Blockchain- and IPFS-Based Data Distribution for the Internet of Things

Simon Krejci, Marten Sigwart, Stefan Schulte

Blockchain- and IPFS-based Data Distribution for the Internet of Things

Simon Krejci, Marten Sigwart, and Stefan Schulte[0000−0001−6828−9945]

Distributed Systems Group, TU Wien, Austria
{s.krejci|m.sigwart|s.schulte}@dsg.tuwien.ac.at
https://www.dsg.tuwien.ac.at

Abstract. Distributing data in a tamper-proof and traceable way is a necessity in many Internet of Things (IoT) scenarios. Blockchain technologies are frequently named as an approach to provide such functionality. Despite this, there is a lack of concrete solutions which integrate the IoT with the blockchain for data distribution purposes.
Within this paper, we present a middleware which connects to IoT devices, and uses a blockchain to distribute IoT data with guaranteed integrity. Furthermore, the middleware also offers that data is distributed in real-time via a second channel. We implement our solution using the Ethereum blockchain and the InterPlanetary File System (IPFS).

Keywords: Internet of Things, Blockchain, Data Distribution, IPFS

1 Introduction

The Internet of Things (IoT) is a worldwide network of interconnected devices, which are able to process and store data, and in many cases provide sensor and actuator capabilities [3]. Blockchains are well-known as the underlying technology for cryptocurrencies like Bitcoin [16], but have also been named an enabling (and potentially disruptive) technology for application areas like supply chains [14], smart healthcare [12], or smart factories [8]. In a lot of these areas, it has been proposed to combine blockchains with IoT technology in order to store data from objects like sensor nodes in a tamper-proof, decentralized way, to process this data using smart contracts or off-chain, to distribute IoT data, and to provide services on top of this data [5, 7, 23].

Despite the manifold options to use blockchain technologies in the IoT, there are a number of challenges which complicate the wide-spread uptake of blockchains in this area. To start with, IoT devices are often hardware- or energy-constrained, and therefore do not provide the computational power necessary to participate in a blockchain network. Also, executing transactions and storing data in blockchains is expensive, which is not in line with the large number of interactions and the large amount of data to be distributed in typical IoT scenarios [30].

Therefore, one particular question is how lightweight IoT devices can interact with blockchains in order to exchange data. Current research focuses on

building specific blockchains for the IoT [6], with the explicit goal to provide more lightweight blockchain protocols. However, novel blockchain protocols do not only contribute to a significant fragmentation of the blockchain research and development field [24], but may also suffer from a smaller user base as well as a higher likelihood of bugs [17]. Therefore, it would be favorable if IoT devices could use existing, mature blockchains.

To achieve this, the work at hand presents a middleware for IoT applications, which facilitates the distribution of data via a blockchain, in case data integrity needs to be ensured. Because of the inherent overhead of using blockchain technologies for data distribution, the middleware explicitly facilitates data exchange also via a second channel. The second channel allows data distribution in (near) real-time, but does not provide the same integrity guarantees as the on-chain data exchange. We implement the middleware and test its performance in a fog setting, i.e., take into account that in many IoT scenarios, it is useful to host such a middleware at the edge of the network.

The remainder of this paper is organized as follows: In Sect. 2, we discuss the related work. In Sect. 3, we present the design and implementation of the middleware. Afterwards, we evaluate the middleware in Sect. 4. Finally, we conclude the paper in Sect. 5.

2 Related Work

The utilization of blockchains in the IoT has been proposed in many different papers. Typical IoT-related use cases are the utilization of blockchains to enable a tamper-proof log of IoT events, e.g., [21, 27], the management of access control data, e.g., [18], or the purchase of assets, such as IoT sensor data, e.g., [30, 31].

With regard to the storage and distribution of IoT data, Huh et al. [9] propose the utilization of Ethereum-based smart contracts in order to enforce policies for smart devices. Prybila et al. [21] present a solution to exchange data about distributed events in supply chains via the Bitcoin blockchain. Liu et al. [13] introduce a blockchain-based data integrity framework for IoT data, based on Ethereum smart contracts. Pešić et al. [19] present a high-level concept for applying blockchains in the IoT, following a blockchain-as-a-service model. Amongst other topics, the authors mention that this could be used for data sharing purposes. The concept has not been implemented so far. Shafagh et al. [25] propose a blockchain-based data management solution for the IoT, but also do not provide an implementation. However, such a proof-of-concept is presented by Sharma et al. [26] in their work on establishing a fog- and blockchain-based infrastructure for data exchange and computational tasks in the IoT.

Ali et al. [2] discuss the utilization of blockchains to enable data privacy in the IoT. Data items are stored in the IPFS, while the according data hashes are stored on the chain. The IPFS [4] is a Peer-to-Peer (P2P) distributed file system. It combines concepts of Distributed Hash Tables (DHTs), the Self-certifying File System (SFS), BitTorrent, and the version control system Git. The aim of the IPFS project is to connect all computing devices with one common file system.

