Received: 28 June 2022 | Revised: 2 November 2022 | Accepted: 8 November 2022

DOI: 10.1002/spe.3173

RESEARCH ARTICLE

WILEY

Exploring the use of blockchain in resource-constrained fog computing environments

Miguel Sánchez-de la Rosa[1] | **Carlos Núñez-Gómez**[1] | **M. Blanca Caminero**[2] | **Carmen Carrión**[2]

[1]High-Performance Networks and Architectures Group (RAAP), Albacete Research Institute of Informatics (I3A), University of Castilla-La Mancha, Albacete, Spain

[2]Department of Computing Systems, University of Castilla-La Mancha, Albacete, Spain

Correspondence
Miguel Sánchez-de la Rosa, High-Performance Networks and Architectures Group (RAAP), Albacete Research Institute of Informatics (I3A), University of Castilla-La Mancha, 02071 Albacete, Spain.
Email: miguel.sanchez@uclm.es

Funding information
Agencia Estatal de Investigación/Ministerio de Ciencia e innovación/European Regional Development Fund, Grant/Award Number: PID2021-123627OB-C52; Ministerio de Ciencia e Innovación; Junta de Comunidades de Castilla-La Mancha (Regional Government), Grant/Award Number: GC-020-017

Summary

Fog computing has become a complementary technology to cloud computing and addresses some of the cloud computing threats such as the response time and network bandwidth demand. Fog computing successes processing data and storing data near to the edge, and usually is combined with container virtualization to provide hardware isolation. Empowered by these capabilities, numerous Internet of Things (IoT) applications are developed as virtualized instances on resource-constrained fog nodes such as single-board computers (SBC). In addition, blockchain has emerged as a key technology that is transforming the way we share information. Blockchain technology represents a decentralised, distributed, and immutable database ledger and is a potential solution for the distributed ecosystem of IoT applications. The distributed structure of blockchain is naturally suitable for IoT applications. However, it introduces new challenges related to CPU overhead or response time. This paper proposes a layered architecture that integrates blockchain technology and OS-level virtualization technology to develop fog-based IoT applications. It also provides insights for future deployments through a proof-of-concept use case harnessing SBCs, in this case Raspberry Pi, as blockchain-enabled fog nodes to drive virtualized IoT applications. The study shows that the maximum CPU overhead added by a permissioned blockchain based on Ethereum on the Raspberry Pi is around a 25% under stress situations while the overhead introduced by the sealer process is negligible. These results support the feasibility of using blockchain on resource-constrained fog nodes for supporting IoT applications.

KEYWORDS

blockchain, fog computing, internet of things, smart contracts, single-board computers

1 | INTRODUCTION

Fog computing emerges as a complementary technology to cloud computing, which aims to shorten the distance between the computation and storage resources from the location where data are actually generated.[1] This allows response time

reductions and bandwidth optimizations, which may result to be critical in numerous Internet of Things (IoT) applications. IoT environments respond to both a digitization of the most worldly aspects of life (i.e., household appliances) and industrial data-gathering environments (i.e., sensors medical facilities or factories).[2] These processes, individually, may not be as heavy as to be processed in a regular server. Thus, popular single-board computers (SBC) can come in handy as a replacement and stay connected via connectivity technologies like Wi-Fi or Bluetooth Low Energy (BLE).

Single-board computers (SBC)-based platforms can off-load both processing and bandwidth when communicating with cloud providers, apparently bringing those remote services closer to the sources of information, just like fog looks like clouds closer to the ground. Moreover, the information generated by IoT sensors can be preprocessed by low-cost SBCs, such as inexpensive Raspberry Pis (RPies) that employ ARM64 quad-core processors with up to 8 GB of RAM, while consuming a maximum of 15 W. This is important for economy of scale and geographical distribution.[3]

Concurrently, cryptocurrency technology appears through the warmth of blockchain technology,[4] providing a highly-distributed transaction ledger, initially applied to support economic transactions, although multiple applications have arisen in several areas. Blockchain offers useful properties such as record immutability, data transparency, decentralization, and high availability of records. Upon this concept, smart contracts[5] were conceived, which specify code stored in the blockchain to get executed when some specific event occurs. Smart contracts are executed independently in all nodes that participate in the blockchain consensus mechanism, which can be seen as a distributed computer. A set of smart contracts executing on a blockchain platform is usually referred to as a Decentralized Application (DApp).

In this work, blockchain technology is used to develop reliable IoT applications in fog environments. A layered architecture that integrates the blockchain technology and the OS-level virtualization technology for fog environments is presented. The proposal includes a simple blockchain-based identity and authentication mechanism for IoT devices and users, that enhances both data security and data transparency while providing different levels of privilege for data access. Moreover, the portability of the application is improved thanks to the use of OS-virtualization. In particular, Docker-based containerization of the blockchain-based IoT application is harnessed to allow agile and efficient management of the software system as well as a reproducible execution environment.

The general objective of this work is to explore the viability of using the blockchain-based IoT application into fog computing environments, focusing on a scarce-resource implementation of the nodes that compose the fog. The analysis is made on the basis of the deployment of a DApp on a permissioned (private) Ethereum blockchain. After assessing the functionality of the smart contracts, non-functional parameters, such as transaction latency or system throughput, are evaluated in order to estimate the limits for applying this technology in low-cost fog computing environments.

The rest of the paper is organized as follows. Section 2 briefly reviews some basic concepts and technologies, necessary to focus the rest of the paper. Section 3 summarizes the contributions found in the literature with goals similar to this research. The reference architecture for this work is described in Section 4. In Section 5, the smart contracts that compose the DApp considered for the case study, which emulate a generic IoT application, are specified. Section 6 details the implementation of the realistic tested where the DApp has been deployed, while Section 7 summarizes the experiments carried out to evaluate the system. Finally, some conclusions and guidelines for future work are included in section 8.

2 | BACKGROUND

In this section, a brief overview of the technologies that support this work is presented, namely, fog computing and blockchain.

2.1 | Fog computing

Applications that make use of cloud environments may not be tolerant to latency or jitter introduced by data interchanged with the cloud. In fact, having to transfer all the raw generated data from sensors or more complex IoT devices may be less efficient than processing that data locally, just before transferring relevant data to the cloud, thus the terms

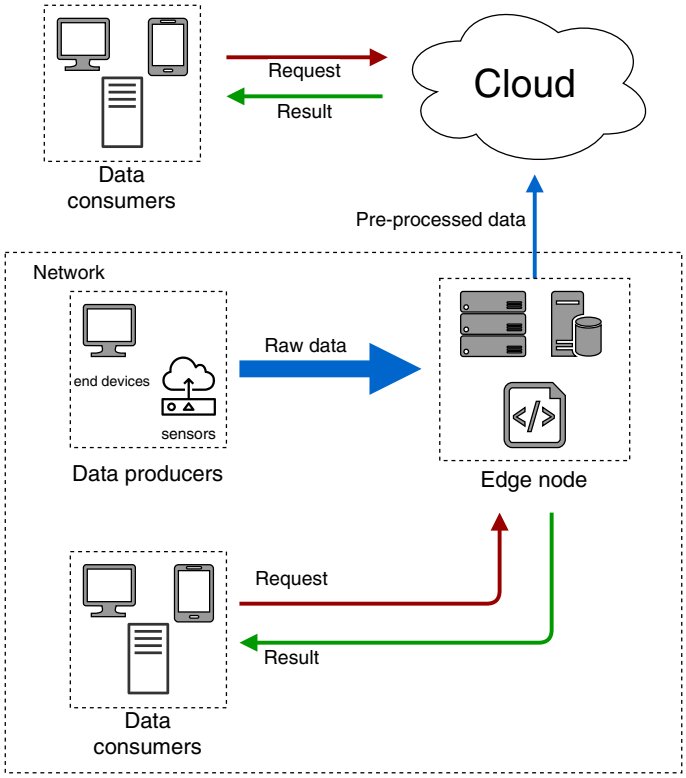

FIGURE 1 Edge computing architecture

edge and fog computing arise (see Figure 1). This is especially noticeable as the processing power of the cloud outclasses any capability that the devices located elsewhere may have; however, the bandwidth to transfer data can become a bottleneck.[6]

Fog computing is an approach to cloud computing that tries to complement its service models* and advantages, such as elasticity and scalability, in order to mitigate the latency due to the distance between the cloud servers (geographical distribution can provide location awareness) and the devices that generate raw data that may end up going to the cloud.[7] A *Fog node* is an intermediate device that produces local processing on the generated information, which results in the upstream data that goes on its route to the Cloud, offloading both traffic (thus also bandwidth) and processing from cloud servers.

