VIRTUAL RESOURCES & INTERNET OF THINGS

A Thesis Submitted to the

College of Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the degree of Master of Science

in the Department of Computer Science

University of Saskatchewan

Saskatoon

By

MAYRA ALEJANDRA SAMANIEGO PALLAROSO

PERMISSION TO USE

ABSTRACT

Internet of Things (IoT) systems mostly follow a Cloud-centric approach. These systems get the benefits of the extensive computational capabilities and flexibility of the Cloud. Although Cloud-centric systems support virtualization of components to interact with IoT networks, many of these systems introduce high latency and restrict direct access to IoT devices. Fog computing has been presented as an alternative to reduce latency when interacting with IoT networks, however, new forms of virtualization are required to access physical devices in a direct manner.

This research introduces a definition of Virtual Resources to enable direct access to IoT networks and to allow richer interactions between applications and IoT components. Additionally, this work proposes Virtual Resources as a mechanism to handle the multi-tenancy challenge that emerges when more than one tenant tries to access and manipulate an IoT component simultaneously. Virtual Resources are developed using Go language and CoAP protocol. This work proposes permission-based blockchain to provision Virtual Resources directly on IoT devices. Seven experiments have been done using Raspberry Pi computers and Edison Arduino boards to test the definition of Virtual Resources presented by this work. The results of the experiments demonstrate that Virtual Resources can be deployed across different IoT platforms. Also, the results show that Virtual Resources and blockchain can support multi-tenancy in the IoT space. IBM Bluemix Blockchain as a Service and Multichain blockchain have been evaluated handling the provisioning of Virtual Resources in the IoT network. The results of these experiments show that permission-based blockchain can store the configurations of Virtual Resources and provision these configurations in the IoT network.

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my supervisor Dr. Ralph Deters for accepting me in his group of students. During these two years, his knowledge and support were fundamental to pursue my master's degree. I am very grateful to his sincerity when suggesting to improve in certain areas, and to his trust when encouraging me to take new challenges.

I want to thank my husband for his love and continually support. His beautiful heart and tenacity in pursuing his dreams makes me feel very proud of him. Thanks to my father for always encouraging me to learn new things and be a model for others to follow. Thanks to my sister for being my best friend and confident. I admire her strength and positive attitude. Thanks to my beautiful cousin for believing in me. Her support during these two years has been very important for me. Thanks to all my family for always sending me the best wishes from home.

Finally, I want to thank my angel in heaven that always encouraged me to pursue my dreams even though that meant the sadness of my absence. My mom.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ADEPT	Autonomous Decentralized Peer-to-Peer Telemetry
AES	Advanced Encryption Standard
CoAP	Constrained Application Protocol
CRUD	Create Read Update Delete
DaaS	Data as a Service
DDS	Data Distribution Service
ESB	Enterprise Server Bus
Golang	Go Language
HTTP	Hyper Text Transfer Protocol
IIoT	Industrial Internet of Things
IoE	Internet of Everything
IoT	Internet of Things
M2M	Machine-to-Machine
MQTT	Message Queue Telemetry Transfer
ms	Milliseconds
NIST	National Institute of Standards and Protocols
PBFT	Practical Byzantine Fault Tolerance
REST	Representational State Transfer
RR	Round Robin
RTT	Round-Trip Time
SoC	System on a Chip
SAaaS	Sensing and Actuation as a Service
SaaS	Software as a Service
SD-IoT	Software-Defined Internet of Things
SEaaS	Sensor Event as a Service
SDN	Software Defined Network
SOA	Service Oriented Architecture
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
WS	Web Service

CHAPTER 1

INTRODUCTION

The Internet of Things adds sensing and actuating capabilities to common “Things” to capture data from the real world [1]. The “Things” can work in different scenarios and under different conditions. Virtual systems use the “Things,” or IoT devices, to offer new forms of communication and services in many areas such as healthcare [2], transportation [3], smart homes [4], public services [5], industry [6], and other processes. According to a study from Gartner [7] (Figure 1.1), in 2009 the number of powerful devices (e.g. laptops and cellphones) connected to the Internet had not reached five billion, and this number is not expected to reach ten billion by 2020. On the other hand, this study says that there were around two billion IoT devices (e.g. sensors and actuators) connected to the Internet in 2009. By 2020 the number of connected IoT devices is expected to increase to twenty-six billion. The low cost of IoT computing supports these estimates. According to Klubnikin A. [8], the price of sensors has dropped by almost 200% between 2004 and 2016.

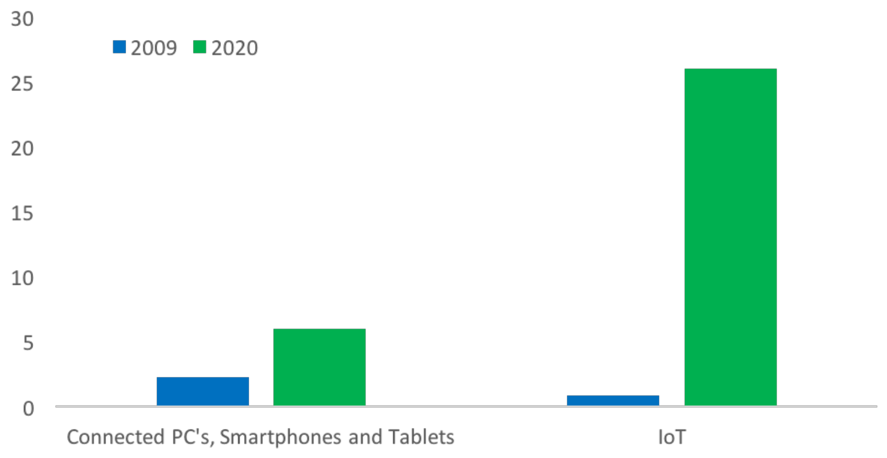


Figure 1.1 Number of connected PC's, Smartphones and Tablets vs. IoT connected devices [7]

Gubbi et al. [9] identify two groups of systems in the IoT space. The first group is Things-centric systems, which highlight the features of devices to provide a richer user experience. The second group is Cloud-centric systems, which focus on IoT services and data processing. Most IoT systems are based on the Cloud-centric approach (e.g. [10], [11], [12]), they are hosted on the Cloud and get the benefits of extensive computational capabilities and virtualization support. Virtualization techniques have already been studied in the IoT space, for example, virtualization of physical sensors on the Cloud for sharing purposes [13] and software-defined IoT units on the Cloud for a unified access [14]. Overall, the primary interest of IoT virtualization on the Cloud is data, which means that the communication is a static reading process.

Even though the Cloud represents a robust and reliable architecture for IoT analytics, its consolidated power does not fit the dynamic characteristics of IoT networks. For example, according to a study by Cortés et al. [15] about IoT in the health field, the centralized Cloud storage cannot handle the velocity of the data flow generated by sensing devices in real-time. Additionally, the significant latency restricts the direct access to IoT components and might affect the decision-making over data [16]. Finally, because the Cloud's architecture does not enable direct access to IoT components, multi-tenancy is not a concern in Cloud-centric systems.

Fog computing extends the Cloud features toward the edge of networks to deal with specific characteristics of some networking scenarios such as a large set of heterogeneous nodes, geographical location, and real-time communication [17], [18]. In the IoT space, Cisco explains that Fog nodes can directly access physical devices, consequently reducing latency and bandwidth consumption [16] (e.g. [18]). According to Bonomi et al. [19], IoT analytic tasks can be moved to a Fog layer as well. Due to the low latency cost, Fog computing facilitates virtualization and access to IoT components, however, when various users try to engage those components at the same time, multi-tenancy issues emerge.

This research proposes a definition of Virtual Resources to allow direct manipulation of IoT components in a multi-tenant manner. Virtual Resources are programmed using Go language and CoAP protocol. Permission-based blockchain is used to handle the provisioning of Virtual Resources directly onto IoT devices.

The remaining parts of this work are organized as follows:

- Chapter 2 - Problem Definition discusses the questions that arise when virtualization of IoT components is intended to be hosted closer to the “Things” layer.
- Chapter 3 - Literature Review analyzes previous IoT works about system architectures, Cloud computing, Fog computing and communication patterns. Challenges regarding multi-tenancy are reviewed. Virtualization of IoT components and permission-based blockchain are studied as relevant concepts to develop and provision Virtual Resources in IoT networks.
- Chapter 4 - Architecture explains the definition of Virtual Resources presented by this work and their provisioning using blockchain.
- Chapter 5 - Implementation presents the technology needed to build and provision Virtual Resources onto IoT devices.
- Chapter 6 - Experiment tests the performance of Virtual Resources and blockchain in the IoT space.
- Chapter 7 - Conclusion and Future Work describes this research's next steps.

CHAPTER 2

PROBLEM DEFINITION

With the advent of Fog computing, some Internet of Things (IoT) tasks can be moved closer to physical devices. A solution to handle the virtualization issues that arise from moving computation closer to the IoT "Things" layer is needed. This work proposes Virtual Resources as the main mechanism to allow richer interactions and to enable multi-tenant access to IoT components. Virtual Resources are required to face the dynamic characteristics of IoT networks [20], [9]:

- Heterogeneous platforms
- Large set of devices
- Limited computational capabilities
- Limited energy consumption
- Geographical distribution
- Real-time operations

The following questions should be answered to implement Virtual Resources:

2.1 How to define Virtual Resources?

Virtual Resources are required to have a light architecture that can be supported by the limited computational capabilities of constrained devices. Virtual Resources should be programmed in a cross-platform language that allows their compilation into different IoT platforms. Finally, Virtual Resources should expose simple interfaces to manipulate their current state and to communicate with other resources.

2.2 How to provision Virtual Resources?

The provisioning of Virtual Resources is required to work on demand. When a tenant requires access to an IoT component, then, the correct configuration of Virtual Resources should be executed in the corresponding device. Provisioning also should include removing Virtual Resources on demand as well.

