A Fog Computing Architecture to Share Sensor Data by Means of Blockchain Functionality

Hendrik L. Cech
Fakultät für Informatik
Technische Universität München
Boltzmannstr. 3, D-85748 Garching, Germany
Email: hendrik.cech@tum.de

Marcel Großmann, Udo R. Krieger
Fakultät WIAI
Otto-Friedrich-Universität
An der Weberei 5, D-96047 Bamberg, Germany
Email: {marcel.grossmann|udo.krieger}@uni-bamberg.de

***Abstract*—Considering rapidly evolving Internet-of-Things services such as smart buildings, smart homes or secure e-health applications, the fog computing concept provides a technical basis for innovative application scenarios of the blockchain technology. We develop an enhancement of the fog computing architecture HCL-BaFog by the blockchain functionality to collect and securely share sensor data. They may arise from private IoT applications such as ambient assisted living which incorporate the need of an immutable, local storage and a sharing of protected real-time sensor data monitored at patients' or people's homes. Our proof-of-concept employs the fully virtualized functionality of a permissioned blockchain and a network of fog computing nodes and utilizes the container orchestration and management system Docker and the MultiChain framework. We investigate some basic performance metrics of our storage-and-sharing approach using a test bed of Raspberry Pi SBCs. Finally, some conclusions on the developed fog computing architecture with its integrated blockchain functionality are discussed and its current limitations are revealed.**

***Index Terms*—Fog computing, Data sharing, Blockchain, MultiChain**

I. Introduction

The *Internet-of-Things* (IoT) describes a fundamental paradigm shift enhancing previously analog devices with effective computing and networking capabilities. It has enabled new machine-to-machine as well as device-to-network communication patterns among billions of smart devices in recent years. The related physical objects or (virtual) logical objects called *things* are capable of being identified and integrated into computing platforms and communication networks and arise in rapidly evolving service scenarios like smart cities, smart homes, or innovative e-health applications. These things have already become the interactive peers of modern high-speed wireless and wired communication infrastructures such as evolving 5G networks (cf. [2], [3], [20]). They generate streams of real-time data by their associated sensor components and potentially react to control commands by actuator components (cf. [5]). From an engineering perspective, an IoT ecosystem faces serious challenges such as long-term stability, interoperability, as well as severe privacy and security issues (cf. [7], [32]). Integrating billions of small sensors, actuators, and data analysis entities provides great challenges to classical cloud computing. Today, the Internet-of-Things requires a fundamental transition of many commonly used paradigms towards new efficient protocols and processing as well as data storage concepts. To cope with these challenges of innovative, rapidly evolving IoT applications, the new paradigm of fog computing has been developed in recent years (cf. [11]). Its challenging objective is to establish in a secure and efficient way the well-determined concepts and services of cloud computing such as IaaS and PaaS at a continuum between the data centers of a cloud and the edge of wireless networks based on WLAN and 5G technologies with their associated next generation optical backbones (cf. [1], [29]).

Considering the diverse IoT application areas and economically relevant vertical market segments, the smart buildings, smart homes, and protected e-health sector are of utmost importance (cf. [2], [27]). In particular healthcare in aging societies may create challenging ambient assisted living scenarios. In these environments a fog computing approach integrating blockchain functionality appears to be very useful regarding the local preprocessing and analysis of real-time sensor data monitored at patients' or old people's homes. It should be combined with an immutable, local storage and sharing approach of these protected data which can partly replace the submission of the data to the remote cloud computing sites. In this way it can enhance the required immutability and privacy of health records facilitating the regulatory conformance of the applied technology, e.g. with regard to the General Data Protection Regulation of EU or requirements in USA (cf. [24]). In this respect the fog computing concept provides a technical basis for innovative application scenarios of the blockchain technology in a protected environment with well-known sensor, actuator, and fog computing nodes as interacting peers (see Figure 1, cf. [11], [14]).

In our ongoing research we focus on the integration of the blockchain functionality into an existing low-cost fog computing environment called HCL-BaFog (cf. [14]). We want to investigate the most efficient way how one can effectively partition the required functional blocks of both concepts and integrate the latter as a layered protocol structure into a software architecture with container virtualization. This approach constitutes a first step towards a realization of blockchain technologies as Function-as-a-Service system. Following this line of reasoning, we investigate in this paper how the blockchain functionality can be exploited to securely store and share

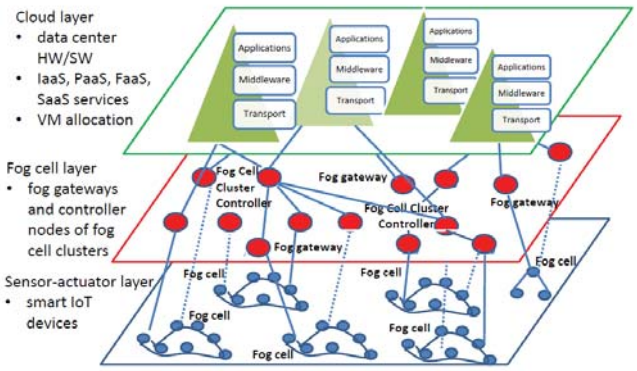

Fig. 1. Architecture of the fog computing environment.

sensor data among more advanced fog computing nodes. Consequently, we develop an enhancement of the fog computing platform HCL-BaFog to collect and securely share sensor data arising from such private IoT applications. It can be incorporated into an available scalable stack of interconnected ARM based SBCs (cf. also [16], [17]). The new proof-of-concept uses the fully virtualized functionality of a blockchain and a network of efficient fog computing nodes and utilizes the container orchestration and management system Docker with its Swarm mode and the MultiChain framework (cf. [21]). We demonstrate the feasibility of our data sharing and management approach with its secure access, authentication, and data integrity policies using a test bed of Raspberry Pi SBCs with their 64-bit ARM processor architecture.

The paper is organized as follows. In Section 2 we present some foundations on the blockchain technology, evaluate its functionality to share sensor data and select a related open source framework. In Section 3 we first sketch our fog computing platform HCL-BaFog and describe its enhancements to incorporate the blockchain functionality. Then we present a prototype implementation of our new fog computing node. A first assessment of its performance is discussed, too. Finally, conclusions on the developed fog computing approach with integrated blockchain functionality, its current limitations, and some useful extensions are discussed.

