sustainability

MDPI

Article

Blockchain IoT for Smart Electric Vehicles Battery Management

Bogdan Cristian Florea *,† **and Dragos Daniel Taralunga** †

Faculty of Electronics, Telecommunications and Information Technology, Politehnica University of Bucharest, 061071 Bucharest, Romania; dragos.taralunga@upb.ro

* Correspondence: bogdan.florea@upb.ro

† These authors contributed equally to this work.

Received: 9 April 2020; Accepted: 12 May 2020; Published: 13 May 2020

Abstract: Electric Vehicles (EVs) have generated a lot of interest in recent years, due to the advances in battery life and low pollution. Similarly, the expansion of the Internet of Things (IoT) allowed more and more devices to be interconnected. One major problem EVs face today is the limited range of the battery and the limited number of charging or battery swapping stations. A solution is to not only build the necessary infrastructure, but also to be able to correctly estimate the remaining power using an efficient battery management system (BMS). For some EVs, battery swapping can also be an option, either at registered stations, or even directly from other EV drivers. Thus, a network of EV information is required, so that a successful battery charge or swap can be made available for drivers. In this paper two blockchain implementations for an EV BMS are presented, using blockchain as the network and data layer of the application. The first implementation uses Ethereum as the blockchain framework for developing smart contracts, while the second uses a directed acyclic graph (DAG), on top of the IOTA tangle. The two approaches are implemented and compared, demonstrating that both platforms can provide a viable solution for an efficient, semi-decentralized, data-driven BMS.

Keywords: blockchain; battery management; electric vehicle; state of charge estimation; Internet of Things; smart contract; Ethereum; IOTA

1. Introduction

In recent years the development of electric vehicles has become one of the main interests for most major automobile manufacturers. One of the most concerning aspects for consumers regarding the adoption of electrical vehicles (EVs) is the autonomy and the lack of infrastructure in most areas, except for large cities.

The history of electric vehicles dates back almost to the introduction of the electric motor. EVs can be classified in the following categories: hybrid electric vehicles (HEVs), plug-in hybrid electric vehicles (PHEVs), and battery electric vehicles (BEVs) [1]. HEVs and PHEVs are powered by two sources: an electric motor and an internal combustion engine which can operate independently or in parallel, while BEVs operate solely using an electric motor powered by batteries or fuel-cells (FCEV) [2]. Moreover, HEVs do not provide an external charger for the batteries, while PHEVs allow external battery charging.

For HEVs and PHEVs, multiple drive-train configurations are available. Series operation EVs use only the electric motor for propulsion, which is coupled with the transmission, while the internal combustion engine is used solely to charge the battery when the state of charge is low. This method is useful for city driving, where frequent start-stops are required. The parallel operation uses both engines for propulsion, by simultaneously transmitting power to the wheels, thus improving the

efficiency of the vehicle. Finally, the series-parallel mode combines the features of the two previous modes, but the design of the vehicle is more complicated [1].

On the other hand, BEVs rely solely on battery power and an electric motor for propulsion, and both BEVs and PHEVs can be externally charged from a power source. There are two types of charges used: (i) on-board chargers, which are often designed for small size and low weight with the usual downside of providing a slow charge, and (ii) dedicated fast chargers that can be used in designated locations such as charging stations. For EV charging, various methods are used, such as constant current (CC), constant voltage (CV), constant power (CP) or a combination of these methods [1,3].

With the recent advances in battery technology [4–7], the autonomy of EVs has increased to well over 200 km and it will probably continue to increase over the next few years. However, EV drivers are still faced with many challenges related to battery life, battery charging time and especially the availability of charging stations. Thus, numerous pilot projects exist for implementing inductive charging on designated road sections, such as parking areas, traffic light stops or airport road segments for electric buses [8,9]. Until the technology becomes mainstream, EV drivers will still depend on fixed charging points.

One solution for this problem could be a decentralized network for battery charging or swapping, where users or charging stations can trade energy or batteries. Moreover, such a decentralized network can also be used by autonomous (self driving) vehicles (a new trend in vehicle development). These vehicles must be able to process a lot of information, so that the safety of the passengers is always ensured. Ideally, these vehicles should be able to communicate directly with other smart vehicles on the road and share information about traffic conditions, incidents, weather, etc. A method of machine-to-machine (M2M) communication is required so that true autonomy can be achieved. Since these vehicles are equipped with an array of sensors, one can perceive them as IoT (Internet of Things) devices.

To implement a decentralized IoT network of charging and swapping stations, which could allow even regular users to provide some of these services, some vital information is necessary, such as the type of batteries used in electric vehicles, the possibility of swapping these batteries, the state of the battery (charge cycles, health, remaining capacity, etc.) and the location of the stations and their availability of charging and swapping services. Such a system can be implemented using IoT, by monitoring the battery parameters while driving and warn the user when a charge is necessary. Due to the fact that the security of IoT applications is essential, new strategies have to be used to offer high security and reliability for IoT networks.

Blockchain is a novel technology, which was first introduced in 2008 as the underlying network architecture for the cryptocurrency Bitcoin [10]. The technology created an environment for secure, anonymous transactions, using a decentralized network of devices [11]. The main goal of the architecture was to create trust between the participants without the regulation of a central authority [10]. This technology has found many applications outside the financial world. Wang et al. proposed a decentralized electricity transaction model for microgrids, defining a continuous double auction mechanism directly between the buyer and the seller, while continuously adjusting the energy price according to market changes [12]. In [13], Khan et al. proposed and developed a method for validating microgrid transactions using signatures of multiple producers based on their attributes. These signatures are verified by consumers with matching attributes, without the regulation of a central authority. Wu and Tran presented an overview for blockchain technology in sustainable energy systems, describing different scenarios for blockchain use in energy systems [14]. In [15], Miller proposes a survey for the use of blockchain and IoT technologies in the industrial sector, providing solutions for supply chain management, autonomous vehicles and manufacturing plant asset management. Florea proposed a blockchain data provider, using IOTA to integrate IoT devices to create a decentralized data provider [11].

One of the main problems of IoT has always been the security of the communication between the devices. Son et al. proposed a directed acyclic graph (DAG)-based IoT protocol, using IOTA for

securing IoT applications [16]. Odysseas and Gialelis introduced a IOTA-based sensor node system [17], while Bartolomeu et al. used IOTA for vehicular applications [18].

This paper presents a blockchain application for electric vehicles' (EVs) battery charge and swap, considering two approaches: custom Ethereum blockchain and the IOTA public tangle. For both scenarios, the performance of the system is analyzed, highlighting the advantages and disadvantages of each method. The application is tested using lithium-polymer (LiPo) batteries driving DC motors in an environment similar to that of an EV. The battery voltage and state of charge (SOC) is continuously monitored and the data are stored on the blockchain, together with the required functionalities, such as user and battery information and charge/swap requests.

In the following sections, the paper will present the general background knowledge for blockchain, smart contracts and the IOTA tangle (Section 2), electric vehicles batteries and SOC estimation (Section 3), the proposed application structure and test build (Section 4), followed by the results and comparison between the two proposed methods (Section 5), the general discussion and the conclusions (Sections 6 and 7).

2. Background

2.1. Blockchain and Distributed Ledgers

Since the introduction of blockchain in 2008, blockchain and cryptocurrencies have become mainstream terms. The technology proposes a network similar to that of a distributed ledger technology(DLT). Specifically, it implements a peer-to-peer (P2P) network of distributed data sets shared over multiple locations, where every change in the ledger is reflected in all copies on the network [19]. This means that once a change is submitted by one of the participants, it must be validated and approved by the entire network before it is added to the ledger. To achieve this, a consensus mechanism is required, so that the information, once accepted, cannot be altered by any user or group of users. Different implementations of the blockchain technology provide different methods of consensus, but in most cases consensus is achieved by means of cryptographic functions.

The first blockchain application was introduced in [10], where the cryptocurrency Bitcoin was introduced, with the goal to replace the trust provided by a 3rd party with proof. The term *blockchain* suggests that the network consists of a series (or *chain*) of *blocks*, linked together by means of a cryptographic hash function (Figure 1). Each block can have multiple transactions (Tx1, Tx2, ...). When a new block is created, the corresponding transactions are stored as a Merkle tree (or hash tree), where each leaf (data) node is labeled with its cryptographic hash and each non-leaf node is labeled with the hash of the labels of its children [20]. When the block is created, only the root node of the hash tree is included in the block, which allows for old blocks to be compacted. Each new block includes the reference to the previous block hash. The network participants (or *miners*) will generate the corresponding hash of the new block with the restrictions imposed by the consensus algorithm (difficulty of the hash function). The miners can inject a *nonce* in order to find a resulting hash that respects the network specifications. Once this hash is created, the new block is added to the chain and the other users must validate and approve the block by verifying the generated hash. Once the block has been confirmed by the network it can no longer be changed, as any change on the block (or any of the previous blocks) will invalidate the computed hash and will be rejected by the network. In this way, data immutability is achieved, which is one of the main advantages of blockchain technologies because it ensures that no single entity can have control over the data.

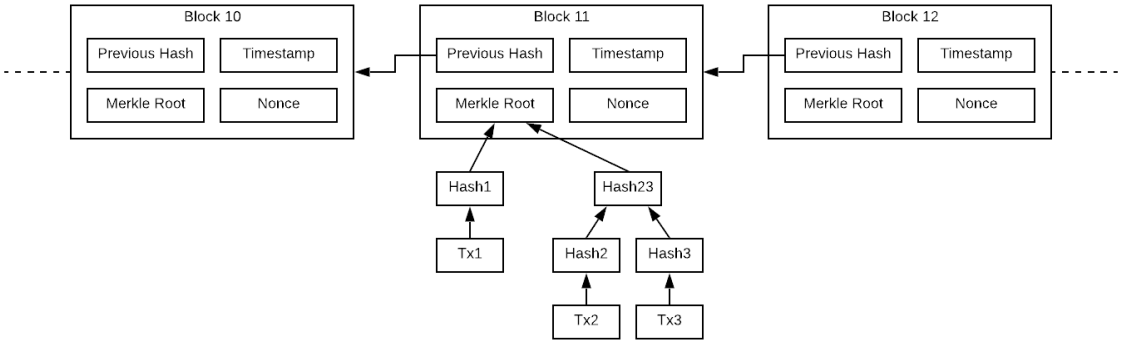


Figure 1. Typical blockchain structure [10,11].

