Journal of Ubiquitous Systems & Pervasive Networks
Volume 9, No. 2 (2017) pp. 01-13

Virtual Resources & Blockchain for Configuration Management in IoT

Mayra Samaniego [a], Ralph Deters [a] *

[a] *University of Saskatchewan, Saskatoon, Canada, S7N 5C9*

Abstract

Until now, most systems for Internet of Things (IoT) management, have been designed in a Cloud-centric manner, getting benefits from the unified platform that the Cloud offers. However, a Cloud-centric infrastructure mainly achieves static sensor and data streaming systems, which do not support the direct configuration management of IoT components. To address this issue, a virtualization of IoT components (Virtual Resources) is introduced at the edge of the IoT network. This research also introduces permission-based Blockchain protocols to handle the provisioning of Virtual Resources directly onto edge devices. The architecture presented by this research focuses on the use of Virtual Resources and Blockchain protocols as management tools to distribute configuration tasks towards the edge of the IoT network. Results from lab experiments demonstrate the successful deployment and communication performance (response time in milliseconds) of Virtual Resources on two edge platforms, Raspberry Pi and Edison board. This work also provides performance evaluations of two permission-based blockchain protocol approaches. The first blockchain approach is a Blockchain as a Service (BaaS) in the Cloud, Bluemix. The second blockchain approach is a private cluster hosted in a Fog network, Multichain.

Keywords: *IoT, Configuration Management, Virtual Resource, Blockchain, Fog, REST*

1. Introduction

The Internet of Things (IoT) connects physical devices (Things) to the web. From this connection, new and varied interactions between things, services, and users emerge.
Traditional network configuration management assumes [1]:

- perennial on-site location
- deliberated device login
- specific understanding of the devices' characteristics, and so forth.

However, IoT networks have distinctive characteristics [2], e.g.:

- dynamic positioning,
- heterogeneity,
- energy constraints,
- geographical distribution.

These characteristics make traditional network management techniques hardly feasible in practice in IoT. Besides, the performance demand and configuration management are different in the IoT space. Most research, to a significant extent, focus on IoT management from a Cloud-centric perspective. Cloud-centric systems mostly rely on static virtualizations such as data streaming and batch processes. The main goal of these systems is to obtain reliable data in an efficient and secure manner, from constrained networks [3][4].

With the emergence of Fog Computing [5], the management focus shifts away from a static one-direction communication towards a dynamic multi-directional interaction over the IoT components. Configuration and processing tasks can be distributed among of the edge of the IoT network. However, the following questions emerge:

- How to manage the configuration of the large and heterogeneous set of devices in the IoT network?
- How to guarantee the provisioning of the correct configurations in the IoT network?

* Corresponding author. Tel.: +1 306-966-2072
Fax: +1 306-966-4884; E-mail: deters@cs.usask.ca

DOI: 10.5383/JUSPN.09.02.001

This work introduces virtualization of IoT components (referred to as Virtual Resources) and permission-based Blockchain protocols in a Fog network to address the mentioned questions. The remainder of this paper is organized as follows. Section 2 introduces Configuration Management & IoT. The communication patterns in IoT are explained in Section 3. Section 4 introduces virtualization in IoT. Section 5 introduces Blockchain technology. The evaluations of Virtual Resources and Blockchain protocols are presented in Section 6. Finally, a conclusion is presented in Section 7.

2. Configuration Management & IoT

The Internet of Things (IoT) vision aims connectivity with the world, anytime, anywhere [6]. This vision has been analyzed by Gubbi et al. [7] from two perspectives. First, object-centric, which highlights the features of IoT devices and supports direct interactions among them. This perspective accomplishes a richer user-experience as users can change the configuration of devices; however, only one user configuration is supported at a time in a specific device. Second, Cloud-centric, which centers around services and implementations that process large data streams contributed by constrained devices. The Cloud-centric view identifies three layers of abstractions: Things, Services, and Applications (Figure 1). The Things layer is the lowest level of abstraction and represents constrained devices [3] (e.g. sensor and actuator networks). The Application layer is the higher-level of abstraction and hosts final solutions such as monitoring, managing, and other processes. The Service layer is the link between the Applications and Things. This layer virtualizes IoT components either as data streaming processes or as virtualized resources (e.g. proxies). The Cloud-centric perspective supports multiple user configurations simultaneously. However, the configurations use the virtualizations hosted in the Service layer, which means that the interaction is a static one-direction communication that focuses on sensor data. Besides, the price for moving computation away from the edge into the Cloud is a significant latency when engaging the constrained devices.

In the IoT space, it is essential to have low latency when engaging the geographically distributed devices to perform configuration management in an efficient manner. A study in the healthcare industry presented by Cortés et al. [8] showed that the centralized Cloud storage cannot handle the velocity of the data flow generated by the IoT Cloud in real-time.

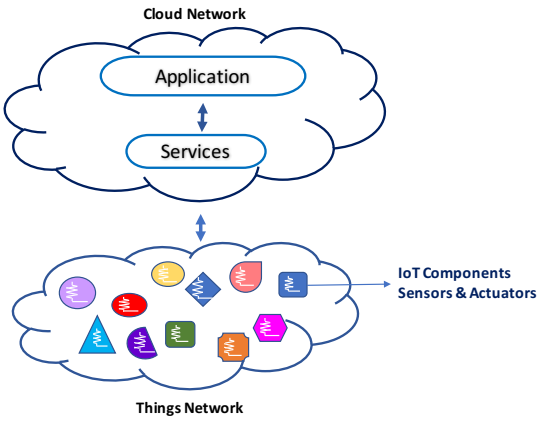

Fig. 1. Typical system design pattern in IoT [9]

The Fog Computing paradigm emerges as a solution to engage networks geographically closer than from the Cloud. Cisco describes Fog as an extension of the Cloud where computing, storage, and networking are the main pieces of both [5]. The characteristics of a Fog network are:

- edge location
- geographical distribution
- large -scale networks
- a considerable number of nodes
- mobility support
- real-time interactions
- wireless connectivity supremacy

Fog Computing benefits IoT in the following aspects [5]:

- location awareness rather than location ignorance, typical of Cloud computing
- geographical distribution of a vast number of nodes rather than centralized clusters
- wireless mobility
- real-time things engagement rather than streaming/batch processes
- resource heterogeneity rather than static features

With Fog Computing capabilities at the edge of the IoT network, more efficient real-time configuration and analytics can be handled as the latency and bandwidth consumption to engage the IoT network components is reduced (see Table for a comparison between the Fog and the Cloud). Additionally, dynamic interactions over IoT components can be supported.

Table 1. Fog vs. Cloud Nodes [5]

	Fog nodes closest to IoT devices	**Fog aggregation nodes**	**Cloud**
Response time	Milliseconds to sub seconds	Seconds to minutes	Minutes, days, weeks
Application examples	M2M communication	Virtualization Simple analytics	Big data analytics Graphical dashboard
How long IoT data remains stored	Transient	Short Duration: perhaps hours, days	Months or years
Geographic coverage	Very local: for example, one city block	Wider	Global

3. Communication & IoT

Communication is the primary concern when working in the IoT space. Reliable management systems require effective communication among all parties in the IoT network e.g.: sensors, actuators, back-end services, and so forth. The communication patterns in IoT can be divided into three groups [10]:

- Data Centric
- Message Centric
- Resource Centric

3.1. Data-Centric Communication

The Data-Centric communication pattern focuses on the reliability and transmission of data.
The Data Distribution Service (DDS) is an Object Management Group's (OMG) [11] standard. DDS is a data-centric publish-subscribe [12] model for distributed application communication and integration [13]:

- DDS is data-centric because it has a central data space where the data and the rules to access to that data are structured.
- DDS is publish-subscribe because it has a middleware where the communication is performed through publications and subscriptions of topics. [14].