II. Storing Sensor Data of IoT Applications by Means of Blockchain Functionality

The Internet-of-Things combined with the paradigm of fog computing will provide a fast growing basis for new, rapidly evolving application scenarios of the blockchain technology (cf. [1], [2], [28]). A blockchain is a decentralized database which is distributed via replication among its contributing peers. Data are not directly updated, but a collection of change records represented by transactions are appended as aggregated entities called *blocks*. All participating peers can validate the state changes and must agree on the state of those data stored in the associated data plane (cf. [9], [12]).

A. Foundations of Blockchain Technology

A decade ago, the blockchain technology called *Bitcoin* has been introduced to realize a truly decentralized digital currency for its users in a peer-to-peer (P2P) interaction mode without the need of a centralized authority (cf. [25]). The central primitive of Bitcoin is determined by the concept of a *transaction*. A transaction has at least one input which can be considered as the digital currency spent by the transaction. Its counterpart is given by a transaction output which is addressed to another peer of the Bitcoin overlay and transfers a specified amount of the digital currency. Transactions are considered valid if a number of domain-specific constraints are met, e.g., that the sum of the currency specified in the outputs does not exceed the amount given by the inputs [12]. Regarding Bitcoin, the category of an *asset* is not a separate entity but represented by transactions. It means that the asset referenced by the input is actually an output of an earlier transaction. At each point in time, a user's current digital balance is equal to all transaction outputs that are addressed to her, and that she did not yet transfer to other users. The set of transactions satisfying those conditions are called *unspent transaction outputs (UTXO)*. User balances are calculated summing up UTXO's grouped by their addresses. Transactions are not ordered individually but in bigger aggregated units created around the same time. In addition to a list of transactions, these so-called *blocks* contain a reference to the most recent block at that time. This reference is realized including a hash value of the preceding blocks. Similar to the transactions, the back-references realize a specific order among these blocks. Thus, the naming *blockchain* of this technology originates from this chaining of blocks.

Bitcoin supports a simple stack-based language, which allows a limited degree of programmability. Fewer than 100 operations are implemented that can support basic cryptographic calculations. A script, usually comprising only a few instructions, may be attached to each transaction output to offer an additional specific functionality. Normally, the ownership of transactions is controlled using public-key cryptography. Users are identified by a public key and assets are locked such that only the owner's private key is able to unlock and authorize the transfer of those assets. To send an asset, a new transaction is created and its input references that asset. The transaction output references the new owner's public key and the whole transaction is signed by the asset owner's private key.

The distributed control model of a blockchain organizes the interworking of a set of identifiable, interconnected peers in such a transaction-oriented system architecture that includes three basic features:

- *Decentralized architecture:* The blockchain functionality is executed by a set of independent peers that can dynamically join and leave the underlying P2P network. These peers can be maintained by different entities that do not even need to know their identities or intentions.
- *Fundamentally anonymous interaction:* The functionality of a public blockchain allows peers to participate in

the P2P network without identification. Read access is not recorded and writing data to the storage part of a blockchain is supported by a pseudonymity concept. This feature is a desirable fundamental property to guard the privacy of the interacting peers.
- *Stability guarantees:* Participanting peers of a blockchain can store data in the storage plane of its P2P network and be assured that these data will not be manipulated, even if they do not trust other peers.

Peers of a blockchain share new transactions with the whole P2P overlay and each peer stores all transactions that were recently created in a block, before restarting to gather transactions for the next block. The goal of negotiating a common history of blocks realizes a *consensus* problem. Nguyen and Kim [26] divide the consensus mechanisms of a blockchain into *proof-based* and *vote-based* schemes. Bitcoin, for instance, applies an expensive proof-based consensus mechanism whereas the Byzantine Fault Tolerant (BFT) replication, i.e., a solution to the *Byzantine generals problem* subject to malicious nodes, constitutes a vote-based scheme (cf. [23], [30]). The algorithms Practical Byzantine Fault Tolerance (PBFT) [10] and BFT-SMART [6] are popular representatives of this latter class. A related but weaker class of protocols is given by crash-tolerant protocols that guard only against crashed but honest nodes. An extensive overview of these different types of consensus mechanisms can be found in [26, Table 3].

The choice of an efficient consensus mechanism coincides with the access model of the peer in a blockchain. Bitcoin is the prime example of an open, *unpermissioned*, i.e. *permissionless*, network. Access to the data in the P2P network or participation in the consensus process are not restricted. An alternative is given by a *permissioned* blockchain, where an access to the network requires authentication and authorization. The latter can be realized by well-known techniques, e.g., public-key cryptography, shared secrets, or white listed IP addresses.

Inspired by the digital currency application Bitcoin, blockchains are often called distributed ledger[s] [9] since they are storing the distribution of assets. Furthermore, a copy of the ledger is maintained by each peer of the underlying network. From a technical point of view, the ledger is considered as a state transition system. The state represents the allocation of units of a digital currency, the transition function takes a state and a transaction, applies the transaction to the state, and yields a new, modified state (cf. [8]). In this regard it is obvious that a blockchain could be a useful means to store and represent distinct types of data that are available for a certain level of public sharing.

B. Evaluating Blockchain Functionality to Share Sensor Data

Considering general IoT usage scenarios, an application of the blockchain technology has already been examined in the literature (cf. [13], [24], [28], [31], [33]). Conoscenti, Vetrò, and De Martin [13], for instance, have classified approaches that are relevant to IoT scenarios into four categories: data storage management, trade of goods and data, identity management, and rating systems. Wörner and von Bomhard [31] have proposed a system where sensors deliver data to buyers along the blockchain and are paid by its digital currency. Their proposal relates to data storage management and trade of goods and data, resulting in severe scaling issues. Shafagh et al. [28] have presented a broad vision on data storage management and identity management with regard to the access control by a blockchain. Blockchain nodes have the task to control the access of clients by enforcing rules stored in the blockchain. Zyskind et al. [33] have described a blockchain system aiming to improve scalability and privacy issues. Their system is split into a blockchain part hosting public data and a private off-chain part.

To explore the application of the blockchain technology to IoT scenarios in the context of a fog computing environment, a suitable blockchain framework must be chosen first and its underlying architecture and APIs must suit the needs of the considered IoT application (cf. [12], [13]). To maintain a network of involved IoT devices, one must operate stable fog nodes with the blockchain functionality to serve its peers within the associated network. Operational logical management entities could coordinate the operation of such a blockchain and, in particular, decide on new membership requests within a given organizational context of an IoT scenario. Thus, a permissioned blockchain with a managed membership is preferred. It does not face the challenges of open networks, such as Sybil attacks, and can exploit this feature to choose a less expensive and more powerful consensus mechanism than a proof-based approach such as Bitcoin's proof-of-work (PoW) scheme (cf. [25], [26]). However, the system must still provide features to enforce security and privacy. While blockchain nodes authenticate themselves to participate in the network, required trust between parties should also be limited as much as possible. For example, a single organizational unit should not be able to allow new peers to participate in the consensus process without the consent from the majority of other peers. In this regard a mechanism for confidential data exchange between selected peers of a blockchain constitutes a central requirement. Regarding our fog-computing approach integrating this blockchain functionality, an implementation on top of an inexpensive ARM processor environment using open source frameworks should be possible in addition.