An advantage of the distributed architecture of IPFS is that the nodes in the network do not have to trust each other and no user or node is privileged. This makes the IPFS an obvious choice to be used as a distributed file system together with a blockchain. Therefore, we also utilize IPFS in the work at hand.

Notably, Ali et al. [2] provide a decentralized data access model for the IoT, while in our work, the focus is on a middleware which helps to use blockchain capabilities in applications to distribute IoT-based data items. The data management part including the IPFS integration of data items presented by Ali et al. has not yet been implemented by the authors. Nevertheless, this concept comes closest to the work at hand.

To the best of our knowledge, none of the so-far discussed approaches to distribute IoT data via blockchains provides a proof-of-concept implementation for blockchain-based data distribution. However, this is done by Meroni et al. [15], who focus on artifact-driven process monitoring and apply both the Ethereum blockchain and IPFS for this.

A second important field of related work is the enhancement of current blockchain technologies to provide more lightweight solutions with regard to the computational power and energy consumption needed by a blockchain. One particular drawback of standard Proof-of-Work (PoW)-based blockchains is the extensive energy consumption [28]. This is especially an issue in the IoT, where devices may be battery-powered. Different specific-purpose blockchains for the IoT have been proposed to overcome this issue: Dorri et al. [6] present a hierarchical blockchain architecture consisting of multiple private immutable ledgers which IoT devices can connect to. A public overlay blockchain links the individual private ledgers. Zyskind et al. [32] present the Enigma platform, which uses a DHT to store data and to perform heavyweight, costly computational tasks off-chain. The IOTA Foundation [20] follows an entirely different approach for providing the benefits of distributed ledgers in the IoT. They deploy the so-called tangle, which does not organize transactions in blocks. Instead, individual transactions reference each other, forming a Directed Acyclic Graph (DAG).

As pointed out in Sect. 1, we believe that the utilization of already existing blockchain protocols is beneficial because of the more mature technology and the bigger user base, compared to novel protocols [17, 24]. Nevertheless, the conceptualization and implementation of IoT-specific blockchain protocols is surely a promising research direction.

3 Solution Architecture

As pointed out in Sect. 1, blockchain technologies can provide different benefits in IoT settings, including, but not limited to (i) exchanging computational power through decentralized smart contracts, (ii) the tamper-proof record of transactions on the blockchain for auditing or accounting purposes or simply for data distribution, and (iii) increasing the trust in distributed data [7, 11, 23]. Within the work at hand, we focus on blockchain-based data distribution in the IoT.

Besides the opportunities the blockchain may provide to the IoT, there are also challenges which need to be addressed. As pointed out above, the energy and resource demands of state-of-the-art blockchain protocols may be problematic for constrained IoT devices.

Latency is another challenge: In many cases, IoT-based applications need to react to real-world events in a timely manner. However, contemporary blockchains possess long block times, which means that it may take a certain amount of time until a transaction is added to a block: A frequently named example is Bitcoin's median inter-block time of 10 minutes, but even Ethereum's inter-block time of 13 to 20 seconds[1] might be too long for time-sensitive IoT applications. Notably, even a short inter-block time does not guarantee that a transaction is added to a block in due time, since a number of different factors play a role by when a transaction is actually added to a blockchain, e.g., the current load of the miners, the number of transactions in the transaction pool, transient connectivity issues, and the transaction fee a participant might be willing to pay [29].

Taking into account the potential benefits as well as challenges when using blockchain technologies, it should be foreseen that the blockchain does *not* become the only means to provide a particular functionality, e.g., data distribution, in the IoT. In order to achieve this, we conceptualize and implement a middleware which is able to collect data from IoT-based data sources like sensor nodes, and to distribute the data via two different channels.

The middleware provides the means to handle the demands of time-sensitive IoT applications, and to distribute data on- and off-chain. For this, the following functional requirements need to be fulfilled:

- Allow data collection from arbitrary IoT-based data sources: Naturally, the middleware should allow interested stakeholders to collect data from different types of IoT data sources, most notably sensor nodes.
- Access data in a time-sensitive manner: Data from the data sources should be accessible within a given, guaranteed timeframe, if necessary.
- Access data with guaranteed integrity: The data which is collected should be provided to stakeholders with guaranteed integrity. Notably, this functional requirement may contradict the need for time-sensitive data access.

3.1 Architecture

Fig. 1 gives an overview of the designed and implemented IoT-blockchain middleware. As it can be seen, the middleware provides the means to integrate arbitrary sensors via so-called sensor drivers. The middleware itself allows IoT clients to communicate with the sensors via two dedicated data distribution channels, i.e., the integrity channel and the real-time channel. The IoT client could be an arbitrary software interested in the IoT data and may get data via any of the two channels. Within the work at hand, we have implemented an IoT client which is able to measure different performance metrics and therefore provides the foundation for our evaluation (see Sect. 4).

