Mullick, Mohammad R. H.; Großmann, Marcel; Krieger, Udo R.

A Feasibility Study of a Lightweight Fog Computing Architecture Integrating Blockchain Technology for Smart E-Health Applications

A Feasibility Study of a Lightweight Fog Computing Architecture Integrating Blockchain Technology for Smart E-Health Applications

Mohammad R. H. Mullick, Marcel Großmann, and Udo R. Krieger(✉)

Fakultät WIAI, Otto-Friedrich-Universität, An der Weberei 5, 96047 Bamberg, Germany
udo.krieger@ieee.org

Abstract. Today, a variety of advanced Internet-of-Things applications like data collection apps for ambient assisted living services supporting elder patients or for the surveillance of smart homes are developed everywhere. Regarding such application scenarios and the requirement to securely disseminate the collected data, we present a fog computing architecture that integrates blockchain technology for privacy aware data sharing. The latter fog-blockchain prototype is based on a private, permissioned blockchain paradigm. We discuss a lightweight implementation of our proposed integrated fog node architecture using Hyperledger Fabric as blockchain with a peer-to-peer network among cheap single-board-computers as basic data collecting peers. Furthermore, we investigate the computing and network performance indices of our prototypical lightweight fog computing architecture.

Keywords: Internet-of-Things · Fog computing · Smart home · Blockchain · Hyperledger fabric

1 Introduction

Nowadays, new advanced Internet-of-Things (IoT) applications are evolving very rapidly and advanced software-defined communication technology like 5G and evolving 6G mobile networks with better network functions virtualization (NFV) techniques are incorporated into the underlying infrastructures of software-defined networks (SDN) for cloud computing, fog computing, and multi-access edge computing (MEC), cf. [1,10,11].

Considering advanced Internet-of-Things applications like data collection apps used for ambient assisted living of patients at their homes or for the surveillance and remote control of heating, ventilation, air condition (called HVAC), washing machines and other entities consuming energy triggered by users' control at their smart homes, one may support an anonymized data collection and the decoupling of personal identities and control functionality by blockchain

technologies, cf. [6,20]. Such privacy protection methods can enable better consumption studies of user communities by service providers. They can also serve the exigency to improve advanced healthcare services or smart energy provisioning at private homes that are raised by current socio-economic developments like aging societies and a period of climate change.

Then the research question naturally arises whether one can incorporate fog computing together with blockchain technology for related IoT-applications and how a resourceful, efficient system design based on an extensible virtualized open source software architecture should look like.

In this paper we investigates the feasibility of using a private, permissioned blockchain concept in a distributed fog computing environment, cf. [13,20,21]. Such a private blockchain aims to work in a controlled setting. It is most suitable for a rapid adoption by IoT-service providers exploiting an SDN/NFV based network environment to interconnect modern IoT devices and cloud, fog, or edge computing systems. On the path between an end device to the cloud every device is maintained by the service provider. Regarding the collection of sensitive personal data in the sketched smart home scenarios like smart ambient assisted living systems, such a private blockchain makes perfect sense to protect the personal data. To study the feasibility and performance aspects of such an integrated architecture we have designed such a lightweight IoT-architecture for the related data collection by personal sensors and implemented it by means of a cheap single-board-computer (SBC). Then the integrated blockchain technology can offer an extensible, immutable, transparent data repository in our fog computing system, cf. [4,10,21]. The latter can be used for securing sensitive smart home applications like the sharing of personal health data that are collected by patients with persistent diseases like diabetes mellitus or arterial hypertension, for instance, or the usage patterns of employed service entities in smart private homes.

We have implemented this distributed architecture by means of Hyperledger Fabric [2] in a test bed of Raspberry Pi SBCs which serve as basic nodes of an underlying peer-to-peer network of this employed permissioned blockchain. Then we have evaluated important performance metrics of our fog computing system by means of a test bed with five SBCs as basic peers of Hyperledger Fabric as its core blockchain component. In the following we discuss our findings regarding this integrated software design and reveal the limits of its implementation by our feasibility study. These technical results may inspire improved fog node architectures and the design of further scalability and feasibility studies in modern smart cities like Bamberg, cf. [17].

The paper is organized as follows. In Sect. 2 we present the foundations of our integrated fog computing system. In Sect. 3 we describe the developed software architecture of the integrated fog-blockchain system and thereafter its implementation in Sect. 4. Then we evaluate some performance metrics of the implemented prototype in Sect. 5. Finally, some conclusions are presented.

2 Foundations of a Fog Computing System Integrating Blockchain Technology

Today, it is a standard edge computing approach to consider the support of advanced IoT-application areas by means of a fog computing architecture shown in Fig. 1, cf. [1,18]. Considering the different IoT-application areas, there are diverse requirements with respect to the underlying architecture of fog nodes and their performance as indicated by Table 1, cf. [1,10].
A fog computing system based on scalable virtualization techniques can process and store a large variety of digital data collected by lightweight clients like sensors in smart homes or in professional healthcare environments like retirement homes. Regarding the next generation of a sharing economy based on the dominance of those digital data, privacy and security issues naturally arise, cf. [14]. The sharing of the collected data by the sensor layer of a hierarchically organized fog computing system can be supported by blockchain technology, cf. [6,20,21].

In particular, healthcare applications have their own unique requirements and challenges such as *security*, *interoperability*, *data sharing* and *mobility* support, cf. [14]. In this situation it is necessary to provide access controls, authentication, and non-repudiation of a patient's electronic medical health record or her/his data generated by various medical devices, for instance, sensors collecting the heartbeat rate and the glucose value at various times of the day.

Standard requirements of typical e-health applications in smart homes such as ambient assisted living services which are one focal area of our fog computing architecture with its blockchain functionality often incorporate the following three basic technical design issues:

- In a considered typical IoT-healthcare context *interoperability* turns out to be a major issue. Regarding typical therapeutical scenarios of an affected patient, the patient's healthcare data need to be shared among different doctors or even hospitals. But in a centralized database oriented architecture this may

Table 1. IoT application sectors supported by fog computing and their infrastructural requirements

Application Sectors	Infrastructural Requirements
sharing economy	trustworthy transaction models
insurance/liability	transparency, tamper-proof data management systems
industrial IoT-sector	transparent product lifecycle management
automotive and mobility management	vehicle lifecycle management by tracking the history of a vehicle, verifying the use of various manufacturing elements, etc.
smart city	huge bandwidth, low-latency, privacy, key management
healthcare, education, government	secured identity management, hiding personal information, sharing personal data, only sharing data with intended parties

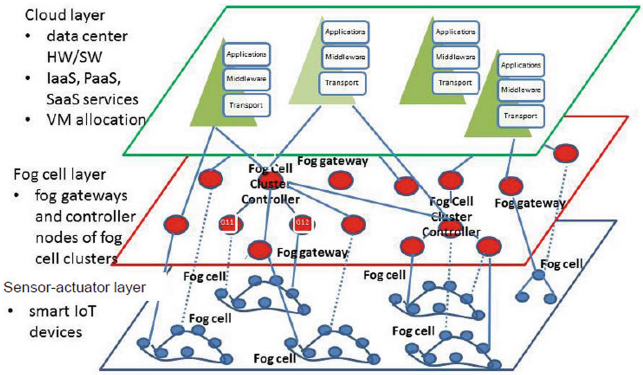


Fig. 1. Hierarchical architecture of a fog computing environment (cf. [21])

be impossible. Furthermore, it also opens up an opportunity for a monopoly regarding a patient's treatment or severe reliability issues.