Figure 2 shows the DDS service diagram. The data-centric publish-subscribe middleware guarantees time-effective and reliable delivery from data writers to data readers. Topics enable the publishing and subscribing. DDS Domains are kept completely isolated from each other. There is no data-sharing across DDS domains [14]. Following the publish/subscribe pattern, writers and readers work in a decoupled environment regarding synchronization and time:

- Time: It is not necessary that both actors be active at the same time
- Synchronization: It is not necessary that any of the actors have information about each other.

The DDS standard highlights that its main goal is the "Efficient and Robust Delivery of the Right Information to the Right Place at the Right Time" [13]. The key features of DDS are:

- Dynamic discovery of participants
- Recovery of data for subscribers
- Quality of Service support
- Publish/subscribe service in real-time
- Peer-to-peer communication between publishers and subscribers
- Scalability

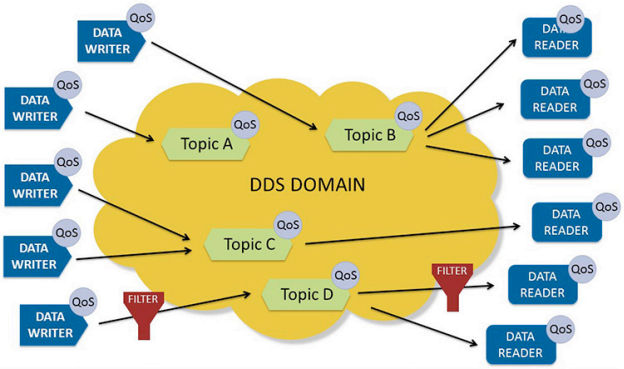

Fig. 2. Data Distribution Service (DDS) Diagram [14]

3.2. Message-Centric Communication

The Message-Centric communication pattern focuses exclusively in the delivery of reliable messages from writers to readers.
The Message Queuing Telemetry Transport (MQTT) [15] is a lightweight message-centric protocol based on the publish/subscribe pattern [16].
Figure 3 shows the MQTT architecture. In MQTT, there is a central broker [17] that supports the communication between writers and readers. The message-orientation feature of MQTT makes it content agnostic and only focuses on the delivery of messages.

MQTT uses TCP for communicating with the message broker. Using TCP, in turn, can lead to high communication costs. Consequently, a UDP-based MQTT for sensors (MQTT-S) [18] was developed.
In the IoT space, MQTT is the most used protocol (e.g.: [19][20]) due to its low overhead, easy implementation, and support from all leading vendors. The common design in systems that implement MQTT is integrating sensors and actuators in constrained nodes and connecting those nodes to the central broker [17], e.g.: the AWS IoT platform [21]. MQTT offers to decouple in time, space, and synchronization.

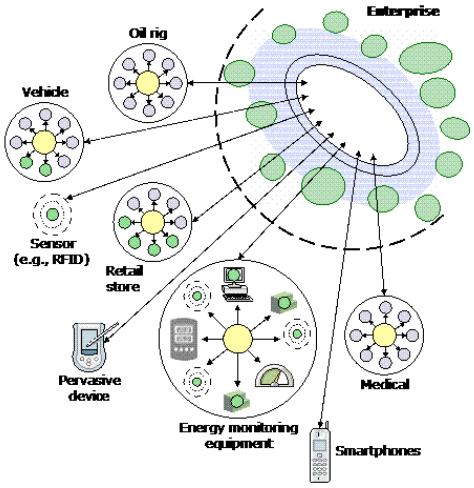

Fig. 3. Message Queue Telemetry Transport Protocol (MQTT) IBM [17]

3.3. Resource-Centric Communication

The focus of the Resource-Centric communication pattern is the resources.
The Constrained Application Protocol (CoAP) [22] is a machine-to-machine (M2M) resource-based protocol whose features work very well in constrained networks. CoAP follows the REST architectural design (Figure 4). The method definitions in CoAP are like the ones in HTPP: GET, POST, PUT, DELETE. Figure 5 shows an interaction between a client and a server using CoAP. The REST pattern of CoAP enforces a resource-oriented view on IoT components [23] [24], e.g. edge devices. CoAP uses UDP as the default method for transmission of data, but TCP can be employed as well. The CoAP package size varies from the minimum 4 bytes (simple GET requests) to a maximum of 1024 bytes.

Even though CoAP does not follow the publish/subscribe pattern, it emulates a publish/subscribe behavior through the observe feature. Figure 6 shows the interaction between an observer and a subject. The observer registers into a subject asking permission to observe a specific resource. The observer will receive the update from the observed resource any time it changes.
Contiki [25] and TinyCoAP [26] are implementation of CoAP in the context of IoT.
Additionally, the use of CoAP provides a unified way to abstract and engage IoT components.

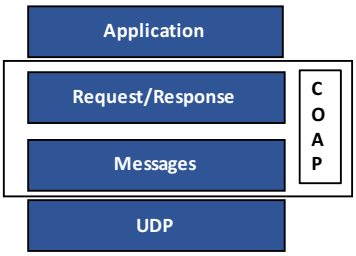

Fig. 4. CoAP Abstraction Layer [22]

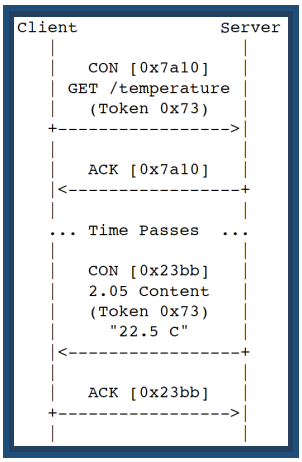

Fig. 5. Example of a CoAP GET acknowledges interaction [22]

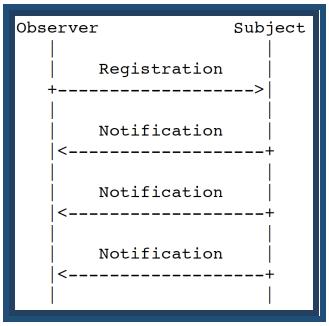

Fig. 6. Observer design pattern [27]

4. Virtualization & IoT

Virtualization in IoT has emerged from the success of common management techniques such as Software Defined Networking (SDN) [28], [29]. SDN abstracts and decouples the control plane (determine destinations of traffic) from data plane (forwarding traffic). The characteristics of the IoT space have led to the rise of Software-defined IoT (SD-IoT) [30] [31]. SD-IoT is a type of virtualization that uses abstractions to simplify provisioning and customization of the network components. Virtualizing components is a typical approach to managing IoT networks, e.g.: virtualized sensors [32] or interface services [33]. However, these virtualization techniques use the Cloud as the processing unit, which means they are not scalable and introduce significant latency and bandwidth consumption. Besides, this kind of virtualizations focus on abstracting individual components but neither defining an effective communication between them nor allowing richer interactions over them.
The development of Fog Computing gives IoT the possibility to introduce management techniques closer to the constrained environment reducing costs and being location and context aware. Implementing configuration management in a Fog network instead of the Cloud would represent a more realistic scenario to engage the IoT components. Solutions at the edge of the IoT network have been proposed to handle the traffic and provisioning tasks [34].

Botta et al. [29] present specific management considerations for IoT and Cloud networks:

- While IoT systems interact with real Things to be the data providers, Cloud systems interact with virtualization of Things to be the service providers.
- While IoT networks focus on a pervasive service using constrained devices and limited storage and energy, cloud networks focus on a ubiquitous service available from everywhere using virtually unlimited computational capabilities.