[1] https://etherscan.io/chart/blocktime, as of January 2020

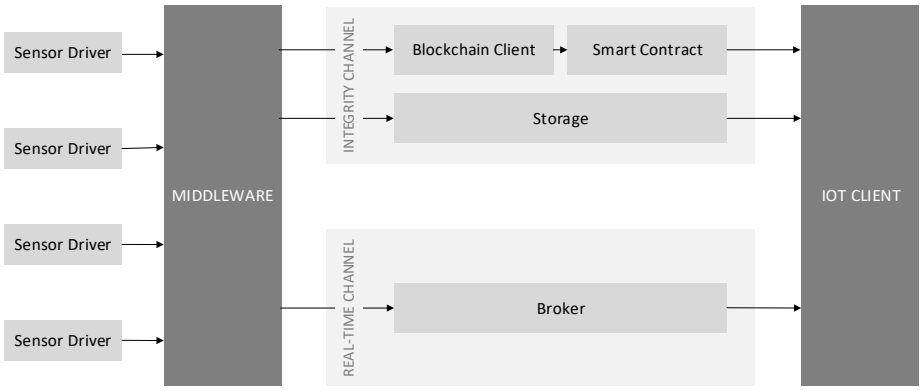


Fig. 1: Architecture Overview

Notably, the middleware may run in the cloud or close to the IoT sensors, i.e., at the edge of the network. A cloud-based middleware provides the benefit that additional computational resources are available. The latter option can be used in order to decrease the communication delay between the sensors and the middleware, to benefit from a distributed architectural approach which mirrors the distribution of data sources in the IoT, and to allow the utilization of lightweight, IoT-specific communication protocols, e.g., the Constrained Application Protocol (CoAP) [1]. The basic approach to use computational devices in the vicinity of the IoT-based data sources is also known as fog computing [22].

In the following subsections, we discuss the core components of our architecture, i.e., the sensor drivers and the middleware including the data distribution channels, in more detail.

Sensor Driver A sensor driver acts as the interface between an arbitrary IoT sensor and the middleware. Thus, it is responsible to collect the data from a specific sensor, with every sensor having its own driver. It may be the case that one sensor senses several phenomena, e.g., temperature *and* humidity. In such a case, one sensor driver collects the values for all supported phenomena.

The collected data is provided via an interface to the middleware. For this, the driver provides a phenomenon's data via IoT-suitable data distribution channels. In order to be able to differentiate the channels for different phenomena, each phenomenon gets its own channel. In the further course of this paper, a sensed phenomenon which is distributed via its own channel is referenced as *data source*.

Before the exchange between the driver and the middleware can be realized, the driver has to register at the middleware. The idea behind this process is that sensors can be added to the middleware dynamically, i.e., during system runtime. This avoids a hard coding of driver connections into the middleware and therefore adds flexibility. The registration workflow is made up from the steps depicted in Fig. 2:

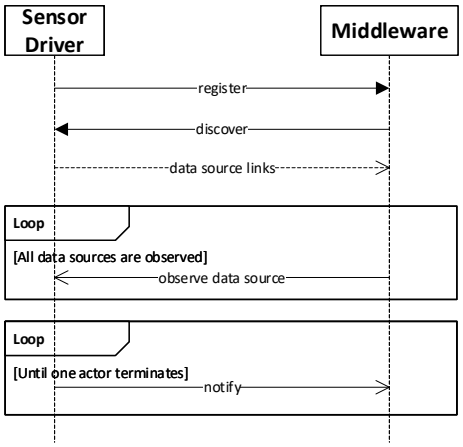


Fig. 2: Interactions between Sensor Drivers and Middleware

1. When registering a new sensor, a driver sends a registry request to the middleware containing its own address.
2. The middleware answers with a discovery request.
3. The driver responds with a list of data source links it offers to the middleware, i.e., one link per phenomenon.
4. For every data source link, the middleware sends an observe request. That means in case the observation is accepted by the driver, the middleware is notified if a new value is received from the sensor.
   While we do so far not provide the means to negotiate Service Level Agreements (SLAs) between sensor drivers (i.e., data providers) and IoT clients (i.e., the data sinks) via the middleware, we foresee that SLAs could be integrated. Also, we facilitate the payment of penalties by a data provider if an SLA is violated. For instance, in case a data provider promises to deliver a data update every n time units, the provider needs to pay a penalty fee if such a data update is not provided by a sensor in time.
5. The notify message from the sensor driver to the middleware contains both the actual notification as well as the sensor reading, i.e., the new value. Thus, a push-based data distribution scheme is provided. As long as the middleware and the driver are running, sensor data is delivered to the middleware.

Middleware & Data Distribution Channels The main functionality of the middleware is to receive data from the sensors (via the sensor drivers), and to distribute this IoT data via the two data distribution channels.