Fog computing implies that the data produced in the end devices is to be processed or pre-processed, contributing to the economy of scale that the pay-as-you-go model of the cloud provides; mainly to reduce activity and raw data sets traveling through the network, on its way to cloud services. This obviously requires some provisioning of resources outside of the cloud, in devices at the edge of the network.[8] This would allow for latency-sensitive elements to be processed locally, for example, for SLA (Service Level Agreement)† compliance, while other tasks which are tolerant to delay or computationally intensive take place in the cloud.

Fog/edge computing is usually related to IoT, because it may be needed for some IoT applications where the cloud paradigm is not feasible.[9]

2.2 | Blockchain

A blockchain is a distributed ledger that allows and records transactions between peers without the need of an intermediary party (mainly, this party is expected to be trusted, unbiased and centralizes the control of the transactions). This,

*Be them infrastructure as a service (IaaS), platform as a service (PaaS), or software as a service (SaaS)

†Commitment between a service provider and its client, which measured in parameters such as mean time to repair (MTTR), mean time between failures (MTBF), or data throughput or jitter; among others.

by definition, makes the network that interacts with the ledger to be purely peer-to-peer and trust-less. It is commonly associated with the Bitcoin cryptocurrency,[4] since it was the first public implementation of blockchain, although it has overcome that initial purpose to reach much wider and diverse application domains.

Blockchain technology is capable of storing data in a distributed manner ensuring data integrity and availability. Underlying peer-to-peer (P2P) networks are used to connect participating nodes together without the need for third parties. A blockchain consists of a data structure that encapsulates data inside transactions. Then, these transactions are sent throughout the P2P network in a broadcast process and grouped into blocks. These blocks are chained together by means of a cryptographic hash function and ordered in time, forming a large database or distributed ledger. A blockchain can be formally defined as a global state machine where transactions are the transitions between states.

Blockchain can be classified as permissionless or permissioned blockchains.[10] Permissionless blockchains allow any entity around the world to participate, which decentralizes the control and maintenance of the blockchain. In addition, the stored information is public and can be consulted by any entity, which increases transparency. Permissioned blockchains, on the other hand, are focused on local and private use cases where higher performance, control, and privacy are required. Only authorized entities can participate in a permissioned blockchain. This means that the control of the blockchain is less distributed, but improves the scalability and network speed.[11]

As blockchains broadcast the transactions among all the participants, a consensus algorithm is required to decentralize the blockchain control, ensuring the enforcement of certain consensus rules to the participating nodes. A consensus algorithm allows to obtain a unique blockchain state shared among all participants by establishing which new transactions and blocks will be added to the blockchain. Examples of consensus algorithms are Proof-of-Work (PoW) and Proof-of-Authority (PoA), commonly used in permissionless and permissioned blockchains respectively.

Ethereum[12] is one of the most popular blockchain platforms. Its main goal is to provide a worldwide computing infrastructure where its cryptocurrency acts as an exchange value for the use of the platform. Ethereum implements its own underlying P2P network which connects all participating nodes on the blockchain. The blockchain transactions (or state transitions) are processed through the Ethereum Virtual Machine (EVM), which is able to execute custom scripts called smart contracts. The smart contracts are developed with high-level Turing-complete languages such as Solidity. These smart contracts are compiled in bytecode format and stored within transactions. Hence, it is impossible to modify a smart contract once it is deployed. Formally, a smart contract can be defined as a deterministic computer program that is automatically executed based on the inputs received.

3 | RELATED WORK

Fields of application for blockchain in IoT devices is increasingly expanding. Some possible applications have been suggested, such as Smart Energy, to buy or sell excess energy generated by solar panels or other 'home' power sources; temperature tracking of medicines on transport and distributions, sometimes required by law; and supply chain tracking, to verify the origin of products or counterfeits; but any other field is subject to revision and can be enhanced by using blockchain.[13]

Moreover, due to the intrinsic distributed nature of fog computing, its confluence with blockchain seems natural. There exist several proposals that harness blockchain technology within an IoT and fog computing context. The survey in reference 14 addresses the integration of blockchain and fog computing. They find that blockchain is mostly used for data management purposes and security, harnessing its reliability and immutability. Also, in most cases, the blockchain is deployed in the cloud layer, in order to support payment/trading operations, and uses a Proof-of-Work consensus algorithm (with its corresponding high energy consumption). From a more general point of view, the work in reference 15 focuses on proposing a methodology to build and evaluate blockchain-based security and privacy systems. Also, the work in reference 16 provides a systematic review of empirical and analytical studies to evaluate blockchains, not necessarily focused on its integration with IoT.

Focusing in IoT-related blockchain solutions, Table 1 shows a feature comparison to summarize the differences between previous proposals found in the literature and this work. A detailed description of them is also provided next.

In BlockEdge,[17] the authors propose to use blockchain as a means for auditing the different processes involved in an industrial IoT application, exemplifying it by means of the use case of the process of building a smart house, with different agents over different industrial processes involved. Their proposal is only evaluated by simulations.

TABLE 1 Comparison between related works and our proposal (main characteristics)

Paper	Platformindept	Blockchain objective	Smart contractsupport	OS virtualization	Evaluated framework	Application	Metrics
Kumar et al.[17]	✓	basic layer	✓	×	Simulation	Industrial IoT	Latency, power consumption, network usage
Memon et al.[18]	✓	Application layer	✓	×	Simulation	Generic IoT	System usage, response time, throughput
Pan et al.[19]	✓	Basic layer	✓	×	Real tested	Generic IoT	Blockchain overhead, resource allocation
Tulie et al.[20]	✓	Data integrity	×	✓	Real tested	Healthcare (sleep apnea)	CPU & RAM usage, QoS expectation, latency (with and without BC), energy, network usage
Islam et al.[21]	×	Not provided	×	×	Not provided	E-healthcare (human activity recognition)	Accuracy of the algorithm
Baker et al.[22]	×	Authentication	×	×	Simulation	Smart transportation	Authentication time, delay, energy
This work	✓	Basic layer	✓	✓	Real tested	Generic IoT	CPU & RAM usage, time between transactions, peak throughput

DualFog-IoT[18] integrates support for IoT applications that use blockchain. The authors identify three types of applications, namely, real-time (RT), non-real-time (NRT) and delay-tolerant blockchain (DTB) applications. Their proposal includes a separate fog infrastructure to process DTB requests, so that the mining processes do not interfere with the performance of RT and NRT requests, which are processed in a conventional fog infrastructure. This approach is justified by the fact that the mining operation in blockchain requires heavy computation capabilities, currently being performed using Application Specific Integrated Circuit (ASIC) chips. However, this is only applicable to PoW-based blockchains. Thus, fog nodes including support for other less compute-greedy consensus algorithms can be suitable for a wide range of IoT applications. Moreover, many use cases would benefit from a permissioned blockchain, due to privacy issues.