2.3 How to guarantee multi-tenant access to IoT components?

Multi-tenancy is the main issue that arises when engaging IoT components from a Fog layer. Virtual Resources are required to support multi-tenant interactions in real-time. Each tenant should have their configuration of Virtual Resources isolated from the other tenants.

CHAPTER 3

LITERATURE REVIEW

Placing computation at the edge of Internet of Things (IoT) networks demands support for virtualization, multi-tenancy and provisioning of resources. This work reviews important literature to address the challenges described in Chapter 2.

- Internet of Things
    - Things-centric systems
    - Cloud-centric systems
- Fog Computing
- Multi-Tenancy
- Software-Defined IoT and Virtualization
- Blockchain
- Architectural Design Patterns
    - Service Oriented Architecture (SOA)
    - Representational State Transfer (REST)
- Communication Patterns in IoT
    - Data-centric
    - Message-centric
    - Resource centric

3.1 Internet of Things (IoT)

The Internet of Things (IoT) enables connectivity with the real world anytime and anywhere [21]. An IoT definition has not been yet formalized. Cisco expands the IoT concept to the Internet of Everything (IoE), including anything that supports sensing and connectivity [22].

IBM refers to IoT as an industrial revolution (IIoT [23]), which enables machine-to-machine and human communication [24]. Microsoft introduces IoT as the adoption of low-cost and pervasive hardware [25]. Although these definitions highlight different aspects of the IoT paradigm, they follow the same vision, which is having a large number of constrained devices connected to the Internet to obtain data from the real world [26]. This vision has been observed by Howard P. [27]. Figure 3.1 shows that between 2011 and 2015, the number of connected devices has grown exponentially.

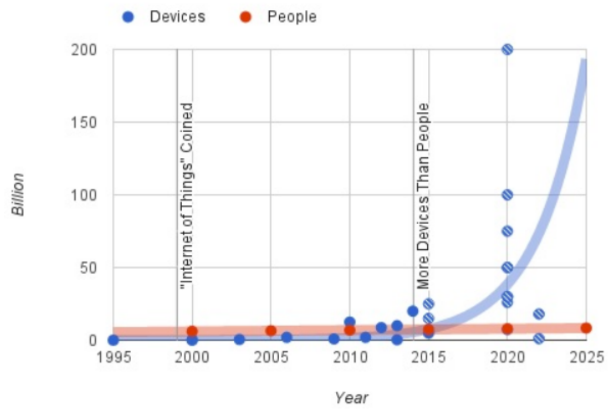


Figure 3.1 Trend of Devices vs. People [27]

According to Wu et al. [20], an IoT network is composed of three main layers:

- Perception
- Network
- Application

The Perception layer groups IoT physical devices either to sense data from its surroundings or to execute specific actions, both in real time. The Network layer represents the connection between IoT systems to handle data transmission. Finally, the Application layer denotes IoT systems that process and share data. Gubi et al. [9] identify two groups of IoT systems, Thing-centric and Cloud-centric.

3.1.1 Things-Centric IoT

"Things," is the generic term to refer to objects with sensing, actuating and connectivity capabilities, which can be reached anytime and anywhere [26]. The Things-centric approach enhances the features of devices to enrich the user experience, for example, smart objects [28] and enchanted devices [29]). Although these systems allow users to change the configuration of their devices, they do not support multi-tenancy because a device can only be engaged by one user (tenant) at the same time.

3.1.2 Cloud-Centric IoT

Even though Cloud Computing [30] and IoT are two paradigms that emerged separately to face different requirements (Table 3-1), both are considered complementary technologies to build a flexible deployment environment for IoT systems (e.g. [10], [12]). While IoT works in the real environment and lacks computational capabilities, Cloud Computing provides access to virtualized and scalable services over the Internet [18]. The Cloud benefits IoT in the following aspects [16], [31]:

- efficient use of resources
- orchestration of resources
- on-demand self-service
- broad network access
- resource pooling
- rapid deployment and elasticity
- planned services

As explained in Chapter 2, Cloud-Centric systems consist of three primary layers: Things, Service, and Application. Figure 3.2 illustrates this architecture. The Things layer is the lowest-level of abstraction and represents constrained devices, for example sensor and actuator networks. The Application layer is the higher-level of abstraction and hosts final solutions such as monitoring, managing, and other processes. Finally, the Service layer is the bridge between Applications and Things. This layer virtualizes IoT components and hosts all main IoT services such as data storage, analytics, and other processes.

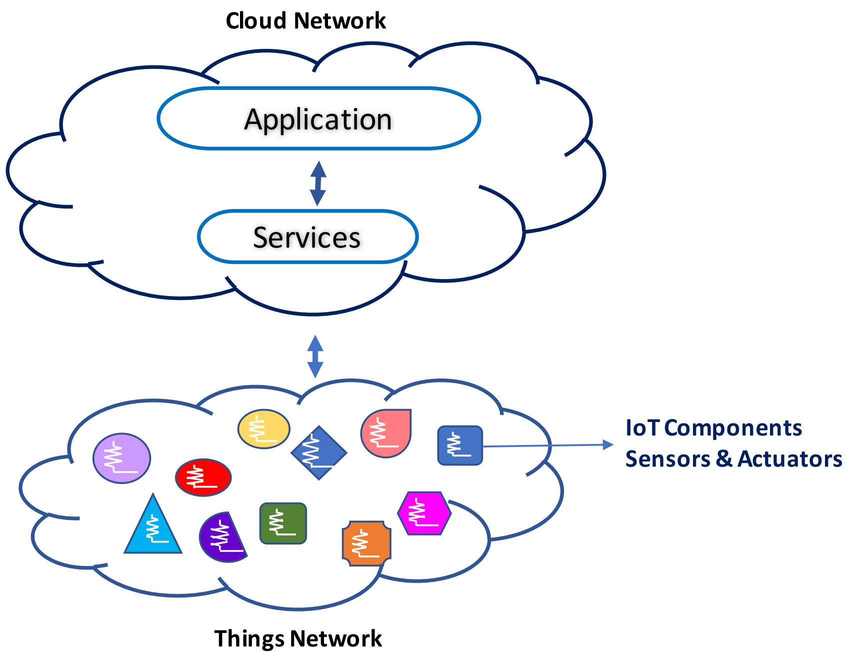


Figure 3.2 Representation of the three layers of Cloud-centric IoT systems

Cloud–centric systems introduce some limitations. For instance, Cloud-centric systems support multi-tenancy as multiple tenants can interact with the virtualizations hosted in the Service layer; however, this interaction with the IoT network is a static one-direction communication that avoids direct access to physical devices and focuses on sensor data. Although Cloud systems make data processing efficient and reliable [32], the time data streams take to reach the Cloud may affect the accurate decision-making over that data [16]. Finally, these systems introduce significant latency, network traffic and bandwidth consumption [16].

In the IoT space, it is mandatory to have low latency when engaging the geographically distributed devices. The next section explores Fog computing as an option to engage IoT networks geographically closer than from the Cloud. Furthermore, the challenges that emerge from hosting IoT components closer to physical devices are addressed.

Table 3-1 IoT vs. Cloud scenarios [31]

IoT	Cloud
Pervasive (things placed everywhere)	Ubiquitous (resources usable from everywhere)
Real world things	Virtual resources
Limited computational capabilities	Virtually unlimited computational capabilities
Limited storage or no storage capability	Virtually unlimited storage capabilities
The Internet as a point of convergence	The Internet for service delivery
Big data source	Means to manage big data

3.2 Fog Computing and IoT

Cisco describes Fog computing as an extension of the Cloud [16]. Fog Computing paradigm moves the features of the Cloud toward the edge of networks. The characteristics of Fog computing are [16]:

- edge location
- geographical distribution
- large-scale networks
- a significant number of nodes
- mobility support
- real-time interactions
- wireless connectivity supremacy
- interoperability and organization
- heterogeneity
- analytic support

A Fog layer benefits IoT in the following aspects [16], [18]:

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

Table 3-2 presents a comparison between Fog and Cloud environments. Overall, the response times in a Fog node are lower than in the Cloud. Although a Fog node works in a local area and the time that data remains stored is short, those conditions are enough to do some processing tasks in real-time and to avoid sending the entire row data to the Cloud. Additionally, analyzing data in a Fog node increases the accurateness of decision-making over that data and makes the analysis time-effective.

In the IoT space, a Fog layer allows engaging physical devices, reducing latency. For example, Aazam and Huh [33] introduce a Fog "Smart Gateway," which processes data in real time and enhance the communication and service provisioning in the Cloud. IoT applications hosted in a Fog layer are capable not only to read data from devices but also to manipulate them, e.g. updating software versions, triggering alarms or engaging actuators. However, the following challenges regarding multi-tenancy emerge:

- virtualization on the constrained and heterogeneous IoT components
- secure & safe software distribution of resources over IoT devices
- access control in the IoT network

Since Fog computing presents new possible interactions with the IoT network, multi-tenancy is demanded as the main feature in IoT systems. The next section explores literature and related work about multi-tenancy in IoT.

Table 3-2 Comparison between Fog and Cloud environments [31]

	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

3.3 Multi-Tenancy in IoT

Multi-tenancy is the characteristic of an architecture that shares resources and serves multiple users (tenants) in a cost-effective and secure manner [34]. Serving multiple tenants means that they must operate within different contexts to share resources in a successful manner. Multi-tenancy has been studied in many areas such as databases [35] and Cloud-hosted services [36], [37]. In the IoT space, many studies identify multi-tenancy challenges, for example, control flow [38] and access rights [19], [31]. Although these studies have observed that multi-tenancy plays a major role in IoT, Cloud-centric and Things-centric systems tend to ignore it [9]. Cloud-centric systems, e.g. [39], avoid direct communication with devices and focuses on the provisioning of services and applications that process large data streams received from constrained networks. Things-centric systems focus on the user experience configuring devices to meet the needs of a single tenant, e.g. the enchanted umbrella that notifies the user of possible weather changes [29].

According to Xu [40], conflicting settings of actuators is another challenge when enabling multi-tenancy over IoT components. Supporting multi-tenancy means that tenants should be able to interact and configure IoT components based on their specific requirements. Each tenant should be able to manage their own configuration in an isolated manner so it does not affect other tenants that can be working with the same components at the same time.