On public blockchains, especially in the cryptocurrency applications, incentives are provided for network participants as rewards for discovering new blocks (generating the block's hash function). The user (miner) that submits the valid block is rewarded using a token specific for each platform. These tokens can be used in further transactions on the blockchain or even be exchanged for fiat currency. This method of submitting and verification of new blocks (consensus) is called *proof of work* (POW).

2.2. Ethereum and Smart Contracts

With the rapid development and adoption of the Bitcoin model, the need for an automated transaction model was evident. In 2014, the Ethereum platform was launched, introducing *smart contracts* to the blockchain [21]. The term *smart contract* was first introduced in [22] as a set of promises, defined in digital form, including the protocols within which the parties will perform them. Ethereum uses the same basic consensus mechanism of POW where the network nodes continuously mine new blocks by means of their hash function. Similar to Bitcoin, a network token is used as an incentive for the network participants, rewarding the node who successfully generates a valid block by a number of tokes (ETH). To support this reward system and keep the network running, any participant who submits a transaction to the blockchain pays a fee, called *gas*, in the same way that banks charge their clients a transaction fees for their operations.

Transactions on the Ethereum blockchain can be created either externally or by smart contracts, introducing the concept of functions to the blockchain [21]. Smart contracts are programs that are recorded and verified on the blockchain (Figure 2). Once the contract is deployed, its code cannot be altered, thus ensuring that the initial conditions will always be respected for any future executions. Just like real world contracts, a smart contract on the blockchain is created between two parties and is executed once some triggering event is set, such as a deadline or a specific target value being reached. In [23], the architecture and applications of blockchain smart contracts are described.

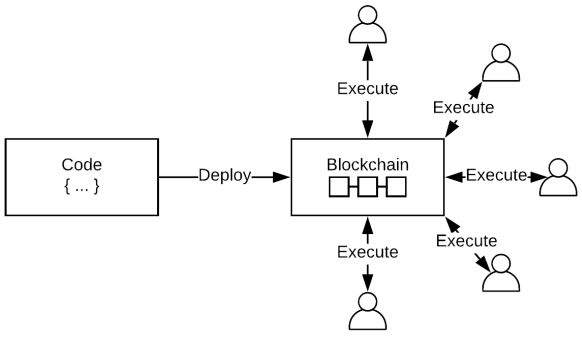


Figure 2. Smart contract anatomy.

Many blockchain implementations provide languages for the development of smart contracts to a certain extent of Turing-completeness. Ethereum, the most popular platform for smart contract and decentralized apps (dApp) development, introduced Solidity, a Turing complete language that allows the creation and deployment of smart contracts on the blockchain [21]. The code of the smart contract is written and compiled using the available language for each specific platform. A contract can have two main attributes: value and state [23]. Once the contract has been created and deployed, users can create calls (*execute*) to the contract functions. These calls can either query the contract state (*read* operations) or make changes which will result in a different state (*write* operations).

In Ethereum a very important difference must be made between these two types of functions. Calling (reading) a contract function is processed immediately, since they are executed locally without consulting the network, by accessing the current state of the blockchain. For this reason, *calls* do not require gas for their execution (they do not involve any fees).

Verified transactions have the potential (although not mandatory) to change the state of the contract (writing data). These types of executions must be validated by the entire network and the new state of the contract must be agreed upon. Since the data are processed by the entire network, these function calls require gas, which is determined by the number of computational steps and the amount of data which is handled by the contract [21]. These transactions are included in blocks (mined),data are and the change is reflected on the network once the new block is added to the blockchain (a new consensus is reached).

When a verified transaction is submitted it can have the following outcomes based on the amount of gas that is assigned for the transaction [21]:

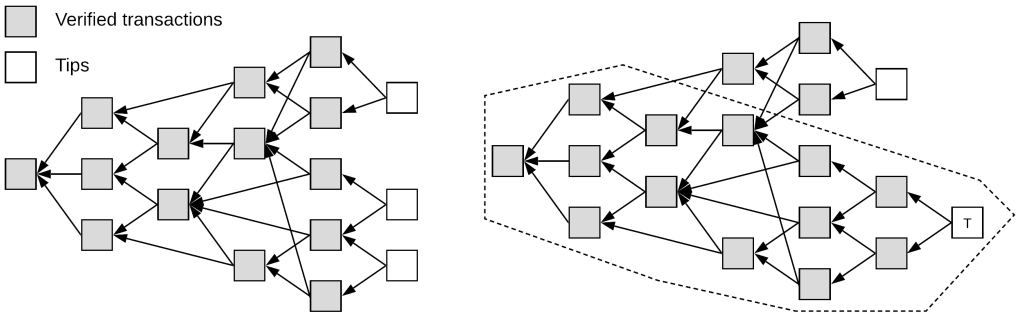
- The transaction is executed successfully. In this case the amount of gas that was required by the processing of data is subtracted from the sender's account and transferred to the mining nodes (which verify and process the transaction). If the sender allocated more gas than was actually required, the difference is returned to the sender's account.
- The transaction exceeded the gas limit set by the sender (the number of computational steps or the amount of data were too high for the fees which the sender was willing to spend). In this case all changes to the contract are reverted. However, the gas is still subtracted from the sender, since the other nodes have executed the code up to a certain point.

On a privately owned network, the gas limit and cost can be controlled so that the users do not run into these problems or their impact will be reduced. Furthermore, for specific applications, a possible solution to the gas-related costs could be the implementation of a semi-decentralized (permissioned) blockchain, where the cost of gas (processing) can be supported by one or more trusted nodes. This will be the network topology proposed in this paper.

2.3. IOTA Tangle

Since the blockchain consists of a series of blocks linked together by means of their cryptographic hash, when the number of transactions increases, the difficulty of the hash function also increases, which means that the creation of new blocks will require longer times and greater processing power [24]. This is currently one of the main drawbacks of public blockchain networks. The so-called *mining* operation is a required mechanism to achieve consensus between all participating nodes. To overcome the resource intensive protocols that are implemented by most platforms, various other consensus methods have been proposed. In [25], a comprehensive study of available and proposed consensus methods is presented. Regardless of the consensus mechanism employed, the scalability of the blockchain will still remain an issue due to the linear fashion in which the data are stored. For data-driven systems, which require a large number of data points, such as IoT applications, this limitation may become an issue. To overcome this issue a novel approach was introduced by the IOTA foundation [26].

The IOTA network was specifically designed with IoT applications in mind, proposing a different method of organizing the transaction data, by using a directed acyclic graph (DAG) instead of a linear blockchain. The IOTA DAG is often referred to as *the tangle*. Its main characteristic is that it allows zero-fee and zero-value transactions [26] (Figure 3a).



(**a**) The tangle (**b**) Transaction validation path

Figure 3. IOTA structure [11,26].

IOTA uses a trinary representation of data, as opposed to the usual binary system. Balanced trinary data can have the following states: 0, 1, or −1 (trits). Similar to how 8 bits form a byte, in IOTA 27 (3^3) from a tryte. IOTA uses special trinary hashing functions for data encoding and POW, which are described in [26].

In the usual blockchain implementation, the users have to perform the POW continuously until a hash is successfully found. This is a power-intensive operation that discourages new users from competing in the mining phase. In IOTA, each new transaction (*tip*) is attached to two previous transactions that it must validate by performing the necessary POW. In this manner, POW is only necessary when a node wants to create a new transaction on the network, thus validating two previous transactions. This means that the node that actually initiates the transaction is the one that computes the POW hash. This allows the elimination of network fees, since it is in the interest of the initiating node for its transaction to be completed. Hence, the network fee is replaced by the "on-demand" POW computation. Since a new transaction validates two previous transactions, the scaling problem can be successfully solved, because more transactions on the network result in more validated transactions.

Once a new transaction is attached to the tangle it also confirms all the previous transactions referenced by the two former tips to which it was attached. This creates a validation path (Figure 3b), which increases the trust for all the transactions inside this path. The IOTA network employs a Markov chain Monte Carlo (MCMC) method of choosing the tips to which the new transaction is attached [26].

3. Electric Vehicles Batteries and SOC Estimation

The battery is the single most important component in an EV. Most of the weight of an EV comes from the battery pack, so an efficient battery system, together with an optimized consumption model, are crucial in electric vehicle development. Most EV models use lithium-based batteries (lithium-ion, lithium-polymer, lithium-iron-phosphate) [27,28].

The energy requirements and number of battery-operated systems have changed drastically over the last decade. Thus, a real battery management system is required for most battery-operated applications and appliances. One of the most important parameters of a battery is the state of charge (SOC), which represents the available battery capacity. SOC can be influenced by a very large number of factors: battery chemistry, age, temperature, load characteristics, etc.

State of charge estimation methods vary depending both on the type of battery and on the application where it is used. In [29], various methods of SOC estimation and their applications were presented. The authors concluded that the most used technique is the Ampere-hour counting,

because of its ease of implementation and good-enough results, but other methods, such as linear modeling, Kalman filter, or artificial intelligence could be used, provided that the system can provide the necessary computational resources or training data.

In [30,31] the authors compared the available methods for lithium-ion batteries with specific applications for EVs. According to the authors, the best-suited methods for EV applications are the electrical circuit model (ECM) or the Kalman filter (KF) model. Machine learning and artificial intelligence models, although very accurate, are not well suited to the field of EV due to the high computational costs involved.