Definition of Virtual Resources

Because a Virtual Resource is a software artifact, there can be many definitions of Virtual Resources for different purposes and scenarios. This paper uses the definition of Virtual Resources proposed by Samaniego [35][36]. For configuration management, Virtual Resources are grouped into two levels of abstraction. Figure 7 illustrates this virtual architecture: first, the Atomic Abstraction Layer (A2L) that faces the IoT physical components (sensors, actuators) and manage a one-to-one configuration relation; second, the View Abstraction Layer (VAL) that is built on top of the A2L and handles the configuration of complex views that use the data from the A2L resulting in a one-to-many relation. Both layers are hosted in a Fog network.
For both layers, Virtual Resources are defined as RESTful micro services [36]. These services are programmed using the features of Go language [37]:

- routines
- channels

Routines allow to deploy multiple Virtual Resources simultaneously (see Figure 8. An Example of Virtual Resources programmed in Go language). Channels are the communication media among Virtual Resources (see Figure 9. An example of a Virtual Resource listening to a channel in Go language).
Virtual Resources in the Atomic and View layers communicate via the Constrained Application Protocol (CoAP). Also, both layers transmit data in a JSON format (see Figure 10. An example of JSON data transmitted by Virtual Resources).
Figure 11 shows the architecture of the Atomic Layer. The Atomic layer is the lowest level of abstraction. Atomic Virtual Resources engage IoT components directly. One Atomic Virtual Resource corresponds to a one physical IoT component, which leads to a one-to-one relation. An Atomic Virtual Resource exposes the CoAP method definitions: GET, POST, PUT, DELETE (Figure 12). By doing this, the simple CRUD operations can be mapped to the CoAP methods. Atomic Virtual Resources can share the current state of the physical component they represent, change their configuration, interact with other Atomic Virtual Resources, and make the physical component stop working.

Figure 13 shows the architecture of the View Abstraction Layer. View Virtual Resources have a higher hierarchy. One View Virtual Resource can engage one or more Atomic Virtual Resources, which leads to a one-to-many relation.
This work applies the Constrained RESTful Environment (CoRE) Link Format to CoAP to manage the two main services in the IoT space:

- the discovery of services
- the retrieving of state

The discovery of services process is exposed in a "/.well-known/core" interface, which returns a list of all available components with their URI's. The retrieving of state process is exposed in a "/state" interface, which returns the value of the IoT component by the time it was requested. View Virtual Resources expose these two interfaces.
A View Virtual Resource can engage not only Atomic resources but also another View Resources as well, in this case, the View becomes a Composite View Virtual Resource with a many-to-many relation.
In the View layer, Virtual Resources are divided into two groups, stateless and state-full. Stateless Virtual Resources use the CoAP REST pattern to pull the states each time a request is received. State-full Virtual Resources use the CoAP observe pattern to receive updates of the resources they are observing. State-full Virtual Resources use caching techniques to keep the state of their linked atomic resources while it is valid. The purpose of making View Virtual Resources state-full is to reduce the traffic load in the IoT network.
The View Layer also works as a processing unit in which the row data is evaluated and stored.

Please see Samaniego M. [9] *for additional information about the definition and implementation of Virtual Resources.*

By integrating Virtual Resources at different levels of abstraction, we can build a structure to facilitate the configuration management of the IoT network. Hence, users can interact with the virtualization of devices in real time and manage a specific virtual configuration without affecting other processes.

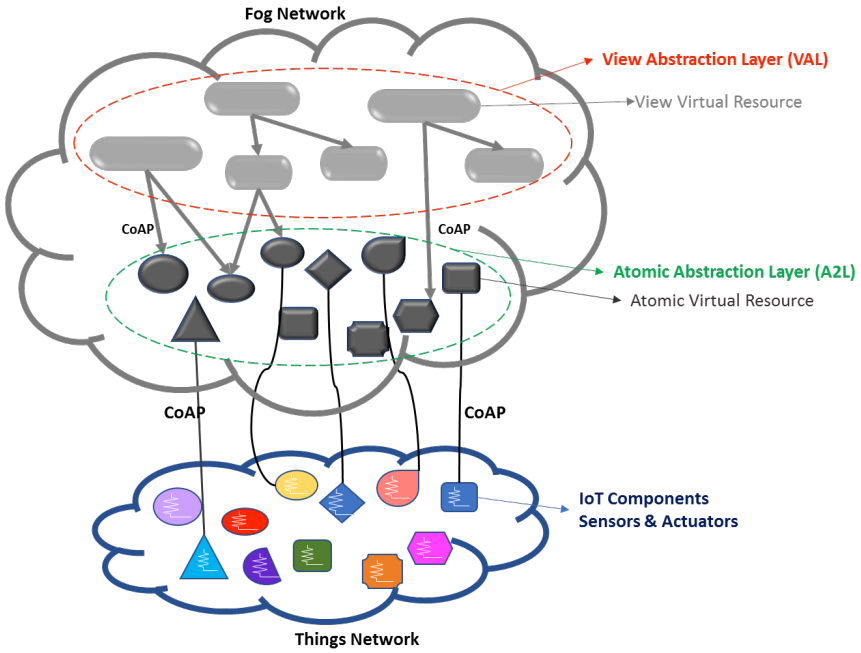

Fig. 7. Virtual Resource Architecture [9]

```
func invoke_code(w http.ResponseWriter, r *http.Request) {
    wg.Add(total_clients)
    go client1.Routine(&wg, max_delay, delay_interval, total_request, "", w, r)
    go client2.Routine(&wg, max_delay, delay_interval, total_request, "", w, r)
    go client3.Routine(&wg, max_delay, delay_interval, total_request, "", w, r)
    go client4.Routine(&wg, max_delay, delay_interval, total_request, "", w, r)
    go client5.Routine(&wg, max_delay, delay_interval, total_request, "", w, r)
    go client6.Routine(&wg, max_delay, delay_interval, total_request, "", w, r)
    go client7.Routine(&wg, max_delay, delay_interval, total_request, "", w, r)
    go client8.Routine(&wg, max_delay, delay_interval, total_request, "", w, r)
    go client9.Routine(&wg, max_delay, delay_interval, total_request, "", w, r)
    go client10.Routine(&wg, max_delay, delay_interval, total_request, "", w, r)
    wg.Wait()
}
```

Fig. 8. Virtual Resources programmed as routines in Go language [9]

```
func listening_to_exit() {
    for {
        select {
        case <-chan_stop:
            log.Println("I am leaving...  god bye")
            os.Exit(0)
        }
    }

}
```

Fig. 9. Function of a Virtual Resource listening to a channel [9]

```
{
  "url": "masp.local/router/vr_router_001.go",
  "routed_resources": [
    {
      "url": "masp.local:51275/door"
    },
    {
      "url": "masp.local:51276/temp"
    },
    {
      "url": "masp.local:51277/water"
    }
  ]
}
```

Fig. 10. Example of JSON data transmitted by Virtual Resources [9]

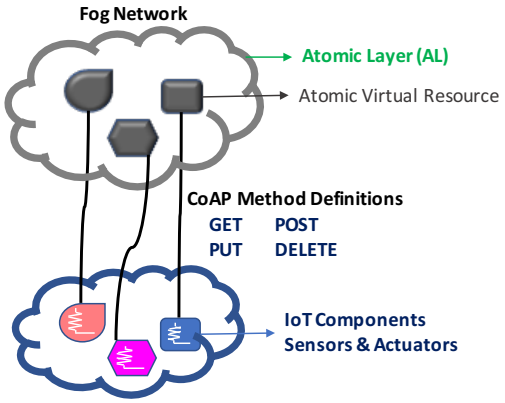

Fig. 11. Atomic Abstraction Layer [9]

```
func handle_request(m *coap.Message) (payload []byte, err error) {
    code := m.Code
    switch code {
    case coap.GET:
        payload, err = handle_GET("temperature")
    case coap.DELETE:
        chan_stop <- true
    }
    return
}
```