The following section analyzes virtualization in IoT and how it can help building multi-tenant systems in a Fog environment.

3.4 Software-defined IoT (SD-IoT) and Virtualization IoT

Software-defined elements emerged from the concept of Software Defined Networking (SDN) [41], [42]. SDN is a management concept that uses virtualization to decouple the control plane (determine destinations of traffic) from the data plane (forwarding traffic) and manage network functionalities. SDN enables programmability to network application development [43], [44]. Network virtualization focuses on virtualizing all elements of the network, which results in the ability to define customized virtual networks [45].

The success of SDN and network virtualization has led to the rise of Software-Defined IoT. According to Nastic et al. [14], "Software-defined IoT units are used to encapsulate the IoT resources and lower level functionality in the IoT cloud and abstract their provisioning and governance, at runtime." Software-defined IoT units can encapsulate the complexity of access and customization of the IoT network. Many works have covered virtualization of IoT components in the Cloud. For example, Sensing and Actuation as a Service (SAaaS) [46], which offers sensors and actuators resources as services hosted in the Cloud. Data as a Service (DaaS) [47], which offers ubiquitous access to data. Sensor Event as a Service (SEaaS), which offers management of events [48]. Similarly to virtualization in the Cloud, many studies have proposed virtualization of IoT components in a Fog layer, for instance, gateways [33] and routers [49]. These studies have focused on virtualizing individual components. According to Biswas and Giaffreda [50], another potential benefit for IoT is building software-defined ecosystems or Virtual Systems.

Building Virtual Systems at the edge of the IoT network opens the possibility of provisioning Virtual Resources directly on constrained devices, such as Raspberry Pi computers or Arduino boards. Configuring Virtual Systems for each tenant is a mechanism for:

- handling multi-tenancy as each tenant uses their own Virtual System's configuration
- distributing the workload to a Fog layer as Virtual Resources deployed on constrained devices can do real-time processing or analytic tasks
- enabling controlled and low-latency access to physical devices

The high number of constrained devices and their resource-limited characteristics represent a challenge when provisioning Virtual Resources in the IoT network. The next section explores blockchain technology as an option for software provisioning and versioning control on IoT devices.

3.5 Blockchain and IoT

Introduced by Bitcoin in 2009 [51], blockchain represents the public ledger that stores Bitcoin transactions in the form of blocks. Blocks are connected through a hash value forming a chain (blockchain). A blockchain is a peer-to-peer network which allows the execution of direct transactions without any central verification authority. Transactions are validated by a consensus mechanism in which participants must invest computation to show trustworthiness. In Bitcoin, this consensus is a proof-of-work task based on cryptography hashes algorithms. Participants must scan the hash value to be able to write new blocks or update existing ones. Changing a block means that the proof-of-work must be redone, in the current block and in the subsequent ones that have been added, that is why changing or updating blocks is not a standard or functional task. All participants of the network have a copy of the blockchain and are notified about new blocks and changes over them. Because Bitcoin transactions can be seen by anyone on the Internet, for example, the transactions of the bitcoin address https://blockchain.info/ [52], Bitcoin systems are considered public blockchains.

A public blockchain protocol gives open access to transactions and blocks [51]. Any user on the Internet can interact with a public blockchain. However, a public blockchain protocol does not trust any participant; participants must validate their transactions by a proof-of-work consensus mechanism. The proof-of-work process involves significant time and computation.

When the use of blockchain is limited to an organization or company, then a private blockchain protocol is a more convenient option. Private blockchains manage the blocks of transactions and permissions in a centralized manner [53]. A private blockchain only trusts a set of registered participants [54]. Even though registered participants do not have to do proof-of-work; this participation has to be managed by a consensus mechanism, such as Practical Byzantine Fault Tolerance (PBFT) [55], Round Robin (RR) scheduling [56], and other consensus algorithms. Transactions in a private blockchain are validated by a small number of participant nodes. The organization running a private blockchain has total control over all the elements of the blockchain for instance blocks, transactions, and permissions.

The most popular example of blockchain in IoT is the IBM's ADEPT system [57]. ADEPT uses Ethereum, an open source blockchain protocol, to manage device coordination functions such as storing the configuration of devices and authentication. IBM also has a private Blockchain as a Service (BaaS), which stands on Bluemix [58]. Bluemix offers a virtual blockchain cluster in a private network.

An area that is keeping attention within blockchain is smart contracts [59]. Smart contracts were proposed by Szabo N. [60] in 1993. A smart contract is defined as "a computerized transaction protocol that executes the terms of a contract" [60]. According to Christidis and Devetsikiotis [54], in the contexts of IoT, smart contracts represent a convenient option to define the business rules to interact with the blockchain.

Using blockchain in the IoT space puts forward new management possibilities. In this work, blockchain is used to manage the provisioning of Virtual Resources and to control access to the IoT network.

How to represent each resource in the IoT network is a question that arises. The next section analyzes different architectural patterns to represent the IoT network.

3.6 Architectural Design Patterns

3.6.1 Service Oriented Architecture (SOA)

The Service Oriented Architecture (SOA) [61] states well-defined and loosely coupled services. The SOA design goal is to build services to fulfill the business need of a company [62]. Many works have studied SOA in IoT, for example, [63] proposes a solution based on SOA to handle some IoT tasks such as discovery and provisioning of resources; [64] proposes an implementation that follows the SOA approach to integrating IoT within enterprise services. Additionally, many studies show that SOA would be a good architecture to deal with service providers and users [65], [66]. The previous studies indicate that a service architecture might work fine for the Application and Network layer of the IoT architecture.

3.6.2 Representational State Transfer (REST)

The Representational State Transfer (REST) is a resource-based architectural style for distributed hypermedia systems proposed by Roy Fielding in 1994 [67]. Rather than focusing on services, REST stands for the concept of state-full resources. Resources on the server side are accessed through a URI (uniform resource identifier).

The REST constraints highlight "scalability of component interactions, generality interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems" [67].

Web APIs and REST-based methods are the basis of many studies in IoT, e.g. [68], [69], [70]. These studies show that adopting the REST design leads to higher scalability, reliability, and decoupling in systems. The REST architectural design, which focuses on resources, satisfies the requirements and characteristic of the three layers of IoT systems. Additionally, the use of lightweight data-exchange formats (being JSON the most used) can reduce the overhead related to network bandwidth and storage capacity in the IoT Cloud.

REST enforces a resource-oriented view on the constrained components. In an IoT network, constrained components can be represented as full state entities, and the interaction can be mapped to CRUD operations. A particularly interesting aspect of using REST to model the Things layer is the possibility to define Virtual Resources on top of existing "Things."

A communication protocol that follows the REST approach is needed to build Virtual Systems for different tenants. The next section analyzes the communication patterns in IoT.

3.7 Communication Patterns in IoT

IoT communication patterns can be classified into three groups [71]: data-centric, message-centric and resource-centric.

3.7.1 Data-centric IoT Communication

The Data-Centric communication pattern focuses on the transmission of data in a reliable and secure manner.

DDS stands for Data Distribution Service. DDS is a standard developed by the Object Management Group's (OMG) [72]. DDS is a data-centric and publish-subscribe (DCPS) model for distributed application communication and integration [73].

- Data Centric because DDS has a Global Data Space in which data is defined and rules to access that data are structured.
- Publish-Subscribe because DDS provides a middleware that allows having multiple readers subscribed to a topic and writers publishing to those topics [74].

DDS enables "Efficient and Robust Delivery of the Right Information to the Right Place at the Right Time" [73]. DDS focuses on delivering data with Quality of Service (QoS) and reliability. DDS offers 23 QoS policies that developers can address such as security, priority, reliability and other policies that can be used when programming DDS [75].

Figure 3.3 shows the communication diagram of DDS. Topics enable the publishing and subscribing processes. DDS Domains keep completely isolated from each other. There is no data-sharing across DDS domains [74]. Following the publish/subscribe pattern [76], writers and readers work in a decoupled environment regarding synchronization and time

- Time: It is not necessary that writers and readers be active at the same time
- Synchronization: It is not necessary that readers have any information about writers, and vice versa

According to Esposito [77], DDS performs well and shows good scalability when the number of participants increases. This behavior would fit well for IoT environments in which the number of writers (sensors) is counted in hundreds and millions. This work seeks to enable multi-tenancy over IoT components that produce data (writers), DDS would not be a suitable option for this purpose as readers and writers are separated.

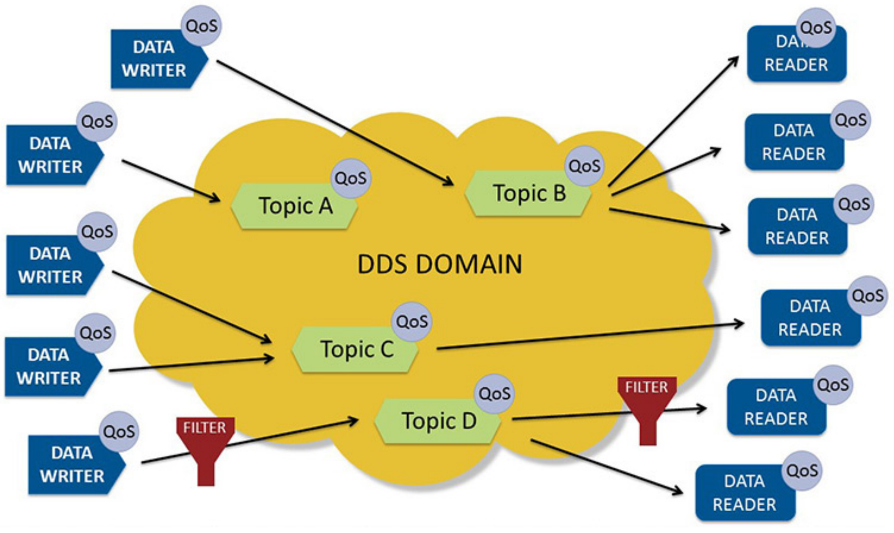


Figure 3.3 Data Distribution Service (DDS) Diagram [74]

3.7.2 Message-centric IoT Communication

The primary focus of the Message-Centric communication pattern is the delivery of reliable messages from writers to readers.