In this section two methods will be briefly presented: the Ampere-hour (Ah) counting and the open circuit voltage (OCV) estimation.

3.1. Ampere-Hour Counting SOC Estimation

The Ah counting (or Coulomb counting) method has become an industry standard and one of the most used methods for SOC estimation, because of its accuracy for short-term calculations [30]. The Ah method defines the SOC as:

$$SOC\left(t\right) = SOC\left(t_0\right) + \frac{1}{C_n}\int_{t_0}^{t} I_{battery} d\tau \times 100\%, \tag{1}$$

where $SOC\left(t_0\right)$ is the initial state of charge, C_n is the nominal capacity and $I_{battery}$ is the charging or discharging current of the battery [30].

The method is very simple, but it does not take into account the loss current. If the battery current is measured incorrectly, the method will accumulate the measuring errors. This can be compensated by better measuring techniques and sensors.

Taking into account the loss current, Equation (1) could be rewritten as:

$$SOC\left(t\right) = SOC\left(t_0\right) + \frac{1}{C_n}\int_{t_0}^{t} \left(I_{battery} - I_{loss}\right) d\tau \times 100\%, \tag{2}$$

where I_{loss} is the current consumed by loss reactions [31].

In order to use this method, the initial SOC must be known. If it is not known, or it is incorrectly estimated, the errors will accumulate throughout the process. This method is generally used in combination with other methods to improve the estimated result.

3.2. Open Circuit Voltage SOC Estimation

The OCV method uses the battery open circuit voltage as a function of the SOC by means of a polynomial equation or look-up table. The SOC is determined as a reverse function of OCV [30]:

$$SOC\left(t\right) = f^{-1}\left(OCV\right). \tag{3}$$

The open circuit voltage is continuously measured and the corresponding SOC is obtained using a look-up table specific for each battery type. The accuracy of the method can be very good if enough rest time is provided to estimate the SOC. Therefore, it is not ideal for real-time operation. The method can be used to calibrate the Ah counting Equation (1). Different OCV measurements can be obtained for the same battery under different charging or discharging currents, so the method may not always provide the same result, depending on the C-rates of the charging/discharging process [30].

In the following sections the OCV method is used for SOC estimation, because of the available battery data (look-up tables and empirical measurements). The proposed blockchain application will base its logic around the computed SOC values that can be improved by different estimation methods and measurement techniques.

4. Proposed Application for EV Battery Charging and Swapping

4.1. System Diagram

The proposed battery swapping and charging system is described in Figure 4. Each EV will have an on-board computer, which will monitor the battery data and send the information to the BMS application. Since the on-board computer may not have sufficient resources for computing the POW hashing functions, the charging stations will act as peers on the implemented blockchain and the transactions will be handled by the stations (computing the necessary POW).

The users can register on the system, add new batteries, and create charge or swap requests. These requests are handled by a smart contract and are deployed on the network.

Two blockchain implementations will be implemented and analyzed: a customized Ethereum blockchain (Section 4.4) and the IOTA tangle (Section 4.5).

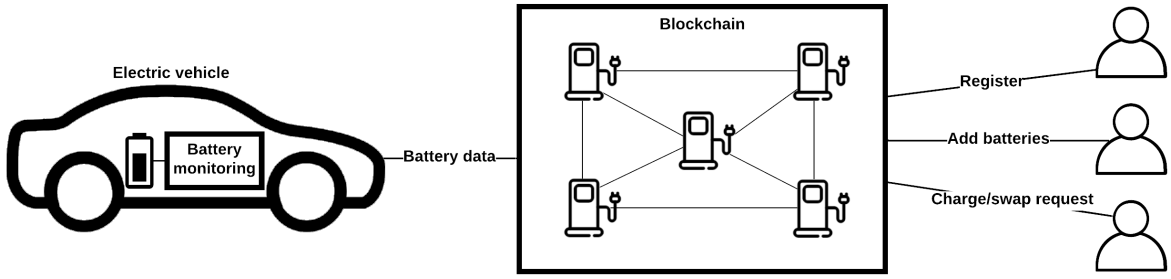


Figure 4. The proposed battery charging and swapping system diagram.

4.2. Battery Level and SOC Monitoring

For the purpose of developing and testing the application, lithium-polymer (LiPo) batteries were used because of their high current and discharge rates, which make them a common solution for EV battery packs.

Four LiPo batteries were chosen:

- Two 2-cell, 800 mAh, 7.4 V, 30 C discharge rate;
- Two 3-cell, 1000 mAh, 11.1 V, 30 C discharge rate.

The chosen SOC estimation method is the OCV. The voltage-SOC table (Table 1) was created using available LiPo look-up tables and checking them empirically.

To extend the life of the battery pack and to avoid deep discharging of the battery [32], it is recommended to remain outside the highlighted area. The data were mapped with a minimum safe battery voltage (V_{min}) of 3.5 V/cell (7.0 V/cell for two cells and 10.5 V/cell for three cells). Three approximations [33] were considered: linear, symmetric sigmoidal, and asymmetric sigmoidal.

Table 1. LiPo Voltage-SOC (state of charge) table.

SOC	Cell Voltage (V)	2 Cells Voltage (V)	3 Cells Voltage (V)
100%	4.20	8.40	12.60
95%	4.15	8.30	12.45
90%	4.11	8.22	12.33
85%	4.08	8.16	12.25
80%	4.02	8.05	12.07
75%	3.98	7.97	11.95
70%	3.95	7.91	11.86
65%	3.91	7.83	11.74
60%	3.87	7.75	11.62
55%	3.85	7.71	11.56
50%	3.84	7.67	11.51
45%	3.82	7.63	11.45
40%	3.80	7.59	11.39
35%	3.79	7.57	11.36
30%	3.77	7.53	11.30
25%	3.75	7.49	11.24
20%	3.73	7.45	11.18
15%	3.71	7.41	11.12
10%	3.69	7.37	11.06
5%	3.61	7.22	10.83
0%	3.27	6.55	9.82

The highlighted values represent the unsafe battery operating range.

The linear approximation is described by the following relation:

$$SOC = \frac{V - V_{min}}{V_{max} - V_{min}}, \tag{4}$$

where V is the battery voltage, V_{max} is the maximum voltage (for 100% charge), and V_{min} is the minimum safe voltage, as described in the previous paragraph.

The symmetric sigmoidal approximation was fitted using a 4-parameter logistics function (4PL):

$$SOC = d + \frac{a - d}{1 + \left(\frac{V_{normalized}}{c}\right)^b}, \tag{5}$$

where:

$$V_{normalized} = \frac{V - V_{min}}{V_{max} - V_{min}}, \tag{6}$$

and the parameters a, b, c, and d have the following meaning: a and d control the position of the horizontal asymptotes (upper and lower, respectively), b controls the slope of the response, and c controls the position of the transition region [34].

The asymmetric sigmoidal approximation was fitted using a 5-parameter logistics function (5PL):

$$SOC = d + \frac{a - d}{\left[1 + \left(\frac{V_{normalized}}{c}\right)^b\right]^g}, \tag{7}$$

where a, c, and d are defined in (5) and $V_{normalized}$ is defined in (6). In this case, the b parameter solely controls the approach to the top asymptote and together with g controls the approach to the bottom asymptote [34].

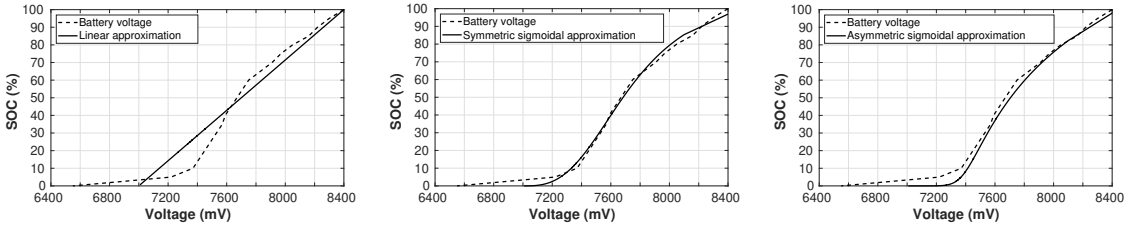
Using curve fitting tools, the parameters for the 4PL function were determined as: $a = 0$, $b = 2.9$, $c = 1/1.9$, and $d = 112$. With these values, Equation (5) becomes:

$$SOC = 112 - \frac{112}{1 + (1.9 \cdot V_{normalized})^{2.9}}. \quad (8)$$

In the case of the 5PL function the parameters are: $a = 0$, $b = 8$, $c = 1/3.5$, $d = 270$, and $g = 0.045$. With these values Equation (7) becomes:

$$SOC = 270 - \frac{270}{\left[1 + (3.5 \cdot V_{normalized})^{8}\right]^{0.045}}, \quad (9)$$

The voltage data in Table 1 are plotted together with the three approximations in Figure 5 for a 2-cell battery, with $V_{max} = 8400$ mV and $V_{min} = 7000$ mV.



(**a**) Linear (**b**) Symmetric sigmoidal (**c**) Asymmetric sigmoidal

Figure 5. Battery state of charge approximations.

It can be noted that both the 4PL (Figure 5b) and 5PL (Figure 5c) approximations provide good results, while the linear approximation (Figure 5a) is not suitable for SOC estimation of LiPo batteries. In the next section the 4PL approximation is chosen, as it requires less computational effort with very good results.

4.3. Battery Monitoring System

The battery monitoring system should be available on any registered EV. For this application, the BMS measures the battery voltage and, using Equation (8), estimates the remaining charge. The data are processed by a micro-controller and sent via I2C (inter-integrated-circuit) communication to an on-board computer (in this case a Raspberry Pi), which generates the transaction and submits the data to the blockchain network.

A monitoring board (Figure 6) was designed and implemented. An ATMega164p micro-controller is used to measure the battery voltage and compute the SOC using the 4PL approximation (8).