In this respect we have performed a systematic survey of published blockchain frameworks. It has excluded proprietary and no longer maintained systems such as Hyperledger Iroha, Corda, Hydrachain, Openchain, and Chain Core. Finally, three blockchain frameworks shown in Table I, namely Hyperledger Fabric [4], Quorum, and MultiChain [15], appeared to be suitable and were closely examined. In conclusion, MultiChain [15] has been chosen as basic blockchain framework to enhance our fog computing architecture. The reason is that its concept is very close to the original blockchain model of Bitcoin with relevant changes to adapt it to a permissioned environment. Useful abstractions such as streams which support publish, subscribe and query functionality are built

TABLE I
COMPARING RELEVANT CONTENDERS' ATTRIBUTES OF THE SELECTED OPEN SOURCE BLOCKCHAIN FRAMEWORKS.

Attributes	Fabric	Quorum	MultiChain
General properties	Apache-2.0 licensed, ∼5700 stars on GitHub, written in Go, originating from IBM.	LGPL-3.0 licensed, ∼2400 stars on GitHub, written in Go, forked from *go-ethereum*.	GPL-3.0 licensed, ∼400 stars on GitHub, written in C++, based on Bitcoin and compatible with its RPC protocol.
Consensus protocol	Pluggable consensus backend, using Kafka as only current production-ready option.	Pluggable consensus backend, applying the PBFT-inspired *Istanbul BFT* algorithm as most promising option.	Round-robin vetting, i.e., each node proposes an equal number of blocks by taking turns.
Application features	Potential use of a variety of general-purpose languages to write *chaincode*.	Supporting Ethereum virtual machine (EVM) smart contracts.	Programmability limited to Bitcoin-like scripts, supporting *stream*-abstraction on top of transactions with a permission system on address granularity level.
Confidentiality features	Access can be configured to *channels* which are independent blockchains.	Supporting *private transactions/contracts*, handling encryption and storage by nodes in a P2P-fashion, exposing users' plain-text to nodes.	Proposing an encryption scheme based on its stream feature with additional support for encrypted off-chain storage.

on top of the transaction primitive. Furthermore, MultiChain has an elaborate permission system based on capabilities of blockchain addresses. For instance, only certain addresses can be allowed to publish data to a stream or to create new transactions. At the time of writing, the major new version 2.0 was about to be released with a bunch of new important features where the integrated off-chain storage represents the most interesting item.

III. A FOG COMPUTING PLATFORM SUPPORTING SECURE DATA SHARING AND ACCESS MANAGEMENT

Fog computing is a distributed computing concept that incorporates a collaborative multitude of end-user clients or near-user edge devices to execute a substantial amount of computation, storage, communication, and control, as well as monitoring and management functionality (cf. [11]). It disseminates virtualized computing, storage, communication and control services closer to the end users along the Cloud-to-Things continuum using edge-driven data and control planes.

A. A Fog Computing Architecture Based on Container Virtualization and SDN Functionality

We have developed the versatile fog gateway HCL-BaFog with its separated data and control planes shown in Figure 2 (cf. [14]). It is exploiting a lightweight Linux container virtualization provided by the Docker framework (cf. [21], [35]). Its technical basis is realized by single board computer (SBC) systems based on a 32-bit or 64-bit multi-core ARM processor architecture with Hypriot Cluster Lab (HCL) as Linux operating system (cf. [16], [36]). Both components can provide the required hardware and software functionality of a powerful, low cost fog computing node at the first aggregation level of the underlying distributed infrastructure within fog cells (see Fig. 1). It includes the services, middleware and protocol stacks to interconnect sensor or actuator entities and appliances of a modern Internet-of-Things environment (see Fig. 2, cf. [2]). This fog gateway has been recently enhanced by SDN functionality that is incorporated in an Open vSwitch (OVS). We have shown that the programmability of the SDN control plane offered by an SDN controller with the ONOS

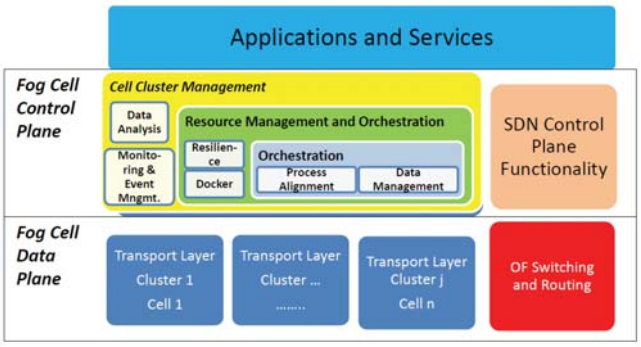

Fig. 2. Hierarchical protocol layering of a fog gateway.

framework can be employed, too. Its development can be effectively supported by a continuous integration approach and corresponding deployment platforms (cf. [19]). This fog gateway can be integrated into a hierarchically structured, enhanced cloud computing architecture as shown in Figure 1. It can use the powerful orchestration and management functionality of Docker Swarm mode to reliably distribute fully virtualized functions among nodes working in either a manager or worker mode subject to high availability constraints. Regarding a private healthcare scenario, for instance, such a fog computing architecture with its distributed processing and storage capabilities enables us to store and process the monitored data flows arising from medical sensors in an extensive neighborhood of interconnected sites. The integration of blockchain functionality can enforce an immutable storing of similar health and treatment profiles of patients and a public sharing and control of this information by the medical experts or healthcare personnel.