MQTT stands for Message Queuing Telemetry Transport [78]. MQTT is a lightweight message-centric protocol based on the publish/subscribe pattern [76]. Figure 3.4 shows the MQTT architecture in which many clients are supported by one broker [79]. MQTT uses TCP for communicating with the message broker. Using TCP can lead to high communication costs. Consequently, a UDP-based MQTT for sensors (MQTT-S) [80] has been developed. The message-orientation feature of MQTT makes it a content agnostic protocol. MQTT focuses exclusively on the delivery of messages.

MQTT has been widely adopted in IoT environments, e.g. [64], [81]. The low overhead, easy implementation, and support from all leading vendors make MQTT a convenient option for IoT. MQTT offers to decouple with respect to time, space, and synchronization. The classical MQTT deployment follows a hub-spoke model in which nodes are linked directly to sensors and actuators.

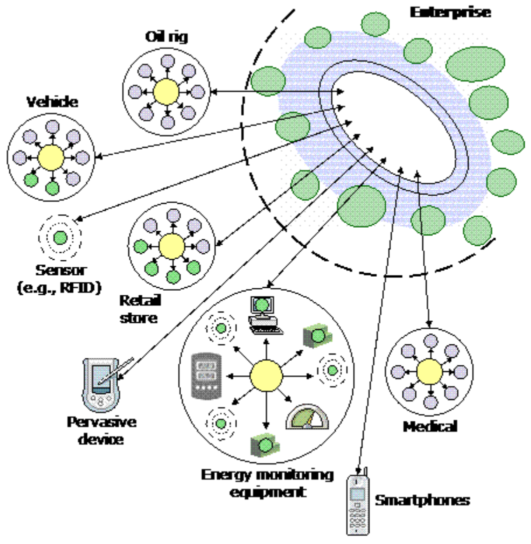


Figure 3.4 MQTT IBM [82]

3.7.3 Resource-centric IoT Communication

As the name suggests, the Resource-Centric communication pattern focuses exclusively on resources.

The Constrained Application Protocol (CoAP) [83] is a machine-to-machine (M2M) resource base protocol designed for constrained scenarios. CoAP follows the REST approach exchanging representations of resources. Like in HTTP, the request methods in CoAP are GET, POST, PUT, DELETE.

Even though the CoAP specification uses UDP protocol as the default transport option, TCP can be used as well. Figure 3.5 presents the logical layers of CoAP. The figure shows the request/response layer that interacts with applications through methods and codes and the messaging layer that works with UDP [83]. The CoAP package size varies from the minimum 4 bytes (simple GET requests) to a maximum of 1024 bytes (Figure 3.6).

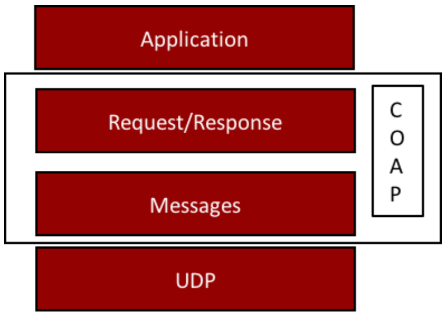
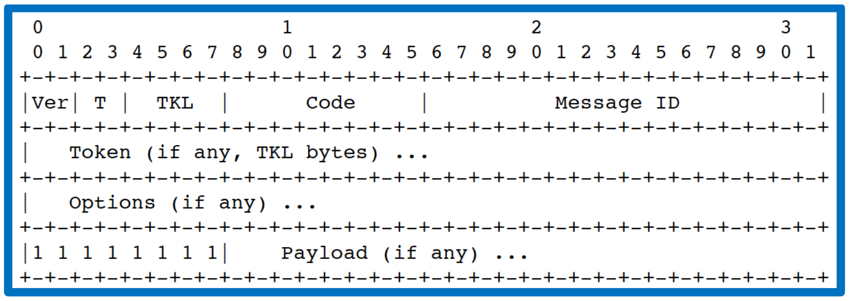


Figure 3.5 Abstract layering of CoAP [83]

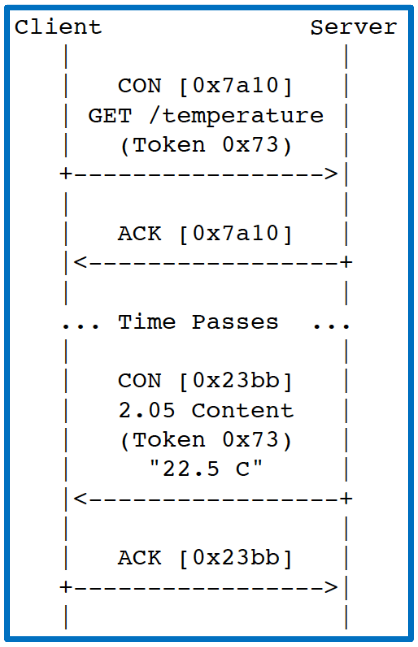


```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.6 CoAP Message Format [83]

CoAP specifies four types of messages: Acknowledgment, Reset, Confirmable (CON), and Non-Confirmable (NON). An acknowledgment message is sent by the recipient confirming the reception of the message. A reset message is an empty confirmable message that the recipient sends to indicate that something was missing in the message. A confirmable message requires an acknowledgment message from the recipient; it ensures reliability when sending CoAP messages. A non-confirmable message does not require an acknowledgment message from the recipient.

Figure 3.7 shows an example of the interaction between a client and a server. The client makes a confirmable CoAP GET request asking for the current temperature value. As soon as the server receives the request, it responds with an acknowledgment message. When the server has the current temperature value, it sends a confirmable message with the temperature value. Finally, the client confirms the reception.



