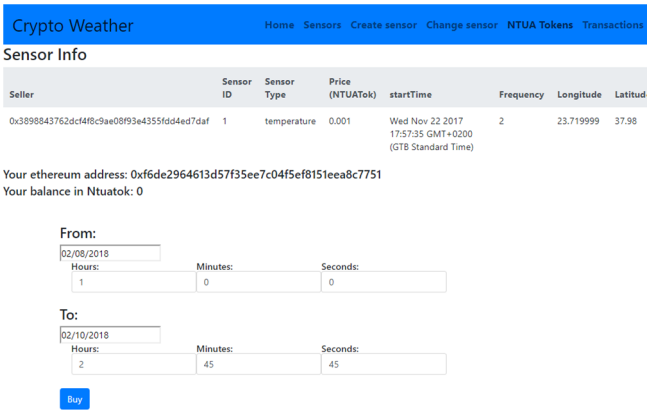

Fig. 11. Buy sensor data screen

We examined two different ways to make IoT sensor data available to the buyers and finally implemented the one that suited best. One way was to create a protocol which would enable buyers to request the data directly from the operators and the other was the application to act as an intermediate between the buyers and the owners. The first way gives to the application a completely decentralized character, but forces the operators of the sensors to set up services to interact with the blockchain, e.g. run an Ethereum node and listen for events. This way every operator should write special code specifically for Ethereum in order to sell the data of his sensors, which generates a barrier against potential sellers. The second way requires much less effort from the owner of the sensor, since he is not required to interact with Ethereum, thus making the application much more easily accessible and user-friendly. However, its disadvantage is that it slightly alters the decentralized character of the application. Both ways were designed but only the second one is implemented and is presented in this paper.

Because the application acts as an intermediate, users request the data from it and it requests the data from the sensor operators. Operators are provided with a unique API key when they register a sensor, with instructions on how the registering sensor's data must be available. This key is generated deterministically using elliptic curve encryption, by signing his account's public key with a (secret) private key. This API key will be later used by the application (as a password) to retrieve the IoT weather sensor data. A challenge in a possible commercial version of the DApp would concern how to ensure that operators of registered sensors will actually provide data of the sensors to the buyers.

When a sensor is registered, its owner provides a URL. The measurements of the sensor must be received with a POST request at this URL, containing a JSON with the API key provided to the sensor owner and an array containing pairs of Linux timestamps that specify the time intervals for which the data are requested, as follows:

```
{ key: apiKey,
[fromTime: 32bit linux Timestamp,
toTime: 32bit linux Timestamp] }
```

Fig. 12. Data sent along the request for sensor data

The response to this request must be a JSON array containing pairs of the timestamp and the value of each measurement taken into the specified intervals.

```
{ [time: 32bit linux Timestamp,
temperature: decimal number] }
```

Fig. 13. Response containing the IoT temperature sensor data requested

In Figure 14, we can see the flow executed every time a user requests data of a sensor. The user is asked to sign a random message (in a similar way he signs Ethereum transactions) for identification purposes. Then the signed data are sent to the server where the reverse sign happens (node "verify user"). If the user is verified successfully, we search whether he has actually purchased measurements of the sensor with the specified ID. If any purchases are found, the API key of the sensor's operator is retrieved and the data are requested with a POST request to the RESTful API the operator maintains. After that, the data are sent to the user that initially requested them.

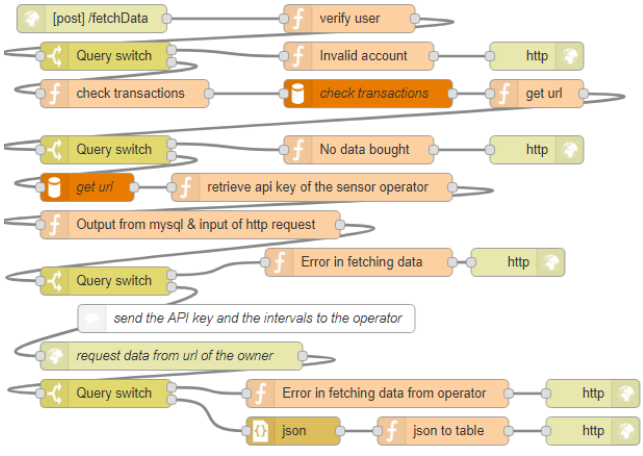

Fig. 14. Node-RED flow which handles user authentication requests data from the operator and serves them to the user originally requested them