The integrity channel is used in order to distribute data items while guaranteeing their integrity. Since the blockchain is a tamper-proof distributed ledger, data integrity is ensured once data items are stored in it. However, storing complex data items in a blockchain may become expensive. To reduce the amount of data which is stored on-chain, our middleware only stores the hash of a data

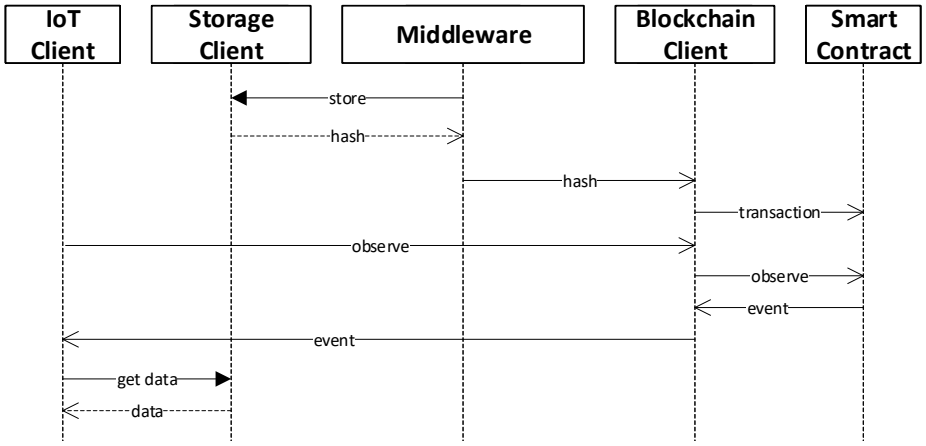


Fig. 3: Storage of Data via the Integrity Channel

item on the blockchain, while the data itself is saved in a content-addressable storage.

The basic workflow of storing data via the integrity channel is depicted in Fig. 3. As it can be seen, the middleware is able to store data via the storage client (not depicted in Fig. 1), which is in turn responsible for the storage process. Once the storage client has successfully persisted a data item, the hash value of the persisted data item is returned to the middleware.

The middleware forwards the hash to the blockchain client in order to store the hash value in a blockchain. As the figure shows, the actual storing of the hash is done via a smart contract (discussed below) running on the blockchain, i.e., by sending a transaction containing the hash as a parameter to a function in the smart contract. Once the function is successfully executed, i.e., the hash is stored in the blockchain, the smart contract generates an event, which can be observed by the blockchain users, e.g., the IoT client. With this hash value, the IoT client is now able to get the data from the storage (via the storage client).

The blockchain client and the storage client are loosely-coupled to the middleware. In our proof-of-concept implementation, we use Ethereum for the blockchain and IPFS for the content-addressable storage, and use the according clients provided by these systems (see below). However, the middleware could also be used with other storage solutions and blockchain protocols. In this case, it is necessary to integrate according clients into the middleware.

For the real-time channel, we use a publish/subscribe mechanism. Via this channel, interested clients are able to subscribe to different data sources and to receive data items in (near) real-time. As Fig. 1 shows, the middleware sends the data to a publish/subscribe broker, which is responsible for the publication of the data and the subscriptions of interested stakeholders. Notably, data items can be distributed via the real-time channel, and at the same time via the integrity

Listing 1.1: Update-Function in Smart Contract "IntegrityService"

```
1  event MeasurementUpdate(
2    address indexed sender,
3    uint8 function_code,
4    uint8 digest_length,
5    bytes32 digest
6  );
7
8  function update(uint8 function_code, uint8 digest_length, bytes32 digest,
9  bytes32 id_hash) onlyBy(client) public {
10   require(registered == true && penaltyPaid == false);
11   if (lastUpdates[id_hash] > 0
12   && block.timestamp > lastUpdates[id_hash] + maxDelay) {
13     penalty += 1;
14   }
15   lastUpdates[id_hash] = block.timestamp;
16   emit MeasurementUpdate(msg.sender, function_code, digest_length, digest);
17 }
```

channel. While this is only possible with a certain delay, this allows to verify the integrity of data items distributed via the real-time channel.

Smart Contract Listing 1.1 provides the core function of the implemented smart contract shown in Fig. 3. The excerpt checks the update rate of a data source, calculates penalties in case an SLA is violated, and emits an event to indicate the reception of new data. As programming language, Solidity is used. The function is called by the middleware (the *caller*) in order to provide the hashes of data items to an IoT client in a push-based manner. Notably, the smart contract is established between one particular data provider and one single IoT client (i.e., data consumer).

As it can be seen in Lines 2–5, the function gets a number of parameters, with *function_code*, *digest_length* and *digest* forming the hash used by IPFS to identify data items, and *sender* providing the address of the caller of the update function. Together, these variables constitute the event *MeasurementUpdate* (Lines 1–6). As the name implies, this event comprises a hash representing an update of a measurement from an IoT sensor.