The board monitors all four test batteries simultaneously, which can also power the microcontroller using the power switch. The battery voltages are read from the analog pins of the microcontroller (A0–A3). Since the battery voltages are higher than the 5V accepted by the microcontroller, the voltage is adjusted using voltage dividers.

For the 3-cell batteries the voltage dividers are formed using $R1$ and $R2$ ($R3$ and $R4$ for the second battery), providing a ratio of 3.12. The divider ratio for the 2-cell batteries is 1.68 ($R5$ and $R6$ for the 3rd battery and $R7$ and $R8$ for the 4th).

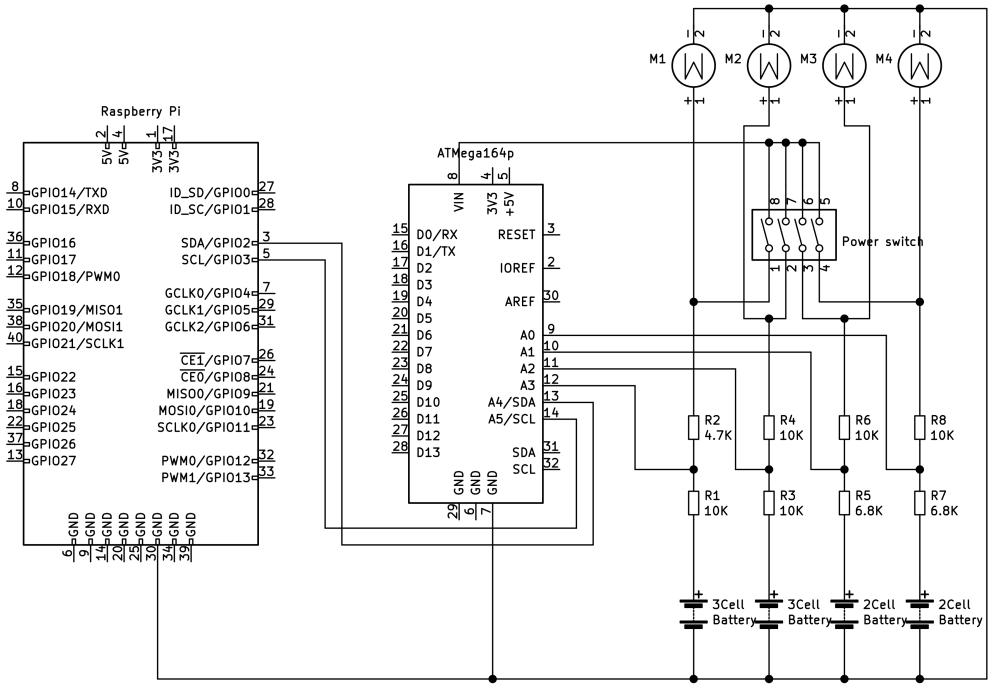


Figure 6. Battery monitoring board.

For the 3-cell batteries the read voltage ranges between 0 and 4.03 V and for the 2-cell batteries between 0 and 5 V.

The microcontroller transmits the data to the Raspberry Pi via I2C communication. The data are averaged over a series of 60 samples and the blockchain transaction is created. The anatomy of the transaction depends on the blockchain implementation and will be detailed in the following sections.

A battery test bed was designed and implemented (Figure 7), using a Raspberry Pi as an on-board computer for collecting the battery data and relaying the information to the blockchain.

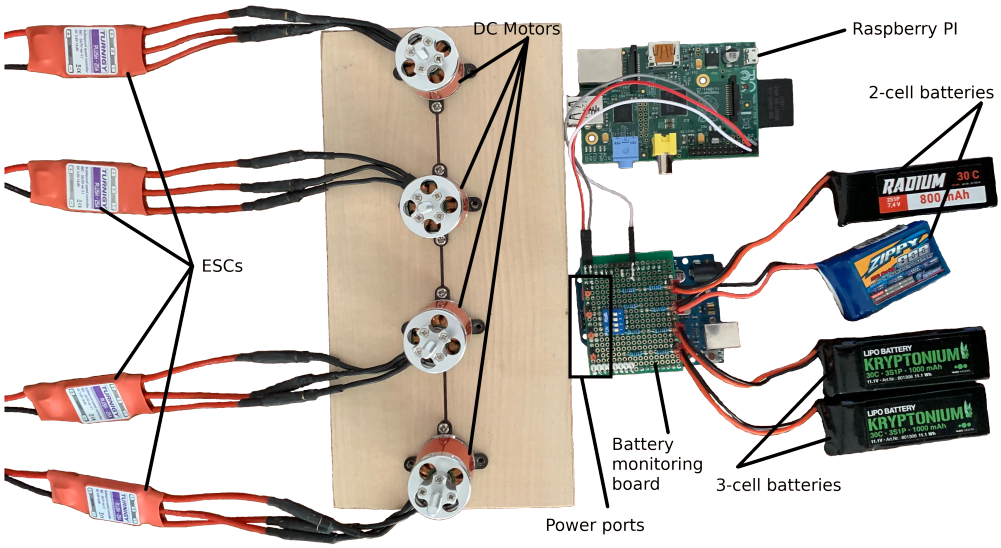


Figure 7. Battery test bed.

The transactions are created using the Python libraries `Web3.py` (Ethereum) and `PyOTA` (IOTA). These libraries implement the necessary functions for creating and submitting the transactions for their respective networks. The methods for each implementation will be described in their respective sections below.

For user interaction, a web application has been developed using `NodeJS` and the libraries specific for each platform (`Web3.js` for Ethereum and `iota.js` for IOTA). The web application allows the users and stations to register on the network, add new batteries and manage their requests.

The overall system architecture is presented in Figure 8.

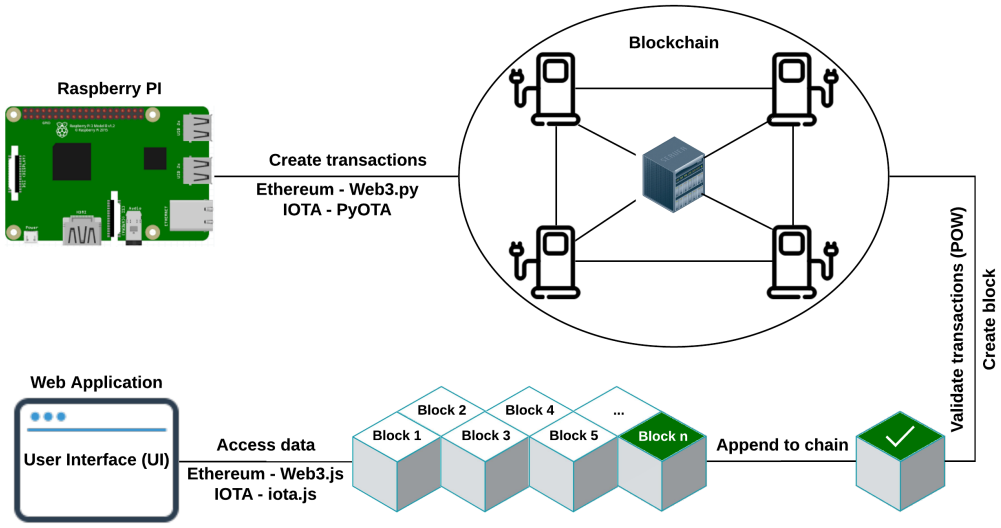


Figure 8. The blockchain interface for the proposed battery swapping and charging system. POW: proof of work.

4.4. Ethereum Blockchain

The first implementation of the charging/swapping application uses a smart contract deployed on a custom Ethereum blockchain.

In Ethereum, participating nodes have to perform POW to discover new blocks. To motivate users into participating in the block discovery process, a transaction fee (called *gas*) is paid by the user submitting the transaction. On the public chain the gas cost can be very high depending on the payload of the transaction. This makes the public Ethereum chain not well suited for data transactions, such as IoT applications.

On a customized chain, the gas cost and difficulty of the hash function can be controlled, in order to be used for specific applications, such as the battery management of electric vehicles.

On the main network any user with sufficiently capable processing resources can join. In a custom deployed network user access can be handled according to the application needs, thus allowing the implementation of permissioned blockchains.

In this paper, a permissioned network is created where the registration of new users or charging stations is handled by a master node. This creates a semi-decentralized network, but it can provide certain advantages, which will be highlighted in this section.

Users of the network are the EV owners. The EV computer (Raspberry Pi in this study) will sync to the network in light mode, where only the current state of the network and block headers are synced. For all processing operations a light node depends on the full node peers on the network (it does not take part in the *mining* process). The semi-decentralized structure ensures that there is at least one full node that is the master node of the application. Transactions and the necessary POW are handled by the charging/swapping station nodes, which will be synced as full nodes on the network.

When a user or station registers on the network via a web application, the master node creates a new address on the blockchain, which will be used to identify the user. The newly created address is stored and managed by the master node.

The registration process is handled by a smart contract deployed on the network. The proposed smart contract is described in Figure 9.

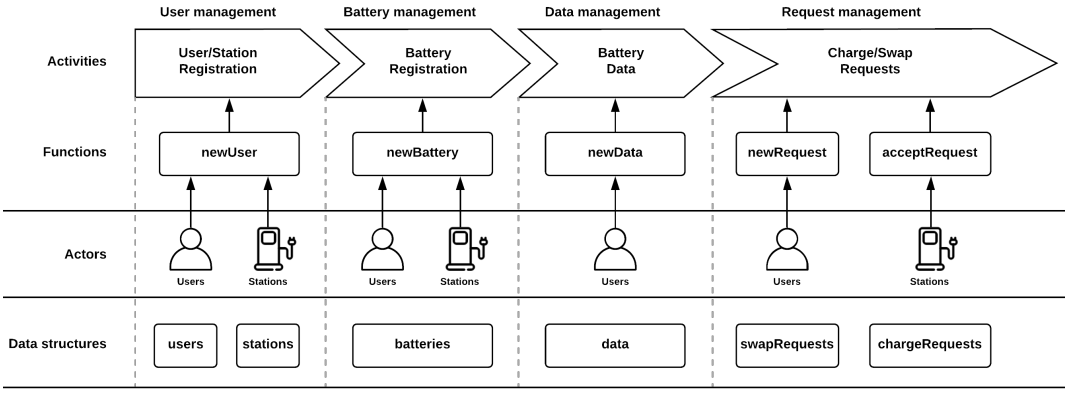


Figure 9. The smart contract structure.