Also, Edgechain[19] uses a credit-based resource management system to control the amount of resources IoT devices can obtain from edge servers, based on predefined rules, which are enforced by means of smart contracts. Also, the blockchain is used to keep secure data logging and auditing of all the IoT activities and transactions. The edge server is implemented with a cloud-like architecture, based on the OpenStack cloud operating system, and runs the miner in the PoW consensus protocol.

Another interesting proposal is FogBus,[20] an architecture that aims at integrating IoT applications running in the edge-fog-cloud ecosystem. FogBus harnesses blockchain technology to *ensure data integrity, protection, and privacy* while transferring confidential data. However, they implement blockchain as an ad-hoc component that uses public-private keypairs validated by a centralized controller in the cloud, and that can be used by fog nodes to check whether data are legitimate or not. Thus, FogBus does not exhibit the decentralization or consensus algorithms that characterize blockchains. Moreover, data processing is made by applications, and not by means of smart contracts, which are not supported by the architecture.

The work in reference 21 proposes a blockchain-based fog architecture for a specific use case in e-healthcare services. A typical 3-layered architecture (end devices, fog, cloud) is described as a context for the specific application domain, before diving into the details of the video processing application. However, the study is only focused on the models used for action recognition in video sequences and their performance evaluation. Moreover, blockchain is only named in the text as part of the fog architecture and not a minimum detail is provided on how it is leveraged, and how it is implemented.

Another blockchain-based fog architecture is proposed in reference 22 for the specific use case of smart public vehicular transportation systems. It is also structured in the typical 3 layers (end devices, fog, cloud), and leverages several technologies, including 5G, blockchain, edge computing, AI, and IoT. It is mainly focused on providing better response time in the matching process among users and smart vehicles, harnessing the capabilities of fog nodes as compared to a cloud-based implementation. The proposal is evaluated by means of the iFogSim simulator.[23] The implementation of blockchain is a custom one, specifically tailored for authentication of vehicles and users, and access control, that relies on a centralized validator in the cloud (i.e., a star topology is emulated rather than a peer-to-peer one). This ad-hoc blockchain is only present in the simulator.

As it can be seen in Table 1, the proposed work is the only one that addresses smart contracts and OS virtualization support at the same time as core technologies. The latter support is suitable for IoT resource-constrained devices. Also, there are other proposals that are evaluated in real testbeds, but ours is the only one aimed at specifically stress-testing the devices where blockchain runs.

4 | SYSTEM DESCRIPTION

The architecture of the proposed system consists of three main entities: the sensors, the users, and the low-cost computational nodes. From a generic point of view, each entity has a role in the system. While the physical sensors collect data in the form of text, audio, or image, the users interact with the processed data and may be the owners of the sensors. On the other hand, nodes consist of low-cost, low-power computers with the ability to support virtualization at the operating system level. They are in charge of processing the data and ensuring private data security, as well as supporting data traceability and auditable access to these data. Note that each entity makes use of a network technology to interconnect to the system. For example, LORA, Wi-Fi or Bluetooth technologies allow sensors to interconnect with computational nodes.

Focusing on the functionalities of each entity, the architecture of the system is composed of four layers, which are detailed in Figure 2. Some details regarding the logical layers and their functions are provided next:

1. **Sensor layer:** A generic IoT application will typically include data sources (i.e., sensors). So, the sensor layer monitors the real environment and collects data with different levels of sensitivity. There are freely accessible data available to

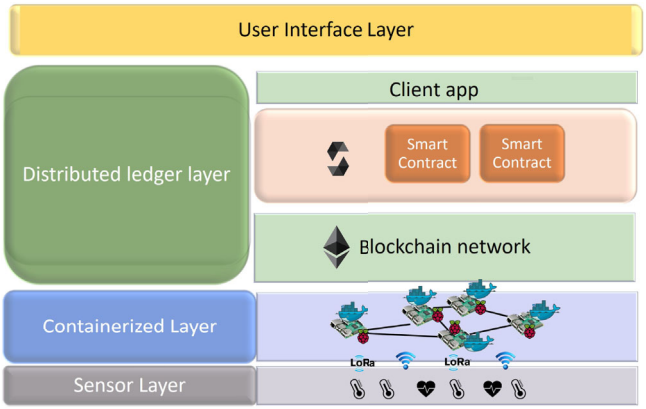

FIGURE 2 Software layered architecture for the blockchain-based IoT application

all users of the system, or alternatively, data that require certain permissions to read. The system supports a dynamic number of data sources and, to increase the security of the system, the new data sources are integrated into the system through a blockchain-based user-registration process. Prior to storing data in a smart contract, some kind of processing might be carried out (i.e., data processing to detect values of interest) to filter out the data to be stored. Every record for any data source is dealt with in its specific contract.

2. **Containerized layer:** To ease application portability, they will be deployed as containers. Hence, the containerized layer aims to manage the deployment of the containers where the blockchain-based IoT application runs. The chosen virtualization system for deploying and managing containers is Docker.[24] Docker is an open-source platform for developing, transporting, and running containerized applications, and is a world-leading software container platform, actively backed and sponsored by a wide community and corporations like Amazon and Microsoft, among many others.
3. **Distributed ledger layer:** This software layer comprises both the blockchain network and the smart contracts of the IoT application. Moreover, a client written in a language like Golang will generate values to be introduced in a smart contract for each data source (see Figure 2). On the one hand, the blockchain network will record, share and synchronize transactions on a private permissionless blockchain, that is, there is a restriction on access and who can transact but there is no restriction on participation in the consensus mechanism. In our case, the geth client for the Ethereum platform will be used.[25] Geth enables participating in permissionless blockchains like the Ethereum main network but also allows to create permissioned blockchains. Nowadays, geth implements two selectable consensus algorithms for permissioned blockchains: a PoW algorithm called Ethash (used by the Ethereum main network), and a PoA algorithm called Clique.

   On the other hand, the smart contracts are part of a decentralized application or DApp that interact with the blockchain network, either to query data of past events or current values, or to set new values. In particular, smart contracts will program the authorization and authentication model of the sensors and users entities.

   DApps differ from the classical server-client paradigm because there is no central server to send information to, and blockchain's status is updated when nodes that have to verify a transaction make sure that the operation in question is legitimate and not fraudulent. As a consequence of this, there is no division between the client and the server, since there is none of the latter. All clients in the network; however, can be different in access to blockchain features and interact within the same infrastructure. This can be used to differentiate clients depending on their domain of usage or in order to restrict functions, although measures of this kind can be taken care of in smart contracts.
4. **User interface layer:** In the IoT application, we can distinguish the system administrator from the other users. The first one will be in charge of registering the users who will be the owners of the sensors in the system. These users will register the sensor devices in the system. Additionally, users can request the information stored in the blockchain, that was previously collected by sensors and processed, in order to, for example, trace a distribution chain, generate historical water level or temperature data for the reservoirs or, generate an efficient solution for healthcare.[26]

5 | CASE STUDY

As the goal of this work is to explore the feasibility of running blockchain in resource-scarce fog computing devices, a DApp representative of a wide range of IoT applications has been developed. Typical IoT applications are based on processing data extracted from different and heterogeneous sources (sensors, cameras, …), and perhaps generating a response back (as feedback on a user front-end or operating an actuator). More precisely, the implemented use case addresses testing some conditions on continuously generated data and identifying data with unusual conditions. Relevant data history is stored in the blockchain for auditing and transparency. Thus, the presented use case captures a general data process workflow applied to healthcare services, but that also mimics a wide range of use cases, such as supply-chain management, smart grid, or food industry.[27]

More precisely, the developed DApp has two main generic functionalities:

1. Monitoring and registering data obtained from sensors. Without loss of generality, certain health-related metrics, such as temperature and heart rate, are considered here.
2. Authorization and authentication. Only authorized users (i.e., accounts) can register a new sensor or read sensed data. A procedure for using an One Time Password (OTP) when new users register into the system has been devised, which takes into account the particularities of the blockchain.