B. Architecture of a Fog Node With Blockchain Functionality

We have developed a proof-of-concept of our proposed fog computing architecture with integated blockchain technology by combining the building blocks of the fog gateway HCL-BaFog and the Multichain framework. It is our main engi-

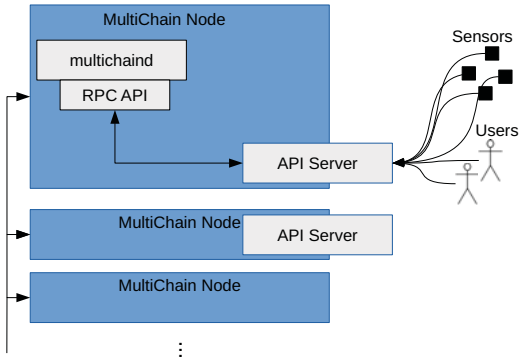

Fig. 3. System architecture of a MultiChain node exposing a custom-built public API to access the internal Multichain daemon and its remote procedure call interface.

neering objective to study how an effective integration of the basic functional blocks of both concepts into a layered protocol structure should be designed.

1) Structure of the Multichain Node: Considering the sketched system architecture of our versatile fog gateway, we have developed an extension of its virtualization concept to incorporate blockchain functionality of the Multichain framework (see Figure 3). The latter can be assigned to a more powerful controller node of a fog cell cluster in a hierarchical fog computing environment (see Figure 1). For this purpose it is required to establish a corresponding overlay network of fog nodes with blockchain processing, distributed data storage, secure access management, and connectivity capabilities. As sensors cannot operate a full blockchain node due to their resource and efficiency constraints, an enhanced fog gateway incorporating this container virtualized MultiChain functionality and a new secure access method which has only minimal client-side requirements are needed. Using MultiChain's node implementation `multichaind` that can be controlled by a JSON-RPC HTTP API, new transactions of the blockchain can be created and the blockchain state can be queried. However, the JSON-RPC API cannot be exposed to external client requests since local node settings are also configured by this original method. For this reason a separate service is needed to ensure a higher level of isolation and security. This new API has access to the JSON-RPC API, provides a domain-specific public API and mediates between requests of external clients and `multichaind` (see Fig. 3).

Publishing transactions to some address of a new peer represents one of the most important services offered by the API of a MultiChain node. A valid transaction references at least one input. When a peer generates a new address, e.g., to publish data on the blockchain, this address has no UTXO to be used as an input for the new transaction. Thus, the implemented client-server solution procedure generates a new MultiChain key pair offline and sends the generated address and transaction content to the API. The API server first publishes a transaction that transfers no value to the new address. Then the server sets up a new transaction draft with the content requested by the client and references the just sent empty transaction as an input. Finally, the server returns the transaction to the client that can inspect its content. If the transaction matches the expectation, the client signs the transaction and publishes the transaction to the blockchain network. Applying this workflow, the private key associated with the address of a peer in the blockchain is only accessible to the client. This feature prevents a rogue API server from altering the transaction content and from publishing additional transactions using that address.

Streams represent an abstraction built on top of the transactions in a blockchain. A stream consists of a list of stream items and is identified by a unique name. A stream item has an optional key and data that may comprise several megabytes in size. To read items from a stream, a blockchain node has to be instructed to subscribe to that stream. The node looks for the special transaction that initialized the stream and scans all subsequent blocks for transactions that add items to that stream. The node builds an index to allow multiple queries, e.g., listing all keys, listing all values, or listing only values associated with a certain key. On transaction level, a stream-creating transaction is identified by a special byte flag. A new stream item is introduced in a transaction output with a transaction output script that follows a certain structure. A transaction can carry multiple stream items (cf. [34]).

MultiChain is designed for a permissioned environment, i.e., by default participation in the network is not enabled for new peers. Thus, we assume in our approach that the fog computing environment will guarantee this property by appropriate means of the address assignment and the establishing of trust relations among the peers. MultiChain works with eight permission capabilities that are granted to addresses: `connect` to the blockchain network, `send`, i.e., create and sign transactions, `receive`, i.e., allow appearance in transaction outputs, `issue` new assets, `create` streams, `mine` new blocks, `activate`, i.e., change connect, send and receive permissions for other peers, and `admin`, i.e., change all permissions for other peers. All these capabilities have default values that are set in the genesis block. Similarly, permission changes are stored in transactions on the blockchain. As we assume the blockchain content to be public, e.g. for reading purposes by supervising medical personnel, we have only enabled `connect` and disabled all other permission attributes in our prototype.

Considering the inclusion of new data in a blockchain, the required distributed coordination process among peers is determined by the choice of the consensus algorithm. The related process of appending new blocks to the blockchain is called *mining*. Regarding MultiChain the consensus algorithm belongs to the class of *proof-based* schemes as no explicit coordination between the peers is necessary like in Bitcoin. In such a mining scheme each selected peer or *miner* collects new transactions into a new block and additionally starts to work on a computationally expensive task. The first miner that solves it broadcasts its block to the network along with its

solution. The receivers validate the solution and all contained transactions. If all tests are passed, the block is appended to the peers' copy of the blockchain and the process starts anew. As two concurrently active miners can find a solution and broadcast a new block almost simultaneously, peers will accept the block that reaches them first and reject the other one. This feature leads to a temporary *fork* of the blockchain. If peers have to decide in this case between two competing forks of a Multichain, they accept the longer chain similar to Bitcoin.

As MultiChain realizes a permissioned blockchain, the right to participate in the consensus process is explicitly granted to the peers. For this reason the system is not susceptible to a Sybil attack and, consequently, able to implement a more efficient, fair and architecturally simple consensus model. The consensus protocol aims to enforce a round-robin block creation schedule. In other words, given N mining peers in the system, each miner has created one of the last N blocks on average. This process is managed by a *mining diversity* parameter $\rho \in [0, 1] \subset \mathbb{R}$. A new block announced by some miner A is only accepted as valid by the network if A did not create one of the last $\lceil N \times \rho \rceil - 1$ blocks. The choice of ρ has to balance safety and availability concerns. A higher value means that more miners have to collude to steer the blockchain in their favoured direction. On the downside, the blockchain can tolerate less node failures before getting stuck and becoming unavailable. The official recommendation of MultiChain proposes the value $\rho = 0.75$ (cf. [15, p. 7f.]). We adopt this recommendation in our approach.

2) Sharing of Confidential Sensor Data: It is a fundamental property of all blockchains that its related data content is public, i.e., readable by everyone who is permitted to the blockchain. As addresses of peers associated with the blockchain are not directly linked to their identities, an observer cannot determine easily the peers of a transaction or calculate someone's total digital balance. The peers of a transaction must become aware of the other peers' identities by some off-chain mechanism, e.g., by simply exchanging addresses among the peers using some messaging application. Peers first publish their identity in terms of a public key on the blockchain. Subsequently, a peer that intends to share data generates a passphrase, encrypts its data with the passphrase and stores the ciphertext at a publicly addressable place. Now access to these data can be granted by conveying the passphrase to the selected receivers. This property again works by employing public-key cryptography, i.e. encrypting the password to the receiver's public key.