Fig. 12. Definition of a CoAP method in Go language [9]

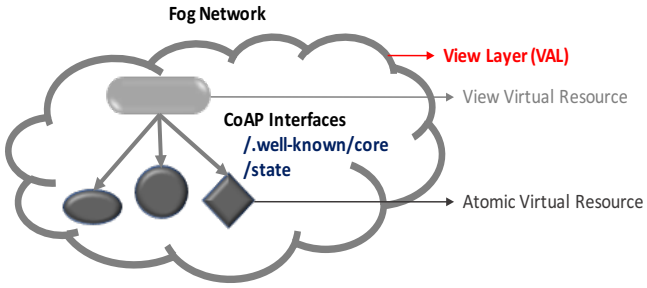

Fig. 13. View Abstraction Layer [9]

5. Blockchain & IoT

The conceptualization of Blockchain is attributed to Satoshi Nakamoto in 2008 [38]. Blockchain is a distributed ledger that stores transactions in the form of connected blocks. A blockchain starts with an initial or genesis block (Figure 14). Any change in any block is immediately visible to all participants. Each subsequent block is related to the previous one through a hash value.

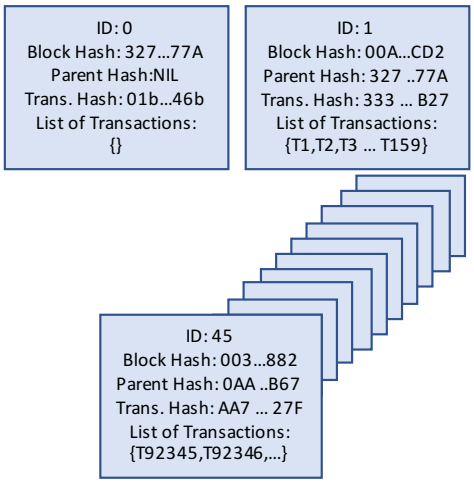

Fig. 14. Sample Blockchain [36]

Blockchain was of public domain when it was initially designed to work as the distributed transaction ledger for the Bitcoin project [39], but nowadays there are few options to install it privatively, e.g.: Multichain [40] or Hyperledger [41]. Multichain is free to use, but it is not open source; Hyperledger is free to use and open source as well.

Each Blockchain technology uses a different mechanism to write only valid transactions, e.g.: Multichain validates

transactions using the round-robin method [42], IBM Bluemix uses the consensus methods [43].
The use of Blockchain has extended to different areas such as health [44], government [45], and manufacturing [46]. In the IoT space, IBM presented the ADEPT system [47] as part of the Bluemix Solution. ADEPT can store the configuration of IoT devices and as a mechanism for pushing code onto devices.
Permission based blockchain works based on a set of registered users. This blockchain technology has the advantage of limiting the participation in the consensus processes as well as who can perform transactions.

6. Evaluation

For testing the definition of Virtual Resources, we deployed them in two IoT platforms: Raspberry Pi and Edison Arduino board.

For testing the Permission-Based Blockchain protocols as a provisioning mechanism, we used IBM Bluemix Blockchain as a Service and a Fog Multichain Blockchain cluster.

Overall, some picks have been observed in the experiments. As the difference between results is minuscule, those picks are attributed to the noise of the network (university wireless connection), memory allocation or background processes of the device.

Please see Samaniego et al. [9][35][36] for more experiments and evaluations.

6.1. Evaluation of Virtual Resources

6.1.1. Evaluation of the Performance of Virtual Resources

Experiment 1

This experiment tests the performance of the definition of Virtual Resources being accessed by a third party.

Figure 15 illustrates the setup of this experiment. The setup includes a Raspberry Pi to deploy a Virtual Resource, a Linux machine to run Elasticsearch database, and a Mac computer to run the clients.

The Raspberry Pi and the Database are part of a Fog network.

The Raspberry Pi hosts a state-full View Virtual Resource. This View goes to the database to read the current state of the Atomic Virtual Resources linked to it. The View Virtual Resource responses with an Acknowledgment CoAP message to all requests. The response includes the state of the resource in the payload.

The database server hosts Elasticsearch, which is a RESTful search engine for analytics [48]. The "query time" and "index time" features of Elasticsearch support the heterogeneity nature of the data generated in the IoT Cloud.

The client computer hosts two clients programmed in Go language. The clients send 500 CoAP GET requests each to the virtual resource. The client waits for the Acknowledgment of the current request before sending the next one.

The communication between the client and the Virtual Resource is performed via CoAP protocol. The client, the Virtual Resource, and the Database are connected to the university Wi-Fi.

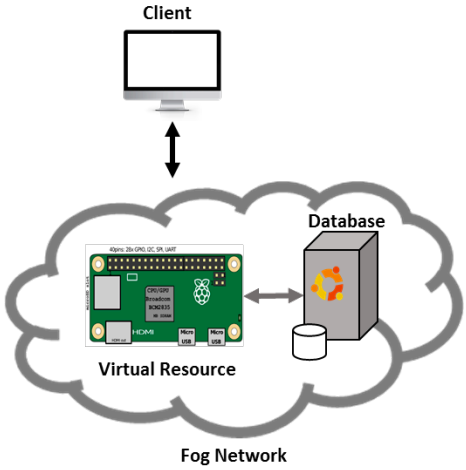

Fig. 15. Setup of Experiment 1. A Fog network formed by an Edison Arduino board that connects to an Elasticsearch database [9]

Table 2 shows the specification of the hardware for this experiment.

Table 2. Specification of the hardware used in experiment 1

Client	Mac OS X 2.5 GHz Intel Core i7 16 GB RAM
Virtual Resource	Raspberry Pi Model B Raspbian. Linux kernel 3.18 900 MHz ARM Cortex-A7 1GB LPDDR2 SDRAM
Database	Linux Ubuntu Intel Core i7-6700 CPU @ 3.40GHz 14 GB RAM Elasticsearch DB

This experiment considers three processes to evaluate the Virtual Resource definition:

1. the Core Link Format (RFC6690) discovery of services through a /.well-known interface
2. the retrieving of the current state of the Virtual Resource through a /state interface
3. the communication performance with the database

Figures 16 to 18 show the results of the above-mentioned processes. In the figures the x axis represents the requests of each client. The first 500 requests belong to Client 1 and the other 500 requests belong to Client 2. The y axis represents the response time in milliseconds.

This experiment introduces delay intervals of 0, 50 and 100 ms to test the response time of the Virtual Resource under different request loads.

The results were measured from the client side.

Figure 16 shows the results of the first part of Experiment 1. The round-trip time of the Discovery-of-Services process (/.well-known/core interface) for the 50-Delay series is between 0.2 ms and 0.6 ms, for the 50-Delay series the round-trip time is between 0.4 ms and 1.4 ms, for the 100-Delay series the round-trip time is between 0.6 ms and 1.6 ms.
From these results, we can argue that the delay intervals directly affect the performance of the virtual resource. The response times are higher as the virtual resource goes to the database to get the current state of the Atomic Virtual Resources. The higher the interval, the higher the response time.