The complete components diagram for the developed DApp is depicted in Figure 3. In this case study, we use the Ethereum platform together with the geth client implemented in Go. Note that the geth client is the gateway into the Ethereum blockchain network thanks to RPC sockets. Hence, since smart contract code is arranged as a collection of methods and variables, their execution will be based on Remote Procedure Calls (RPCs). The entity that calls the methods of the smart contract has a contract interface that allows to execute its methods. In particular, Ethereum has a binary interface, namely the Application Binary Interface (ABI). This means that a program needs to have the methods' definitions (signatures) and bytecode to execute on an Ethereum node that runs the EVM in order to call contract methods. Note that the bytecode of the smart contract is generated by the Solidity compiler (`solc`).

To sum up the IoT DApp consists of the following components:

1. The Go client, which binds itself to the Ethereum node in order to access data and create transactions. Additionally, the client can query the blockchain for information and generate an HTML report.
2. Smart contracts transformed into bytecode. In particular, three main contracts have been developed for the DApp, namely, `GeneralContract`, `AuthContract` and `GenericSensorContract`. From the later, two contracts which inherit its logic have been created: `TemperatureContract` and `HeartRateContract`.

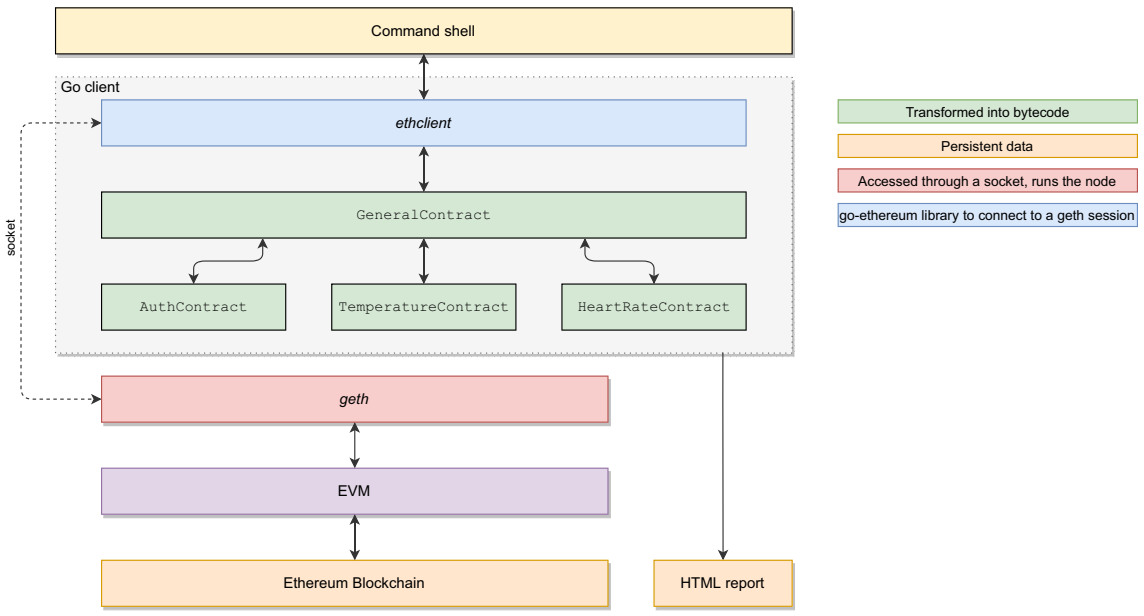

FIGURE 3 Components diagram for the developed DApp.

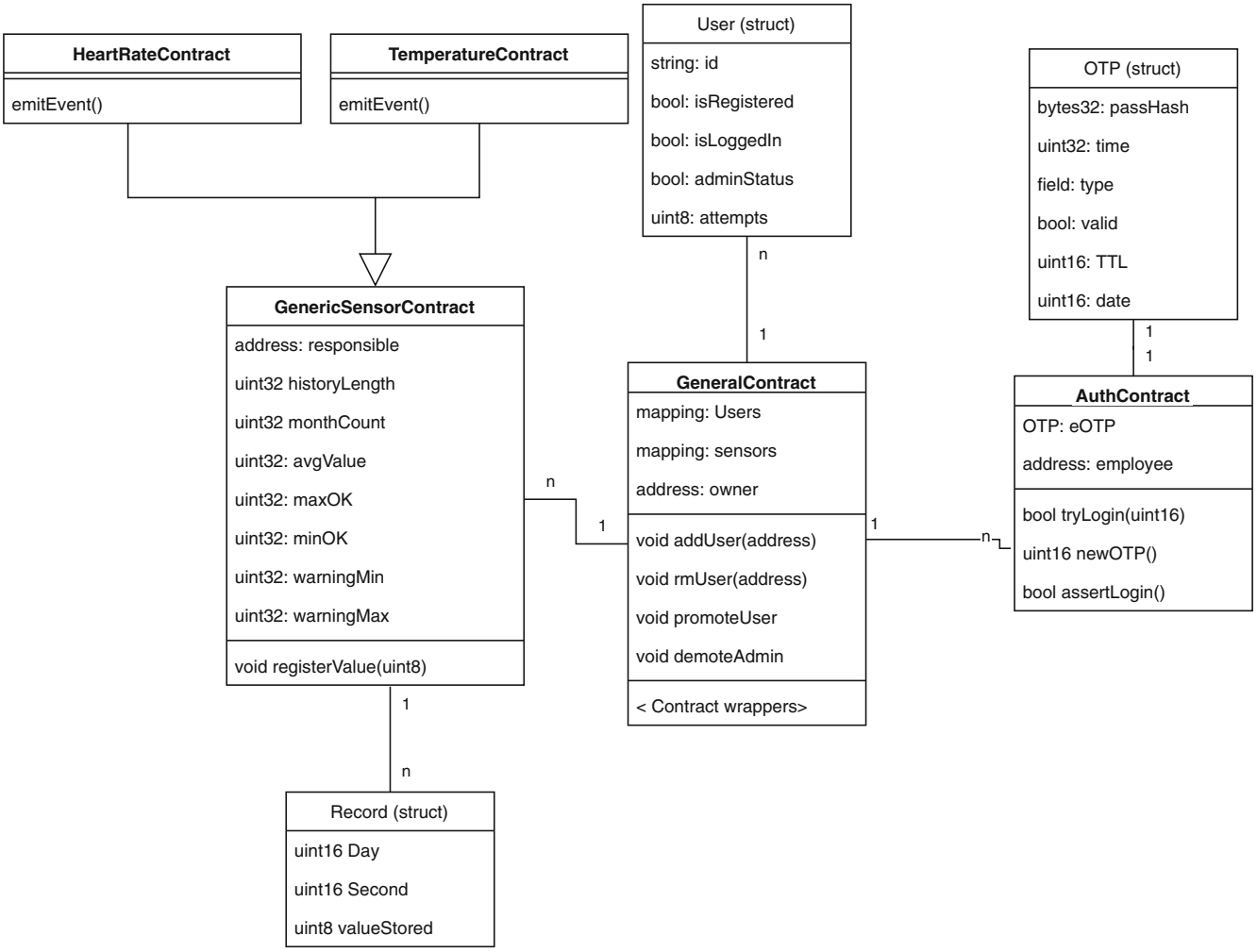

FIGURE 4 Resulting contract relationships.

3. EVM and `geth`. The IoT DApp attaches to the socket created by `geth`. Then, `geth` executes transactions and queries through the EVM. The result of the transactions is stored in the blockchain.