The core functionality of the smart contract is provided by the function *update* (Lines 8–17). Apart from the already mentioned parameters, the function also is provided with the *id_hash*, which is the hash of the data source's ID. The identification of the data source is necessary to track the update rates of every single data source. The function modifier *onlyBy(client)* (Line 9) controls the access to the function.

In Line 10, we make use of Solidity's *require* construct to check conditions. In case the condition is not true, an exception is thrown and all changes on the state of the smart contract are undone. More concretely, we check if the caller of the function is registered, and if a penalty has already been paid out by the caller. As pointed out above, a penalty is due in case a data item (or rather

hash) is not delivered in time, and despite this having been specified between a data provider and a data sink (via sensor nodes and middleware) in an SLA.

Notably, more than one SLA violation may occur during data distribution, leading to multiple penalties. However, the total penalty fee is only disbursed once, i.e., when an authorized person calls an according function (not shown in the listing) in the smart contract. If the total penalty has been paid, no further updates are accepted, and the function is aborted. The reason for a defined ending of the contract is the limited validity of an agreement between the middleware and an IoT client, i.e., that a client is not able to get data items from a source for an indefinite amount of time.

Lines 11–12 are used to check if a penalty needs to be paid: When *update* is called and the last update timestamp plus a maximum allowed delay (defined in the SLA) is less than the current timestamp, i.e., an update is delayed, a penalty is calculated in line 13. In the current implementation, we make use of a fixed penalty fee, i.e., one *Wei*, which is the smallest unit of currency in Ethereum. The penalty could also be calculated in a more sophisticated way, e.g., following a linear penalty function, but since the penalty function is not in the focus of the work at hand, we opted for a simplified approach.

Line 15 sets the timestamp for the last update from a particular data source to the timestamp of the block where the hash of this data item has been added to. Notably, the time a transaction is added to a block depends on the chosen blockchain and other factors not under the control of the sensor driver (see above). This can be problematic, since a penalty may become due even though a data item has been delivered in time, but its hash having been added to a block too late. This is a general problem if blockchains are used in potentially time-sensitive settings. Solutions for this are part of our future work. At the moment, the allowed delay has to be chosen so that blockchain-inherent delays do not become an issue.

By calling the *emit*-command (Line 16), the event *MeasurementUpdate* is broadcast. Events exploit the logging facilities of the Ethereum Virtual Machine. These logging facilities can be used to create callbacks in applications (here: an IoT client) which listen to the events. The events are stored in the transaction's log whereas the logs are associated with the smart contract which emitted the events. Parameters of the event can have the attribute *indexed*. Thus, these parameters are not stored themselves, but it is possible to search for the parameters and filter them.

When a listening IoT client receives the event, the client is able to build the hash of a data item and retrieve the data item from IPFS.

3.2 Implementation

Our approach to IoT-blockchain integration has been realized in a proof-of-concept implementation, using Python for the sensor drivers and Java for the middleware and IoT client.

The communication protocols applied by the integrity channel are determined by the used blockchain and storage technologies, i.e., Ethereum and IPFS. For

the data exchange between the sensor drivers and the middleware and for the real-time channel, suitable communication protocols have been selected. Regarding the former, it is necessary to take into account typical communication issues in the IoT, e.g., potentially lossy links and the need for low-power communication [10]. Hence, we select CoAP as communication protocol [1].

Notably, CoAP offers the necessary request/response mechanism for the registration of new sensor drivers as well as the publish/subscribe mechanism used for value updates. Also, CoAP already provides mechanisms for device discovery and device registration, as needed by the middleware. However, CoAP's publish/subscribe mechanism is rather basic. Therefore, we select MQTT for the real-time channel [1].

We use Californium[2] as CoAP framework and Mosquitto[3] for the MQTT broker. For connecting the middleware as well as the IoT client to the Ethereum network, Web3j[4] and the Geth client[5] are used.

As pointed out above, the presented approach could be realized for other blockchain protocols and storage technologies, but for our proof-of-concept implementation, Ethereum and IPFS have been selected. To use different technologies, it is necessary to integrate according blockchain and storage clients into the middleware, and to implement a smart contract for the chosen blockchain.

The presented middleware is available as open source software at Github[6].

4 Evaluation

The goal of the evaluation is to test the performance of the implemented solution. We assume that the middleware runs on an edge device, i.e., we apply a fog-based system architecture. We use a single-board computer, i.e., a Raspberry Pi 3 Model B, as a typical IoT edge device.

To test the performance, we measure message delays using the real-time and integrity channels, i.e., the delay from the occurrence of a new data item in a sensor driver to the point of time it is received by a user (here: the IoT client).