A corresponding access protocol to share confidential data in an enhanced fog computing environment has been implemented as shown in Figure 4. To facilitate the emission of confidential data to a peer of the blockchain, two MultiChain streams named `identities` and `access` are used. Regarding our example in Figure 4, an additional third stream `data` is used which is optional. A simple alternative could be provided by a HTTP link to the data. Here the

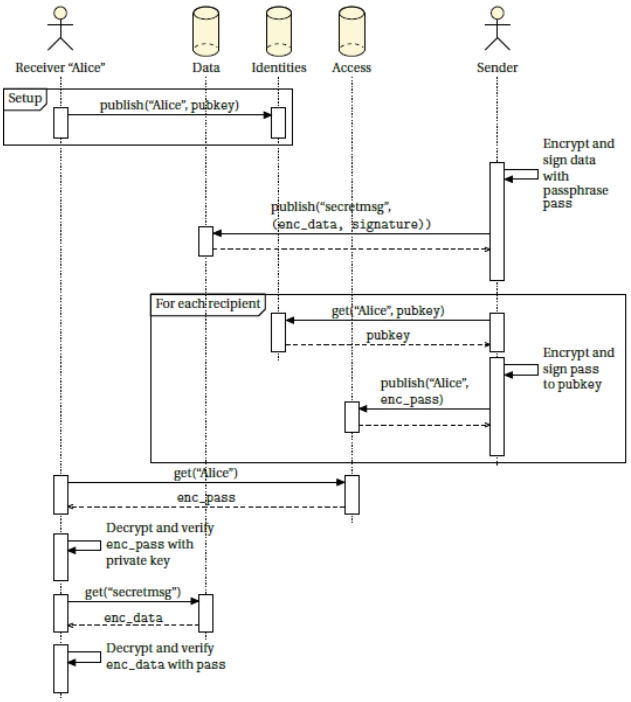

Fig. 4. The process model to share confidential data among a sender and the receiver Alice in a permissioned blockchain.

identity stream maps a label to a public RSA key whereas the *access* stream is used to deposit the decryption password for the receiver of a data item. The key of each item in that stream corresponds to the key that the receiver used to publish his public key to the *identities* stream. The content of an *access* item is encrypted by the receiver's public key. It contains a tuple where the first value is a pointer to the location of the transmitted data and the second value is the AES encryption password. In this example the pointer is the identifier of a transaction published to the *data* stream.

In all those cases where data are encrypted, a digital signature that provides authentication and integrity guarantees for the transmitted sensor data is created and stored along with the ciphertext. Fortunately, the blockchain offers convenient ways to provide message authentication. Regarding MultiChain, for instance, addresses are built around keys stemming from elliptic curve. When encrypted data are published in a transaction with a certain address of a peer, the sender has access to the private key of the addressee. Using the latter, the ciphertext can be signed and placed next to the ciphertext in the transaction. Before trying to decrypt the data, the signature should be checked with the public key of the transaction sender. To sign data and verify signatures, one can use the `signmessage` and `verifymessage` API's offered by MultiChain. Then the authentication code is relatively small and inexpensive to compute.

3) Sharing of Real-Time Sensor Data: The simplest mechanism to store data directly on the blockchain is provided by an attachment of the data to transactions. Then the data become part of the immutable blockchain history and are stored by all peers of the network. The transaction identifier is sufficient to unambiguously identify the data without integrity concerns. However, storing data directly on the blockchain leads to a massive growth in size. As old transactions cannot simply be omitted from the history of a blockchain, increasing storage requirements would quickly turn a fog node that is operating as peer of the blockchain into a major cost factor and prohibit the participation of nodes with limited resources.

Therefore, storing only the hash value of the data in the blockchain can provide the same guarantees with much lower storage demands. Independent of the data size, a computed hash value always has a constant length. The actual data can then be stored by some other means. On retrieval the integrity of the data can be verified recomputing its hash value and comparing it with the one that is immutably stored on the blockchain.

Retrieving the actual data referenced by a transaction can be accomplished in a multitude of ways such as *references* and *content-addressing*. These references to the location of the data along with the hash value in the transaction could be HTTP links, database queries or the identifier of another transaction (see also [22]).

MultiChain offers an integrated solution to indirectly store data on the blockchain by including the hash value of data in the transactions. This content-addressable storage and retrieval scheme is tightly integrated with the rest of the blockchain system, but it follows common principles that are similarly implemented by other frameworks. Publishing data to MultiChain works by sending a regular transaction with the attached data. MultiChain extracts the data, splits it into smaller chunks of 1MB in size by default and replaces the data in the transaction by the hash values of the chunks. The transaction is pushed to the network in the usual way, but the data stays with the publishing node at first. Only once another node is interested in the actual data, the chunks are requested and exchanged in a peer-to-peer fashion between MultiChain nodes. A common way to distribute these data and to provide stronger reliability guarantees is to publish the transaction to a MultiChain stream. By default, all subscribed peers fetch the referenced data chunks. As chunk hashes are included in the transaction, fog nodes acting as peers of MultiChain can easily verify their integrity. As long as at least one fog node possesses a data chunk, all other fog nodes can retrieve it.

We consider the storage and sharing of streamed real-time data arising from IoT applications and its access and retrieval management by means of the blockchain functionality and realize a corresponding protocol component based on a client-server programming paradigm. Five logical entities are involved in this process as shown in Figure 5: a user-controlled logical device (e.g., a smartphone) or its associated fog node proxy, a networked sensor, a time series database (TSDB), a MultiChain API server, and a MultiChain fog node. With

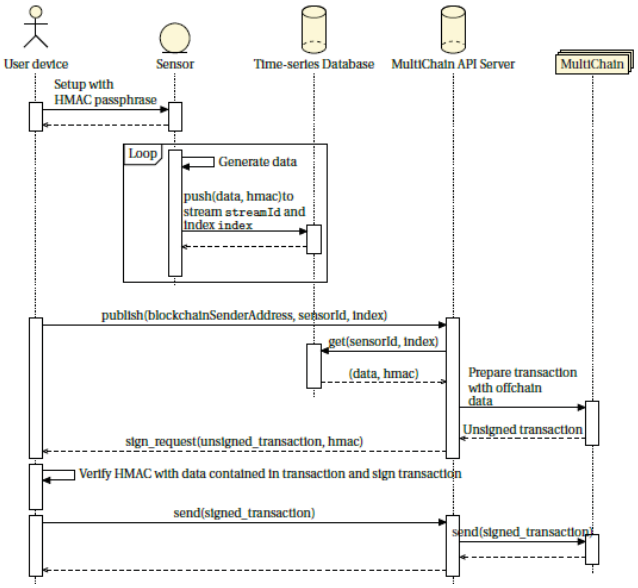

Fig. 5. The interworking procedure with an integrated fog-MultiChain node and the time-series database starting from sensor setup to data storage using the MultiChain platform.