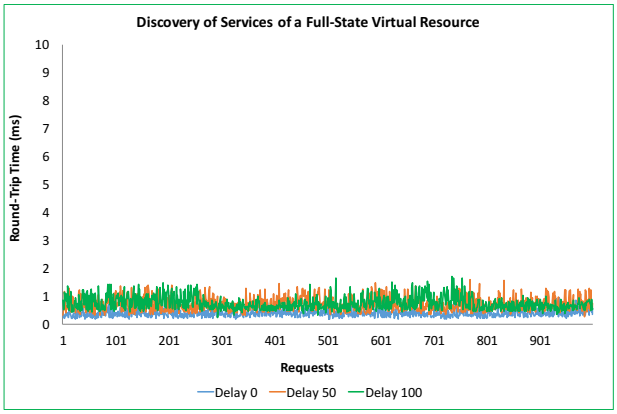

Fig. 16. Results of the Virtual Resource performance – Discovery-of-Services process (/.well-known/core interface).

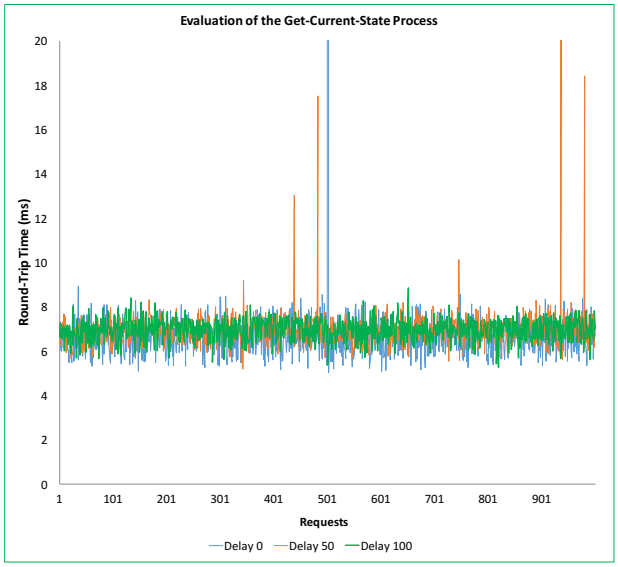

Fig. 17. Results of the Virtual Resource performance – Current State process (/state interface).

Figure 17 presents the results of the second part of Experiment 1. The round-trip time of the get-current-state process (/state interface) for all Delay series is between 5 ms and 9 ms. In this case, the Virtual Resource makes the processing locally, thus the delay times do not affect its performance.

Figure 18 shows the results of the third part of Experiment 1. The communication performance between the View Virtual Resource and the database. The database is Elasticsearch. This graph shows that the delay intervals do not affect the response times of the database server. For all three delay intervals, the round-trip time result is between 4.8 ms and 7 ms.
The database is accessed through an HTTP REST API programmed in Go language.

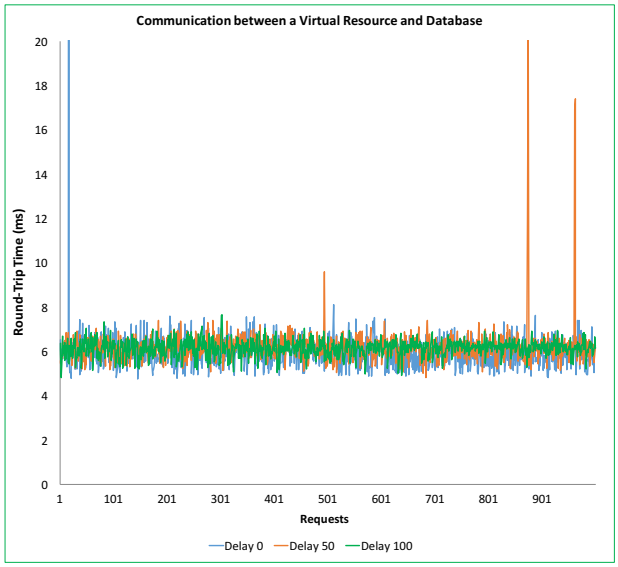

Fig. 18. Results of the Communication between the Virtual Resource and the Database.

6.1.2. Evaluation of the Communication among Virtual Resources

Experiment 2

The following experiment tests the communication performance among two Virtual Resources.
Figure 19 shows the setup of this experiment. The setup includes two Edison Arduino boards connected to the university Wi-Fi. The Edison boards is a System on a Chip (SoC) with the following characteristics: 500 MHz dual-core, dual threaded Intel Atom and a 100 MHz 32-bit Intel Quark microcontroller.

The first Edison board hosts an Atomic Virtual Resource that responds to all requests with an Acknowledgment CoAP message. The second Edison board hosts 10 Stateless View Virtual Resources that send CoAP POST requests to the Atomic Virtual Resource.

The payload sizes are 8 bytes and 512 bytes. All View Virtual Resources must wait for the Acknowledgment message before sending the next one.

Security is mandatory for data transmission in the IoT Cloud. Due to the limited resources of the Edison board, the payload is encrypted using Advanced Encryption Standard (AES) [49] with a key of 16 bytes. Hence the total size of the payload is 8/512 bytes + 18 bytes of AES encryption.

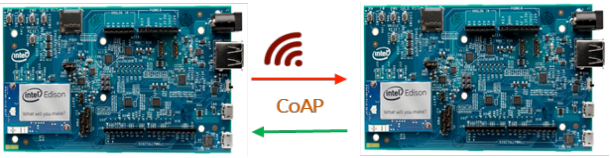

Fig. 19. Setup for Experiment 2. Two Edison Arduino boards communicating via CoAP protocol [9]

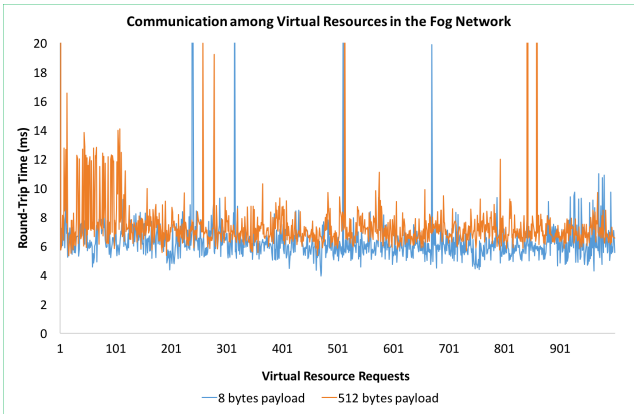

Fig. 20. Results of the communication between 2 Virtual Resources with different payload size.

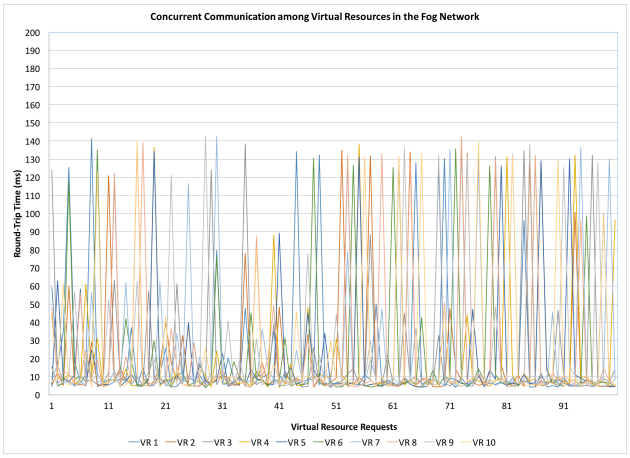

Fig. 21. Results of the concurrent communication between 2 Virtual Resources with a delay time of 300 ms.

Figure 20 shows the results of the first part of Experiment 2. In this part of the experiment, the second Edison Arduino board hosts 1 Stateless View Virtual Resource that sends 100 of CoAP requests to the Atomic Virtual Resource.