```
Client              Server
   |                  |
   |   CON [0x7a10]   |
   | GET /temperature |
   |   (Token 0x73)   |
   +----------------->|
   |                  |
   |   ACK [0x7a10]   |
   |<-----------------+
   |                  |
   ... Time Passes  ...
   |                  |
   |   CON [0x23bb]   |
   |   2.05 Content   |
   |   (Token 0x73)   |
   |     "22.5 C"     |
   |<-----------------+
   |                  |
   |   ACK [0x23bb]   |
   +----------------->|
   |                  |
```

Figure 3.7 Example of a CoAP message with separated acknowledgment of receipt [83]

When the server has the requested value immediately available, the value can be sent directly in the acknowledgment message. This kind of message is called a piggybacked response. Figure 3.8 shows an example of a piggybacked message. The client asks the current temperature value in a confirmable CoAP GET message. When the server receives the request, the value is immediately attached to the acknowledgment message.

CoAP does not have a formal implementation of the publish/subscribe pattern. However, the publish/subscribe pattern is addressable by making resources observable [84] (subject/observer design pattern). When a resource is observed, the observer will receive the updates of any change in the resource. Figure 3.9 shows an example of the CoAP Observe pattern. The observer registers into a subject. When the state of the subject changes, the observer receives the updated value.

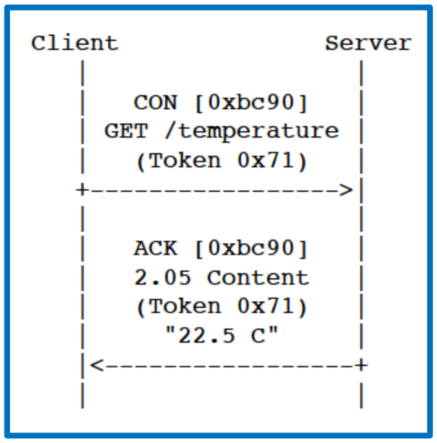


Figure 3.8 Example of a CoAP piggybacked message [83]

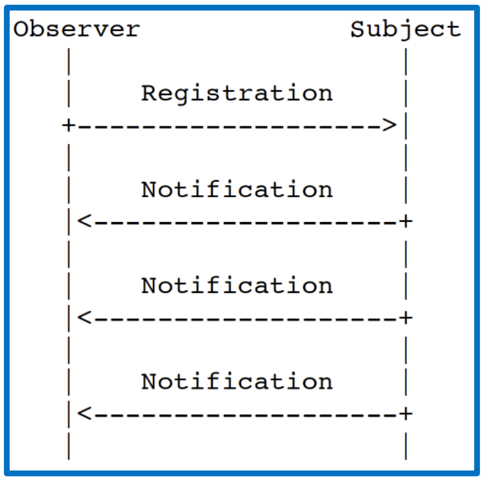


Figure 3.9 Observer design pattern [84]

According to Hemdi [85], the REST pattern of CoAP enforces a resource-oriented view over IoT components. CoAP represents a unified manner to abstract and engage IoT components. Many studies in IoT have used CoAP protocol. For example, Kovatsch [86] presents a special implementation of CoAP for Contiki operating system, and Ludovici [87] presents an implementation of CoAP for wireless sensor networks.

The REST orientation of CoAP makes it a good option to represent constrained elements with URI's and operate them through CRUD interfaces.

3.8 Summary

With the rapid increase of the number of devices connected to the Internet, new challenges have emerged in research. Table 3-3 presents the topics that have been revised to provide an efficient solution to handle the issues that emerge from implementing virtualization in the IoT network.

Table 3-3 Summary of the Literature Review

Topic	Papers	Results
Internet of Things (IoT)	[9]	The IoT architecture states three basic layers: perception, network, and application.
IoT applications	[2], [3], [4], [5], [6], [88], [89], [90]	Nowadays, IoT plays an important role in different areas such as healthcare, transportation, smart homes, public services, industry, and other processes.
Fog Computing	[16], [18], [33], [36]	Fog computing decreases the latency to engage IoT components; however, many challenges emerge due to the features of IoT networks, such as a large set of nodes and heterogeneity of components.
Multi-tenancy & IoT	[19], [31], [38]	Multi-tenancy has been identified by many studies as a challenge in IoT; however current systems do not see it as a concern because they do not support multi-tenant access.
Virtualization & IoT	[46], [47], [48], [50], [33], [49]	Virtualization of IoT components in the IoT network is considered by this work as a solution to handle multi-tenancy in IoT. Some studies have covered IoT virtualization in the Cloud and Fog networks. This work proposes virtualization directly onto IoT devices to create Virtual Systems for each tenant (this topic is covered in the Architecture chapter).

Blockchain and IoT	[51], [52], [53], [57], [60]	Permission-based blockchains are proposed by this work as an alternative to storing the configuration of the Virtual Resources and Virtual Systems for each tenant.
SOA & REST	[63], [64], [65], [66], [68], [69], [70]	Many works show that the flexibility of SOA is a good feature for the Application and Network layers of the IoT architecture. However, that flexibility may introduce complexity when defining services for the variety of IoT components. REST represents a light way to define services for IoT components. REST CRUD mapping makes the engagement of IoT components simple.
Communication Patterns in IoT	[72], [73], [74], [77], [78], [64], [81], [83], [85], [86], [87]	This review has found three focuses on IoT communication, Data-Oriented, Message-Oriented, and Resource-Oriented. Resource Oriented communication is the one that fits better to the characteristics of components of constrained networks. Each component can be abstracted as a resource.

CHAPTER 4

ARCHITECTURE

The Literature Review presented in Chapter 3 evidenced the need to manipulate the Internet of Things (IoT) network in a multi-tenant manner. The review discussed Virtualization on the Cloud and a Fog layer. This chapter presents an architecture based on Virtual Resources to deal with the characteristics of IoT networks and support multi-tenancy. Additionally, this chapter introduces blockchain technology to handle the provisioning of Virtual Resources directly on IoT devices.

4.1 Definition of Virtual Resources

Virtual Resources are digital artifacts, which can be defined using different technologies. This work defines Virtual Resources as RESTful micro services. Virtual Resources communicate via CoAP protocol exposing the methods GET, POST, PUT, and DELETE. When processing requests, Virtual Resources can either engage other Things or use their internal state.

REST architecture was selected over SOA to model Virtual Systems. REST fits better for constrained networks as each IoT component can be represented as an individual resource using a URI [91]. Also, the REST approach is useful to manipulate Things through mapped CRUD (create, read, update and delete) operations. Figure 4.1 shows a representation of Virtual Resources in the IoT Things layer. The figure shows three levels of abstraction. The first level is the IoT Components, which perform the sensing or actuating actions. The second level is the Atomic Layer, which groups Atomic Virtual Resources. The third level is the View Layer, which groups View Virtual Resources. Atomic Virtual Resources and View Virtual Resources are explained in the following sections.

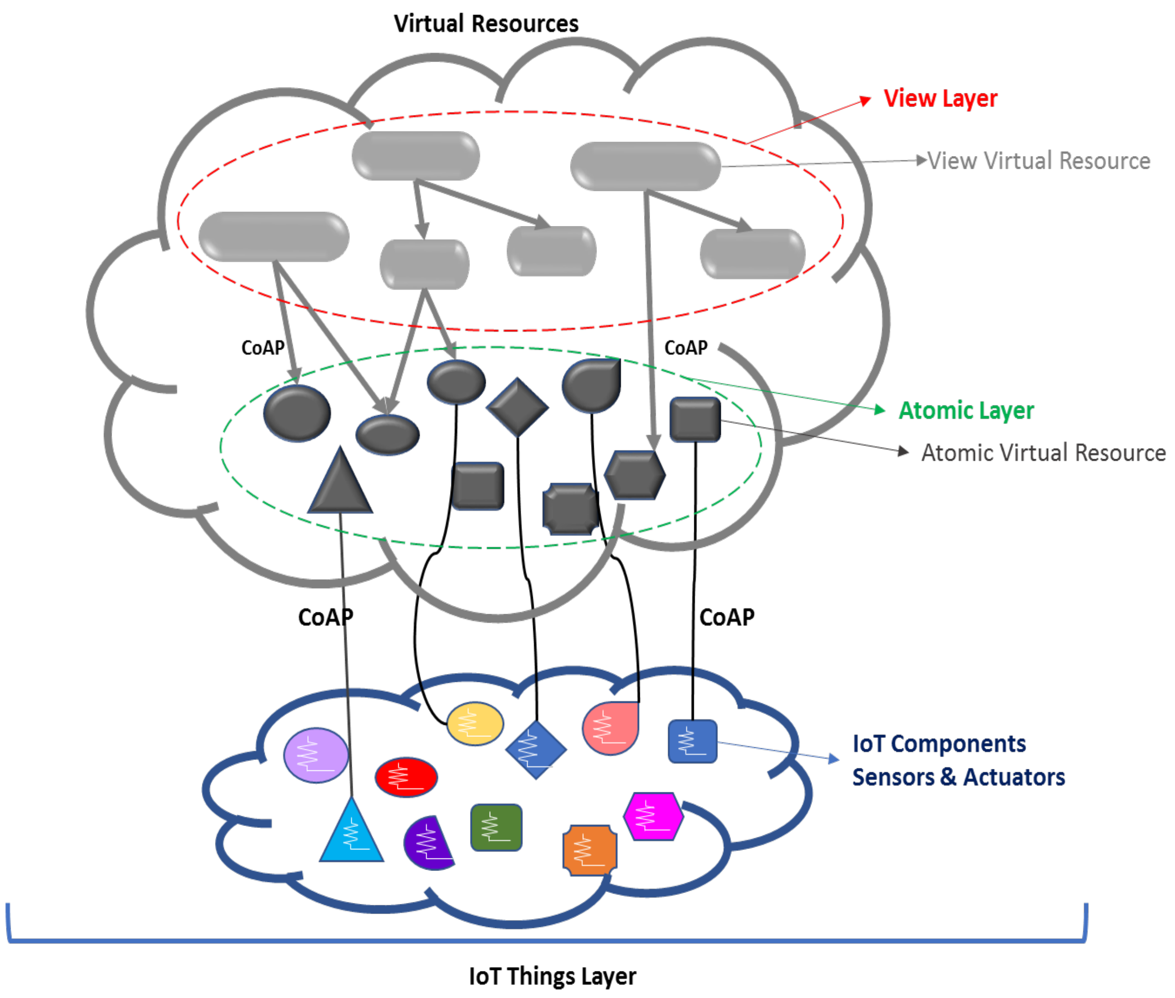


Figure 4.1 Representation of Virtual Resources in the IoT Things layer.

4.1.1 Atomic Virtual Resources

Atomic Virtual Resources are individual abstractions of elements of the Things layer. Figure 4.2 shows the relation between an element in the Things layer and the Atomic Virtual Resource. Each element of the Things layer is linked to its Atomic representation, building a one-to-one relation. This relation is aimed to control the access to physical devices. Atomic Virtual Resources can communicate between them before responding to any request.