the exception of MultiChain (which is an off-the-shelf tool), all components were implemented in our prototype using the programming language Go. The communication is realized by custom-built HTTP APIs. The sensor component (or its unambiguously associated fog gateway proxy) is configured with a reference to a TSDB instance and a password for authentication by a keyed-hash message authentication code (HMAC). At the beginning the sensor produces measurements in JSON format that are queued for submission to the database. A separate part of the sensor program handles communication with the latter. Failure cases, e.g. network problems, are handled transparently to the sensor program part. It is expeceted that the sensor (or its associated fog gateway proxy) also computes a HMAC based on the produced data. When new data are available, the associated client function performs HMAC computation, reads the intermediate result and sends it along with the data to the TSDB instance. As a HMAC is the outcome of a hash function, it can be computed incrementally on the fly, i.e. the client function does not need to store its measurements only for the sake of this computation.

In the model of our fog computing architecture effective, known frameworks such as InfluxDB or Prometheus may be applied as instance of a time series database. The feasibility on the SBC platform has already been shown (cf. [18]). In our prototype we have emulated the related storage functionality by an abstraction into an API component that realizes an append-only log with only a few additional features. Data are organized into *streams*, which are identified by *stream identifiers*. Data chunks of a stream are addressed by incrementing an integer *index* which starts at zero. New data can only be appended to stream indices and only the current index

is writable. As soon as data are written to the next index, the previously current index is locked. All write requests to the old index will fail with a reference to the current index. Read requests, however, are unaffected by this restriction. The only special case is that the locking process is also triggered if the current index is read. This behaviour enables a coordination between sensors (or their uniquely associated fog gateways) and the blockchain. The TSDB buffers data such that they can be bundled and published to the blockchain. This feature is necessary since data that are already stored on the blockchain cannot be extended. Each TSDB index corresponds to one blockchain transaction. The sensor (or its associated fog gateway proxy) always writes to the current index until the blockchain reads from it to retrieve the data for publishing. At that point the sensor should switch to the next index to buffer data for the next transaction which is accomplished by the locking mechanism. In addition to the data each index is associated with a HMAC that is updated by the client function along each storage request.

In the prototype an interaction with the time series database occurs by a HTTP API that offers two points of interaction. First, a specific index of a stream can be read. No authentication is required in the secured fog computing environment. Secondly, data can be appended to a specific index of a stream and a corresponding HMAC will be passed in a special request header. If the stream does not exist yet, it is created and a generated password is returned to the requesting client function. Alternatively, the password needs to be present in an authorization request header.

Regarding our prototype, the MultiChain API server is extended by an endpoint to pull data from a TSDB instance and to push it onto the blockchain. This process is initiated by the user device (or its associated fog node) and triggered by a timer, for instance, or an explicit user request. The latter sends a HTTP request that contains the URL of the TSDB instance, the stream identifier, index and a blockchain address where the storage transaction will originate.

To realize the sharing functionality for real-time sensor data, our implementation uses the offchain storage feature of MultiChain. While it is rather seamlessly integrated into the existing blockchain model, one needs to be aware of a few characteristics. First, if data of a transaction exceed a threshold, they are split into multiple chunks. By default chunks are 1 MB in size and a transaction is limited to 1024 chunks. The HMAC is stored along with the data to allow the user (or its fog node proxy) to which the sensor is associated in a sharing event, to verify the integrity of the data before signing the storage transaction with an address that it controls. When MultiChain receives an offchain transaction, the data is extracted and replaced by an array of chunk hashes. The transaction is passed back to the initiating entity for signing in this form which entity can compute the HMAC over the hash of the chunk and to compare it with the one generated by the sensor. If both items match, no intermediary (e.g., the TSDB instance or the API server node) has messed with the data.

Unfortunately, MultiChain's automatic splitting feature conflicts with the described scheme as it breaks HMAC authentication. Therefore, only data smaller than the maximum chunk size are stored in a single transaction in our setting.

C. Evaluation of the Fog Computing Node With Integrated Blockchain Functionality

Our second objective concerns an investigation of the performance results that are achievable by the implemented arrangement of the blockchain components in a virtualized fog gateway on a low cost SBC platform with its constrained resources such as HCL-BaFog.

1) The Fog Node Test Bed: For this purpose we have developed a test bed to evaluate some performance metrics of the MultiChain implementation. Following the HCL-BaFog approach, a cluster of three Raspberry Pi SBCs that is controlled by a single PC as management node has been used as fog gateway network. Each Raspberry Pi 3 model B Rev 1.2 node was equipped with a QuadCore 1.2GHz 64-bit CPU, 1GB RAM, and a 100 Mbit/s Ethernet port.

The used software environment was built on top of an adapted HCL Linux system and supports container virtualization and orchestration by means of Docker (cf. [35], [36]). The central component is given by a *Docker daemon* which employs Linux kernel features to provide an isolated environment to execute the virtualized functions. The blockchain functionality is provided by a Docker image of a blockchain generated by the MultiChain version 2.0 alpha 3 framework. As MultiChain supports officially only the x84 processor architecture, the framework had to be compiled for an ARM processor architecture from scratch. Docker Swarm has been applied to orchestrate the execution of Docker containers across multiple hosts that run Docker daemons. Nodes in a Docker Swarm are either assigned to a *worker* mode, a *manager* mode, or both roles. Manager nodes can decide which functions are executed by which worker nodes. Docker Swarm has been instructed to deploy only a single container running the master-multichain image on a manager node. To effectively run a MultiChain node on each host, it should try to deploy node-multichain containers on each node that is not a manager. After startup the master-multichain image first initializes a new blockchain. The associated master-multichain container does not have any inherent privileges after setup and could in fact be replaced by any container based on the node-multichain image.