The graph shows that the payload influences the Atomic Virtual Resource behavior. With a payload of 8 bytes + 16 bytes of AES, the round-trip time is between 4 ms and 12 ms. The average round-trip time is 6.45 ms. With a payload of 512 bytes + 16 bytes of AES, the round-trip time is between 4 ms and 14 ms. The average round-trip time is 7.54 ms.
There are some picks in the graph that can be attributed to the noise of the Wi-Fi network and the background processes of the device.

Figure 21 presents the results of the second part of Experiment 2. In this part of the experiment, the second Edison Arduino board hosts 10 Stateless View Virtual Resources that send 100 of concurrent CoAP requests to the Atomic Virtual Resource. In this case, the payload is static, 8 bytes. A delay interval of 300 ms was introduced.

The graph shows many fluctuations in the round-trip time responses. This is because the Virtual Resource is sending the requests asynchronously. From the graph, we can comment that all requests were responded. The round-trip time for all requests is between 4 ms and 60 ms with picks up to 145 ms.

The concurrency introduced affects the response times of the Atomic Virtual Resource. However, the response times are acceptable as it is a constrained device the one that is facing concurrency.

6.2. Evaluation of Permission-Based Blockchain

6.2.1. Evaluation of a Blockchain as a Service

Experiment 3

This experiment evaluates the communication performance between a Virtual Resource and the Blockchain as a Service IBM Bluemix [50].
Figure 22 shows the setup for this experiment. The experiment involves a free Bluemix account and an Edison Arduino board that communicate via HTTP protocol.

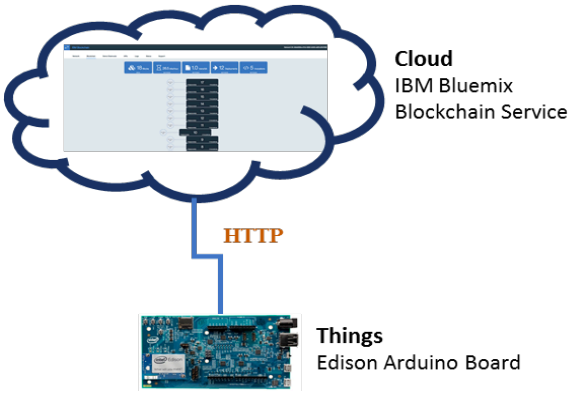

Fig. 22. Setup of Experiment 3. One Edison Arduino board is communicating with a free account of IBM Bluemix Blockchain as a Service. [9]

The API deployed in Bluemix to handle the communication with the blockchain cluster is programmed in Go language. The Edison board hosts 10 View Virtual Resources that send 100 HTTP POST requests to the blockchain service

asynchronously. Each request is a means to write a block in the blockchain. The payload is 712 bytes. In the payload, it is included 256 bytes of encrypted data using AES with a key of 16 bytes.

Figures 23 and 24 show the results of Experiment 1 with 0 ms and 300 ms delay time introduced respectively. Having the blockchain as a service in the Cloud means a significant impact on the communication performance with the Edison Arduino board. The two graphs show that the variation in the arrival rate of the requests does not lead to a better communication performance. The round-trip time for the requests with no delay time is between 100 ms and 2200 ms. The round-trip time for the requests with 300 ms delay time is between 100 ms and 1900 ms.

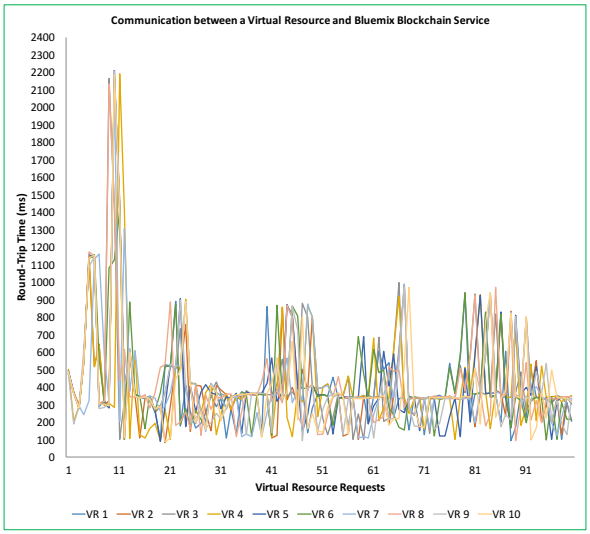

Fig. 23. Results of the communication between a View Virtual Resource and IBM Bluemix as a Service. No delay time introduced [9]**.**

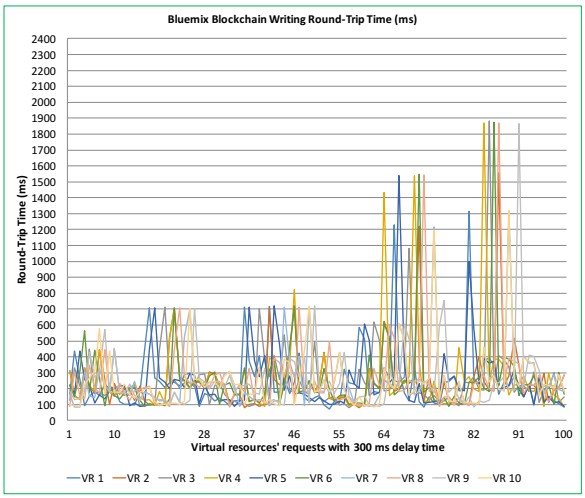

Fig. 24. Results of the communication between a View Virtual Resource and IBM Bluemix as a Service. 300 ms delay time introduced [9]**.**

6.2.2. Evaluation of a Blockchain Cluster hosted in a Fog network

Experiment 4

This experiment evaluates the communication performance between a Virtual Resource and a Multichain [40] Blockchain cluster hosted in a Fog network.
Figure 25 shows the setup for this experiment. The setup includes an Edison Arduino board and a Fog network of 3 machines running Multichain software. The characteristics of these machines are:

- Operating System: Linux Debian 8.5 (Jessie)
- CPU: Intel Core i7-6700 CPU @ 3.40GHz
- RAM: 14 GB

One of the Multichain nodes includes a python API that handles the communication with the cluster. The Edison board hosts 10 View Virtual Resources that send 100 HTTP POST requests each to the Multichain cluster asynchronously. The payload is 712 bytes. The POST requests ask permission to write in the blockchain.

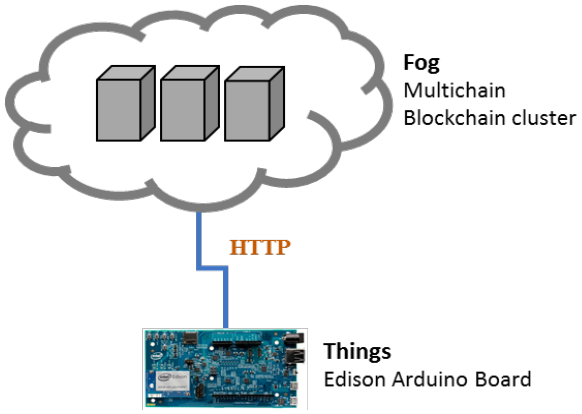

Fig. 25. Setup of Experiment 4. One Edison Arduino board is connected to a Multichain Blockchain cluster in a Fog network [9]