4.1 Evaluation Setup

In order to execute performance tests, we have implemented a *virtual driver*, i.e., a sensor driver which has no connection to a physical sensor, and therefore represents a number of simulated *data sources* which regularly emit data items. Hence, the virtual driver generates artificial messages with hard-coded values. The amount of data sources and the amount of sent messages can be user-specified in the virtual driver, allowing us to use the virtual driver in order to execute reproducible performance tests. To be able to test varying loads, we use

[2] https://www.eclipse.org/californium/
[3] https://mosquitto.org/
[4] https://docs.web3j.io/
[5] https://geth.ethereum.org/
[6] https://github.com/mcmon-dev/iot-middleware

virtual drivers with 2, 7, 11, and 22 data sources in the evaluation setup. In addition, we use GrovePi+ in connection with the Raspberry Pi to also evaluate the setup with a real-world sensor which emits two different phenomena (i.e., represents two data sources).

For the Raspberry Pi, we use Raspbian 8.0 as operating system. The IoT client is a desktop application, running on a standard desktop PC. As blockchain, we make use of the Ethereum test network Ropsten[7]. In order to mitigate the influence of varying network loads, IPFS and the real-time channel are installed in a local network. The experiments are repeated three times to further mitigate varying loads in the local network and on the nodes.

During each experimental run, each (physical and virtual) data source emits a data item every 5 seconds. This is repeated 60 times, leading to an overall duration of 5 minutes per experimental run.

We use different statistical metrics, i.e., median, mean, quantiles, and standard deviations in order to assess the evaluation results. In addition, we make use of notched boxplots to compare the data.

4.2 Results

Table 1: Delays in the Real-Time Channel (in ms)

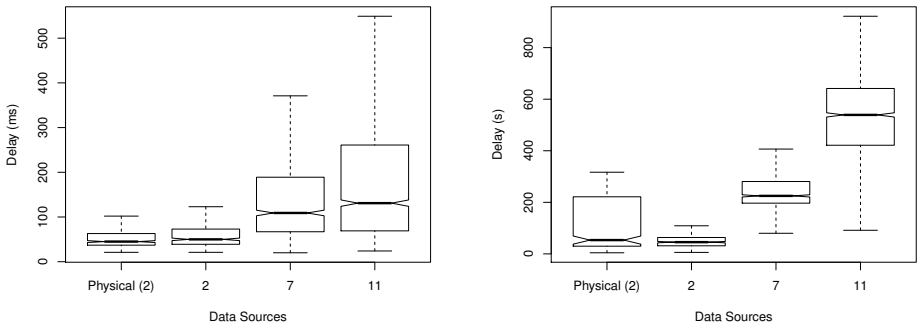
	Min	Q1	Median	Mean	Q3	Max	σ
Phy (2)	21.00	37.00	45.00	86.93	63.00	1785.00	169.234
2	21.00	39.00	50.00	96.26	73.00	3238.00	258.605
7	20.00	67.00	109.00	200.90	189.00	1856.00	268.4895
11	24.00	69.00	131.00	253.80	261.00	2407.00	350.524
22	26.00	129.00	321.00	2628.80	864.50	146121.00	15662.40

Tab. 1 provides an overview of the delays for the real-time channel, i.e., the MQTT-based data distribution, which does not provide data integrity guarantees, for the 2 physical data sources (Phy (2)) and the 2, 7, 11, and 22 virtual data sources. The numbers provide the minimum, Q1, median, mean, Q3, and maximum for the abovementioned three evaluation runs. Fig. 4a visualizes the delays of the real-time channel[8], but does not show a boxplot for the 22 data sources. The reason for this is the high increase of the median for 22 data sources, which would make the differences between the boxplots very difficult to identify.

As it can be seen in Fig. 4a, the median is increasing with the amount of data sources. Also, it can be seen that the boxes for the 2 physical data sources and the 2 virtual data sources provide similar results, indicating that the virtual driver resembles the performance of the physical data sources in a sufficient way. Another observation is that the upper whisker of the boxplot increases with the number of data sources, while the lower whisker is almost stable.

[7] https://ropsten.etherscan.io

[8] With the boxplot notch indicating a 95% confidence interval.



(a) Real-time Channel (b) Integrity Channel

Fig. 4: Delays

The mean numbers shown in Table 1 confirm the results from the median and boxplot analysis. Interestingly, the maximum numbers for 2 virtual data sources are a significant outlier, but are compensated by other evaluation runs, i.e., do not influence the overall numbers significantly.

It should be noted that the usage of 22 data sources led to a quite high number of missing values for the real-time channel. In fact, roundabout 18% of the data updates were lost in the real-time channel when using 22 data sources. This indicates that this number of data sources might put a too high load on the Raspberry Pi, since for the other three test scenarios, no data items got lost. This shows that the implemented solution is suitable only for a limited number of data sources, which can be traced back to the limited computational resources of the used IoT device, i.e., the Raspberry Pi, and the high resource demand of some of the used APIs, especially the IPFS client.

Tab. 2 shows the numbers for the integrity channel, i.e., where the data hashes are stored in the blockchain, and the actual data items in IPFS. Not surprisingly, this leads to a large overhead, since it takes time until a transaction is added to the blockchain. Hence, the numbers in Tab. 2 (as well as Fig. 4b) are given in seconds, while the numbers for the real-time channel in Tab. 1 and Fig. 4a are given in milliseconds.