– *Data sharing* is necessary for a patient to have a unified view of the whole medical data. One may also need to share only some part of the data with some other party. A typical medical diagnostics step may require a patient's thorough medical history over several years depending on her/his age. A centralized solution where data sharing is not possible due to the control by a single entity without securely controlled Internet access may become a bottleneck in such a process, in particular in case of emergency. Therefore, sharing the required personal data is an important factor where a blockchain based solution can substantially improve the medical treatment.
– *Mobility* is becoming another important factor due to the expansion of mobile health applications where various body sensors are collecting personal data continuously, e.g., by means of wearables. Maintaining the privacy of such generated data during the data sharing, consent management, access control, and authentication processes as well as the users' trust in those new technologies constitute key issues of such advanced IoT-scenarios of ambient assisted living and remote e-health services. In this respect various research efforts are currently going on to figure out how blockchain technology can be helpful in a fog computing environment, cf. [4,10,14,18].

A fog computing infrastructure incorporating blockchain functionalities can enable several key features required by these new smart home applications. Here the IoT-environment normally applies a client-server paradigm and the fog computing layer constitutes an intermediary level between the clients and servers. As indicated by Table 2, combining both fog computing and blockchain technology can offer advanced application features like an anonymous sharing of the collected personal sensor data, trustworthy transactions among different parties involved in the considered smart home scenarios and e-health applications, and a secure peer-to-peer communication among all those parties. It can also provide

new value-added features like the tracking of the data history, transparency, and a tamper-proof data management, etc., see [20].

Table 2. Key application features offered by an integrated fog-blockchain infrastructure

Application features	IoT	Fog	Fog + Blockchain
Privacy	No	No	Yes
(Pseudo-) anonymous sharing of sensor data	No	No	Yes
Transparency	No	No	Yes
Temper-proof data management	No	No	Yes
Trustworthy transactions	No	No	Yes
Data history tracking	No	No	Yes
Peer-to-peer communication along an overlay network	No	No	Yes
Huge bandwidth	No	Yes	Yes
Low-latency communication	No	Yes	Yes

Regarding the design of an extensible, virtualized, distributed service platform for smart home applications, we have decided for all these reasons to adopt a fog computing approach with an integrated blockchain functionality.

3 Design of a Lightweight Integrated Fog-Blockchain Architecture for Smart E-Health Applications

Considering smart home applications such as e-health apps for ambient assisted living scenarios, the effective integration of blockchain technology into a fog computing environment is a challenging technical issue. First of all, it is necessary to specify the scenarios where the collected sensor data can be shared within the fog-blockchain computing environment. Furthermore, one has to figure out the roles of the stakeholders of such a distributed ledger system.

For this purpose we follow here the fog architecture with its three basic layers depicted in Fig. 1 that we have developed in our previous research on fog gateways, cf. [21]. Regarding this three-layer computing model and its protocol stack, similar design studies and reference architectures already exist with respect to a blockchain integration, see, e.g., that one by Yang et al. [20]. To implement a scalable system it is reasonable to implement virtualized fog gateways on top of cheap SBCs like Raspberry Pi or Arduino systems, cf. [21].

The major problem in setting up such an architecture and its underlying communication network is to meet the required trustworthiness, the tracking, tamper-proof data management, and the privacy, the security, as well as the SLA and QoS related criteria of a considered IoT-application. The latter criteria have to be developed along the three basic dimensions of the *network*, *storage*, and *computation* components in the architecture of the fog gateways, cf. [20]. Regarding a blockchain concept to support the privacy, security, and

immutability, one may use the concept of a plain public or a private blockchain, or of a multichain, or the sharding approach, or to use off-chaining or sidechains like Plasma, cf. [4,16,21]. Considering our smart home scenario with personal e-health applications, we can immediately exclude a public blockchain at the edge of an IoT infrastructure and argue that a private, permissioned blockchain has to be chosen to fulfill the functional and non-functional requirements in an optimal way.

Considering the design options with respect to secured smart e-health applications, the use of such a blockchain at the resource constrained IoT-devices is not apparent. In an IoT-application scenario like the healthcare sector where a patient's data may need to be shared amongst multiple parties, e.g., in case of an emergency, a delicate data sharing technique is required for such setting. In this case it is certainly not convenient to handle the issues of smart data processing and the creation of transactions at the edge device with its sensors due to the restricted storage and processing capabilities. Rather than using the edge device itself, one can adapt the blockchain functionality at their logical counterparts within the fog layer, namely the fog gateways. The latter are managing the data collection of the end devices including all their associated sensors. In this case it is a plausible idea to utilize the cloud as a persistent storage of the patient's data. This design can also offer the integration of new machine learning techniques. Regarding the fog computing layer with its distributed gateways and the logical layers of a blockchain, we follow this integrated approach and its related protocol stack shown by Fig. 2.

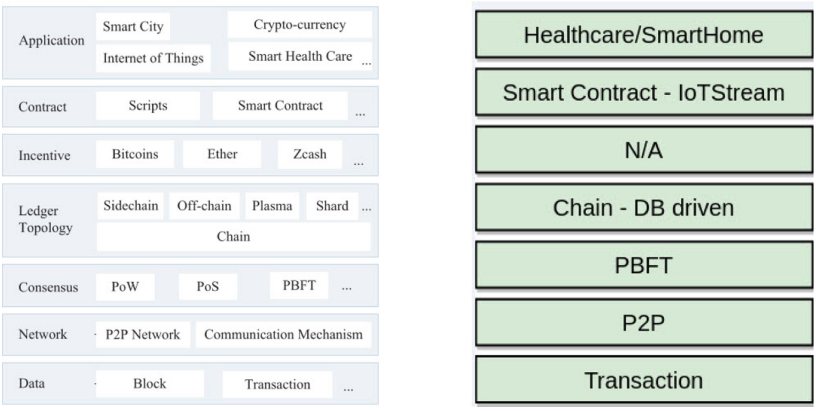


Fig. 2. Functional decomposition of the blockchain stack and its matching with the realized integrated design of our lightweight fog-blockchain prototype (cf. [20])

The fog nodes serve as peers of the peer-to-peer (P2P) network of the blockchain. The consensus protocol of a blockchain and the topology of the distributed ledger platform that is supporting the interaction of the blockchain nodes employing smart contracts can be derived from a given distributed ledger

platform, cf. [4,18,20]. Our previous research [4,21] has shown that the Practical Byzantine Fault Tolerance (PBFT) scheme may provide an adequate consensus algorithm for resource constrained fog gateways based on Raspberry Pi SBCs. Therefore, we attempt to incorporate such a scheme to establish the ledger topology among the peers of the realized blockchain.

Deploying a private, permissioned blockchain at the cloud layer gives us the opportunity to share the time stamped and timely ordered data to multiple parties. Regarding the selection of a blockchain platform, we may argue that an open source, enterprise oriented solution where multiple organizations or parties can get involved in both the consensus process and the data sharing process is the right option of the considered smart e-health scenarios. Therefore, we have adopted Hyperledger Fabric [2] as our basic permissioned blockchain. This private blockchain platform is used in our cloud layer for an immutable storage of the collected personal data.