To monitor important metrics of the deployed computing system of a fog node such as the processor utilization, memory usage, file system writes, etc., the ready-made Docker image *docker-swarm-monitor* was applied. This template sets up an instance of the monitoring system *Prometheus* and installs system monitoring daemons on all Swarm nodes. The latter are regularly contacted by Prometheus to collect data. In addition, the template deploys an instance of *Grafana* which visually represents the collected performance data. To keep the overhead on the Raspberry Pi hosts as low as possible during our tests, a laptop PC was also added to the cluster as passive management node. Using Docker node tag rules, the execution of the Prometheus and Grafana instances

was scheduled on the latter PC. This additional member of the test bed did not run any functions related to MultiChain.

2) Brief Performance Assessment: We know that transactions constitute the central element of blockchains, even with regard to advanced features of the MultiChain framework like permissions. Therefore, it is of critical importance to determine the maximal rate of a fog gateway by which this node can accept and process single transactions and include them into a block.
For this purpose the MultiChain network with three fog nodes has been used, where each node runs on a separate Raspberry Pi SBC. All nodes were tasked with mining blocks. In multiple rounds a fixed number of clients repeatedly called each node's API to send transactions. The transaction was minimal, transferring a zero amount of the built-in currency to an address not owned by anyone of the three nodes in the test bed. At the beginning of a test round, the clients were started steadily over a period of 60 seconds to avoid an overloading of a node by simultaneous requests. While the clients have already sent requests, the metrics were recorded only after another 60 seconds. This precaution made the measurements more coherent. Then each test was running for 5 minutes. Before continuing with the next test, the test executor waited until all nodes had included all received transactions into the blockchain. Then the MultiChain nodes were shut down and a new blockchain was initiated before we started the next round with an increased number of requesting clients.
The results of repeated test runs with a different number of clients are summarized in Figure 6. At first view, the median number of transactions per second (Tx/s) seems to fluctuate. However, the number of clients ranges from 30 to 330 (a difference of factor 11). The difference between the maximal and the minimal measured median rate is only 34.5 Tx/s.
The constant number of published transactions per second may be explained by the implementation of MultiChain. A developer clarified that the API only serves one node at a time due to the use of locks. The API code is effectively single-threaded. This matches the CPU utilization during our tests. At sight, one CPU core is almost constantly fully occupied, while the other three are almost idle. As stated by the MultiChain developers, increasing the throughput above two concurrent connections should not be expected. The developers consider the CPU as primary bottleneck and reported a rate of about 1000 Tx/s on a modern x86 CPU (cf. [37]).

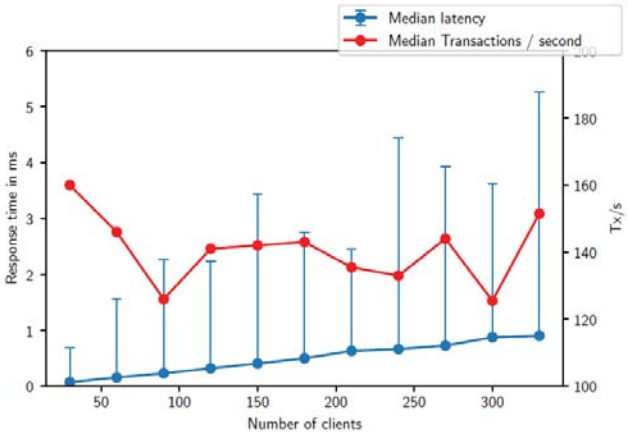

Fig. 6. The median transactions per second, the median response time in ms, and the standard deviation are shown for a different number of requesting clients during repeated stress tests. As the x-axis depicts the *total* number of clients, each node was only targeted by $1/3$ of the clients.

IV. Conclusions

In recent years Internet-of-Things (IoT) architectures have initiated a fundamental paradigm shift towards new machine-to-machine and device-to-network communication patterns among billions of smart devices and given rise to an enormous collection of sensor data and pervasive computing efforts. Manufacturers and users of the technology are facing severe challenges regarding long-term stability, interoperability, as well as privacy and security issues. In this respect the blockchain technology provides a set of unique characteristics that can be employed to solve some of those IoT related research and development issues.
In this paper we have focussed on the problem of storing and securely sharing sensor data by means of a fog computing node with blockchain functionality. Regarding the landscape of open-source software offering blockchain functionality and its suitability for different IoT use cases, a number of blockchain projects that enable developers to build an application on top of a ready-made fog computing infrastructure have been surveyed and rated. These projects differ primarily in their choice of the used consensus algorithm, their programmability features, and confidentiality support.
We have chosen the blockchain framework MultiChain to integrate it as a first approach into the virtualized, modular fog computing gateway HCL-BaFog. The reason for this decision is that MultiChain is close to Bitcoin and its original blockchain idea. Regarding this integration effort of MultiChain, its features based on conventional transactions of a blockchain have proved to be very valuable. Using the basic blockchain framework MultiChain in our fog computing node, two new protocols for data storage and secure access management were implemented and extensively discussed. The first one supports a sharing of data with selected entities over a public blockchain channel. The proposed scheme is flexible since the access part can later be added and the storage location of the data is not prescribed. The second component enables a storage of streamed real-time sensor data on the blockchain. It bridges the gap between continuously produced sensor data and the immutable storage semantics of a blockchain. This featue is accomplished using an additional service that buffers data while not compromising their integrity or uncovering the confidentiality of the data. Using these two protocols as building blocks, a scheme has been presented to disclose non-sensitive data publicly while restricting access to its sensitive

parts. This approach highlights possible applications of the framework, e.g. in healthcare, also pointing to further potential capabilities of the sketched integrated fog-blockchain approach such as agriculture, logistics, or smart city applications. As the proposed algorithmic schemes of the new fog computing node are dependent on the behaviour of the underlying blockchain platform, the performance of MultiChain has been explored in a cluster of Raspberry Pi SBCs.

In conclusion we are convinced that our new protocols built on top of well-understood primitives of MultiChain provide a composable virtualized functionality of a fog computing node to realize bigger data processing chains for more advanced IoT applications. Future research will focus on an incorporation of advanced cryptographic methods to enable more advanced services of the discussed IoT scenarios.

ACKNOWLEDGMENT

The authors express their appreciation to the developers of the used blockchain framework MultiChain. They are also indebted to the reviewers whose constructive comments guided an improved presentation of the material.

REFERENCES

[1] M. Aazam, E.-N. Huh, "Fog Computing and Smart Gateway Based Communication for Cloud of Things," in 2014 International Conference on Future Internet of Things and Cloud (FiCloud). Barcelona, Spain, 27–29 August 2014, pp. 464–470.