The user management section provides the functions for registering new users or stations and for displaying their information.

The user and station information are stored inside the smart contract using the following structure:

```
struct User {
    uint id;
    uint balance;    // user balance
    uint8 userType; // user or station
    bool set;        // differentiate between unset and zero struct values
}
```

Upon registration, each user will have a unique id generated (similar to the primary keys in relational databases) and it can have an initial balance of virtual tokens that can be used to perform transaction on the network. The `userType` field specifies if the user is an EV owner or a charging station.

Each user or station is associated with an address by using a special `mapping` type, which maps the user address to the user data. All users can be accessed through the `user` variable.

```
mapping (address => User) public users;
```

The `newUser` function has the following prototype:

```
function newUser(address _address, uint8 _type) public
```

The function checks if a user with the specified address exists. If the address is not found, a new user object is created and its mapping is assigned. The `newUser` transaction is shown in Table 2 and the result is described in Table 3. The new user has an initial balance of 50 tokens.

In Table 2, the **From** address is different from the new user address (in the **Decoded input** field). The **From** address is the address of the master, as this is the node that actually creates the transaction. In this way, the gas cost required by the transaction is actually paid by the master instead of the user.

The gas used for this transaction would correspond to 0.00234 ETH on the main Ethereum network. For an ETH price of 258 USD (as of May 2019) the transaction cost would be around 0.60 USD. By running a separate network and allowing only mining nodes to submit transactions, the user is not required to have an actual Ethereum token (ETH) balance. Table 3 shows that for a **call** (read) operation, there is no cost involved, as the function only queries and returns the user data.

Table 2. New user transaction on the Ethereum blockchain.

Transaction hash	0x6cf7bdea3c4c8be4c023558e9a7089d840bb18115da1b11aa3ba55f77dd42b2c
From	**0x468fa9e5c2e87816688bcc96176bbe3e711ea4be**
To	BatteryContract.newUser(address, uint8) **0x6b2010a939adce6728d0e61d68c14d19c797a380**
Gas	117179
Input	0x7636a94c000000000000000000000000062650b2f80d471d29372a4be9acf28365cd419c 1000000000000000000000000000000000000000000000000000000000000000

Decoded input	{
	"address _address": "**0x62650b2f80d471d29372a4be9acf28365cd419c1**",
	"uint8 _type": 0
	}

Logs	[
	{
	"from": "0x6b2010a939adce6728d0e61d68c14d19c797a380",
	"topic": "0x05cdcffc20f5068d145a89a6ba9848186e191098da4d73e256368398e0a76ff7",
	"event": "newUserEvent",
	"args": {
	"_address": "0x62650b2f80d471d29372a4be9acf28365cd419c1",
	"_id": "0",
	"length": 2
	}
	}
	]

Value	0 wei

Table 3. New user result on the Ethereum blockchain.

Transaction hash	**call** 0x468fa9e5c2e87816688bcc96176bbe3e711ea4be 0x6b2010a939adce6728d0e61d68c14d19c797a380 0xa87430ba000000000000000000000000062650b2f80d471d29372a4be9acf28365cd419c1
From	0x468fa9e5c2e87816688bcc96176bbe3e711ea4be
To	BatteryContract.users(address) 0x6b2010a939adce6728d0e61d68c14d19c797a380
Input	0xa87430ba000000000000000000000000062650b2f80d471d29372a4be9acf28365cd419c1

Decoded input	{
	"address": "0x62650B2f80D471d29372a4Be9aCf28365cd419c1"
	}

Decoded output	{
	"0": "uint256: id 0",
	"1": "uint256: balance 50",
	"2": "uint8: userType 0",
	"3": "bool: set true"
	}

Once users are registered they can add new batteries for their EVs. The battery information is stored in a structure as follows:

```
struct Battery {
    uint id;
    bytes32 manufacturer;    // eg. Yuki
    bytes32 model;           // eg. Radium
    bytes32 batteryType;     // eg. LiPo
    uint32 capacity;         // eg. 800 (mAh)
    uint8 cells;             // eg. 2
    uint manufactureDate;    // Unix timestamp
    uint32 maxChargeCount;   // eg. 200
```

```
    bool set;                // differentiate between unset and zero struct values
}
```

The `newBattery` function has the following prototype:

```
function newBattery(address _address, bytes32 _manufacturer, bytes32 _model,
    bytes32 _batteryType, uint32 _capacity, uint8 _cells,
    uint256 _manufactureDate, uint32 _maxChargeCount) public
```

The function requires the battery owner's address (which was provided after registering) and the battery information. The resulting transaction is presented in Table A1. Since the amount of data written to the contract is much higher than the `newUser` transactions, the gas cost is almost double.

When a battery runs low, the user can manually submit a charge or swap request to the contract `newRequest` method (Table A2):

```
function newRequest(address _address, uint _batteryId, uint8 _requestType,
    uint _timestamp) public
```

The `_requestType` parameter specifies if it is a charge or swap request. Note that the request does not specify a specific station. All stations can view the request and whichever one can provide the requested service can accept it (Table A3). The function `acceptRequest` has the following definition:

```
function acceptRequest(address _address, uint _userIndex, address _station,
    uint _stationIndex, uint _approved) public
```

where the `_address` and `_userIndex` represent the user address and their battery identifier is used to locate the battery in the structure mapping; the `_station` and `_stationIndex` are used to specify the station's battery and the `_approved` parameter is the timestamp of the approval. The function changes the owners of the two batteries and the change will be reflected on the network once the transaction is confirmed. The result after the request is accepted is presented in Table A4.

The previous requests are created by the users/stations using the web interface. The battery information is submitted by the on-board computer and the data are stored in the smart contract using the following structure:

```
struct BatteryData {
    uint id;
    uint8 SOC;          // percentage
    uint32 voltage;     // mV
    uint32 chargeCount; // eg. 7
    uint timestamp;     // eg. 1562662637000
    bool set;
}
```

The Raspberry Pi collects the data and averages them over 60 samples (30 samples/minute). The `newData` contract function is called from the Python script:

```
function newData(address _address, uint _id, uint _index, uint8 _soc,
    uint32 _voltage, uint32 _chargeCount, uint _timestamp) public
```

where the `_address`, `_id`, and `_index` parameters are used to identify the user and the battery, `_voltage` and `_soc` are the actual measurements, and `_chargeCount` and `_timestamp` are the current charge count and the date and time of the measurement.

In this section, a full implementation of a battery swapping/charging system is described, using an Ethereum smart contract on a custom network. This approach allows further actions or improvements to be added to the smart contract, while the transaction costs are supported by the master node due to the semi-decentralized, permission-based approach.

Using the same contract on the main Ethereum network would be unfeasible, due to the high transaction costs (gas). The nature of the application is perfectly suited for a permission-based network, since the aim of the application is to transfer data rather than tokens on the blockchain.

4.5. IOTA Tangle

The Ethereum implementation uses the power and flexibility of smart contracts to implement the application logic on the blockchain ensuring that all nodes run the same code and data immutability.

The first difference between IOTA and other blockchain platforms is the way the data are stored (directed acyclic graph vs. linear chained blocks). This solves the scalability issue by linking a new transaction to two previous transactions by validating them, as described in Section 4.5.

Since the IOTA network does not have transaction fees, the implementation can run on the public network. One important disadvantage is that IOTA does not (*yet*) support smart contracts, and thus the application logic has to be handled by a master node, resulting in a semi-decentralized system. In this case, the master node has to perform extra operations to extract and filter the data from the tangle, whereas in the Ethereum version these operations were handled directly by the smart contract functions.

To implement the required functionality, the transaction structure is used (Table 4) by routing the different types of transactions to their specific actions.

Table 4. IOTA transaction anatomy [26].

Field	Description	Length (trytes)
address	Sender's or recipient's address, depending if the transaction withdraws or receives tokens	81
signatureMessageFragment	A signature if the transaction withdraws tokens or a tryte-encoded message otherwise. This can be split across multiple transactions	2187
value	The amount of tokens transferred	27
tag	User-defined tag	27
timestamp	Unix timestamp (seconds since Jan. 01 1970) of when the transaction was issued. In IOTA, the timestamp is not currently enforced and can be arbitrary	9
bundle	The hash of the bundle of the transaction	81
currentIndex	Index of the current transaction in the bundle	9
lastIndex	Index of the last transaction in the bundle	9
trunkTransaction	Hash of a parent transaction	81
branchTransaction	Hash of a parent transaction	81
attachmentTimestamp	Unix timestamp of when the POW was completed	9
nonce	The POW field of the transaction	27

To route the transaction to specific actions, the **tag** field is used, similar to a function call of a smart contract. Based on this the application will filter the data and assign it to the appropriate structures.

For example, the equivalent of the `newUser` operation from the Ethereum implementation would have the structure described in Table 5.

The tag field specifies the action of the transaction. Since the application is running on the public development network, any user can create a transaction with this tag. However, since all transactions are still handled by the master node as the sender, the application will only filter its own transactions from the tangle.

Since the IOTA implementation uses only raw transactions, their structure for the other operations are similar to the one presented in Table 5. Each operation has a corresponding tag field: `IOTABMSNEWUSER` for the new user transactions; `IOTABMSNEWBATTERY` for the new battery operation; `IOTABMSNEWDATA` for battery data information; and `IOTABMSNEWREQUEST` and `IOTABMSACCEPTREQUEST`

for the request operations. Because the IOTA implementation uses the public Devnet tangle, these tags can be used to examine the transaction details at https://devnet.thetangle.org.