The fog gateways of the fog computing layer will work as the distributed clients of this selected blockchain. It is an advantage of our integrated design that the end devices of the deployed IoT-application with their attached sensors will remain unchanged. Hence, they will not be impacted by underlying design changes in either the fog computing layer or the cloud layer.

In our integrated fog-blockchain approach we also have to look at the network dimension and, in particular, at the peer-to-peer network of the blockchain. The latter is established among the nodes of the blockchain related to the end devices and their controlling fog gateways. The latter nodes incorporate an SDN/NFV functionality, and the employed consensus algorithm of the selected blockchain model, see Figs. 1 to 4. Considering the fog layer, there can be multiple gateways within several sublayers. Regarding the sketched e-health scenario, only a single layer has been studied in the realized fog-blockchain prototype of our feasibility study in Sect. 5.

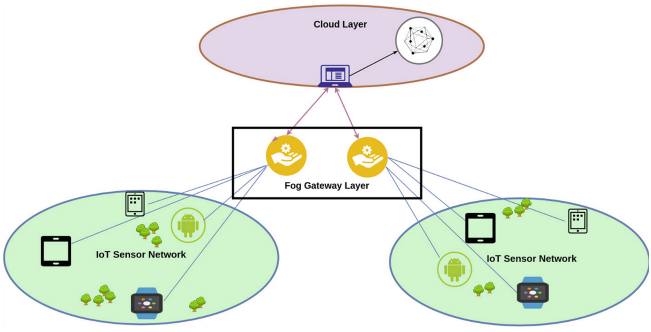


Fig. 3. Overview on the integration of a permissioned blockchain into a fog computing environment of our prototype

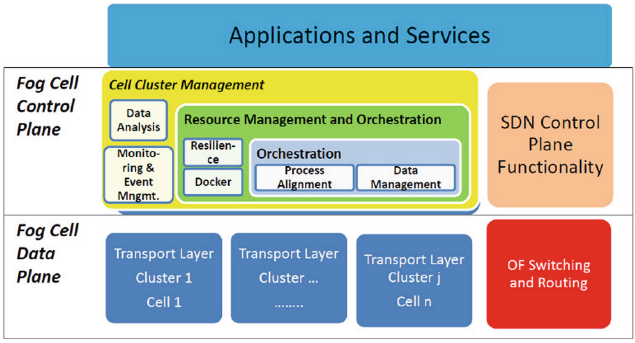


Fig. 4. Prototypical protocol design of a fog gateway (cf. [4])

In general, the application layer of our integrated protocol design can be realized by any IoT-application for smart healthcare, which is backed up by the use of smart contracts. In our approach the latter components are derived from the underlying contract layer of the blockchain platform. As Hyperledger Fabric is a private blockchain, it does not require any incentives in our assumed basic e-health scenario, see Fig. 2.

In conclusion, we adopt to use the blockchain platform Hyperledger Fabric which realizes the topology layer of the distributed ledger instances by a single chain. Its consensus layer uses a PBFT-based solution while the data are processed by means of a transaction from the bottom layer of the chosen protocol model, cf. [2,21]. Considering our overall system architecture, its hierarchical fog computing layer with the associated fog gateways contains an SDN-enabled device layer for real-time data analysis and communication as well as the delivery of service elements to the attached local IoT-devices, see Fig. 4.

We believe that the use of such effective, virtualized fog gateways is a systematic way to provide the required QoS-features of low latency and high throughput for the resource constrained end devices and their associated fog nodes, respectively. Then the latter gateways are the logical counterparts within the established P2P-network of the associated private blockchain. A considered fog gateway will also work as a data broker receiving data streams from the logically attached IoT-end devices. It will only forward the data which are required for a given transaction of the blockchain at a certain epoch. Then the client at the cloud layer is capable of talking to this data broker and it can also create a transaction on demand. As not all data streams may be interesting for a given e-health application scenario, they may not be part of a considered transaction. Then they are also not inserted into the instantiated ledger of the blockchain. In a typical healthcare scenario, involved actors like doctors, for instance, may only be interested to know how often the blood pressure of a particular patient crossed the normal level. Therefore, it makes sense to record only those hashed references as associated information records in the distributed ledger of the realized blockchain.

In summary, we are convinced that this adopted classical three-layer fog computing model in Fig. 1 can provide an efficient technical basis of our integrated fog-blockchain approach. Moreover, it has been designed to satisfy all QoS and SLA requirements of a time-critical IoT-application and can achieve a maximal throughput of processed data requests. The latter features of our prototype have to be validated by adequate experiments in a test bed.

To keep the blockchain architecture as simple and functionally effective as possible, we include here the miner nodes of the private, permissioned blockchain Hyperledger Fabric [2] with their main functionalities. The latter incorporate the verification and validation features and the immutable storage of the collected ledger data at the top-level of our protocol stack given by the cloud layer. We know that Hyperledger Fabric follows the state machine replication approach perceived from a BFT-based consensus mechanism, cf. [19]. It also offers various storage facilities like the ability to use a database, which makes it suitable for various critical IoT-application scenarios. Regarding its application design one can take into account multiple organizations and apply an ordering service.

In Fig. 5 we illustrate the concept of our realized prototype and the interactions among its basic components where we have used only two organizational units for simplicity. Each of these organizations contains two nodes where one of them plays the role of an anchoring peer. The red line indicates a cross-organizational gossiping. The client communication with an anchoring peer and the ordered service are shown in black and green colours, respectively. The blue line indicates a communication relationship between the orderer and the anchoring peer of a transaction.

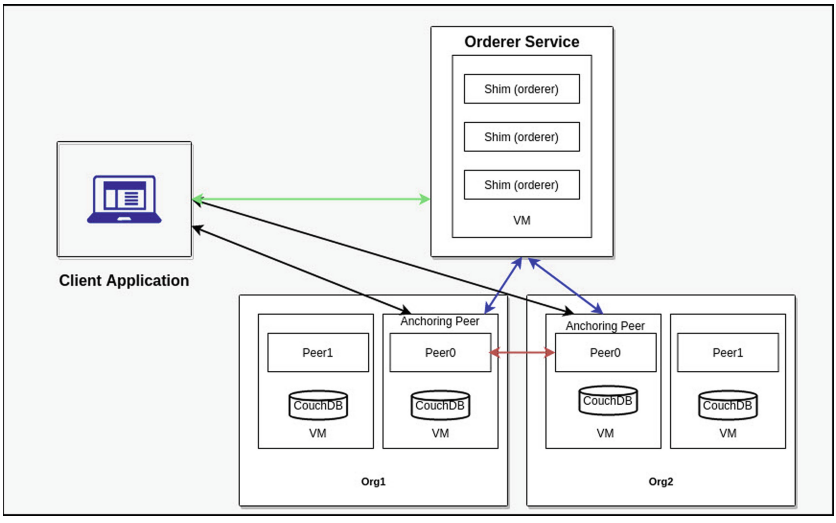


Fig. 5. Overview of the Hyperledger Fabric components among the peers of two different organizations and their interaction in the designed protocol stack of our fog-blockchain prototype (see also [2])