[2] A. Al-Fuqaha, et al., "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," IEEE Communication Surveys & Tutorials, vol. 17, no. 4, pp. 2347–2376, 2015.

[3] L. Atzori, et al., "Internet of Things: A Survey," Computer Networks, vol. 54, no. 15, pp. 2787–2805, 2010.

[4] E. Androulaki, et al., "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in R. Oliveira, P. Felber, and Y. C. Hu, Eds., Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018, ACM, 2018, pp. 30:1–30:15.

[5] K. Ashton, et al., "That Internet Of Things Thing," RFID Journal, vol. 22, no. 7, pp. 97–114, 2009.

[6] A. Bessani, J. Sousa, E. Alchieri, "State Machine Replication for the Masses with BFT-SMART," in 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) 2014. IEEE, 2014, pp. 355–362.

[7] P. Brody, V. Pureswaran, "Device democracy: Saving the future of the Internet of Things," IBM, September, 2014.

[8] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," 2013, accessed on Nov. 22, 2017. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[9] C. Cachin, M. Vukolić, "Blockchains consensus protocols in the wild," arXiv preprint arXiv:1707.01873, 2017.

[10] M. Castro, B. Liskov, "Practical Byzantine fault tolerance," in M. I. Seltzer, P. J. Leach, Eds., Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI). New Orleans, Louisiana, USA, February 22-25, 1999, USENIX Association, 1999, pp. 173–186.

[11] M. Chiang, T. Zhang, "Fog and IoT: An Overview of Research Opportunities," IEEE Internet of Things Journal, vol. 3, no. 6, pp. 854–864, December 2016.

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

[13] M. Conoscenti, A. Vetrò, J. C. De Martin, "Blockchain for the Internet of Things: A systematic literature review," in 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA). Agadir, Morocco, 29 Nov.–2 Dec. 2016.

[14] A. Eiermann, M. Renner, M. Großmann, U. R. Krieger. "On a Fog Computing Platform Built on ARM Architectures by Docker Container Technology," in G. Eichler, C. Erfurth, and G. Fahrnberger, Eds., International Conference on Innovations for Community Services (I4CS 2017), Innovations for Community Services. Springer, 2017, pp. 71–86.

[15] G. Greenspan, "Multichain Private Blockchain - White Paper," 2015.

[16] M. Großmann, A. Eiermann, M. Renner, "Hypriot Cluster Lab: An ARM-Powered Cloud Solution Utilizing Docker," in 23rd International Conference on Telecommunications (ICT 2016). Thessaloniki, Greece, 16–18 May 2016.

[17] M. Großmann, A. Eiermann, "Automated Establishment of a Secured Network for Providing a Distributed Container Cluster," in 28th International Teletraffic Congress (ITC28). Würzburg, Germany, 13–15 September 2016.

[18] M. Großmann, et al., "SensIoT: An Extensible and General Internet of Things Monitoring Framework," Wireless Communications and Mobile Computing, vol. 2019, Article ID 4260359, 2019.

[19] M. Großmann, C. Ioannidis, "Continuous Integration of Applications for ONOS," in 5th IEEE Conference on Network Softwarization (NetSoft 2019). Paris, France, 24–28 June 2019, to appear.

[20] J. Gubbi, et al., "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Generation Computer Systems, vol. 29, no. 7, pp. 1645–1660, 2013.

[21] S. Holla, Orchestrating Docker, Packt Publishing Ltd., Birmingham, 2015.

[22] T. Jin, X. Zhang, Y. Liu, K. Lei, "BlockNDN: A bitcoin blockchain decentralized system over named data networking," in Ninth International Conference on Ubiquitous and Future Networks (ICUFN) 2017. Milan, Italy, July 4–7, 2017, pp. 75–80.

[23] L. Lamport, R. E. Shostak, M. C. Pease, "The Byzantine Generals Problem," ACM Transactions on Programming Languages and Systems, vol. 4, no. 3, pp. 382–401, 1982.

[24] W. Liu, et al., "Advanced Block-Chain Architecture for e-Health Systems," in 19th International Conference on E-health Networking, Application & Services (HealthCom) - 2nd IEEE International Workshop on Emerging Technologies for Pervasive Healthcare and Applications (ETPHA 2017). Dalian, China, October 12, 2017.

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

[26] G.-T. Nguyen, K. Kim, "A Survey about Consensus Algorithms Used in Blockchain," Journal of Information Processing Systems, vol. 14, no. 1, pp. 101–128, 2018.

[27] S.M. Riazul Islam, et al., " The Internet of Things for Health Care: A Comprehensive Survey," IEEE Access, vol. 3, pp. 678–708, 2015.

[28] H. Shafagh, et al., "Towards Blockchain-based Auditable Storage and Sharing of IoT Data," arXiv preprint arXiv:1705.08230, 2017.

[29] O. Skarlat, et al., "Resource Provisioning for IoT Services in the Fog," in 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA 2016). Macau, China, 4–6 November 2016.

[30] M. Vukolić, "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication," in J. Camenisch, D. Kesdogan, Eds., Open Problems in Network Security - IFIP WG 11.4 International Workshop, iNetSec 2015. Zurich, Switzerland, October 29, 2015, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9591, Springer, 2015, pp. 112–125.

[31] D. Wörner, T. von Bomhard, "When Your Sensor Earns Money: Exchanging Data for Cash with Bitcoin," in UbiComp '14 Adjunct, ACM, 2014, pp. 295–298.

[32] B. Zhang, et al., "The Cloud is Not Enough: Saving IoT from the Cloud," in 7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15). Santa Clara, CA, USENIX Association, 2015.

[33] G. Zyskind, et al., "Enigma: Decentralized Computation Platform with Guaranteed Privacy," CoRR, abs/1506.03471, 2015.

[34] Coin Sciences Ltd., "Multichain data streams," 2018. [Online]. Available: https://www.multichain.com/developers/data-streams/

[35] Docker Inc., "Docker overview," 2018, accessed on Aug. 28, 2018. [Online]. Available: https://docs.docker.com/engine/docker-overview/

[36] HCL, "Hypriot Docker Image for Raspberry Pi," accessed on Sept. 2, 2018. [Online]. Available: https://blog.hypriot.com/downloads/

[37] MultiChain, "How to optimize multichain read and write transaction/sec performance," 2017, accessed on Sept. 2, 2018. [Online]. Available: https://www.multichain.com/qa/7648/optimize-multichain-read-write-transaction-sec-performance