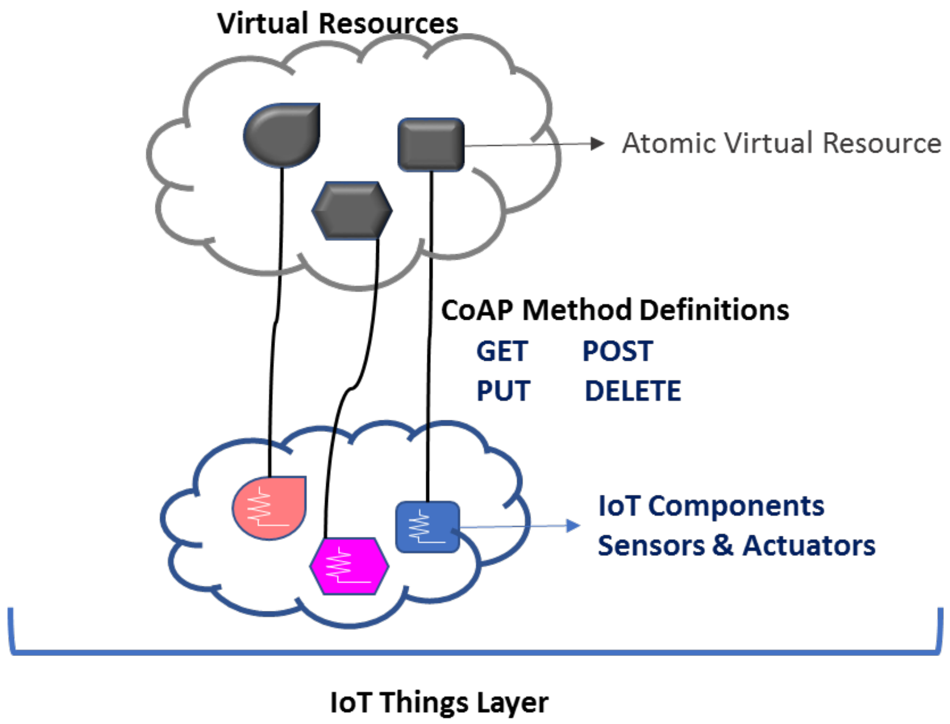


Figure 4.2 Representation of Atomic Virtual Resources. Each physical component has an Atomic Virtual representation.

4.1.2 View Virtual Resources

This work defines View Virtual Resources as abstractions of one or more Atomic Virtual Resources. View Virtual Resources work as processing units that expose interfaces and present data for tenants.

View Virtual Resources are built on top of Atomic Virtual Resources. This model results in a one-to-many relation. Figure 4.3 shows the model of a View Virtual Resource. In this figure, a View Virtual Resource engages three Atomic Virtual Resources through the CoAP interfaces "/.well-known/core" [92] and "/state" (these interfaces are explained in Chapter 5).

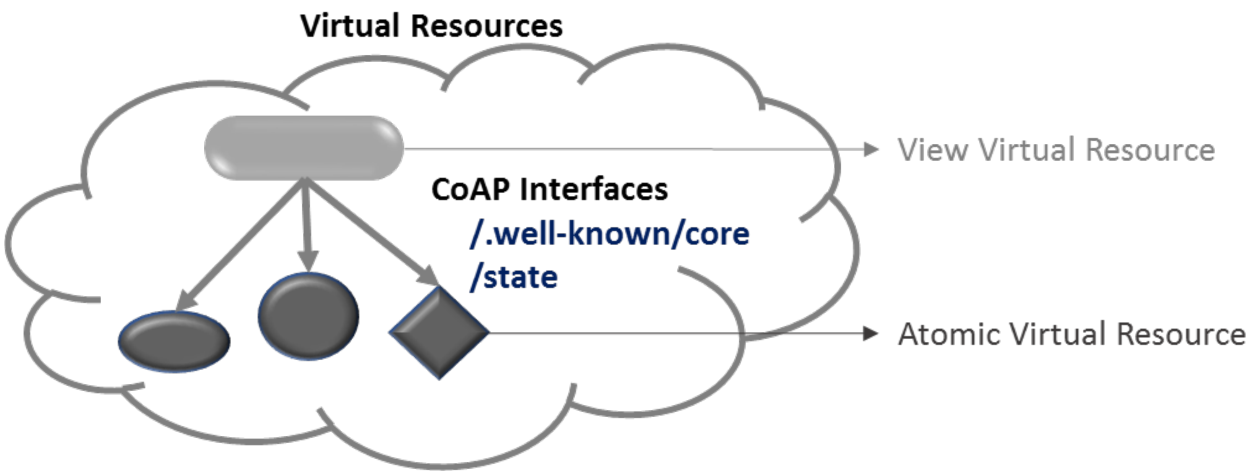


Figure 4.3 Representation of View Virtual Resources. A View Virtual Resource engages two Atomic Virtual Resources.

When building the data representation for each tenant, View Virtual Resources require to integrate not only Atomic Virtual Resources but also other View Virtual Resources as well. This integration results in a Virtual System for each tenant.

4.1.3 Virtual Systems

Virtual Systems integrate many Virtual Resources (Atomic or View). Figure 4.4 shows the integration of different Virtual Resources into a Virtual System. This integration results in a hierarchical composition with Atomic Virtual Resources as root elements and View Virtual Resources as child nodes. The figure shows a View Virtual Resource on the top, which integrates two other View Virtual Resources and one Atomic Virtual Resource. The View Virtual Resources in the second level engage other Atomic Virtual Resources.

Users can build their own customized and dedicated Virtual Systems on top of other Virtual Resources. Virtual Systems defined on top of existing virtualizations make possible to create N virtual IoT systems.

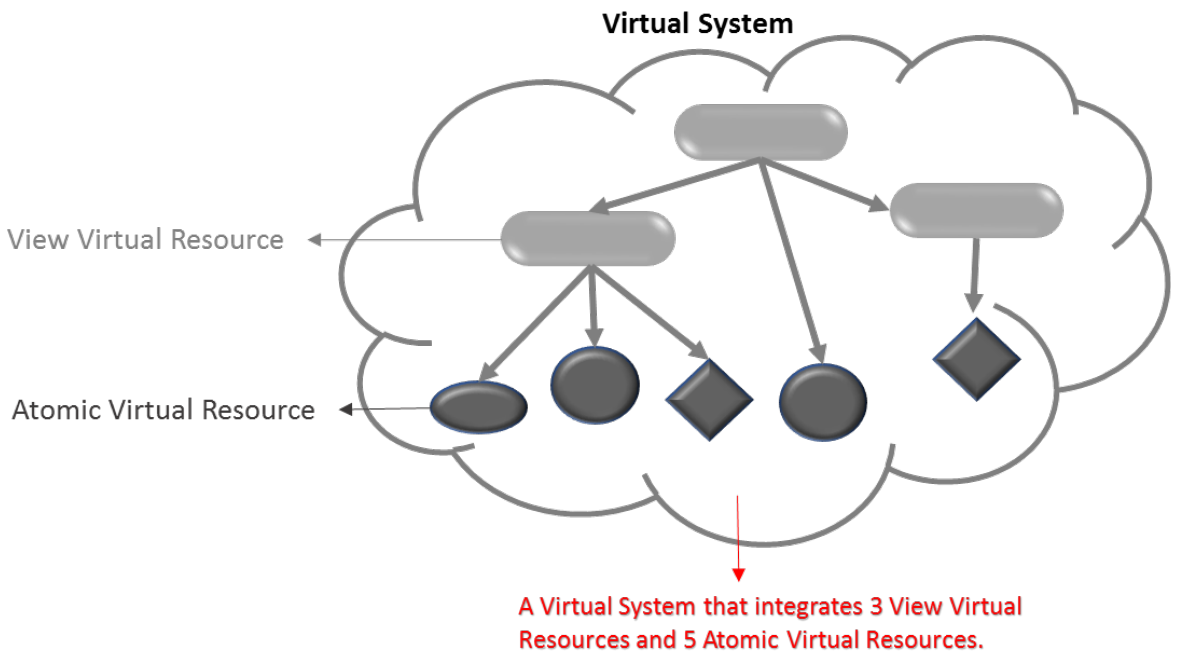

Virtual System
View Virtual Resource
Atomic Virtual Resource
A Virtual System that integrates 3 View Virtual Resources and 5 Atomic Virtual Resources.

Figure 4.4 Representation of a Virtual System.

4.2 Provisioning of Virtual Resources

Provisioning of Virtual Resources in the IoT network represents a challenge regarding security. Virtual Resources must be correctly distributed over the large set of heterogeneous platforms that can work on a single IoT network.

Permission-based blockchain protocols handle the provisioning of Virtual Resources in a secure manner. The configuration of Virtual Resources (code or metadata) is stored in the blockchain. Only the configurations that come from trusted nodes remain in the blockchain in the form of blocks.

Having the configurations of Virtual Resources available in a blockchain makes it possible to create Virtual Resources at runtime. This functionality allows each tenant to self-define and self-deploy their Virtual Resources reading the blocks of the blockchain. Only, the users registered in the blockchain can read or write blocks.

A virtual IoT system is the deployment of multiple Virtual Resources pulled from the blockchain working together.

4.3 Multi-tenant access to IoT Components

Moving computation to the edge of the IoT network enables direct and dynamic manipulations of physical devices. In this scenario, multiple tenants demand access to IoT devices to configure them at the same time. This multi-tenancy issue is handled by giving the tenant the capacity to deploy their own Virtual Systems.

Tenants must be listed in the registry of the permission-based blockchain to be able to read or write in the blocks. Registered tenants can read existing Virtual Resources and write and deploy new ones.

This architecture guarantees security in the deployment of Virtual Resources by encrypting the data stored in the blocks. The encryption process is necessary to ensure that only registered tenants with the correct key can decrypt the configurations stored in the blocks.

4.4 Summary

This chapter proposes an architecture to achieve the virtualization challenges presented in Chapter 2:

- Virtual resources are defined as light RESTful micro services communicating via CoAP protocol. Virtual resources expose CoAP methods: GET, POST, PUT, DELETE.
- The provisioning of Virtual Resources on IoT devices is handled by permission-based blockchain technology. Blockchain hosts the configuration of Virtual Resources and allows tenants to build their unique Virtual Systems.
- Multi-tenant access to IoT components is granted by defining IoT Virtual Systems for each tenant. Permission-based blockchain handles the configuration of Virtual Resources in encrypted blocks, which can be decrypted exclusively by the tenant with the correct key.

CHAPTER 5

IMPLEMENTATION

The architecture to support virtualization and multi-tenancy on Internet of Things (IoT) networks is presented in Chapter 4. Each user (tenant) can configure their own hierarchical Virtual System to access the IoT network. Virtual Resources are modeled as light RESTful micro services. Provisioning of Virtual Resources on IoT devices is handled by permission-based blockchain.

This chapter explores the necessary technologies to implement the previous architecture.

5.1 Definition of Virtual Resources

Virtual Resources are written in Go language. Routines in Go language are a lightweight option to run concurrent Virtual Resources [93]. Each routine implements a micro service following the CoAP protocol. Go language allows to compile Virtual Resources into different IoT platforms.

Virtual Resources expose two interfaces:

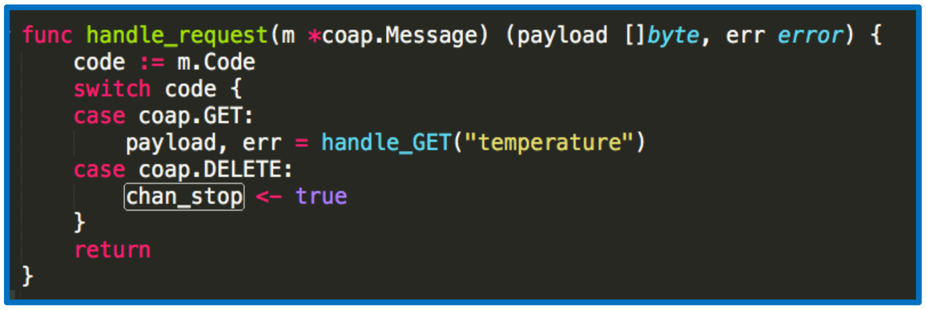
- "/.well-known/core" [92], which identifies the available services or resources of the current Virtual Resource
- "/state", which gets the current state of the Virtual Resource

Virtual Resources are manipulated through CRUD (create, read, update, delete) operations mapped by the CoAP methods GET, POST, PUT, and DELETE. Figure 5.1 shows the Go language syntax to expose CoAP methods.

Channels are another feature of Go language, which is useful to send and receive values [94]. Virtual Resources communicate between them sending CoAP messages through channels.

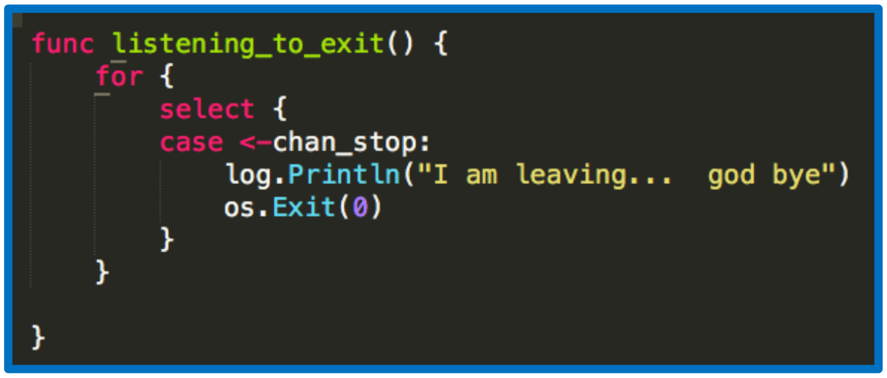
A channel can be heard by N Virtual Resources simultaneously. Figure 5.2 shows an implementation of a channel in Go language. Virtual Resources listen to the channel. As soon as a value is received through the channel, all Virtual Resources that are listening to that channel change their state.

Virtual Resources format the responses using JSON notation. Figure 5.3 shows an example of a JSON response from a Virtual Resource. The name/value pairing of JSON is an efficient option to transmit data between IoT devices.