As Figure 3 shows, the DApp is composed of the main smart contract (`GeneralContract`) the contract to manage the authorization to the application and smart contracts (`AuthContract`), and the contracts that collect the data source from the different types of sensor devices (`TemperatureContract` and `HeartRateContract`). The later ones are inherited from the abstract `GenericSensorContract`, which implements generic logic common to any type of sensor. Some details follow:

1. `GeneralContract` is the main contract where access rights are collected on. It tracks registered users, whether they are logged in, and also provides wrappers for other contracts, so that any interaction with them is entirely up to its logic.
2. `AuthContract` deals with user authorizations. Per-user instances of this contract will be deployed from the `GeneralContract` in order to keep every user login details isolated. Further details will be given in Section 5.1.
3. `GenericSensorContract` is an abstract[‡] contract which records a history of any data received by a sensor. In this particular case, received values are selected based on several thresholds, which decide whether the record is stored and/or if an event must be propagated through the network.

Figure 4 depicts the class diagram for the system. It must be highlighted that the smart contract attached to a sensor device includes an array to store the tuple $< timestamp, data >$ with the recorded information of the sensor (*Record struct*

[‡]In object-oriented languages, an abstract class is not supposed to be instantiated as-is, but through another class which inherits all of its logic and attributes, with possible variations, for abstraction purposes.

in Figure 4). In addition, each user entity is identified by an *id*, a set of boolean status variables (to register whether this particular user is registered into the system, whether it is logged in, and whether it is a user with an administration role) and the number of login attempts (*User* struct in Figure 4).

5.1 | Authorization management at application level

Since the application is theoretically working with sensitive data, users should need some kind of authorization when using the application across many IoT devices. Since the client should be installed in many things, an approach would be having a machine at the organization's doorway, where the user, if registered in the application, receives an OTP (One-Time Password). If they want to interact with a device that contains sensors, they should enter the password for a successful login. This process can be done through different devices, thanks to the use of smart contracts.

A record of authorized users is needed to prevent sensed data to be queried by anyone. Thus, the list of users that are registered (together with some metadata) should be stored in a main/general contract, which will be the one users interact with. There is also the possibility of members being restricted from using the system.

Then, an additional Authorization Contract (`AuthContract`) is used for the user's login, having one contract instance per user. This makes the main/general contract behave like a *contract factory* which keeps every OTP isolated in a particular `AuthContract` instance. Methods are then called via the `GeneralContract`, allowing for user management only through the `GeneralContract` instance.

Generating temporal passwords on a centralized server is trivial since access to the server is restricted and communication between the server and the channel, as insecure as it can be, can be ignored if data is protected. This, however, does not happen in blockchain-based applications since the data stored in the blockchain can be accessed by any peer (regardless of their permissions), so these passwords cannot be generated by the contract. Moreover, password generation should also be verified by other peers to generate the transactions anyway. Generating the password via the corresponding contract and storing the hash is not suitable either, since the method to generate it should return the password value. Also, if the password is generated by the client and written within the contract, it will be both readable by transaction verifiers first, and by any other network peer after being eventually written into the contract's state.

Therefore, a safer way to generate the password and check its value is generating it using the off-chain client. However, the verification should be done by means of a contract to preserve the authentication process through several different devices, if needed. This can be solved by generating its hash in the client, alongside the password. Then, what is actually written into the contract is the password hash. To check that a password is valid, the contract just checks that the presumed password's hash corresponds to the one stored in its state.

More precisely, the process to generate an OTP is depicted in Figure 5. When the *User* requests a new OTP through the client user interface (*UI*), the *Goclient* creates the OTP and issues a call to the corresponding wrapper method in `GeneralContract`, which in turn, calls the method `newOTP()` in `AuthContract` to register the hash of the new OTP into the blockchain, associated to this particular user. This process implies the creation of one transaction into the

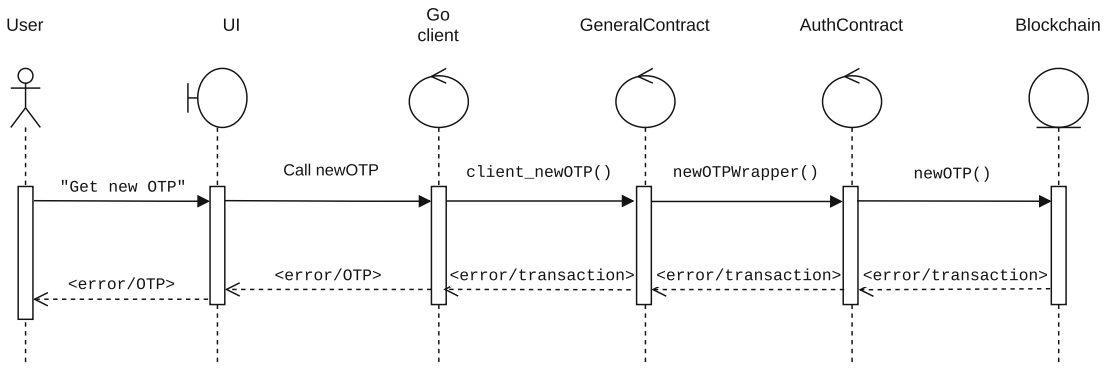

FIGURE 5 Creating a new OTP and storing the password hash in a contract.

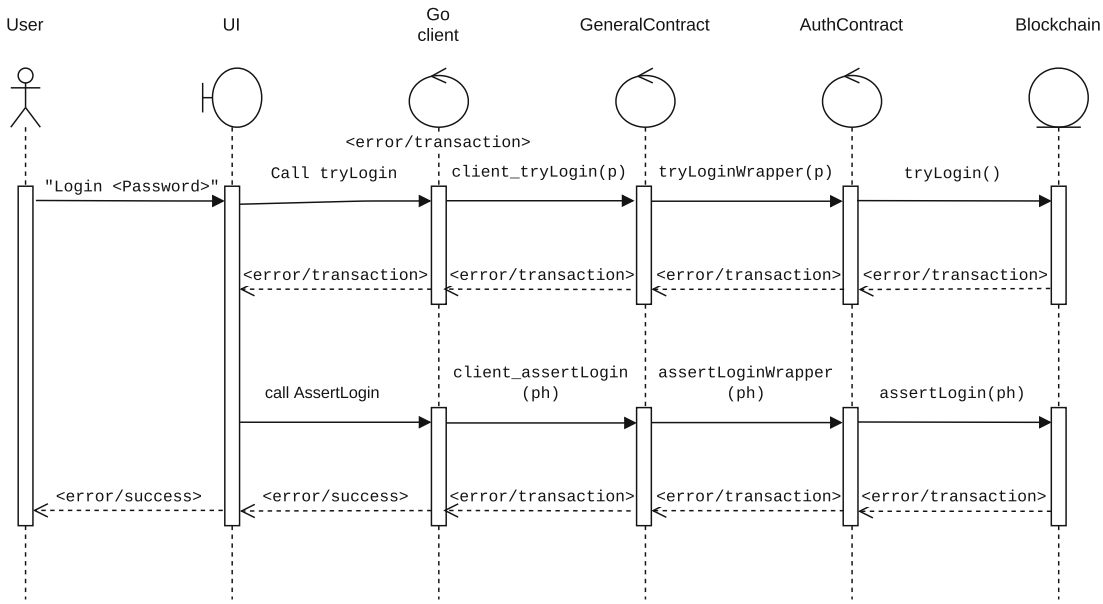

FIGURE 6 Login attempt two-step process.

blockchain to store the metadata related to the new OTP associated to that particular user. Also, recall that `GeneralContract` keeps the information on users roles and permissions, thus all transactions need to be issued by means of its wrappers.