Table 2: Delays in the Integrity Channel (in s)

	Min	Q1	Median	Mean	Q3	Max	σ
Phy (2)	4.302	29.858	53.499	99.600	221.404	316.739	97.614
2	5.739	31.448	45.538	51.359	63.503	143.668	29.289
7	20.15	197.19	224.95	238.71	280.20	513.61	115.634
11	52.11	422.03	539.03	526.52	641.58	1180.60	217.532
22	47.45	723.11	992.98	1172.06	1503.20	3409.93	694.025

As it can be seen in Tab. 2 as well as Fig. 4b, similar to the real-time channel, the median increases with the number of data sources. However, for the integrity channel, the median increases stronger. This indicates that the integrity channel scales not as well as the real-time channel. This is also confirmed by the number of missing data items for the 22 virtualized sensor, which is about 33% and therefore significantly higher than for the real-time channel.

4.3 Discussion

In general, the real-time channel provides acceptable delays in case the number of data sources is not too high, i.e., with up to 11 data sources, the mean delay is roundabout 254 ms, which is acceptable for many IoT scenarios.

As originally assumed, the usage of blockchain technologies in the integrity channel leads to a very high overhead, with the mean delay being roundabout 527 seconds in the case of 11 data sources. Hence, the blockchain-based data distribution should not be used in scenarios where latency is critical. For scenarios where both data integrity and low delay times are needed, it is necessary to develop novel blockchain technologies (see Sect. 2).

However, it should be noted that the presented solution allows to distribute data in real-time, and to store hashes of these data items also in the blockchain, thus allowing to validate the data integrity afterwards.

Due to space constraints, we do not discuss the confirmation delays caused by the blockchain. However, we have made such measurements, which show that the blockchain delays do not differ significantly for the different evaluation scenarios. Therefore, the abovementioned delays for the integrity channel are influenced in a similar vein by the blockchain confirmation delays, and the increase in the overhead (compared to the real-time channel) can be traced back to the restricted amount of resources the hosting node (i.e., the Raspberry Pi) provides.

5 Conclusions

Blockchain technologies are frequently named as an enabler for data integrity in IoT scenarios, as well as facilitator of other functionalities in the IoT. Despite this, there is still a lack of proof-of-concepts which show how blockchain technologies can be used to distribute data between data sources and data sinks in the IoT. Therefore, within this paper, we have proposed a middleware which is able to collect data from IoT sensors and to distribute data via two different channels. The first data distribution channel utilizes IPFS as data storage, and stores data hashes within an Ethereum blockchain. The second channel is based on MQTT and therefore allows to distribute data in a timely manner. We have implemented the middleware and evaluated it with regard to its capability to be run at the edge of the network, i.e., on a single-board computer like a Raspberry Pi, and the overhead introduced by the usage of a blockchain for data distribution purposes.

In our future work, we want to extend the presented middleware in order to investigate further research questions in the blockchain/IoT realm. As has already been stated above, it would be interesting to discuss the integration of full-fledged SLAs including SLA negotiations, while also allowing more sophisticated penalty schemes. Especially, the applied penalty scheme should be extended by a mechanism to reflect the blockchain-inherent transaction delays, i.e., that a penalty does not become due because of such a delay.

Acknowledgements. The work presented in this paper has received funding from Pantos GmbH within the TAST research project.

References

1. Al-Fuqaha, A.I., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys and Tutorials **17**(4), 2347–2376 (2015)
2. Ali, M.S., Dolui, K., Antonelli, F.: IoT Data Privacy via Blockchains and IPFS. In: Seventh International Conference on the Internet of Things. pp. 14:1–14:7. ACM (2017)
3. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: A survey. Computer Networks **54**(15), 2787–2805 (2010)
4. Benet, J.: IPFS – Content Addressed, Versioned, P2P File System (DRAFT 3). CoRR **abs/1407.3561** (2014)
5. Christidis, K., Devetikiotis, M.: Blockchains and Smart Contracts for the Internet of Things. IEEE Access **4**, 2292–2303 (2016)
6. Dorri, A., Kanhere, S.S., Jurdak, R., Gauravaram, P.: LSB: A Lightweight Scalable Blockchain for IoT security and anonymity. Journal of Parallel and Distributed Computing **134**, 180–197 (2019)
7. Fernández-Caramés, T.M., Fraga-Lamas, P.: A Review on the Use of Blockchain for the Internet of Things. IEEE Access **6**, 32979–33001 (2018)
8. Fernández-Caramés, T.M., Fraga-Lamas, P.: A Review on the Application of Blockchain to the Next Generation of Cybersecure Industry 4.0 Smart Factories. IEEE Access **7**, 45201–45218 (2019)
9. Huh, S., Cho, S., Kim, S.: Managing IoT devices using blockchain platform. In: 19th International Conference on Advanced Communication Technology. pp. 464–467. IEEE (2017)
10. Ko, J., Terzis, A., Dawson-Haggerty, S., Culler, D.E., Hui, J.W., Levis, P.: Connecting Low-Power and Lossy Networks to the Internet. IEEE Communications Magazine **49**(4), 96–101 (2011)
11. Kshetri, N.: Can Blockchain Strengthen the Internet of Things? IT Professional **19**(4), 68–72 (2017)
12. Li, M., Xia, L., Seneviratne, O.: Leveraging Standards Based Ontological Concepts in Distributed Ledgers: A Healthcare Smart Contract Example. In: 2019 IEEE International Conference on Decentralized Applications and Infrastructures. pp. 152–157. IEEE (2019)
13. Liu, B., Yu, X.L., Chen, S., Xu, X., Zhu, L.: Blockchain Based Data Integrity Service Framework for IoT Data. In: 2017 IEEE International Conference on Web Services. pp. 468–475. IEEE (2017)