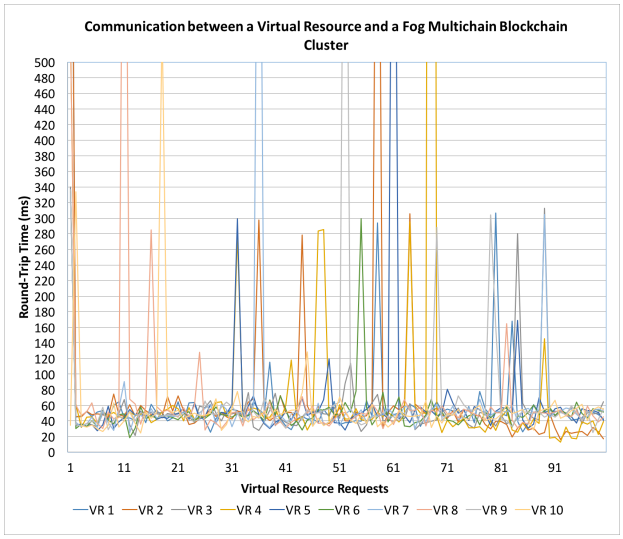

Fig. 26. Results of the communication between a View Virtual Resource and a Multichain Blockchain cluster in the Fog. No delay time introduced [9]**.**

Figures 26 and 27 show the results of Experiment 4 with 0 ms and 300 ms delay intervals respectively. The graphs show that hosting the blockchain closer to the IoT cloud network means better response times. However, with a delay interval of 300 ms, the Fog Multichain blockchain performs better. The round-trip time for the requests with no delay is between 20 ms and 80 ms. Some picks are observed in the graph, but we can attribute them to the fact of sending the requests concurrently and asynchronously. The round-trip time for the requests with 300 ms delay is between 20 ms and 60 ms. From the graph, we can comment that intervals introduced, affect the performance of the Multichain cluster. The scenario with 300 ms delay interval demonstrated a better performance.

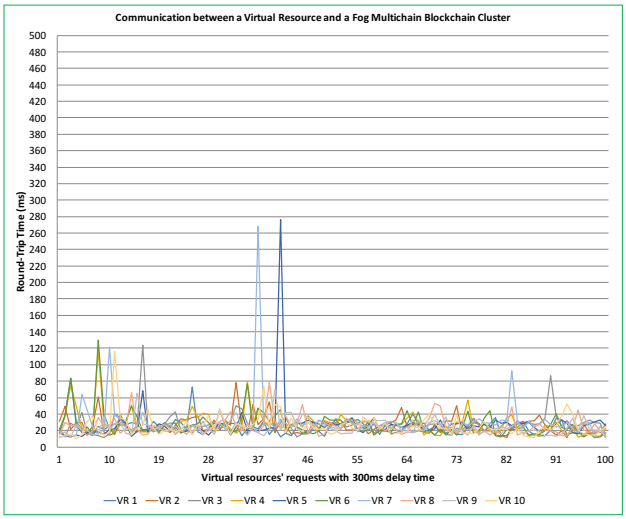

Fig. 27. Results of the communication between a View Virtual Resource and a Multichain Blockchain cluster in the Fog. 300 ms delay time introduced [9]**.**

7. Conclusions and Future Work

This work discussed the use of Virtual Resources and permission-based Blockchain to build an architecture to handle the configuration management in IoT networks.

The proposed architecture encapsulates the complexity of the configuration of IoT components into a Fog layer, which is accessed via Constrained Application Protocol (CoAP). The CoAP-based interfaces fulfill the performance requirements and reduce the complexity of configurations mapping CoAP requests to CRUD operations.

Hosting Virtual Resources in a Fog layer results in an economic solution regarding latency and bandwidth consumption because only meaningful information is transferred to the Cloud. Additionally, the evaluation of the data is done at the edge of the IoT network, which makes the decision-making process time-effective.

Overall, the virtualization of components performes in an expected manner, responding to all requests no matter the concurrency level it faced.

Permission-Based Blockchain adds security to the configuration management of the IoT network as only registered users can access to the blocks in the chain to write and read configurations. Storing these configurations in the form of blocks makes the access to IoT components dynamic and concurrent [51] as each user uses their own configurations to access to a specific physical device. Additionally, as the blocks in the chain are encrypted, only the user with the correct key can read the blocks and store new ones. Multichain cluster obviously performed better than the IBM Bluemix service in the Cloud due to the location in a Fog network, closer to the IoT network. However, Bluemix as a Service can be used to store Virtual Resources of higher hierarchy that are accessed by third parties in the Cloud.

The architecture composed by Virtual Resources and permission-based Blockchain fulfills the two challenges stated by this work, managing the configuration of IoT components and provisioning those configurations in the IoT network.

The experiments presented in this work consider a small number of devices. Future work will consider testing the definition of Virtual Resources in a larger scenario integrating other constrained devices. Additionally, other Blockchain protocols will be evaluated, such as Hyperledger and Ethereum.

References

[1] S. Nastic, S. Sehic, D. H. Le, H. L. Truong, and S. Dustdar, “Provisioning software-defined IoT cloud systems,” *Proc. - 2014 Int. Conf. Futur. Internet Things Cloud, FiCloud 2014*, pp. 288–295, 2014. https://doi.org/10.1109/ficloud.2014.52

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog Computing and Its Role in the Internet of Things,” *Proc. first Ed. MCC Work. Mob. cloud Comput.*, pp. 13–16, 2012. https://doi.org/10.1145/2342509.2342513

[3] S. Bandyopadhyay and A. Bhattacharyya, “Lightweight Internet protocols for web enablement of sensors using constrained gateway devices,” *2013 Int. Conf. Comput. Netw. Commun. ICNC 2013*, pp. 334–340, 2013. https://doi.org/10.1109/iccnc.2013.6504105

[4] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols and Applications,” *IEEE Commun. Surv. Tutorials*, vol. PP, no. 99, pp. 1–1, 2015.

[5] Cisco Systems, “Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are,” 2015.

[6] L. Tan, “Future internet: The Internet of Things,” *2010 3rd Int. Conf. Adv. Comput. Theory Eng.*, pp. V5-376-V5-380, 2010.

[7] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013. https://doi.org/10.1016/j.future.2013.01.010

[8] R. Cortés, X. Bonnaire, O. Marin, and P. Sens, “Stream Processing of Healthcare Sensor Data: Studying User Traces to Identify Challenges from a Big Data Perspective,” *Procedia Comput. Sci.*, vol. 52, pp. 1004–1009, 2015. https://doi.org/10.1016/j.procs.2015.05.093

[9] M. A. Samaniego Pallaroso, “Virtual Rsources & Management in IoT,” University of Saskatchewan,

2016.

[10] H. Shi, N. Chen, and R. Deters, "Combining Mobile and Fog Computing: Using CoAP to Link Mobile Device Clouds with Fog Computing," *Proc. - 2015 IEEE Int. Conf. Data Sci. Data Intensive Syst. 8th IEEE Int. Conf. Cyber, Phys. Soc. Comput. 11th IEEE Int. Conf. Green Comput. Commun. 8th IEEE Inte*, pp. 564–571, 2016.

[11] "About the Object Management Group." [Online]. Available: http://www.omg.org/gettingstarted/gettingstartedindex.htm. [Accessed: 16-Nov-2016].

[12] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003. https://doi.org/10.1145/857076.857078

[13] "Data Distribution Service (DDS)." [Online]. Available: http://www.omg.org/omg-dds-portal/. [Accessed: 28-Oct-2016].

[14] "What is DDS?" [Online]. Available: http://portals.omg.org/dds/what-is-dds-3/. [Accessed: 28-Oct-2016].

[15] A. Ghosh, "Message Queuing Telemetry Transport (MQTT) Protocol," 2014. [Online]. Available: https://thecustomizewindows.com/2014/07/message-queuing-telemetry-transport-mqtt-protocol/. [Accessed: 31-Oct-2016].

[16] OASIS, "MQTT Version 3.1.1," *OASIS Standard*, 2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. [Accessed: 09-Aug-2016].