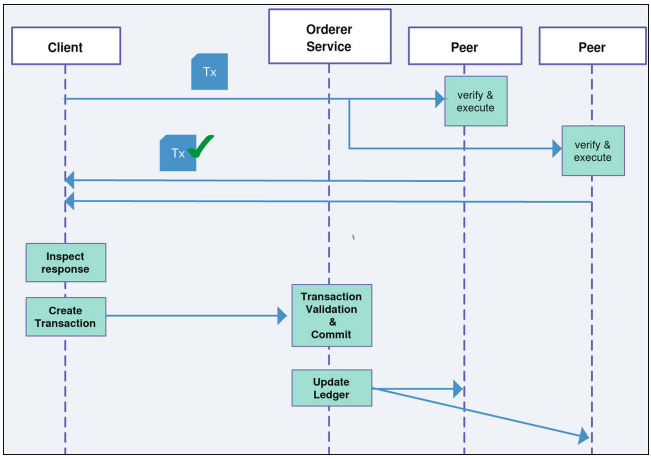


Fig. 6. A sequence diagram of the typical transaction processing steps in Hyperledger Fabric (see also [2])

In Fig. 6 we show the typical steps of a transaction processing of the blockchain following the execute-order-commit pattern which is applied by Hyperledger Fabric [2]. The executed steps of a transaction initiated by the sample of collected personal e-health data records, their analysis and storage are as follows:

- *Step-1:* A client sends a transaction request to the endorsing peers of the blockchain.
- *Step-2:* An endorsing peer verifies the signature of the requestor and executes the indicated transaction related to the collected data.
- *Step-3:* A client receives the response from the endorsing peer.
- *Step-4:* The client inspects the response and assembles the endorsements into a transaction.
- *Step-5:* The client sends the endorsed transaction to the orderer service.
- *Step-6:* The transaction is validated and committed by the orderer service.
- *Step-7:* The orderer service updates the ledger and broadcasts the information to all the other peers in the P2P-network of the blockchain.

In Fig. 7 we have depicted how the creation process of a typical transaction of the blockchain is augmented by a new collected data stream and its hashed reference value, respectively. The Hyperledger Fabric client (or Hyperledger wallet) is started by showing its interest in a particular data stream of an IoT-end device and its associated fog node, respectively. For this purpose a subscribe request is issued to the data broker in a fog gateway. Then the data broker only forwards this stream to the Hyperledger client which has previously subscribed for this stream. The Hyperledger client receives the request and checks for particular, configurable criteria to be met. Then it requests the creation of a corresponding transaction. From this point on every step proceeds as shown in Fig. 6.

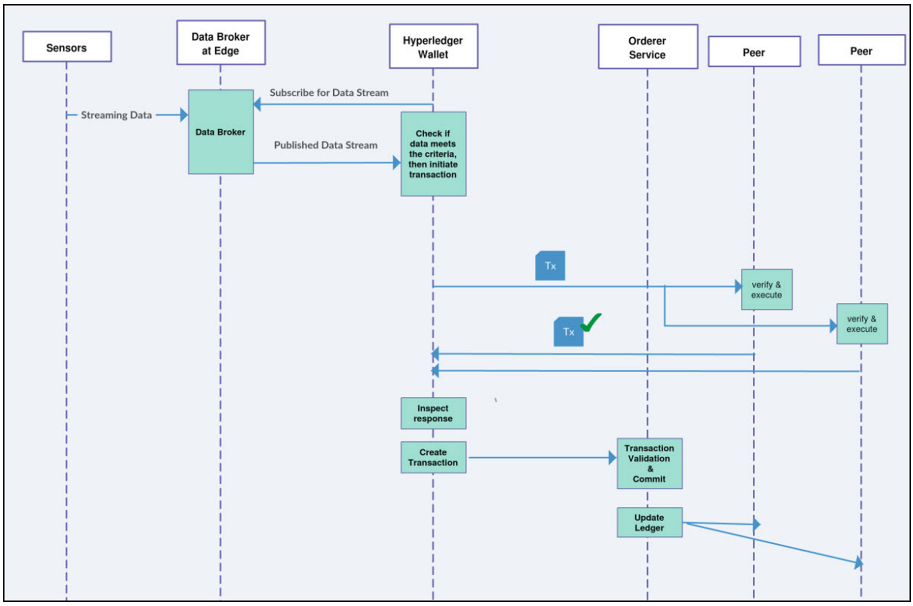


Fig. 7. Workflow of the Hyperledger Fabric starting from an end device with its attached sensors and its associated fog gateway with the data broker functionality directed towards the cloud (see also [2])

Our goal is to provide an effective implementation of this integrated fog-blockchain architecture within a test bed of cheap SBCs and to evaluate its achieved performance with respect to important computing and networking metrics. In this manner we are able to validate the effectiveness and efficiency of our design decisions.

4 Implementation of the Integrated Fog-Blockchain Prototype by the Hyperledger Fabric Platform

We have realized our prototype of the integrated fog-blockchain architecture in a virtualized test bed using appropriate JavaScript elements and related basic Docker images and Docker compose files for all basic elements of our generic software system. By these means we have implemented the sketched integated fog-blockchain architecture with the help of Hyperledger Fabric, version 2.0, cf. [2]. In our feasibility study we closely follow the well-known Hyperledger Fabric samples and trim it to serve our experimental purposes. In this section we describe the object structures and functions of our implementation in detail to enable the reproducibility of our experimental study.

To conduct related performance experiments in our test bed, the implementation requires to set up a minimalistic network scenario involving end devices. The latter simulate the emission of sensor data. A fog computing gateway is used

as their logical counterpart, and a node with the Hyperledger blockchain setup is employed as cloud server. The latter framework offers various services and pluggable consensus mechanisms with the ability to write customized applications in multiple languages. The first requirement is to set up a basic P2P-network by defining the organizations, the orderer service, an optional Certificate Authority (CA) and a Membership Service Provider (MSP) which is also optional.

As shown in Fig. 5, two different organizations *Org1* and *Org2* are used in our test bed. Each of these entities contains two peers, called Peer0 and Peer1 here. Peer0 is the endorsing peer from each organization that is involved in the consensus process and Peer1 adopts the role of a validator. An endorsing peer is also regarded as an anchoring peer that is responsible for the cross-organizational gossiping, cf. [2]. It is the role of a CA to work as a trusted third party by providing digital certificates to the organization's peers and the orderer service. In our tests the use of a CA has been replaced by the manual invocation of the *cryptogen* tool of Hpyerledger Fabric. All necessary certificates which are required for a secure communication between the orderer and the peers are created by this *cryptogen* tool, see [9]. Once the necessary organizations, certificates and configuration files are generated, the *configtxgen* function is used to create the genesis block of the freshly instantiated blockchain, cf. [9].

4.1 Implementing the Peer-to-Peer Network for Hyperledger Fabric

In our implementation a *network.sh* script is used to set up the peer-to-peer network for Hyperledger Fabric. This script *network.sh* is utilized in the script *startFabric.sh* to establish the blockchain's peer-to-peer network, to create channels, to deploy the required chaincode using the script *deployCC.sh* and to test the generated transactions. The script *startFabric.sh* also specifies the employed database and tells the peers which chaincode language is to be used (see *configtxgen* in [9]). The script *envVar.sh* provides the environment variables of the test bed. An overview of all the required scripts and their calling scheme can be seen in Fig. 8.