C. Data Sources

During the development of the system we simulated the IoT weather sensors. For this simulation we used an API provided by Dark Sky [28]. With Node-RED tool we created flows that request current weather data for several locations from the Dark Sky API and then save these data (air temperature, relative humidity, pressure, visibility, wind speed and direction, sky cloud coverage, dew point, UV radiation and the columnar density of total atmospheric ozone layer) into a Maria DB database.

We also exposed a RESTful API in order to serve the data to the users when requested. This API is a model of the API, which will be maintained by operators of IoT weather sensors, in order to serve data to the buyers. When a request is received, the API key is checked. If it is correct, the data responding to the specified time intervals is retrieved from the database and then sent to the requester.

V. Experimental Results

After the deployment of the smart contracts on the Ropsten Test Net the gas usage of each (non-internal) function as well as the delay in the transaction confirmation was measured.

TABLE I. NtuaToken's Functions' Gas Usage

Function	Gas Usage
Contract deployment (NtuaToken constructor)	1000484 gas
Fallback function (payable)	21320 gas when invoked by the contract's owner 50294 gas when invoked by someone who is not the owner and has never before invoked it 35294 gas every time invoked by someone who is not the owner and has invoked it before
changePrice	Gas ranging from 27114 to 27562 depending on the parameters. For example: 27114 gas to change price to 1 wei[1] 27434 gas gas to change price to 1 ether 27562 gas to change price to 10^9 ether
retrieveEthereum	Gas ranging from 30184 to 30568 depending on the parameters. For example: 30184 gas to retrieve 1 wei[a] 30568 gas to retrieve 100 ether
transfer	~ 52000 gas for transfers from one account to another, if the receiver never before owned NTUATok. ~37000 gas for transfers from one account to another for the rest of the cases
brokerTransferFrom	Gas is already paid for buyData function of Broker contract

[a] 1 wei = 10^{-18} ether

TABLE II. Broker's Functions' Gas Usage

Function	Gas Usage
Contract deployment (Broker constructor)	1052408 gas
createSensor	Typically ranging from 142600 to 142800, but in some cases can variate much more. This is because there are many parameters and one of them is a string (not fixed sized)
changeSensorSeller	Typically ranging from 30106 to 30166 depending on the sensorID (size of the integer) and the address in the parameters.
changeSensorPrice	Typically ranging from 28723 to 30000 depending on the parameters.
changeSensorUrl	Greatly variates. Depending exclusively on the url parameter size, which is type of string (not fixed sized). The bigger it is the grater the gas requirement.
buyData	Approximately 120000 gas.

The fee in ether paid for the execution of each function can be found with the following formula:

$$fee = gas\ amount * gas\ price \quad (1)$$

If we want to find an approximation of the fees we set a specific gas price. For example, on the 30th of September the average gas price was equal to 16556363668 Wei. Assuming that the average gas price was offered by the sender of the transaction, the fee for every function of our smart contracts in USD can be found in TABLE III. Please note that the gas price is set by the user who sends the transaction.

TABLE III. Fee Paid For The Execution Of Each Function (Fixed Gas Price)

Function	Fee in ether
Contract deployment (NtuaToken constructor)	0,016564377
Fallback function (payable)	0,00058434 – 0,000832686
changePrice	0,000448909 – 0,000456326
retrieveEthereum	0,000499737 – 0,000506095
Transfer	0,000612585 – 0,000860931
Contract deployment (Broker constructor)	0,01742405
createSensor	0,002360937 – 0,002364249
changeSensorSeller	0,000498446 – 0,000499439
changeSensorPrice	0,000475548 – 0,000496691
buyData	0,001986764

An interesting fact is that the gas requirement of the functions remains the same no matter how many sensors are stored in the blockchain and no matter how old the block that contains the sensor which we want to change or buy data from is. Furthermore, the gas usage is the same whether the smart contract is deployed on the Ethereum Main Net, or a Test Net or even a private Net.