```
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
}
```

Figure 5.1 Example of a function programmed in Go language that exposes CoAP methods



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

}
```

Figure 5.2 Example of a function programmed in Go language that implements a channel.

5.2 Provisioning of Virtual Resources

This work implements two permission-based blockchains to store the configuration of Virtual Resources. First, Multichain, a private blockchain cluster, which validates transactions using the round-robin process. Round-robin process requires that each party put a signature in every block they attempt to create [95]. After adding a block, parties must stop creating new blocks for a certain time. The Multichain cluster is hosted in a Fog node.

Second, IBM Bluemix, a blockchain as a service (BaaS), which validates transactions using a consensus process. This consensus process establishes a quorum of at least fifty percent of nodes plus one to approve transactions [96].

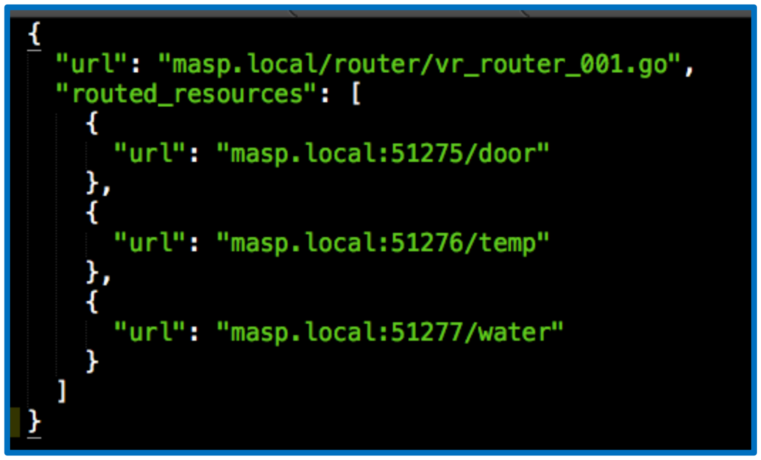
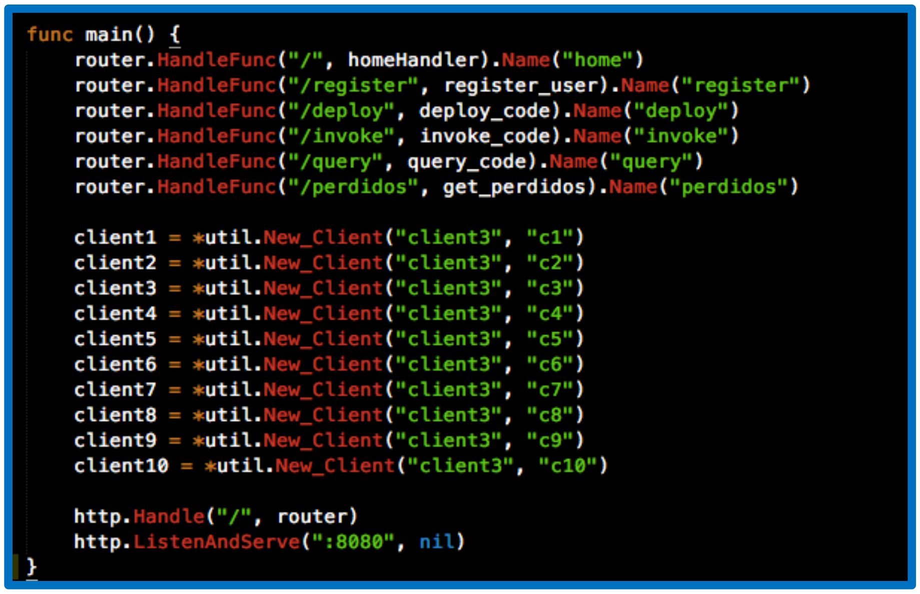

```
{
  "url": "masp.local/router/vr_router_001.go",
  "routed_resources": [
    {
      "url": "masp.local:51275/door"
    },
    {
      "url": "masp.local:51276/temp"
    },
    {
      "url": "masp.local:51277/water"
    }
  ]
}
```

Figure 5.3 Example of a JSON file

An API is required to face the blockchain nodes and ask permission to write or read blocks. In this work, all the code is written in Go language. The methods to interact with the blockchain follow the standard naming of the Hyperledger project [97].

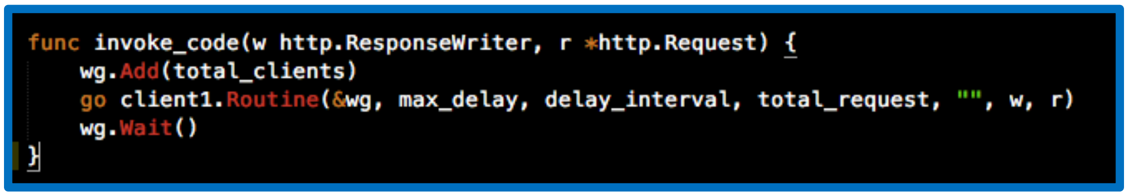
Figure 5.4 shows the declarations of the routes to interact with the blockchain and the declaration of ten Virtual Resources. Virtual resources execute the method *invoke_code* to write data in the blockchain. Figure 5.5 and Figure 5.6 present examples of one and ten Virtual Resources requesting for write operations to the blockchain respectively.

The data written in the blockchain is encrypted using the Advanced Encryption Standard (AES) [98] with a key of 16 bytes.



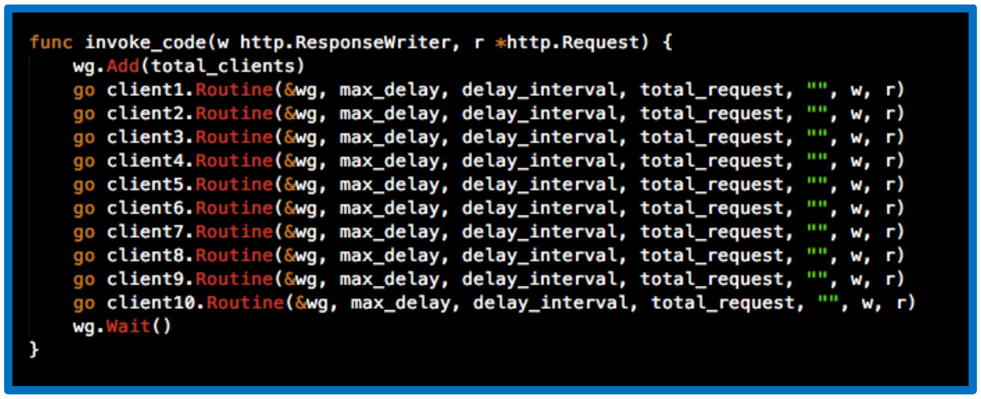
```go
func main() {
    router.HandleFunc("/", homeHandler).Name("home")
    router.HandleFunc("/register", register_user).Name("register")
    router.HandleFunc("/deploy", deploy_code).Name("deploy")
    router.HandleFunc("/invoke", invoke_code).Name("invoke")
    router.HandleFunc("/query", query_code).Name("query")
    router.HandleFunc("/perdidos", get_perdidos).Name("perdidos")

    client1 = *util.New_Client("client3", "c1")
    client2 = *util.New_Client("client3", "c2")
    client3 = *util.New_Client("client3", "c3")
    client4 = *util.New_Client("client3", "c4")
    client5 = *util.New_Client("client3", "c5")
    client6 = *util.New_Client("client3", "c6")
    client7 = *util.New_Client("client3", "c7")
    client8 = *util.New_Client("client3", "c8")
    client9 = *util.New_Client("client3", "c9")
    client10 = *util.New_Client("client3", "c10")

    http.Handle("/", router)
    http.ListenAndServe(":8080", nil)
}
```

Figure 5.4 Example of the declaration of the routes to interact with the blockchain and ten Virtual Resources. Code programmed in Go language.



```go
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)
    wg.Wait()
}
```

Figure 5.5 Example of one Virtual Resource requesting access to the blockchain. Code programmed in Go language.



```go
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()
}
```

Figure 5.6 Example of ten Virtual Resource requesting access to the blockchain. Code programmed in Go language.

5.3 Multi-tenant access to IoT components

The registration of tenants is handled by the permission-based blockchain. Both blockchain technologies, Multichain and IBM Bluemix, manage a registry of tenants to allow correct read and write operations. Additionally, the methods to approve transactions (round robin process for Multichain and consensus process for Bluemix) balance the access to the blocks.

5.4 Summary

This chapter shows the most suitable technologies to implement the architecture proposed in Chapter 4.

Go is the base programming language. Go has useful features to implement the definition of Virtual Resources presented by this work, such as routines and channels. Running concurrent routines in Go language allows to deploy multiple Virtual Resources on runtime. Channels distribute the state of Virtual Resources to multiple listeners.

Multichain and IBM Bluemix are two permission-based blockchain technologies that have the ideal characteristics to handle the provisioning of Virtual Resources and multi-tenant access. The configuration of Virtual Resources is stored in the form of encrypted blocks, which guarantees security and a correct deployment through a decryption key.