During the network setup procedure the *network.sh* script creates the necessary Docker containers for the orderer and the involved peer hosts. To run the *orderer service* depicted in Fig. 7, a Docker image for a `hyperledger/fabric-orderer` is used and started by an *orderer* command. Similarly, a Docker image for the needed `hyperledger/fabric-peer` is used to run inside a peer and started by a *peer node start* command. A Docker image for *eclipse-mosquitto* is applied as the MQTT-based *data broker* in our test bed, cf. [1].

4.2 Realizing the Hyperledger Fabric Client

A Hyperledger Fabric client, also known as Hyperledger wallet, has been defined as depicted in Fig. 6. Two scripts *enrollAdmin.js* and *registerUser.js* are used to create the *Administrator* role and the user instances of the blockchain network, see Fig. 9. The *Administrator* of the network is a single built-in peer whereas

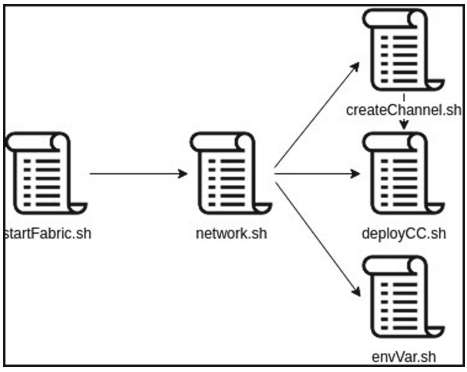


Fig. 8. Created scripts of the fog-blockchain prototype and their interactions in our test bed

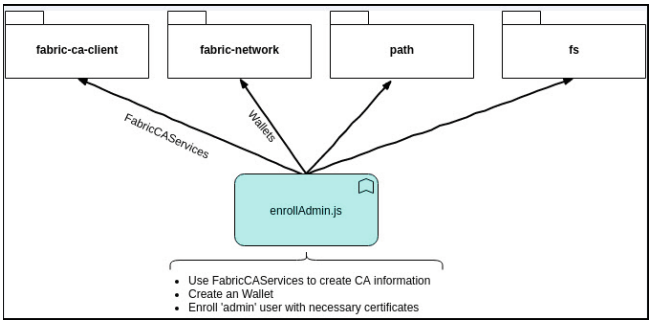


Fig. 9. Overview on the creation script of the admin role

the number of users in a network is configurable. A required *Administrator* id is created by taking all necessary parameters from a specification file *connection-org1.json*, which is generated during the network setup process. This *Administrator* id is used to create the other users of the permissioned blockchain. All the credentials of the users are saved under a *wallet* directory. All the operations in the P2P-network require an *appUser* id from the *wallet*.

The implementation of the Hyperledger client is realized by a JavaScript *pushData.js*. It uses the *fabric-network* module to talk to the fabric network and an *mqtt* server module for its communication with the data broker, cf. [1,2]. Figure 10 shows the imported packages used in *pushData.js*. By standing in the middle it shapes the behaviour of the realized application. It also decides when to create a transaction and which data stream should be kept in the blockchain. In our current test bed implementation the logic for filtering a data stream is hardcoded. It checks the values of the streamed data to see if it passes a certain threshold and only then it pushes the stream to be part of the ledger. It also uses an *appUser* profile before submitting a transaction within the blockchain.

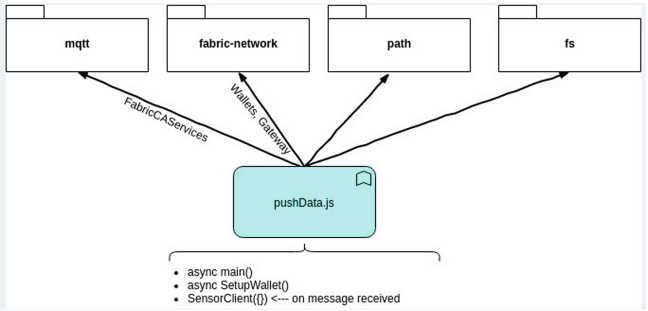


Fig. 10. Design of the hyperledger wallet

At the data stream emission site a *SensorClient* is realized in terms of an *mqtt* client of the data broker service in the fog layer. Hence, it receives all messages from the data broker. Due to its event-driven working mechanism it gets called each time when it receives a message. After executing some threshold checking it calls an asynchronous function *SubmitTransaction* which uses a *globalContract* instance and invokes a *createStream* with necessary parameters. *globalContract* is an instance of the chaincode, which was deployed during the network creation phase. In our test case it is called *IoTStream*, see Fig. 11.

4.3 The Chaincode for Handling the Sensor Data

In a Hyperledger Fabric blockchain the chaincode provides a key functionality. It covers the ability to write a customized application logic using popular programming languages. Basically, it is a container holding multiple smart contracts with a unique identifying name, cf. [2,9]. It allows application developers to concentrate more on developing their business logic rather than concentrating on the complexity of the underlying blockchain network.

In our test bed and its experiments the JavaScript language has been picked for the development of a smart contract and *node.js* has been used as coding framework. In Hyperledger Fabric there are mainly two primary package interfaces for developing smart contracts, namely *fabric-contract-api* and *fabric-shim*, see [15]. The *fabric-contract-api* provides a *Contract* interface and *fabric-shim*, on the other hand, requires to write a smart contract class mainly having two callable functions, namely *Init* and *Invoke*. A defined class needs to be called from *shim.start()*. In our test bed the *fabric-contract-api* has been picked. The developed smart contract is simple and planned to handle data streams by means of a key-value pair where the key is the *timestamp* and the value is the *stream* of the gathered IoT-sensor data set itself.

In our implementation we use the package *fabric-contract-api*, which contains all the contract related functionalities and creates a class called *IoTStream* by

extending the class *Contract*. Figure 11 shows the class diagram of the written contract. It depicts an overview of this class diagram that handles the transaction data.

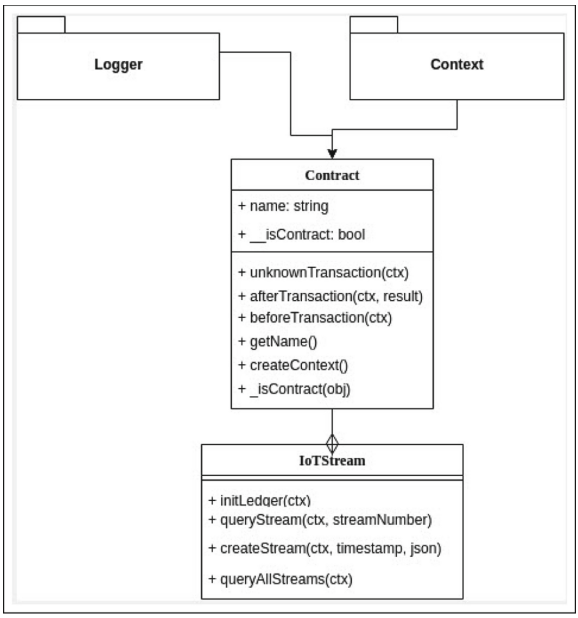


Fig. 11. A smart contract as a key element of an application development based on Hyperledger Fabric

In our experiments the *initLedger* function puts a dummy element into the ledger during the initialization of the smart contract. The class *IoTStream* contains three other functions for creating streams, querying a particular stream and querying all the streams. The prerequisite for creating a stream is to know the timestamp or the range of the timestamp, i.e., start and end epochs.