When the transaction confirmation delay was measured, we noticed variation in the results. In the Ropsten Test Net the delay ranged between a few seconds up to 5 minutes. Generally, it is not possible to predict the transaction confirmation delay, because it depends on the state of the network (e.g. pending transactions, the gas price of the pending transactions) and on the gas price offered from the sender of the transaction. According to Ethereum developers, this delay is expected to be minimized in the future, because the processed transactions per second (TPS) will be increased from 15 to a million [29].

VI. Conclusion

IoT networks are constantly growing, more and more sensors are added, costs are also constantly growing, so it is increasingly important to find ways to maximize the value of data generated by IoT sensors. Applying the S^2aaS business model is a way to increase the value of the data. S^2aaS brings new data monetization opportunities to IoT sensor operators, by enabling them to easily sell sensor data while it makes IoT sensor data more accessible to customers, by enabling them to easily access those data. However, for a successful implementation of an S^2aaS application a secure and easy to use payment system efficiently integrated with such applications is required. We believe that the blockchain may be this system.

We created a decentralized S²aaS application which uses smart contracts in the Ethereum blockchain. Using this application we created a marketplace for IoT weather sensor data. Sensor operators can register their sensors on the Ethereum blockchain and users can buy measurements of the registered sensors. By utilizing blockchain technology, we provide the users with the ability to perform P2P transactions on the Ethereum blockchain using cryptocurrencies. The currency used in this marketplace is a custom experimental token named NTUA token (NTUATok) along with Ether. By leveraging the flexibility Ethereum smart contracts provide, we created a platform that is far more usable, more easily accessible by humans, requiring the less technical knowledge, time and work from the sensor operators and the buyers than previously proposed applications that used Bitcoin. The capability of performing secure P2P transactions with cryptocurrencies, provided by Ethereum, and usability are the main advantages of the created DApp. The most important issues of the application are the sometimes high transaction confirmation delay, which is something that we expect to change in the future and the difficulty to prevent scam (registered sensor that does not supply data). The solution of such problem is outside the scope of our current research work as it also affects social engineering aspects. We are planning to extend the current research work in the future covering also such multidisciplinary aspects.

Regarding future extensions of the system, we are currently working on designing an API, so that machines could automatically carry out transactions and receive the corresponding data. We are also constantly watching for advancements in the blockchain and IoT sensors technology and examine if and how sensors could connect to the blockchain, receive requests and serve data, in order to examine if it is feasible and efficient to reduce or even eliminate centralized databases.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Commission under the H2020 Programme's project Bloomen (grant agreement nr. 762091).

REFERENCES

[1] S. Jernigan, S. Ransbotham and D. Kiron, "Data sharing and analytics drive success with IoT," 8 9 2016. [Online]. Available: https://sloanreview.mit.edu/projects/data-sharing-and-analytics-drive-success-with-internet-of-things/. [Accessed 26 6 2018].

[2] B. Buntz, "IoT Data: Strategic Sharing Will Yield Big Innovations," 20 6 2017. [Online]. Available: http://www.ioti.com/strategy/iot-data-strategic-sharing-will-yield-big-innovations. [Accessed 22 7 2018].

[3] G. Papadodimas, "Ανάπτυξη Έξυπνων Συμβολαίων στο Blockchain και εφαρμογή στο IoT," 31 5 2018. [Online]. Available: http://dspace.lib.ntua.gr/handle/123456789/46997.

[4] G. Papadodimas, "Broker.sol," [Online]. Available: https://github.com/george500/diplwmatiki_ergasia/blob/master/smart%20contracts/Broker.sol.

[5] G. Papadodimas, "Demo εφαρμογής Διπλωματικής Εργασίας," 2018. [Online]. Available: https://www.youtube.com/watch?v=Zcdm52Uow8g.