For attempting to login, a transaction for checking the password in the `AuthContract` and noting that the OTP has not been used is needed. That is, the stored information on that OTP in the blockchain is not modified so that the same OTP cannot be used twice. As a consequence, for the `GeneralContract` to read that new change and change the user's status to *logged in*, another transaction (*assertion*) is also required. This is depicted in Figure 6. Similarly to the OTP generation process, the *User* initiates the login attempt through the *UI*, which in turn calls the appropriate function in the *Goclient*. This issues a call to the wrapper method in `GeneralContract` which consequently invokes the method `tryLogin()` in `AuthContract`, after checking the login status of the user. The latter computes the hash of the provided password and compares it to the hash stored within the contract's structure for the OTP. As calls to methods belonging to smart contracts do not return any value (apart from the transaction receipt), another action must be made in order to assess the success of the login attempt and provide feedback to the user. That is the reason behind the use of the `assertLogin()` method. This method returns whether or not a valid and not yet expired OTP has been used for login. This has to be called on a separate transaction since a successful execution of `try_login` modifies the state of the contract and as such, its changes need to be checked on a separate transaction.

After this process, if the login attempt was successful the status of the OTP is changed to invalid so that it cannot be used again. Thus, what can be done in a two steps process with a centralized server (generating the OTP and using it) now has to be split in three steps when OTPs are stored in the decentralized blockchain.

5.2 | Per-sensor contracts and permission management

Just like Authentication Contracts, each sensor can have its own contract that stores every relevant value before storing it in the blockchain. Data sensors are pre-processed in the Fog nodes and only some data are recorded in the blockchain. In our case, and just as a simple use case, we have defined an upper and lower bound to determine whether the value generated is to be stored in the blockchain, and another pair of thresholds determine if an event should be emitted, while also storing the value.

These sensor contracts need restricted access that allow for both per-user permissions and general permissions, just in case a sensor can be read or written by anyone. A collection of permission records is needed so that permissions for individual users can be stored and read, therefore a mapping is needed.

6 | SYSTEM DEPLOYMENT DETAILS

This section presents the experimental procedures to deploy the system including the setup of the network and the node, as well as the generation of container images.

Running the system on a real blockchain requires an Ethereum node running, which generates a socket the client attaches to, in order to actually call contract methods and submit transactions. The needed contracts can be deployed on a public blockchain like Ethereum's mainnet, at the cost of gas, ETH and ultimately, money. However, as this system can collect personal data (medical in this particular use case), it is more sensible for it to run using a permissioned private blockchain, especially since Ethereum supports its implementation of Proof-of-Authority (`clique`) as a consensus mechanism.

6.1 | Defining the network

The network is first specified via a *genesis file*. This JSON file contains configuration items for the network such as:

1. Chain ID: positive integer number to identify the network.
2. Block gas limit.
3. Clique period: constant block throughput, specified in seconds. If this value is 0, new blocks are only added to the chain when transactions arrive.
4. Initial sealing addresses.
5. Pre-funded addresses and their balance.

The genesis file also includes meta-data about Ethereum versions and features. For ease of use, `go-ethereum` provides a command-line wizard to generate the genesis file called `puppeth`.

6.2 | Setting up nodes

For nodes to mine in an Ethereum network (validate, in the case of PoA), they need to first initialize the genesis file and then start mining. Both operations can be done by invoking `geth` with their corresponding parameters. Invoking `geth` to mine also requires specifying the network ID, in case this node is part of several networks; the listening port, the data directory where blockchain data is stored, and the node key-pair is located. This key-pair is stored as a text file which contains the public key (address), the ciphered private key, and some information about the algorithm used to protect it. For peer-to-peer connectivity, peers can be discovered, specified via parameters, or manually added after starting mining.

6.3 | Setting up Docker containers

The resulting 'binary' application is composed of the Ethereum node that grants it the ability to mine and create new transactions, and the application client, which will call contracts and store business logic. To ease application portability, it will be deployed as a container. The chosen virtualization system for deploying and managing containers is Docker.[24] Containers are deployed on a Raspberry Pi 4,[28] which runs Ubuntu Server 20.04,[29] in its ARM64 variant.

The specification of the Docker image for the application includes the following details:

1. The base image is Ubuntu, due to its variety of architectures and its ad-hoc repository from Ethereum developers.
2. The developed DApp needs `geth` for accessing the Ethereum console, but it is not found on either Ubuntu or Debian repositories, so it has to be downloaded from the original authors.
3. Both the Ethereum mining script and the application client are copied into the image, meaning that there is no need for source code in order to run it.
4. Network ports have to be opened to allow interoperability between applications. In the case of `geth`, ports to be exposed are the following:
   - 8545 TCP, used by the HTTP based JSON RPC API.

- 8546 TCP, used by the WebSocket based JSON RPC API.
- 8547 TCP, used by the GraphQL API.
- 30303 TCP and UDP, used by the P2P protocol running the network.

However, some of these network ports can be omitted, since the *ethclient* can connect to geth using Inter-Process Communication (IPC), which is similar to a socket that gets created as a file when executing *geth*.

Running the produced image requires some data regarding `port-forwarding` with the host operating system as well as directories that are shared and kept in sync by having them 'mounted' into the container. Both of these configuration parameters require both the container's internal port/directory and its counterpart in the host machine. This allows any given machine to launch as many instances of the container as possible as long as there are free ports on the operating system. If the user wants the container to provide them with an interactive shell or be launched in the background, it can also be specified in the running parameters `-it sh` (or Bash if installed).

7 | PERFORMANCE EVALUATION

The aforementioned system has been deployed on the testbed depicted in Figure 7, which includes a computer and an SBC. The computer is a laptop and is used mainly for comparison purposes. The SBC is a Raspberry Pi 4 with 4 GB RAM. The laptop has the following components:

1. CPU: Intel Core i5-6300HQ (Quad-Core 2.3 GHz).
2. Memory: 8GiB DDR4.
3. Storage: SATA SSD.
4. Network: Gigabit Ethernet.

The laptop and the Raspberry Pi run one Ethereum node each. The Ethereum chain is using Proof-of-Authority as the consensus mechanism, and the only address that can seal and add new blocks to it is the one used on the node run by the computer, unless stated otherwise. The Ethereum node on the computer is directly run on the host OS 'on bare metal', while the one on the Raspberry Pi runs inside a container.

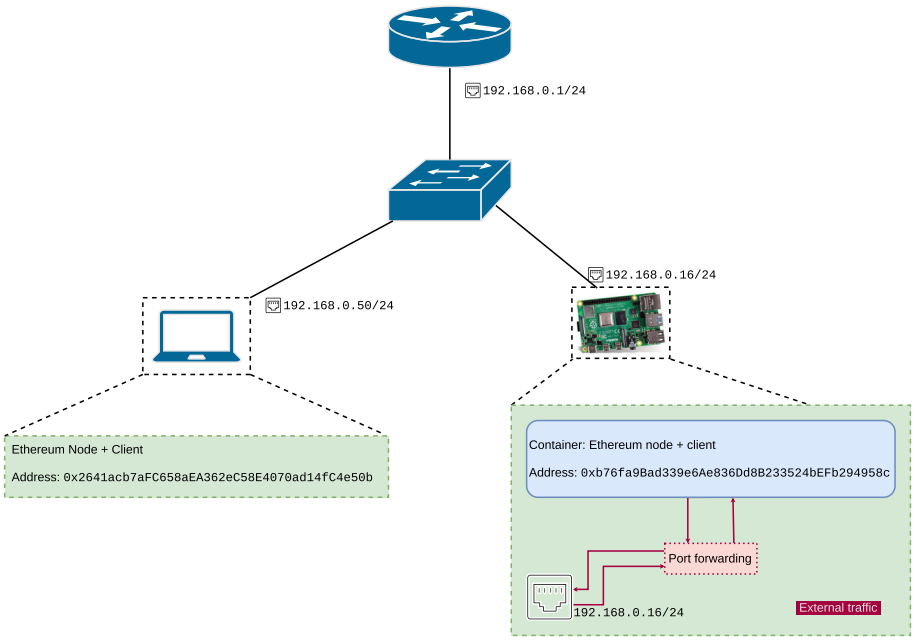

FIGURE 7 Testbed composed of a computer and the Raspberry Pi.