The listing in Fig. 12, for example, shows the relevant code for creating a stream.

Inside the procedure it calls the *putState* function to store the values. A JSON-string is created from the parameters passed to the function and put as a value to the *putState* function. The *queryStream* takes the timestamp as parameter and returns that specific stream. *queryAllStreams* returns all the streams stored so far. One of the limitations appearing in *putState* is that all the inserted values are stored and must obey a key-value pair style. However, *putState* does not have any support for multi-mapping of the key-value pair so far.

```
async createStream(ctx, ctime, desc, value, lat, lng, type, id, unit) {
 const stream = {
      desc,
      docType: 'stream',
      value,
      lat,
      lng,
      type,
      id,
      unit,
 };
 await ctx.stub.putState(ctime, Buffer.from(JSON.stringify(stream)));
}
```

Fig. 12. Sample code for creating a stream for Hyperledger Fabric

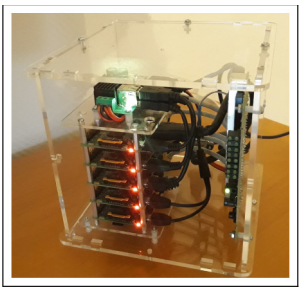


Fig. 13. A Hypriot empowered RPi3 based cluster

5 Performance Evaluation of the Fog Computing System with Its Integrated Blockchain Functionality

In our prototype fog gateways with the Hyperledger Fabric blockchain have been realized by cheap Raspberyy Pi SBCs. To analyze the efficiency and limitations of this implemented fog-blockchain architecture, at least three different entities have to be considered during an evaluation of the resulting performance indices, namely end devices, fog gateways and blockchain peers. To simulate an end device, first a script is used to generate sensor data. Typically, IoT-end devices of smart home applications are known to communicate using MQTT, cf. [1]. Our experimental test system simulates the end devices' behaviour at various speeds to create the transaction load of the blockchain within our virtualized fog computing environment. A cluster of Raspberry Pis (RPis) based on the HypriotOS [7] operating system is used as fog gateways with virtualized functionalities. The test bed server has been powered by a x86_64 based Quad Core machine enabling hyperthreading (HT) to host the Hyperledger Fabric blockchain. Various configuration details of our test bed are outlined in Table 3.

Figure 13 shows the fog gateway that has been used in the experiments of the test bed. The RPi cluster is made of five RPis. They use ARM processors which are very suitable for a lightweight fog gateway. The RPis are also known for their low power consumption. Each of these used fog gateways has a 64 bit

Table 3. Hardware and software specification of the test environment

Properties	Fog Gateway	Server
OS	Hypriot	Ubuntu
CPU	ARMv7 rev4@1.4 GHz	x86_64@1.6 GHz
Docker	18	18.06.3-ce
Kernel	4.14.34	5.4.0-rc5
Core	Quad Core	Quad Core + HT

Quad Core ARM processor with a clock speed of 1.4 GHz. Our prototypical fog gateway is running HypriotOS as kernel, cf. [7]. The latter is a GNU/Linux based operating system derived from Debian which allows it to run containerized applications. HypriotOS based RPi computers appear as a prime candidate of a cheap fog node as they make the cluster effective regarding a smooth application deployment. They can also offer suitable monitoring capabilities.

On the other hand, the server system used in our test bed is backed by an Intel based 64-bit processor with 16 GB memory running Ubuntu 18.06 as operating system. An enterprise grade server would come with different grades of processors. But regarding the experiments of our feasibility study, we only try to understand the suitability of our design settings. We shall illustrate that the performance results inferred from our experiments can be multiplicatively applied to more powerful servers.

To set up a peer-to-peer test network of the blockchain all the involved devices have been connected within a single communication network. Each data generator of a sensor and its fog gateway are connected via a hub over a Gigabit Ethernet interface. The server hosting Hyperledger Fabric has been connected to this network via a rather slow wireless interface. In a real world environment it is expected that a sensor network is connected directly to the fog gateway without an intermediary and by a relatively low latency connection compared to the more powerful connection from a fog gateway towards the cloud.

To properly utilize the fog computing gateway, an overlay network has been created within the blockchain topology of Hyperledger Fabric using *Docker Swarm*, cf. [5]. Running Docker in swarm mode requires a declaration of a *manager* node and some *worker* nodes. The *manager* node is responsible for distributing the transaction load of the blockchain amongst all *worker* nodes. In our test bed five client instances were used to generate sensor data streams. Each client's transaction requests were load balanced via the *swarm manager* node amongst all the available worker nodes. Likewise, at the server end five instances of a Hyperledger client were created and each of them started to receive transaction requests from all of these cluster nodes. The clients were generating data at an overall rate of 100 packets/second where each instance generated data at a rate of 20 packets/second.

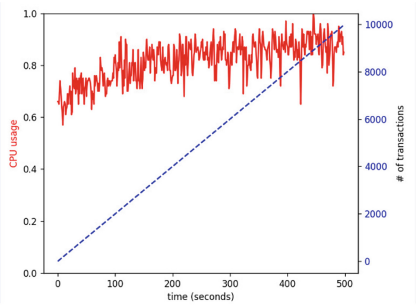
Based on this described setup of the fog-blockchain prototype within our simple test bed, CPU usages were measured at the fog gateway nodes. A snapshot

of these CPU usages during the experimental evaluation can be seen in Fig. 14. The experiments are showing how the block creation timeout value impacts on the CPU usage of a gateway. The CPU usage has been scaled in a window of 0.0 to 1.0. The greater the timeout value is, the more visible are the CPU usage spikes due to the time difference regarding the creation of transactions. The oscillation of the CPU utilization indicates that the CPU of a gateway is not utilized to its fullest level. Being in a container environment within an operating system based on a Linux kernel, which only allocates a portion of the adjustable CPU capacity to the containers, it can only utilize a part of the whole physical CPU capacity. Therefore, the interpretation of the evaluation is a relative measure. Moreover, the peers running inside a deployed Docker container are not CPU intensive. Thus, the transaction requests were not utilizing the CPU to its fullest level. Apart of that, the results also depend upon the synchronization characteristics of the containerization environment which requires that they may also wait for resources other than the physical CPU such as the memory and I/O capacities.

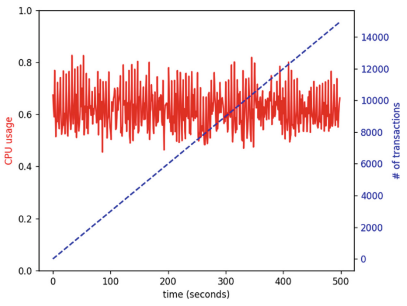
In our performance experiments the CPU usages have been gathered from the *sysfs* interface of our fog computing system. *sysfs* is a RAM-based filesystem which is used to export low level details regarding various system resource usages for user space applications as well as debugging. Reading data from *sysfs* has very low overhead compared to running system monitoring software like cAdvisor [3]. *sysfs* also allows us to pick up only the necessary information. In our case only memory and CPU usages of a fog gateway were needed. To gather these memory and CPU usage values a simple bash script has been used. This script sleeps for 1 s, then wakes up and dumps the CPU usages from `/sys/fs/cgroup/cpu/cpuacct/cpuacct.usage` into a file. The CPU usage values exposed via *sysfs* are recorded on a nanosecond scale. The formula used to obtain the CPU utilization is given by *v1-v0/(1000000000*cpunr)*. Here *v0* and *v1* are the values read at time *t0* and *t1*, whereas *cpunr* is the number of used CPUs within the physical system of a fog node. Similarly, the memory usage values were taken from `/sys/fs/cgroup/memory/memory.usage_in_bytes`.