For this implementation, the monitoring board collects the data over 60 samples, at the same rate of 30 samples/minute. The data are stored directly on the tangle, so the Raspberry Pi has to create a new transaction using the `ProposedTransaction` class from the `PyOTA` Python library:

```
tx = ProposedTransaction(
    address=Address(IOTAAddress),
    value=0,
    tag=Tag(b'IOTABMSDATA'),
    message=TryteString.from_string(IOTAJSONData)
)
```

where `IOTAAddress` is the address of the master node and `IOTAJSONData` is a JSON object containing the measurement information, with the same properties as the `newData` function from the Ethereum implementation. The `ProposedTransaction` object is a transaction that was created locally and has not yet been submitted to the network. To actually broadcast the transaction, the `send_transfer` function is used:

```
IOTAApi.send_transfer(
    depth=3,
    transfers=[tx],
    min_weight_magnitude= 14,
    inputs=BatteryAddress
)
```

where `BatteryAddress` is the address of the registered battery (acting as the sender of the message), `depth` is the maximum depth in the tangle for the tip selection mechanism, and `min_weight_magnitude` is an optional parameter used to specify the POW difficulty. Note that the `transfers` parameter contains a list of transactions (in this case only one), since there is no value transferred between the two addresses. When transferring IOTA tokens, at least two transactions are required: one that adds tokens to the recipient and one that subtracts the same amount from the sender.

Ideally, the POW should be performed locally. However, the Raspberry PI does not have the necessary resources to compute the POW, so the transaction is handled by the master node of the system, which computes the validation hashes for two selected tips. If the master node is not reachable, the transaction data are stored locally and resubmitted when the connection can be established.

The biggest challenge and limitation of this implementation is the battery swapping operation. Since the swap request is not sent to a particular station (similar to the Ethereum implementation), the station that accepts the request has to do so by creating a new transaction. This new transaction has to somehow reference the swap request submitted by the user. Two solutions can be outlined for this problem: either reference the swap request transaction hash in the `message` field of the transaction, together with the rest of the information, or override the IOTA's tip selection mechanism and attach the `acceptRequest` transaction directly to the related request transaction using the `trunkTransaction` or `branchTransaction` fields (Table 4). For this implementation, the first method was preferred.

The main difference between the IOTA implementation and the Ethereum one is that the code which handles the data storage and processing is not contained in a smart contract (distributed across all nodes), but runs solely on the master node. This makes the IOTA implementation more centralized than its Ethereum counterpart, due to the fact that the master node still represents a single point of failure for the application.

Table 5. New user transaction on the IOTA tangle.

Transaction hash	AMCSQBBEUNDBITQMW9ZYDXA9H9YBMO9QQZMAQARQGFFWUZX9HOUYHFI ENRBAUTQ9IZZ9ZSWBMULFZ9999
From	UHLEMW9QSZBRM9QVQGICTIMKWQNWDPQLYPHCMMHR9JDDYJXHGAOVLJR LEWACCLJTXFOJLJAAJLXVRUDIW
To	SMVIZBPLNBFOALXONUIREQZZWN9HTLVJQVEDUDVNGMLYP9SQDOLMMWC 9WXRKXFJZMLQZU9TMRSWJCDZKD
Bundle	STODQIOYCBKLUQWWHARYLQUEKLFDZNNSVASDQSHCHEXWZEG9ZKCH9LJ KV9YTHJZDTUJDGSESUHUQYYDA9
Index in bundle	0/0
Trunk transaction	Y9YHCOTRWEMFREDJ9HWRWSKONSUGADUCSXZDDMJLVD9IJYDWSNCG9PN PWTCEWWTOXHKKBWYVDUIOXM999
Branch transaction	Y9YHCOTRWEMFREDJ9HWRWSKONSUGADUCSXZDDMJLVD9IJYDWSNCG9PN PWTCEWWTOXHKKBWYVDUIOXM999
Tag	**IOTABMSNEWUSER**9999999999999
Message	ODGAPCSCSCFDTCGDGDGADBGABCWBECSBICLBZBVBXBLBPBYBKBVBGCY BXBDCSBACOB9CICICFCXBCBRBCCVBECTB9CECOBNBDCNBECXBQBWBVBH CZBCBBC9CNBYBVBWBWBFCMBCBFCGCACUBGCPBTBICWBVB9CICDCCBCC WBACBCFCTBMBNBICUBNBGAQAGAHDMDDDTCGADBGAUAGAQAGAQCP C9DPCBDRCTCGADBZAUAQAGAHDXCADTCGDHDPCADDDGADBGAVAZAZ ABBYAYAWAXAWACBVAYAWAGAQD
Decoded message	{ "address": "SMVIZBPLNB…MRSWJCDZKD", "type": "0", "balance": 50, "timestamp": "1558442329142" }

In the case of smart contracts, the corresponding function is executed whenever a transaction is submitted. In the IOTA implementation the master node has to monitor the network state continuously using the transaction **tag** field and decide which function should be executed locally. Here, the tangle is used only as a decentralized data storage (instead of a traditional database management system). When the user wants to access the information from the web application, the master node queries the tangle for the raw data and any necessary processing is done before presenting the data to the user.

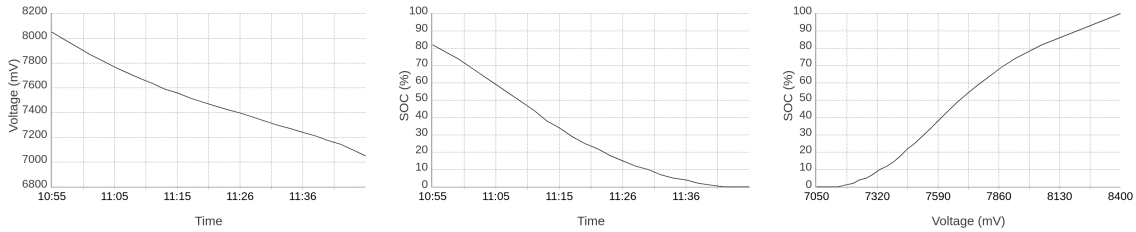
Off-chain processing is recommended when working with distributed applications, since computational power is expensive on any blockchain implementation, but it is much more difficult to work with unstructured data than with organized data structures achieved through smart contracts.

One important difference between the IOTA implementation and the Ethereum counterpart is the fact that Ethereum requires at least two full nodes (the master node and at least one charging station) to perform the POW and create new blocks on the network. In IOTA, the charging stations can be connected to the tangle, but this is not mandatory, since the actual POW is done by the device that creates the transaction (the master node in this implementation). This reduces the setup time and the necessary hardware requirements for the stations, but has the downside that if the master node is not available, the system may encounter some down-times.

5. Results

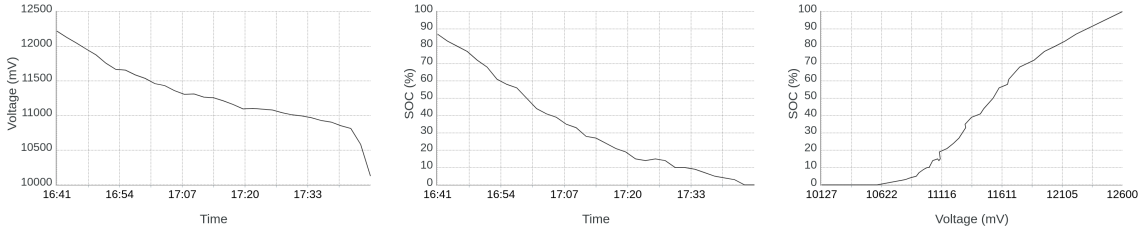
Figures 10 and 11 show the voltage and state of charge plots for a 2-cell and 3-cell battery, respectively. These results are obtained directly in the front-end application and can be viewed by the users. They can help improve the SOC estimation by accounting for battery age and other environmental factors, and possibly allow stations or users to properly service or replace batteries in

safe operating conditions. It can be noted that the SOC vs. Voltage plot (Figures 10c and 11c) is similar to the theoretical 4-PL approximation in Figure 5b, which is the expected result.



(**a**) Voltage vs. Time (**b**) SOC vs. Time (**c**) SOC vs. Voltage

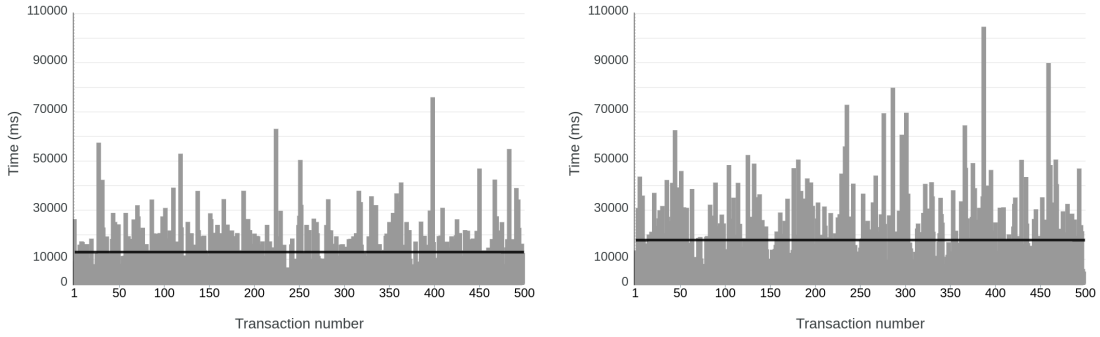
Figure 10. Experimental results for 2-cell battery.



(**a**) Voltage vs. Time (**b**) SOC vs. Time (**c**) SOC vs. Voltage

Figure 11. Experimental results for 3-cell battery.