[17] V. Lampkin, "What is MQTT and how does it work with WebSphere MQ? (Application Integration Middleware Support Blog)," 2012. [Online]. Available: https://www.ibm.com/developerworks/community/blogs/aimsupport/entry/what_is_mqtt_and_how_does_it_work_with_websphere_mq?lang=en. [Accessed: 23-Nov-2016].

[18] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks," *2008 3rd Int. Conf. Commun. Syst. Softw. Middlew. Work. (COMSWARE '08)*, pp. 791–798, 2008. https://doi.org/10.1109/comswa.2008.4554519

[19] P. Spiess *et al.*, "Soa-based integration of the internet of things in enterprise services," *2009 IEEE Int. Conf. Web Serv. ICWS 2009*, pp. 968–975, 2009.

[20] V. Lampkin, W. T. Leong, L. Olivera, S. Rawat, N. Subrahmanyam, and R. Xiang, "Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry," *IBM Redbooks*, p. 270, 2012.

[21] "How the AWS IoT Platform Works - Amazon Web Services." [Online]. Available: https://aws.amazon.com/iot-platform/how-it-works/. [Accessed: 17-Jan-2017].

[22] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," *The Constrained Application Protocol (CoAP)*, 2014. [Online]. Available: https://tools.ietf.org/html/rfc7252. [Accessed: 27-Oct-2016].

[23] M. Hemdi and R. Deters, "Data Management in Mobile Enterprise Applications," *Procedia - Procedia Comput. Sci.*, vol. 94, pp. 418–423, 2016. https://doi.org/10.1016/j.procs.2016.08.064

[24] M. Hemdi, "Using REST based protocol to enable ABAC within IoT systems," *Inf. Technol. Electron. Mob. Commun. Conf. (IEMCON), 2016 IEEE 7th Annu.*, pp. 1–7, 2016. https://doi.org/10.1109/iemcon.2016.7746297

[25] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A low-power CoAP for Contiki," *Proc. - 8th IEEE Int. Conf. Mob. Ad-hoc Sens. Syst. MASS 2011*, pp. 855–860, 2011. https://doi.org/10.1109/mass.2011.100

[26] A. Ludovici, P. Moreno, and A. Calveras, "TinyCoAP: A Novel Constrained Application Protocol (CoAP) Implementation for Embedding RESTful Web Services in Wireless Sensor Networks Based on TinyOS," *J. Sens. Actuator Netw*, vol. 2, no. 2, pp. 288–315, 2013. https://doi.org/10.3390/jsan2020288

[27] K. Hartke, "Observing Resources in the Constrained Application Protocol (CoAP)," 2015. [Online]. Available: https://tools.ietf.org/html/rfc7641. [Accessed: 27-Nov-2016].

[28] C. Doukas and I. Maglogiannis, "Bringing IoT and cloud computing towards pervasive healthcare," *Proc. - 6th Int. Conf. Innov. Mob. Internet Serv. Ubiquitous Comput. IMIS 2012*, pp. 922–926, 2012. https://doi.org/10.1109/imis.2012.26

[29] A. Botta, W. De Donato, V. Persico, and A. Pescape, "On the integration of cloud computing and internet of things," *Proc. - 2014 Int. Conf. Futur. Internet Things Cloud, FiCloud 2014*, pp. 23–30, 2014. https://doi.org/10.1109/ficloud.2014.14

[30] A. R. Biswas and R. Giaffreda, "IoT and Cloud Convergence: Opportunities and Challenges," *2014 IEEE World Forum Internet Things*, pp. 375–376, 2014. https://doi.org/10.1109/wf-iot.2014.6803194

[31] N. D. E. La, "T Elecom N Etwork a Rchitectures a N a Rchitecture for S Oftware D Efined W Ireless N Etworking," no. June, pp. 52–61, 2014.

[32] S. Alam, M. M. R. Chowdhury, and J. Noll, "SenaaS: An event-driven sensor virtualization approach for internet of things cloud BT - 1st IEEE International Conference on Networked Embedded Systems for Enterprise Applications, NESEA 2010, November 25, 2010 - November 26, 2010," 2010.

[33] D. Yang, T. Ng, and H. Lim, "Demo abstract: A service-oriented application programming interface for sensor network virtualization," *... Process. Sens. Networks*, pp. 143–144, 2011.

[34] M. Jutila, "An Adaptive Edge Router Enabling Internet of Things," *IEEE Internet Things J.*, vol. 4662, no. c, pp. 1–1, 2016. https://doi.org/10.1109/jiot.2016.2550561

[35] M. Samaniego and R. Deters, "Hosting Virtual IoT Resources on Edge-Hosts with Blockchain."

[36] M. Samaniego and R. Deters, "Using Blockchain to push Software-Defined IoT Components onto Edge Hosts," 2016.

[37] "Concurrency — An Introduction to Programming in Go | Go Resources." [Online]. Available: https://www.golang-book.com/books/intro/10. [Accessed: 06-Sep-2016].

[38] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System."

[39] "Innovation - Bitcoin." [Online]. Available: https://bitcoin.org/en/innovation. [Accessed: 16-Jan-2017].

[40] "MultiChain | Open source private blockchain platform." [Online]. Available: http://www.multichain.com/. [Accessed: 15-Jan-

2017].

[41] "Hyperledger Fabric," *2016*. [Online]. Available: http://hyperledger-fabric.readthedocs.io/en/latest/. [Accessed: 22-Nov-2016].

[42] A. Lewis, "In a nutshell: MultiChain (Epicenter Bitcoin interview – Nov 2015) | Bits on blocks," 2016. [Online]. Available: https://bitsonblocks.net/2016/03/07/in-a-nutshell-multichain-epicenter-bitcoin-interview-nov-2015/. [Accessed: 27-Nov-2016].

[43] "Testing consensus and availability." [Online]. Available: https://console.ng.bluemix.net/docs/services/blockchain/etn_pbft.html. [Accessed: 27-Nov-2016].

[44] K. Peterson, R. Deeduvanu, P. Kanjamala, and K. Boles, "A Blockchain-Based Approach to Health Information Exchange Networks," no. 1, pp. 1–10.

[45] Ø. Svein, "Beyond Bitcoin Enabling Smart Government Using Blockchain Technology," pp. 253–264, 2016.

[46] X. Li, F. Baki, P. Tian, and B. A. Chaouch, "A robust block-chain based tabu search algorithm for the dynamic lot sizing problem with product returns and remanufacturing," *Omega (United Kingdom)*, vol. 42, no. 1, pp. 75–87, 2014. https://doi.org/10.1016/j.omega.2013.03.003

[47] V. Pureswaran, S. Panikkar, S. Nair, and P. Brody, "Empowering the Edge: Practical Insights on a Decentralized Internet of Things," 2015. [Online]. Available: https://www-935.ibm.com/services/multimedia/GBE03662USEN.pdf. [Accessed: 22-Nov-2016].

[48] "Elasticsearch." .

[49] "AES encryption." [Online]. Available: http://aesencryption.net/. [Accessed: 22-Nov-2016].

[50] "IBM Bluemix - Cloud infrastructure, platform services, Watson, & more PaaS solutions." [Online]. Available: https://www.ibm.com/cloud-computing/bluemix/?S_PKG=&cm_mmc=Search_Google-_-IBM Cloud_Bluemix Program-_-WW_CA-_-bluemix ibm_Exact_&cm_mmca1=000002FP&cm_mmca2=10001882&mkwid=6af31007-4324-45da-aac1-ab29662f1bc1%7C401%7C236774&cvosrc=ppc.google.bluemix. [Accessed: 15-Jan-2017].

[51] M. Zhu, P. Grogono, O. Ormandjieva, and H. Kuang, "A Categorical Approach to Verifying Concurrency between Design and Implementation," vol. 8, no. 1, pp. 7–13, 2017.