Let us look at a typical experiment within our fog-blockchain test bed. Applying a block timeout of 2 s for a block creation step during the runtime of 365 seconds, 13000 transactions were created and amongst them 7500 were successful. The maximum limit was 10 messages per generated block of the blockchain. All the transactions were write-only transactions. In this setting it was possible to handle approximately 20 transactions per second on a RPI fog gateway. A failure in transaction handling indicates a timeout event received from the peers before a commit event of the blockchain. When the timeout value is 5 s, is has been possible to process all the transactions. In order to achieve this outcome, changes by increasing the key parameters *maximum message count* and *preferred max size* of our transactions were executed.

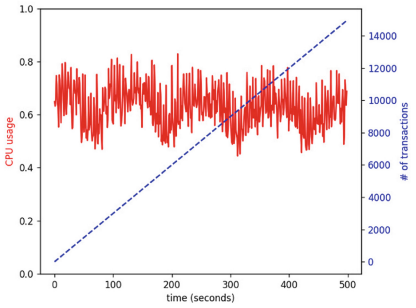
Table 4 shows the key configuration options for each timeout value. A lower block timeout value provides low latency, but it may not scale well. To achieve scalability, an increasing timeout value is necessary. This strategy is a trade-off between latency and scalability within a RPi fog gateway which needs to be addressed based on the requirements of a smart home application.



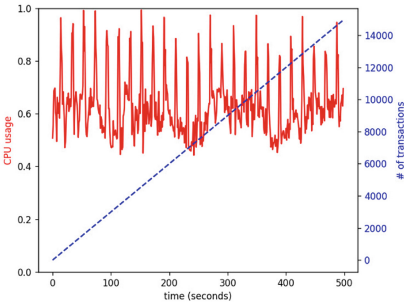
(a) Timeout 2 seconds



(b) Timeout 5 seconds



(c) Timeout 10 seconds



(d) Timeout 20 seconds

Fig. 14. CPU usages with different timeout values over 500 s with the same amount of load

Our experiments show that other factors to achieve an adequate throughput of approved transactions in Hyperledger Fabric are determined by the size of a block and the state management. The size of a block depends upon the application logic, the timeout value and the limit regarding the number of transaction messages which can be incorporated into a block of the blockchain. To achieve a satisfactory throughput, necessary changes can be made to these parameters which depend upon the application requirements. Our experiments only deal with a few variations in the timeout and the message limit size.

Regarding the state management two primary databases are used in Hyperledger Fabric, namely *LevelDB* and *CouchDB*, cf. [9]. Although, LevelDB is well known for an adequate throughput, the performance of CouchDB has been increased since the deployment of Fabric version 2.0. Thus, we have used CouchDB for the state management and Hyperledger Fabric version 2.0 has been applied during the evaluation phase of our settings. Moreover, CouchDB

Table 4. Configuration options for the transactions per second (TPS) with various timeout (seconds) and message size parameters

Timeout (sec)	Preferred Max. Size (KB)	Max. Message Count	TPS
2	512	10	20
5	8192	10000	200
10	8192	10000	200
20	8192	10000	200

provides better query features which is an important aspect for developing smart home applications.

To figure out how much load the Hyperledger server can handle, a separate load generation technique has been used. The sensor data have been generated in a gradually increasing order, each end device instance has been generating 20 transactions per second. The intensity of each of these transaction instances has been increased in a close to exponential back-off algorithm. But they differ in terms of choosing the inter-arrival time between two transaction instances which was driven by the load intensity. The number of instances can have a highest value of thirty units. From each transaction instance to create a next instance, the delay between two instances is driven by the number of instances currently running. After the first instance and before creating the second instance the delay is one second. The delay slot is like $[1, 2, 3, 4, ..., N]$ where N is the highest number of transaction instances. For this experiment N is set to 30, although CPU usages reached the limit even before that and 30 threads were not really used. The delay is given in terms of seconds and picked sequentially. So, after the first instance, it will wait for one second, after the second instance it will wait for two seconds and so on until it reaches its limit. In this way it would be possible to find out the amount of transaction load which is required to fully utilize the CPU. To generate the load in this way, the sensor data generator script *mqttgen.py* has been modified to enable a multi-threading, which is configurable via the configuration file *config.json*. Figure 15 shows the CPU usages during the runtime of this transaction load. Hyperledger Fabric was configured with a timeout of 5 s. The load was generated in a gradually increasing order and the transaction rate per second has been indicated by the blue line. It is shown that it took 80 s to reach this peak utilization.

Figure 16 shows the corresponding memory usage of the server in the test bed. The blue line indicates the transaction rate per second. The server ran out of memory during this operation and the Out-of-Memory (OOM) [8] manager got into an action to rescue the situation. This outcome is indicated by a drop in the memory usage graph.

Figure 17 shows the default network usage statistics in terms of the number of kilobytes sent and received over the bridged P2P-network of the blockchain. Here cAdvisor [3] has been used to collect the network statistics.

These experiments can only indicate the feasibility of our proposed system design. More extensive validations of the integrated fog-blockchain architecture

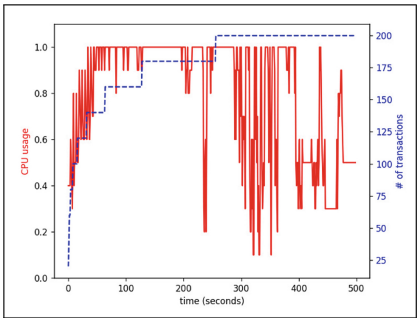


Fig. 15. The time it took for the server to get to its peak usage scenario in the experimental setup is shown.

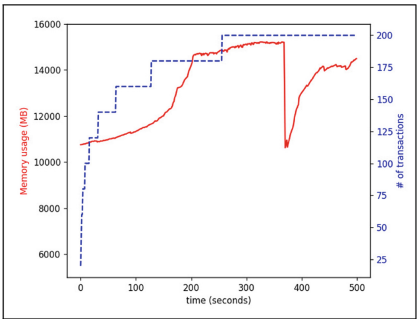


Fig. 16. Memory usages (in megabytes) snapshot during the evaluation phase

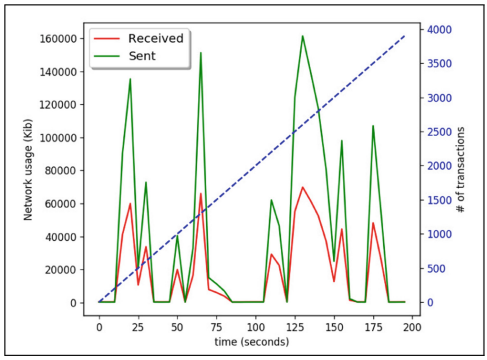


Fig. 17. Network usages (in kilobytes) during the evaluation phase with default timeout 2 s

with real-life smart home and e-health applications are required to assess the efficiency and scalability aspects of our design decisions in more realistic scenarios.