Next, some experiments run to measure resource utilization on the Raspberry Pi as well as testing system throughput will be described.

7.1 | Resource utilization

The Raspberry Pi, while idling, has a CPU load of no more than 2%. When a container running *geth* is launched and is waiting for transactions, it gets to barely 5% on a single core. The memory usage remains between 600 and 800 MB.

After launching the client, the CPU goes to a 100% load on a single core while executing the instructions to configure the account to load and generate the *ethclient*. Then, when it starts generating transactions, the load as measured with respect to one core is increased to 102%. As the load is split between the four cores of the device, each one is loaded at 25% approximately. Memory occupation also increases by 600 MB until it finishes execution. The transaction throughput, based on the *geth* console, is on average 1.366 ms, and the median is 1.1314 ms. However, these timings oscillate between 2.5 and 0.5 ms, thus this upper bound is the safest interval to have between transactions.

If the Ethereum node at the Raspberry Pi is configured as a sealer, CPU usage does not rise significantly above the aforementioned 25% when also sealing blocks. This negligible difference is probably due to the consensus algorithm being very simple.

7.2 | System throughput

In order to evaluate system throughput, that is, the transaction rate which can be achieved on the system, continuous requests to the system must be carried out. These requests are performed on the `GenericSensorContract` to mimic a realistic situation where data from sensors might be generated at a higher pace than creating new users or OTPs.

So, to ensure the correct operation of the IoT application, it is necessary to take into account the asynchronous behavior of the decentralized blockchain. In other words, the transaction confirmation that modifies the state of the blockchain is not immediate, imposing special restrictions on the requests made in the system that modify the state of the contracts. Thus, executing those methods on a real *Ethereum* blockchain requires manual delays to be used in the code.

Focusing on this aspect, a test case has been set up, where a `GenericSensorContract` instance is created and new data is continuously stored into it. Moreover, to stress the system, the delay between every storage call is continuously reduced. Eventually, as the delay gets reduced, a transaction does not get properly validated before the next one is executed. For example, if the transaction increments a counter when the value of the counter is read between transactions and the one for the following one is the same. In our tests, the starting value for the delay between transactions is 10 ms, and for every storage the value is reduced by 0.1 ms. In every test, the time between consecutive transactions is reduced until the transaction gets lost.

More precisely, the test conducted using a node running on the Raspberry Pi and the computer using the main sealer address with the conditions stated above provide the results plotted in Figure 8, with statistics obtained from 100 executions of the test Box plots are used to illustrate the distribution of time between transactions, including their minimum and maximum values, the first quartile, median, and third quartile. It can be seen as the minimum delay observed in the Raspberry Pi is around 3 ms, while up to nearly 90% of the requests achieved delays lower than 4 ms. The difference between the 75th percentile upwards on the RPi can be caused by sustained load on *geth*. The same behavior is observed in the laptop, with delays between 1 and 2 ms for 90% of the requests. Thus, the performance obtained on the Raspberry Pi is about half of that obtained on the laptop, while its resources are significantly lower.

Next, the throughput of the IoT DApp is analysed when the constrained device is on stress to analyse if it has any impact. In particular, the experiment consisted on applying increasing CPU loads to the RPi using programs like *go-cpu-load*[§] This mimics the more realistic use case where the fog node (i.e., the Raspberry Pi) is performing some complex processing on sensed data before storing relevant information on the blockchain. In this case, the time between transactions starts rising, although time differences are usually sub-milliseconds (see Figure 9).

[§]https://github.com/vikyd/go-cpu-load.

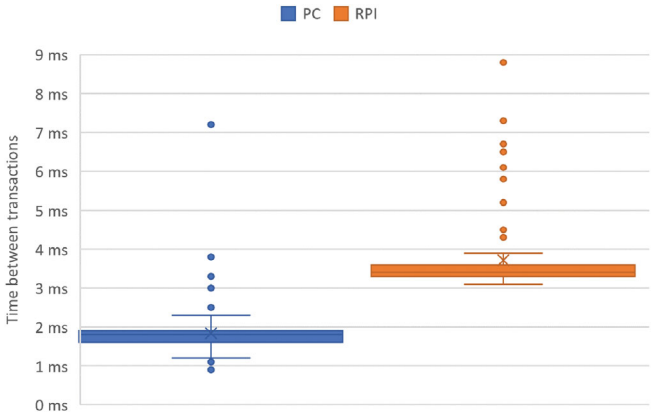

FIGURE 8 Intervals between transactions until malfunctioning

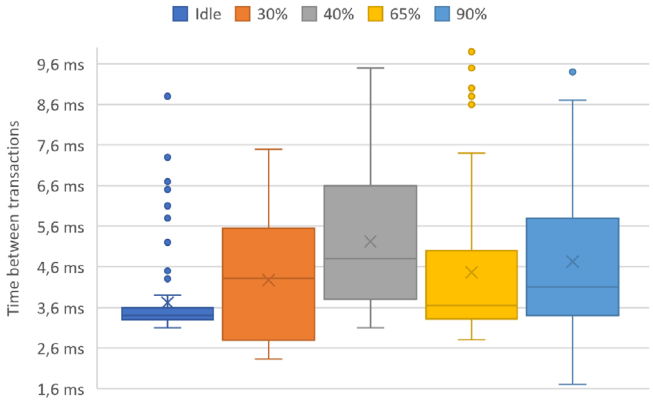

FIGURE 9 Intervals between transactions until malfunctioning, by CPU load

In the worst tested case, the minimum suitable interval on the RPi should be 9 ms or higher, although this value can further increase based on the Raspberry Pi's CPU load, and bandwidth latency.

Overall, the theoretical peak throughput is approximately over 300 transactions per second for the lowest minimum value for the time between transactions, while the minimum would be 100 transactions per second.

Given the ubiquitous and inexpensive resources used, the results seem reasonable for working in small scales. Since the continuous insertion of data to the blockchain is not expected to happen in real scenarios, where only critical or unexpected data would actually be stored, this deployment could be enough for geographically reduced areas. However, if an area presented much higher device density and thus more data generated, it means that there would be more information that should be stored on-chain. In these cases, other strategies like inter-connecting several different blockchains, even hierarchically,[30] should be contemplated; or alternatively, make use of other distributed ledger technologies such as IOTA.[31]

8 | CONCLUSIONS

The IoT ecosystem benefits from the distributed processing and storage capabilities of a fog computing architecture. Moreover, SBCs such as Raspberry Pi, are a common and affordable implementation option for fog nodes. Also, blockchain has emerged as a disruptive technology able to provide support to the decentralization of storage and business logic among untrusted parties. The IoT enables to enrich the blockchain with information from the real world which can trigger the execution of business logic by means of smart contracts deployed on the blockchain. Thus, this work has mainly tackled the feasibility of implementing blockchain on low-cost, scarce-resource fog nodes.

A set of generic smart contracts aimed at registering data of interest retrieved from sensors has been developed, as well as a client which can run on Raspberry Pi devices. The client and smart contracts also address the problem of user authentication and authorization to access data from sensors, by means of a procedure for issuing and using one-time passwords suitable to the particularities of the blockchain. These have been deployed on a private Ethereum blockchain running the Proof-of-Authority consensus protocol.