14. Lu, D., Moreno-Sanchez, P., Zeryihun, A., Bajpayi, S., Yin, S., Feldman, K., Kosofsky, J., Mitra, P., Kate, A.: Reducing Automotive Counterfeiting using Blockchain: Benefits and Challenges. In: 2019 IEEE International Conference on Decentralized Applications and Infrastructures. pp. 39–48. IEEE (2019)
15. Meroni, G., Plebani, P., Vona, F.: Trusted Artifact-Driven Process Monitoring of Multi-party Business Processes with Blockchain. In: BPM Blockchain and Central and Eastern Europe Forum. LNBIP, vol. 361, pp. 55–70. Springer (2019)
16. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2008), Whitepaper
17. Nofer, M., Gomber, P., Hinz, O., Schiereck, D.: Blockchain. Business & Information Systems Engineering **59**(3), 183–187 (2017)
18. Novo, O.: Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT. IEEE Internet of Things Journal **5**, 1184–1195 (2018)
19. Pešić, S., Tošić, M., Iković, O., Radovanović, M., Ivanović, M., Bošković, D.: Conceptualizing a Collaboration Framework Between Blockchain Technology and the Internet of Things. In: 20th International Conference on Computer Systems and Technologies. pp. 56–61. ACM (2019)
20. Popov, S.: The tangle (2017), IOTA Whitepaper v1.3
21. Prybila, C., Schulte, S., Hochreiner, C., Weber, I.: Runtime Verification for Business Processes Utilizing the Bitcoin Blockchain. Future Generation Computer Systems (2020)
22. Puliafito, C., Mingozzi, E., Longo, F., Puliafito, A., Rana, O.: Fog Computing for the Internet of Things: A Survey. ACM Transactions on Internet Technology **19**(2), 18:1–18:41 (2019)
23. Reyna, A., Martín, C., Chen, J., Soler, E., Díaz, M.: On blockchain and its integration with IoT. Challenges and opportunities. Future Generation Computer Systems **88**, 173–190 (2018)
24. Schulte, S., Sigwart, M., Frauenthaler, P., Borkowski, M.: Towards Blockchain Interoperability. In: BPM Blockchain and Central and Eastern Europe Forum. LNBIP, vol. 361, pp. 3–10. Springer (2019)
25. Shafagh, H., Burkhalter, L., Hithnawi, A., Duquennoy, S.: Towards Blockchain-based Auditable Storage and Sharing of IoT Data. In: 2017 Cloud Computing Security Workshop. pp. 45–50. ACM (2017)
26. Sharma, P.K., Chen, M.Y., Park, J.H.: A Software Defined Fog Node Based Distributed Blockchain Cloud Architecture for IoT. IEEE Access **6**, 115–124 (2017)
27. Sigwart, M., Borkowski, M., Peise, M., Schulte, S., Tai, S.: Blockchain-based Data Provenance for the Internet of Things. In: 9th International Conference on the Internet of Things. pp. 15:1–15:8. ACM (2019)
28. Tschorsch, F., Scheuermann, B.: Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies. IEEE Communications Surveys and Tutorials **18**(3), 2084–2123 (2016)
29. Weber, I., Gramoli, V., Ponomarev, A., Staples, M., Holz, R., Tran, A.B., Rimba, P.: On Availability for Blockchain-Based Systems. In: 36th IEEE Symposium on Reliable Distributed Systems. pp. 64–73. IEEE (2017)
30. Wörner, D., von Bomhard, T.: When your sensor earns money: exchanging data for cash with bitcoin. In: The 2014 ACM Conference on Ubiquitous Computing Adjunct. pp. 295–298. ACM (2014)
31. Zhang, Y., Wen, J.: The IoT electric business model: Using blockchain technology for the internet of things. Peer-to-Peer Networking and Applications **10**(4), 983–994 (2017)
32. Zyskind, G., Nathan, O., Pentland, A.: Enigma: Decentralized Computation Platform with Guaranteed Privacy. CoRR **abs/1506.03471** (2015)