5.1 Limitations of Our Experiments

Measuring the relevant performance indices of a blockchain framework in a fog computing environment is challenging, cf. [21]. The first problem is to simulate the fog computing infrastructure properly. In such a scenario typically multiple entities are involved and they are connected to each other via different types of network connectivity. In a real world this connectivity variation can be very diverse. Measuring the system performance based on CPU and memory usages is only one part of the whole performance analysis setting. The delay along the used diverse communication channels will also severely impact on the performance measurements of a blockchain, cf. [12].

As the blockchain is an emerging technology, all the applications and tools surrounding this framework are under rapid development. The tools for measuring the performance are also not mature enough right now. Furthermore, due to the involvement of multiple components over a P2P-network of varying latency, it would require a different set of tools for analyzing the network performance indices. This situation generates a complex scenario to collect reliable numbers and to draw solid conclusions. Due to all these limitations, the results excerpted from our experiments can only be used for an indicative purpose, rather than being an absolute measure of the fog node performance.

6 Conclusions

Nowadays, new advanced Internet-of-Things applications are required to support the effective processing of smart home data, e.g., arising from HVAC applications, cf. [1]. Other examples are given by smart e-health applications like ambient assisted living services which may support diagnostic and therapeutic healthcare protocols for elder people in a modern society, cf. [10]. To cope with privacy and security issues of these data collected by advanced sensors in such an advanced IoT-infrastructure dedicated to these specific application areas, the integration of blockchain technology into a fog computing architecture has been considered recently, cf. [18,20]. It can provide a feasible way to realize a secure distributed data processing and storage architecture which is required by this advanced fog computing approach.

Regarding the IoT-application scenarios for smart homes, the basic research question arises how a resourceful, efficient design of an integrated fog-blockchain system based on an extensible virtualized open source software architecture should look like. We have investigated these technical issues by means of a feasibility study and developed a lightweight software architecture. It integrates a set of fog gateways and the blockchain technology, cf. [4]. We have implemented this distributed architecture by means of Hyperledger Fabric [2] in a test bed of cheap Raspberry Pi SBCs as basic fog nodes of an underlying peer-to-peer network in such a private, permissioned blockchain. The key issue whether these fog nodes with their virtualized functions based on a Docker orchestration framework can meet the required computing and networking goals has been addressed in terms of a series of performance tests within our test bed. Our experiments have revealed the feasibility of the proposed software architecture as well as its benefits regarding potential smart home and smart e-health applications. However, theses tests have also illustrated the intrinsic performance limitations of the developed fog-blockchain architecture if simple SBC systems like Raspberry Pis are used for the implementation.

Further performance studies are needed to evaluate the full potential of the developed fog-blockchain approach for the secure provisioning of required e-health data, for instance, to collect the blood pressure and glycose values of elder patients with diabetes mellitus at their homes or for new healthcare services in smart cities. Moreover, the scalability and efficiency of the adopted three-layer architecture should be investigated more carefully by several performance

experiments within an extended real-life test bed. Furthermore, the adoption of new federated learning techniques to analyze the collected data at the fog or cloud layer and to trigger immediate actions if needed may be an interesting extension of our distributed edge computing system. These technical issues shall become a subject of our future research.

Acknowledgment. This feasibility study was done while Mr. Mullick was working in the Computer Networks group at the University of Bamberg. The other authors are very much indebted to his efforts in implementing a first prototype of the integrated fog-blockchain architecture with Hyperledger Fabric.

References

1. Al-Fuqaha, A., et al.: Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Commun. Surv. Tutorials **17**(4), 2347–2376, Fourth Quarter (2015)
2. Androulaki, E., et al.: Hyperledger fabric: A distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference, ser. EuroSys '18, New York, NY, USA, pp. 30:1–30:15, ACM (2018)
3. cAdvisor (Container Advisor). https://hub.docker.com/r/google/cadvisor/
4. Cech, H.L., Großmann, M., Krieger, U.R.: A Fog Computing Architecture to Share Sensor Data by Means of Blockchain Functionality. In: IEEE International Conference on Fog Computing (ICFC) 2019, pp. 31–40 (2019)
5. Docker Engine: Swarm mode overview. (2023). https://docs.docker.com/engine/swarm/
6. 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)
7. Fichtner, G., et al.: HypriotOS. https://blog.hypriot.com/
8. Gorman, M.: Understanding the Linux Virtual Memory Manager, Chapter 13 Out Of Memory Management. https://www.kernel.org/doc/gorman/html/understand/understand016.html
9. Hyperledger: Hyperledger-Fabric - A Blockchain Platform for the Enterprise, (2023). https://hyperledger-fabric.readthedocs.io/en/release-2.5/
10. Islam, S.M.R., Kwak, D., Kabir, M.H., Hossain, M., Kwak, K.S.: The internet of things for health care: a comprehensive survey. IEEE Access **3**, 678–708 (2015)
11. Kreutz, D., et al.: Software-Defined Networking: A Comprehensive Survey. In: Proceedings of the IEEE, vol. 103, no. 1, pp. 14–76, January (2015)
12. Krieger, U.R., Ziegler, M.H., Cech, H.L.: Performance modeling of the consensus mechanism in a permissioned blockchain. In: Gaj, P., Sawicki, M., Kwiecień, A. (eds.) CN 2019. CCIS, vol. 1039, pp. 3–17. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21952-9_1
13. Li, C., Zhang, L.: A blockchain based new secure multi-layer network model for internet of things. In: IEEE International Congress on Internet of Things (ICIOT) 2017, pp. 33–41, June (2017)
14. McGhin, T., Choo, K.-K.R., Liu, C.Z., He, D.: Blockchain in healthcare applications: research challenges and opportunities. J. Netw. Comput. Appl. **135**, 62–75 (2019)
15. npm.community, fabric-contract-api, (2020). https://www.npmjs.com/package/fabric-contract-api

16. Poon, J., Buterin, V.: Plasma: Scalable Autonomous Smart Contracts, (2017). https://plasma.io/plasma.pdf
17. Stadt Bamberg, Stabsstelle Smart City, 96047 Bamberg, Germany. https://smartcity.bamberg.de/ueber-das-programm-smart-city-bamberg/
18. Tuli, S., Mahmud, R., Tuli, S., Buyya, R.: Fogbus: a blockchain-based lightweight framework for edge and fog computing. J. Syst. Softw. **154**, 22–36 (2019)
19. Vukolić, M.: The quest for scalable blockchain fabric: proof-of-work vs. BFT replication. In: Camenisch, J., Kesdoğan, D. (eds.) Open Problems in Network Security, pp. 112–125. Springer International Publishing, Cham (2016)
20. Yang, R., Yu, F. R., Si, P., Yang, Z., Zhang, Y.: Integrated Blockchain and Edge Computing Systems: A Survey, Some Research Issues and Challenges. IEEE Commun. Surv. Tutorials, **21**(2), 1508–1532, Secondquarter (2019)
21. Ziegler, M.H., Großmann, M., Krieger, U.R.: Integration of Fog Computing and Blockchain Technology Using the Plasma Framework. In: IEEE International Conference on Blockchain and Cryptocurrency (ICBC) 2019, pp. 120–123 (2019)