[6] K. Noyen, D. Volland, D. Wörner and E. Fleisch, "When Money Learns to Fly: Towards Sensing as a Service Applications Using Bitcoin," 20 9 2014. [Online]. Available: https://arxiv.org/abs/1409.5841. [Accessed 28 6 2018].

[7] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

[8] M. Conoscenti, A. Vetro and J. C. D. Martin, "Blockchain for the Internet of Things: a Systematic Literature Review," in *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, Agadir, 2016.

[9] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," *IEEE Access,* vol. 4, pp. 2292-2303, 2016.

[10] H. Hou, "The Application of Blockchain Technology in E-Government in China," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, Vancouver, 2017.

[11] A. Alketbi, D. Q. Nasir and D. M. A. Talib, "Blockchain for government services — Use cases, security benefits and challenges," in *2018 15th Learning and Technology Conference (L&T)*, Jeddah, 2018.

[12] BlockchainHub, "Initial Coin Offerings – ICOs," BlockchainHub, [Online]. Available: https://blockchainhub.net/ico-initial-coin-offerings/. [Accessed 20 6 2018].

[13] J. Ray, "Ethereum Introduction," 2 6 2018. [Online]. Available: https://github.com/ethereum/wiki/wiki/Ethereum-introduction.

[14] J. Pfeffer, "EthOn—introducing semantic Ethereum," 4 1 2017. [Online]. Available: https://media.consensys.net/ethon-introducing-semantic-ethereum-15f1f0696986. [Accessed 22 6 2018].

[15] "White Paper," [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper.

[16] Ethereum, "What is Ethereum?," [Online]. Available: http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html. [Accessed 25 1 2018].

[17] Ethereum, "Solidity," Ethereum, [Online]. Available: http://solidity.readthedocs.io. [Accessed 20 6 2018].

[18] B. Garner, "Off to the Races: Creating the Best Dapps Platform (Ethereum, NEO, QTUM, Lisk, Cardano)," 30 1 2018. [Online]. Available: https://coincentral.com/best-dapps-platform/.

[19] D. Wörner and T. v. Bomhard, "When your sensor earns money: exchanging data for cash with Bitcoin," in *2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, New York, 2014.

[20] S. Dolan, "Revise Electronic Transactions Act/blockchain/smart contracts," 23 5 2018. [Online]. Available: https://legiscan.com/OH/text/SB300/id/1795258.

[21] S. Huh, S. Cho and S. Kim, "Managing IoT devices using blockchain platform," in *2017 19th International Conference on Advanced Communication Technology (ICACT)*, Bongpyeong, 2017.

[22] L. Moore, "[Part One] Building an Ethereum Node on a Raspberry Pi 3B," 20 2 2018. [Online]. Available: https://medium.com/@thelucasmoore/part-one-building-an-ethereum-node-on-a-raspberry-pi-3b-481104974cf7.

[23] J. Pfeffer, "Over 12,000 Ether Are Lost Forever Due to Typos," 9 3 2018. [Online]. Available: https://media.consensys.net/over-12-000-ether-are-lost-forever-due-to-typos-f6ccc35432f8.

[24] eth, "How is the address of an Ethereum contract computed?," 24 3 2017. [Online]. Available: https://ethereum.stackexchange.com/questions/760/how-is-the-address-of-an-ethereum-contract-computed. [Accessed 29 6 2018].

[25] JS Foundation, "Node-RED," JS Foundation, [Online]. Available: https://nodered.org/. [Accessed 5 7 2018].

[26] "About MariaDB," MariaDB Foundation, 2018. [Online]. Available: https://mariadb.org/about/. [Accessed 25 6 2018].

[27] "web3.js - Ethereum JavaScript API," 2018. [Online]. Available: https://web3js.readthedocs.io/en/1.0/. [Accessed 30 6 2018].

[28] The Dark Sky Company, LLC, "Dark Sky," The Dark Sky Company, LLC, [Online]. Available: https://darksky.net/dev.

[29] K. Smith, "Vitalik — Ethereum en route to a million transactions per second," 7 6 2018. [Online]. Available: https://bravenewcoin.com/news/vitalik-ethereum-en-route-to-a-million-transactions-per-second/. [Accessed 28 6 2018].