A set of experiments have been carried out in order to assess the overhead of the client on the Raspberry Pi, and to test the throughput of the system. Results show the feasibility of the proposal because there is not a significant load on the Raspberry Pi and the transaction rate in the worst case suffices for most of the practical cases related to IoT sensing.

Future work includes testing the scalability of the private blockchain, that is, finding an estimation on the maximum number of fog nodes, sensors, measuring rates, and so forth, that the presented deployment is able to support.

AUTHOR CONTRIBUTIONS

Miguel Sánchez-de la Rosa: Writing - Review & Editing; Writing - Original Draft Preparation; Visualization; Software; Investigation. **Carlos Núñez-Gómez:** Writing - Review & Editing; Writing - Original Draft Preparation; Visualization; Software; Investigation. **M Blanca Caminero:** Writing - Review & Editing; Writing - Original Draft Preparation; Investigation; Conceptualization. **Carmen Carrión:** Writing - Review & Editing; Writing - Original Draft Preparation; Investigation; Conceptualization.

ACKNOWLEDGMENTS

This work was supported under PID2021-123627OB-C52 project, funded by MCIN/AEI/10.13039/501100011033 and by European Regional Development Fund (ERDF), "A way to make Europe" and under GC-020-017 grant, funded by the Regional Government of Castilla-La Mancha for Consolidated Research Groups.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available on request from the corresponding author.

ORCID

M. Blanca Caminero https://orcid.org/0000-0003-3312-7393

REFERENCES

1. Dastjerdi A, Gupta H, Calheiros R, Ghosh S, Buyya R. Fog computing: principles, architectures, and applications. In: Buyya R, Vahid Dastjerdi A, eds. *Internet of Things*. Elsevier; 2016:61-75. doi:10.1016/B978-0-12-805395-9.00004-6
2. Mahmud R, Ramamohanarao K, Buyya R. Application management in fog computing environments: a taxonomy, review and future directions. *ACM Comput. Surv*. 2021;53(4):1-43. doi:10.1145/3403955
3. Mahmud R, Toosi AN. Con-pi: a distributed container-based edge and fog computing framework. *IEEE Internet Things J*. 2022;9(6):4125-4138. doi:10.1109/JIOT.2021.3103053
4. Nakamoto S. Bitcoin: a peer-to-peer electronic cash system, Tech. Rep; 2008. https://bitcoin.org/bitcoin.pdf
5. Szabo N. *Smart Contracts: Building Blocks for Digital Markets*. Vol 16. Extropy Institute; 1996. http://www.extropy.org/publications.htm
6. Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge computing: Vision and challenges. *IEEE Internet Things J*. 2016;3(5):637-646. doi:10.1109/JIOT.2016.2579198
7. Bonomi F, Milito R, Zhu J, Addepalli S. Fog computing and its role in the internet of things. Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC'12, Association for Computing Machinery, New York, NY, USA; 2012:13–16. doi:10.1145/2342509.2342513
8. Carrión C. Kubernetes scheduling: Taxonomy, ongoing issues and challenges. *ACM Comput. Surv*. 2022. doi:10.1145/3539606
9. Mouradian C, Naboulsi D, Yangui S, Glitho RH, Morrow MJ, Polakos PA. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Commun Surv Tutor*. 2018;20(1):416-464. doi:10.1109/COMST.2017.2771153
10. Ferdous MS, Chowdhury MJM, Hoque MA, Colman A. Blockchain consensus algorithms: a survey. ArXiv:2001.07091; 2020.
11. Ali MS, Vecchio M, Pincheira M, Dolui K, Antonelli F, Rehmani MH. Applications of Blockchains in the Internet of Things: A Comprehensive Survey. *IEEE Commun Surv Tutor*. 2019;21(2):1676-1717. doi:10.1109/COMST.2018.2886932
12. Buterin V. A next-generation smart contract and decentralized application platform. Tech. Rep; 2013. https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf
13. Govil N. Applications of blockchain in IoT. 2020; Accessed May 14, 2021. https://iotdunia.com/applications-of-blockchain-in-iot/
14. Baniata H, Kertesz A. A Survey on Blockchain-Fog Integration Approaches. *IEEE Access*. 2020;8:102657-102668. doi:10.1109/ACCESS.2020.2999213

15. Ferrag MA, Shu L. The performance evaluation of blockchain-based security and privacy systems for the internet of things: a tutorial. *IEEE Internet Things J*. 2021;8(24):17236-17260. doi:10.1109/JIOT.2021.3078072
16. Fan C, Ghaemi S, Khazaei H, Musilek P. Performance evaluation of blockchain systems: a systematic survey. *IEEE Access*. 2020;8:126927-126950. doi:10.1109/ACCESS.2020.3006078
17. Kumar T, Harjula E, Ejaz M, et al. BlockEdge: blockchain-edge framework for industrial IoT networks. *IEEE Access*. 2020;8:154166-154185. doi:10.1109/ACCESS.2020.3017891
18. Memon RA, Li JP, Nazeer MI, Khan AN, Ahmed J. DualFog-IoT: additional fog layer for solving blockchain integration problem in internet of things. *IEEE Access*. 2019;7:169073-169093. doi:10.1109/ACCESS.2019.2952472
19. Pan J, Wang J, Hester A, Alqerm I, Liu Y, Zhao Y. EdgeChain: an edge-IoT framework and prototype based on blockchain and smart contracts. *IEEE Internet Things J*. 2019;6(3):4719-4732. doi:10.1109/JIOT.2018.2878154
20. Tuli S, Mahmud R, Tuli R, Buyya R. FogBus: a blockchain-based lightweight framework for edge and fog computing. *J Syst Softw*. 2019;154:22-36. doi:10.1016/j.jss.2019.04.050
21. Islam N, Faheem Y, Din IU, Talha M, Guizani M, Khalil M. A blockchain-based fog computing framework for activity recognition as an application to e-healthcare services. *Future Gener Comput Syst*. 2019;100:569-578. doi:10.1016/j.future.2019.05.059
22. Baker T, Asim M, Samwini H, Shamim N, Alani MM, Buyya R. A blockchain-based fog-oriented lightweight framework for smart public vehicular transportation systems. *Comput Netw*. 2022;203:108676. doi:10.1016/j.comnet.2021.108676
23. Gupta H, Vahid Dastjerdi AV, Ghosh SK, Buyya, R. ifogsim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Softw Pract Exp*. 2017;47(9):1275-1296. doi:10.1002/spe.2509
24. Docker. Accessed January 27, 2021. https://www.docker.com/
25. Geth. Accessed May 24, 2021. https://geth.ethereum.org/
26. Adere EM. Blockchain in healthcare and IoT: a systematic literature review. *Array*. 2022;14:100139. doi:10.1016/j.array.2022.100139 https://www.sciencedirect.com/science/article/pii/S2590005622000108
27. Dai H-N, Zheng Z, Zhang Y. Blockchain for internet of things: a survey. *IEEE Internet Things J*. 2019;6(5):8076-8094. doi:10.1109/JIOT.2019.2920987
28. R. P. Foundation, Teach, learn, and make with raspberry pi. 2021; Accessed May 28, 2021. https://www.raspberrypi.org/
29. Canonical I. Install ubuntu on a raspberry pi. 2021; Accessed May 28, 2021. https://ubuntu.com/download/raspberry-pi
30. Oktian YE, Lee S-G, Lee HJ. Hierarchical multi-blockchain architecture for scalable internet of things environment. *MDPI Electron*. 9(6):1050. doi:10.3390/electronics9061050
31. Popov S. IOTA: feeless and free, IEEE Blockchain Technical Briefs.

How to cite this article: Sánchez-de la Rosa M, Núñez-Gómez C, Caminero MB, Carrión C. Exploring the use of blockchain in resource-constrained fog computing environments. *Softw Pract Exper*. 2022;1-17. doi: 10.1002/spe.3173