To compare the two implementations, the transaction times were measured for a set of 500 transactions submitting new battery data (Figure 12). For the Ethereum blockchain (Figure 12a), running with two mining nodes (one master node and one charging station), the time from submitting a transaction until its first confirmation by the network ranged between 1.65 and 75.45 s, with an average of 12.97 s per transaction. These confirmation times can vary depending on the network congestion, the number of operations required in the contract function and the difficulty of the network, which, in the case of a permission-based chain, can be controlled by the network administrator.



(**a**) Ethereum blockchain (**b**) IOTA tangle

Figure 12. Transaction confirmation times.

For the IOTA tangle the results for the same transactions are presented in Figure 12b. It can be seen that the transaction times are slightly higher than the Ethereum implementation, with an average of 17.86 s and the minimum and maximum values of 3.78 and 104.14 seconds, respectively. These differences can be easily explained by the different approach of the two platforms. While in Ethereum, mining nodes perform the POW operations continuously, in IOTA these operations are performed only when a new transaction is submitted. In the proposed implementation the IOTA POW is not done locally due to the limited resources of the Raspberry Pi board. Instead, the required operations are performed by a public node outside the application infrastructure (in this case, the IOTA Devnet node).

Depending on the load of the node and the network congestion, the expected transaction recording time can vary. However, the benefit is an easier setup and fewer resources necessary for the master and station nodes of the BMS application.

6. Discussion

The Ethereum blockchain demonstrates that the technology is mature enough to be used in many fields of industry, offering the possibility of developing any kind of application logic on the blockchain in a decentralized manner. The main disadvantage of this approach is the rather complicated setup required to deploy the blockchain and smart contract. Since the application would not be suited to function on the main network, due to the high transaction fees, the creation of a separate network is absolutely necessary. Hence, one advantage is the possibility to manage the network settings and difficulty, so that the response time can be reduced, compared to a transaction on the public blockchain. Another advantage, which will be considered in future works, is to setup the blockchain using a different consensus mechanism, such as proof-of-authority (POA), where designated nodes can add new blocks to the chain without requiring a swarm of mining nodes.

IOTA was considered as an alternative because of its zero-fee transaction model and its scalability, which allows the application to run on the main tangle with minimal additional setup. The lack of smart contracts is the main disadvantage of this approach, because the application logic has to be maintained by one or more central nodes and it is more susceptible to failures, while possible changes of the application may affect its overall functionality on the network. These problems could be resolved when smart contracts become available in IOTA (which were already announced as of 2019), but the implementation may provide additional difficulties that cannot yet be estimated.

The main problem of the proposed IOTA implementation is the tag-based routing system, which can be replicated by users not registered on the battery management application. This can be solved either by having one or more trusted nodes (or *oracles*) that will generate the transactions on behalf of their users (as proposed in this paper), or by using masked authenticated messages (MAM) [35] as a means to create direct channels of communication between data publishers (EV owners) and subscribers (stations). This approach will be further studied in future works.

Battery charging requests on IOTA can be easily created because they do not involve changing the ownership of the battery, but, without the capabilities of on-chain processing, swap requests can be more difficult to handle, as they require the front-end application to search through the transaction history to find the latest state of the requested batteries. Due to this fact, IOTA is better related to producer–consumer applications, where sensors or embedded devices are the data producers and the users query and use the data off-tangle (data flow is unidirectional).

An issue for any application which relies on user information and value exchange (tokens, electricity, batteries, fiat currency, etc.) must deal with the privacy issues that may arise from these interactions. Several methods can be considered, depending on the type of blockchain that is being used and on the visibility of the blockchain network. Feng et al. [36] and Jia et al. [37] introduced such methods that deal with blockchain privacy.

The Ethereum-based implementation presented in this paper uses a private blockchain network where user access control (UAC) methods can be implemented to protect the user information. Furthermore, sensitive user information can be encrypted using a private–public key mechanism.

The proposed IOTA implementation uses the public tangle; thus, MAM can be used to establish a direct channel between the involved parties when an exchange is requested (battery swap or driver-to-driver charging).

These issues are an interesting extension for this study and the authors plan on extending the results of the current paper with an in-depth study on blockchain data privacy and protection.

7. Conclusions

In this paper, a complete battery management system for electric vehicles was presented using blockchain technology to create a semi-decentralized network of electric vehicles and charging stations that are able to share data (battery information and condition) based on continuous monitoring.

Blockchain is a relatively young technology that has seen major growth and adoption in the last 10 years with the development of cryptocurrencies. The underlying architecture makes it a strong candidate for data-driven applications, such as electric and autonomous vehicles.

To analyze the feasibility of the proposed application, two distinct implementations were considered and tested: the first method uses an Ethereum blockchain, powered by a smart contract, which allows the distributed processing and sharing of data, while at the same time ensuring data immutability and privacy. The second approach uses the IOTA network, which lacks the support for smart contract development, but provides zero-fee transactions and is built with data-driven applications in mind, thus allowing better application scaling.

The results show that both platforms can be used for developing blockchain IoT applications having acceptable transaction confirmation times for the purpose of the proposed battery monitoring system, which does not require real-time confirmations. Ethereum is still the main platform for developing decentralized applications, which can be deduced from the large number of projects and research papers available. This may change in the future especially for IoT and data-driven applications, as alternative approaches specially designed for this field mature and become available.

Author Contributions: Methodology, B.C.F. and D.D.T.; implementation, B.C.F.; writing—original draft, B.C.F.; writing—review and editing, D.D.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

4PL	4-Parameter Logistics
5PL	5-Parameter Logistics
Ah	Ampere-Hour
BEV	Battery Electric Vehicle
BMS	Battery Management System
DAG	Directed Acyclic Graph
DLT	Distributed Ledger Technology
ECM	Electrical Circuit Model
ESC	Electronic Speed Control
EV	Electric Vehicle
FCEV	Fuel-Cell Electric Vehicle
I2C	Inter-Integrated Circuit
IoT	Internet of Things
HEV	Hybrid Electric Vehicle
KF	Kalman Filter
LiPo	Lithium-Polymer
M2M	Machine to Machine
MAM	Masked Authenticated Messages
OCV	Open Circuit Voltage
P2P	Peer to Peer
PHEV	Plug-in Hybrid Electric Vehicle
POA	Proof of Authority
POW	Proof of Work
SOC	State of Charge
UAC	User Access Control

Appendix A. Ethereum Transactions

Table A1. New battery transaction on the Ethereum blockchain.

Transaction hash	0xe3bdb085f2dc0c7423bc7a8a50c3c75f431faccf7fe9ced9c8889e755845455d
From	0x468fa9e5c2e87816688bcc96176bbe3e711ea4be
To	BatteryContract.newBattery(address,bytes32,bytes32,bytes32,uint32,uint8,uint256,uint32) 0x6b2010a939adce6728d0e61d68c14d19c797a380
Gas	**213278**
Input	0x2b069245 . . . 000000c8

Decoded input	{
	"address _address": "0x62650b2f80d471d29372a4be9acf28365cd419c1",
	"bytes32 _manufacturer": "0x59756b69",
	"bytes32 _model": "0x4b727970746f6e69756d0000",
	"bytes32 _batteryType": "0x4c69506f",
	"uint32 _capacity": 1000,
	"uint8 _cells": 3,
	"uint256 _manufactureDate": "1526989237000",
	"uint32 _maxChargeCount": 200
	}

Logs	[
	{
	"from": "0x6b2010a939adce6728d0e61d68c14d19c797a380",
	"topic": "0xf01d0b1897e49a087d6d16276d60ab14a343d4e4b451af73bfc055cab1888c7c",
	"event": "newBatteryEvent",
	"args": { . . . }
	}
	]

Table A2. New swap request transaction on the Ethereum blockchain.

Transaction hash	0xdb90386e3ab9877e7da06699cb975fcf97020211f3c351755302961146d1d5b5
From	0x468fa9e5c2e87816688bcc96176bbe3e711ea4be
To	BatteryContract.newRequest(address,uint256,uint8,uint256) 0x6b2010a939adce6728d0e61d68c14d19c797a380
Gas	141145
Input	0xc286525a . . . e5153348

Decoded input	{
	"address _address": "0x62650B2f80D471d29372a4Be9aCf28365cd419c1",
	"uint256 _batteryId": "0",
	"uint8 _requestType": 0,
	"uint256 _timestamp": "1558621533000"
	}

Logs	[
	{
	"from": "0x6b2010a939adce6728d0e61d68c14d19c797a380",
	"topic": "0x28d61e4489683b71fb18e12434540b0f2e409d794b88c706069480866c22f893",
	"event": "newRequestEvent",
	"args": { . . . }
	}
	]

Table A3. New accept swap request transaction on the Ethereum blockchain.

Transaction hash	0x3a78d06023f5d493cb1a55f21c0bbf0f803d98a6373dba5bd0c92e87401bf26b
From	0x468fa9e5c2e87816688bcc96176bbe3e711ea4be
To	BatteryContract.acceptRequest(address,uint256,address,uint256,uint256) 0x6b2010a939adce6728d0e61d68c14d19c797a380
Gas	469526
Input	0xeca01e5a0000000000000000000000062650b2f80d471d29372a4be9acf28365cd419c10000 0000000000000000000000000000000000000000000000000000000000000000 0000001365af95d86cf447dbe6b991be87d5c4a59a3e3800000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 000000016ae5153348

Decoded input	{
	"address _address": "0x62650B2f80D471d29372a4Be9aCf28365cd419c1",
	"uint256 _userIndex": "0",
	"address _station": "0x1365af95D86cF447dbe6B991bE87d5c4A59A3E38",
	"uint256 _stationIndex": "0",
	"uint256 _approved": "1558621533000"
	}

Logs	[
	{
	"from": "0x6b2010a939adce6728d0e61d68c14d19c797a380",
	"topic": "0xf1e21ec30699d34039ffbb472baadcc06326aa1a6fc32b220f936f2a58a8a495",
	"event": "acceptedRequestEvent",
	"args": {…}
	}
	]

Table A4. Accepted request result on the Ethereum blockchain.

Transaction hash	**call** 0x468fa9e5c2e87816688bcc96176bbe3e711ea4be 0x6b2010a939adce6728d0e61d68c14d19c797a380 0x9ecebe2a0000000000000000000000062650b2f80d471d29372a4be9acf28365cd419c10000 0000000000000000000000000000000000000000000000000000000000000000
From	0x468fa9e5c2e87816688bcc96176bbe3e711ea4be
To	BatteryContract.requests(address, uint256) 0x6b2010a939adce6728d0e61d68c14d19c797a380
Input	0x9ecebe2a0000000000000000000000062650b2f80d471d29372a4be9acf28365cd419c10000 0000000000000000000000000000000000000000000000000000000000000000

Decoded input	{
	"address": "0x62650B2f80D471d29372a4Be9aCf28365cd419c1"
	"uint256": "0"
	}

Decoded output	{
	"0": "uint256: id 0",
	"1": "uint256: batteryId 0",
	"2": "uint8: requestType 0",
	"3": "uint256: timestamp 1558621533000",
	"4": "address: station 0x1365af95D86cF447dbe6B991bE87d5c4A59A3E38"
	"5": "uint256: approved 1558621533000"
	"6": "bool: set true"
	}

References

1. Yong, J.Y.; Ramachandaramurthy, V.K.; Tan, K.M.; Mithulananthan, N. A review on the state-of-the-art technologies of electric vehicle, its impacts and prospects. *Renew. Sustain. Energy Rev.* **2015**, *49*, 365–385. [CrossRef]
2. Eberle, U.; von Helmolt, R. Sustainable transportation based on electric vehicle concepts: A brief overview. *Energy Environ. Sci.* **2010**, *3*, 689–699. [CrossRef]
3. Dharmakeerthi, C.H.; Mithulananthan, N.; Saha, T.K. Modeling and planning of EV fast charging station in power grid. In Proceedings of the 2012 IEEE Power and Energy Society General Meeting, San Diego, CA, USA, 22–26 July 2012; pp. 1–8.
4. Hannan, M.A.; Hoque, M.M.; Hussain, A.; Yusof, Y.; Ker, P.J. State-of-the-Art and Energy Management System of Lithium-Ion Batteries in Electric Vehicle Applications: Issues and Recommendations. *IEEE Access* **2018**, *6*, 19362–19378. [CrossRef]
5. del Valle, J.A.; Anseán, D.; Carlos Viera, J.; Antuña, J.L.; González, M.; García, V. Analysis of Advanced Lithium-Ion Batteries for Battery Energy Storage Systems. In Proceedings of the 2018 IEEE International Conference on Environment and Electrical Engineering and 2018 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I CPS Europe), Palermo, Italy, 12–15 June 2018; pp. 1–6.
6. Liu, C.; Liu, L. Optimizing Battery Design for Fast Charge through a Genetic Algorithm Based Multi-Objective Optimization Framework. *ECS Trans.* **2017**, *77*, 257–271. [CrossRef]
7. O'Malley, R.; Liu, L.; Depcik, C. Comparative study of various cathodes for lithium ion batteries using an enhanced Peukert capacity model. *J. Power Sources* **2018**, *396*, 621–631. [CrossRef]
8. Helber, S.; Broihan, J.; Jang, Y.J.; Hecker, P.; Feuerle, T. Location Planning for Dynamic Wireless Charging Systems for Electric Airport Passenger Buses. *Energies* **2018**, *11*, 258. [CrossRef]
9. Liu, H.; Tan, L.; Huang, X.; Zhang, M.; Zhang, Z.; Li, J. Power Stabilization based on Switching Control of Segmented Transmitting Coils for Multi Loads in Static-Dynamic Hybrid Wireless Charging System at Traffic Lights. *Energies* **2019**, *12*, 607. [CrossRef]
10. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 3 March 2020).
11. Florea, B.C. Blockchain and Internet of Things data provider for smart applications. In Proceedings of the 2018 7th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 10–14 June 2018.
12. Wang, J.; Wang, Q.; Zhou, N.; Chi, Y. A Novel Electricity Transaction Mode of Microgrids Based on Blockchain and Continuous Double Auction. *Energies* **2017**, *10*, 1971. [CrossRef]
13. Khan, S.; Khan, R. Multiple Authorities Attribute-Based Verification Mechanism for Blockchain Mircogrid Transactions. *Energies* **2018**, *11*, 1154. [CrossRef]
14. Wu, J.; Tran, N.K. Application of Blockchain Technology in Sustainable Energy Systems: An Overview. *Sustainability* **2018**, *10*, 3067. [CrossRef]
15. Miller, D. Blockchain and the Internet of Things in the Industrial Sector. *IT Prof.* **2018**, *20*, 15–18. [CrossRef]
16. Son, B.; Lee, J.; Jang, H. A Scalable IoT Protocol via an Efficient DAG-Based Distributed Ledger Consensus. *Sustainability* **2020**, *12*, 1529. [CrossRef]
17. Odysseas, L.; Gialelis, J. An IOTA Based Distributed Sensor Node System. In Proceedings of the 2018 IEEE Globecom Workshops (GC Wkshps), Abu Dhabi, UAE, 9–13 December 2018; pp. 1–6.
18. Bartolomeu, P.C.; Vieira, E.; Ferreira, J. IOTA Feasibility and Perspectives for Enabling Vehicular Applications. In Proceedings of the 2018 IEEE Globecom Workshops (GC Wkshps), Abu Dhabi, UAE, 9–13 December 2018; pp. 1–7.
19. Ibáñez, L.; Simperl, E.; Gandon, F.; Story, H. Redecentralizing the Web with Distributed Ledgers. *IEEE Intell. Syst.* **2017**, *32*, 92–95. [CrossRef]
20. Merkle, R.C. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology (CRYPTO '87)*; Pomerance, C., Ed.; Springer: Berlin/Heidelberg, Germany, 1988; pp. 369–378.
21. Buterin, V. A Next Generation Smart Contract and Decentralized Application Platform. 2014. Available online: https://github.com/ethereum/wiki/wiki/White-Paper (accessed on 3 March 2020).

22. Szabo, N. Smart Contracts: Building Blocks for Digital Markets. 1996. Available online: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html (accessed on 3 March 2020).
23. Wang, S.; Ouyang, L.; Yuan, Y.; Ni, X.; Han, X.; Wang, F.Y. Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends. *IEEE Trans. Syste. Man Cybern. Syst.* **2019**, *49*, 2266–2277. [CrossRef]
24. O'Dwyer, K.J.; Malone, D. Bitcoin mining and its energy footprint. In Proceedings of the 25th IET Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014), Limerick, Ireland, 26–27 June 2014; pp. 280–285.
25. Wang, W.; Hoang, D.T.; Hu, P.; Xiong, Z.; Niyato, D.; Wang, P.; Wen, Y.; Kim, D.I. A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks. *IEEE Access* **2019**, *7*, 22328–22370. [CrossRef]
26. Popov, S. The Tangle. 2018. Available online: https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf (accessed on 3 March 2020).
27. Miao, Y.; Hynan, P.; von Jouanne, A.; Yokochi, A. Current Li-Ion Battery Technologies in Electric Vehicles and Opportunities for Advancements. *Energies* **2019**, *12*, 1074. [CrossRef]
28. Iclodean, C.; Varga, B.; Burnete, N.; Cimerdean, D.; Jurchiş, B. Comparison of Different Battery Types for Electric Vehicles. *IOP Conf. Ser. Mater. Sci. Eng.* **2017**, *252*, 012058. [CrossRef]
29. Piller, S.; Perrin, M.; Jossen, A. Methods for state-of-charge determination and their applications. *J. Power Sources* **2001**, *96*, 113–120. [CrossRef]
30. Rivera-Barrera, J.P.; Muñoz-Galeano, N.; Sarmiento-Maldonado, H.O. SoC Estimation for Lithium-ion Batteries: Review and Future Challenges. *Electronics* **2017**, *6*, 102. [CrossRef]
31. Zhang, R.; Xia, B.; Li, B.; Cao, L.; Lai, Y.; Zheng, W.; Wang, H.; Wang, W. State of the Art of Lithium-Ion Battery SOC Estimation for Electrical Vehicles. *Energies* **2018**, *11*, 1820. [CrossRef]
32. Prochazka, P.; Cervinka, D.; Martis, J.; Cipin, R.; Vorel, P. Li-Ion Battery Deep Discharge Degradation. *ECS Trans.* **2016**, *74*, 31–36. [CrossRef]
33. Weng, C.; Sun, J.; Peng, H. A unified open-circuit-voltage model of lithium-ion batteries for state-of-charge estimation and state-of-health monitoring. *J. Power Sources* **2014**, *258*, 228–237. [CrossRef]
34. Gottschalk, P.G.; Dunn, J.R. The five-parameter logistic: A characterization and comparison with the four-parameter logistic. *Anal. Biochem.* **2005**, *343*, 54–65. [CrossRef] [PubMed]
35. Brogan, J.; Baskaran, I.; Ramachandran, N. Authenticating Health Activity Data Using Distributed Ledger Technologies. *Comput. Struct. Biotechnol. J.* **2018**, *16*, 257–266. [CrossRef]
36. Feng, Q.; He, D.; Zeadally, S.; Khan, M.K.; Kumar, N. A survey on privacy protection in blockchain system. *J. Netw. Comput. Appl.* **2019**, *126*, 45–58. [CrossRef]
37. Jia, B.; Zhou, T.; Li, W.; Liu, Z.; Zhang, J. A Blockchain-Based Location Privacy Protection Incentive Mechanism in Crowd Sensing Networks. *Sensors* **2018**, *18*, 3894. [CrossRef]