CHAPTER 6

EXPERIMENTS AND EVALUATIONS

This chapter describes the experiments designed to evaluate the performance of the definition of Virtual Resources presented by this research. In addition to the evaluation of Virtual Resources, experiments on blockchain working in the IoT space are detailed.

Seven experiments have been designed. Table 6-1 presents a description of each section of the experiments:

Table 6-1 Description of Experiments

Experiment Sections	Description
Virtual Resources deployed on Raspberry PI	Test the performance of Virtual Resources deployed on Raspberry PI computers.
Virtual Resources deployed on Edison Arduino Boards	Test the performance of Virtual Resources deployed on Edison compute modules.
Provisioning and multi-tenant access to Virtual Resources using blockchain	Test the performance of blockchain used to manage the provisioning of Virtual Resources and multi-tenant access.

6.1 Virtual Resources deployed on Raspberry PI

6.1.1 Experiment 1:

This experiment evaluates the performance of Virtual Resources deployed on Raspberry PI computers. Figure 6.1 shows the setup of this experiment. The setup includes a Raspberry Pi computer, a Mac OS computer, and a Linux server. The three elements of this experiment are connected to the university's Wi-Fi network. The Raspberry Pi hosts a state-full View Virtual Resource. This View Virtual Resource connects to a NoSQL database to get the current state of the resources linked to it. The View Virtual Resource responses with a CoAP Acknowledgment message to all requests. The responses include the state of the resources in the payload. The Raspberry Pi computer represents the IoT Things layer.

The Mac OS computer hosts a Client routine programmed in Go language to test the View Virtual Resource. The Client sends 1000 CoAP GET requests to the View Virtual Resource. Before sending a request, the Client must wait for the previous acknowledgment message.

The Linux server hosts Elasticsearch, which is a RESTful search engine for analytics [99]. Although Elasticsearch is not formally considered a NoSQL database, it can be contemplated as a document-oriented database due to its schema-less document storage. The "query time" and "index time" features of Elasticsearch support the heterogeneity nature of the data generated in the IoT Cloud. The Linux database server represents the Fog layer.

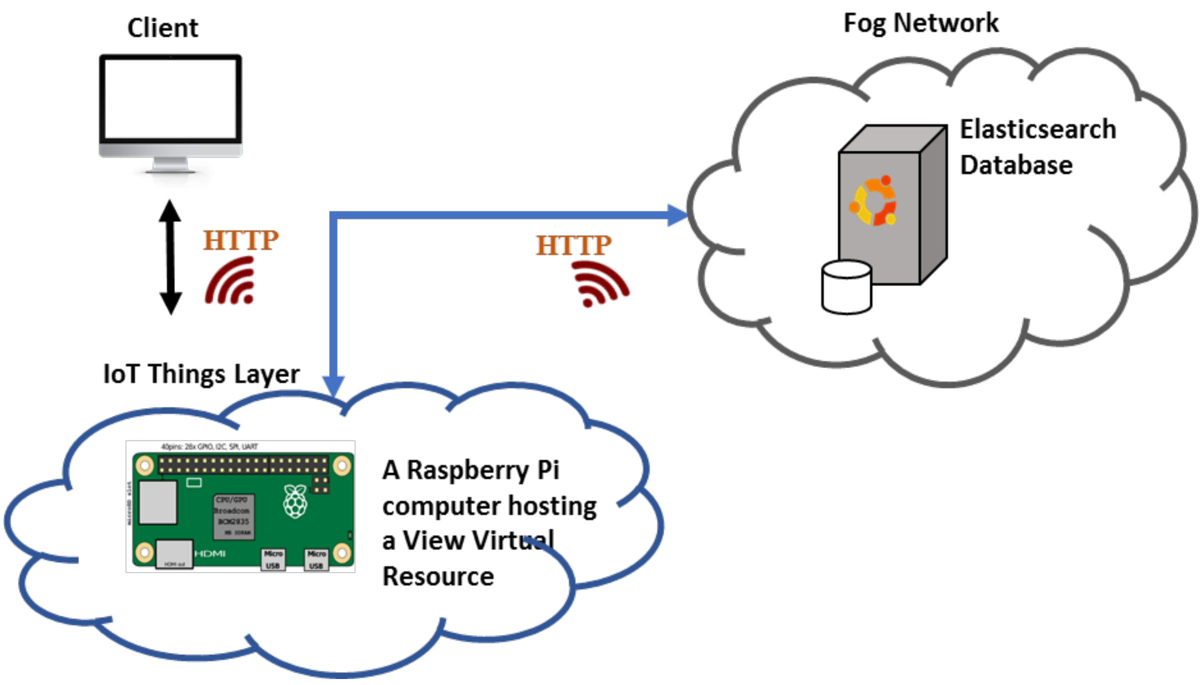


Figure 6.1 Setup of Experiment 1. An IoT Things network formed by a Raspberry Pi. A Fog network formed by a database server. A Client testing the Virtual Resource.

Table 6-2 details the characteristics of the hardware used in this experiment.

Table 6-2 Specification of the hardware used in Experiment 1

Hardware	Details
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 evaluates three processes of View Virtual Resources:

- the Discovery-of-Services process, which is exposed in a REST interface ("/.well-known/core" - Core Link Format - RFC6690)
- the current-state process, which is exposed in a REST interface ("/state")
- the communication with the database

A. Discovery of Services

The first part of Experiment 1 evaluates the Discovery-of-Services process. Virtual Resources expose a REST interface ("/.well-known/core"), which handles this process. Figure 6.2 shows the results of this evaluation. The y-axis of the graph represents the round-trip time in milliseconds measured from the Client side. The x-axis of the graph represents the Client's requests (1-1000). This experiment introduces delays of 0, 50, and 100 ms in sending the requests from the Client to the Virtual Resource. Overall, the round-trip time of the requests is between

0.2 ms and 2.2 ms. The delay intervals directly affect the response times. The higher the interval, the higher the response time. The round-trip time of the 0ms-delay series is between 0.2 ms and 0.6 ms, and the average round-trip time is 0.38 ms. These results do not show any pick. The round-trip time of the 50ms-delay series is between 0.6 ms and 01.2 ms, and the average round-trip time is 0.79 ms. These results evidence some picks, which can be attributed to the noise of the network, memory allocation or background processes of the device. Additionally, as the difference between the results of each series is minuscule, any noise could have affected the response times. The round-trip time of the 100ms-delay series is between 0.6 ms and 1.2 ms, and the average round-trip time is 0.89 ms. These results do not show any picks.

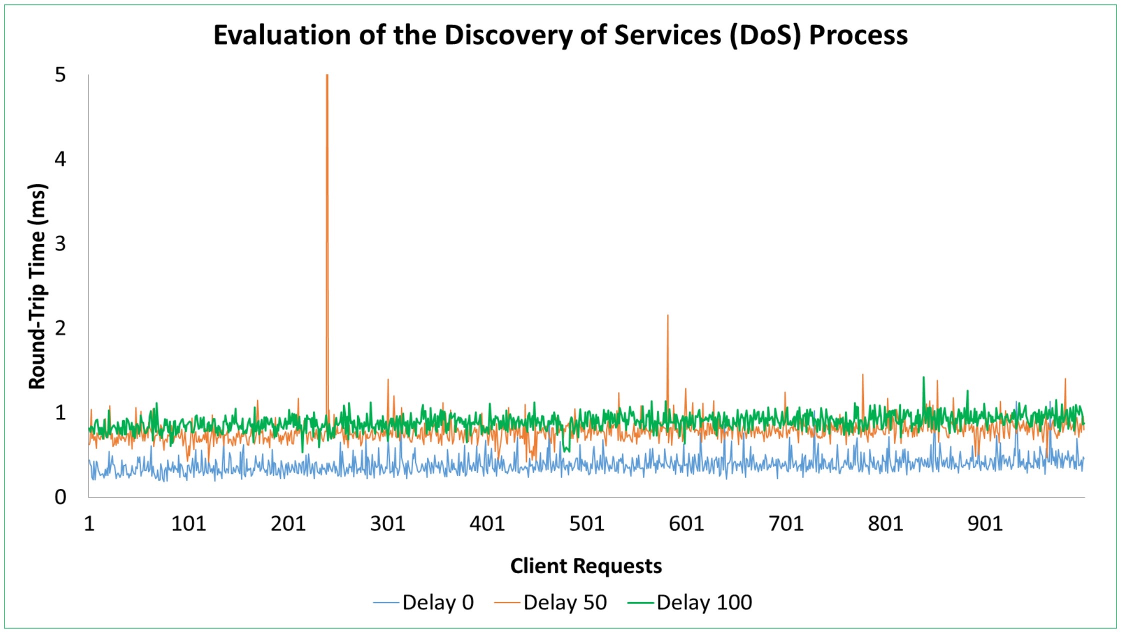


Figure 6.2 Results of the evaluation of the Discovery-of-Services process ("/.well-known/core") of the Virtual Resources.

B. Current State

The second part of Experiment 1 evaluates the "current-state" process. Virtual Resources expose a REST interface ("/state") to handle this process. Figure 6.3 presents the results of this evaluation. The y-axis of the graph represents the round-trip time in milliseconds measured from the Client side. The x-axis of the graph represents the Client's requests (1-1000). This experiment introduces delays of 0, 50, and 100 ms in sending the requests. Overall, the round-trip time from the client to the Virtual Resource is between 5.4 ms and 9.8 ms.

Like the first part of Experiment 1, the delay intervals affect the communication performance. The round-trip time increases when a higher delay is introduced. The round-trip time of the 0ms-delay series is between 5.2 ms and 7.2 ms, and the average round-trip time is 6.96 ms. The round-trip time of the 50ms-delay series is between 6.2 ms and 8.4 ms, and the average round-trip time is 7.67 ms. The round-trip time of the 100ms-delay series is between 6.9 ms and 9 ms, and the average round-trip time is 8.29 ms.

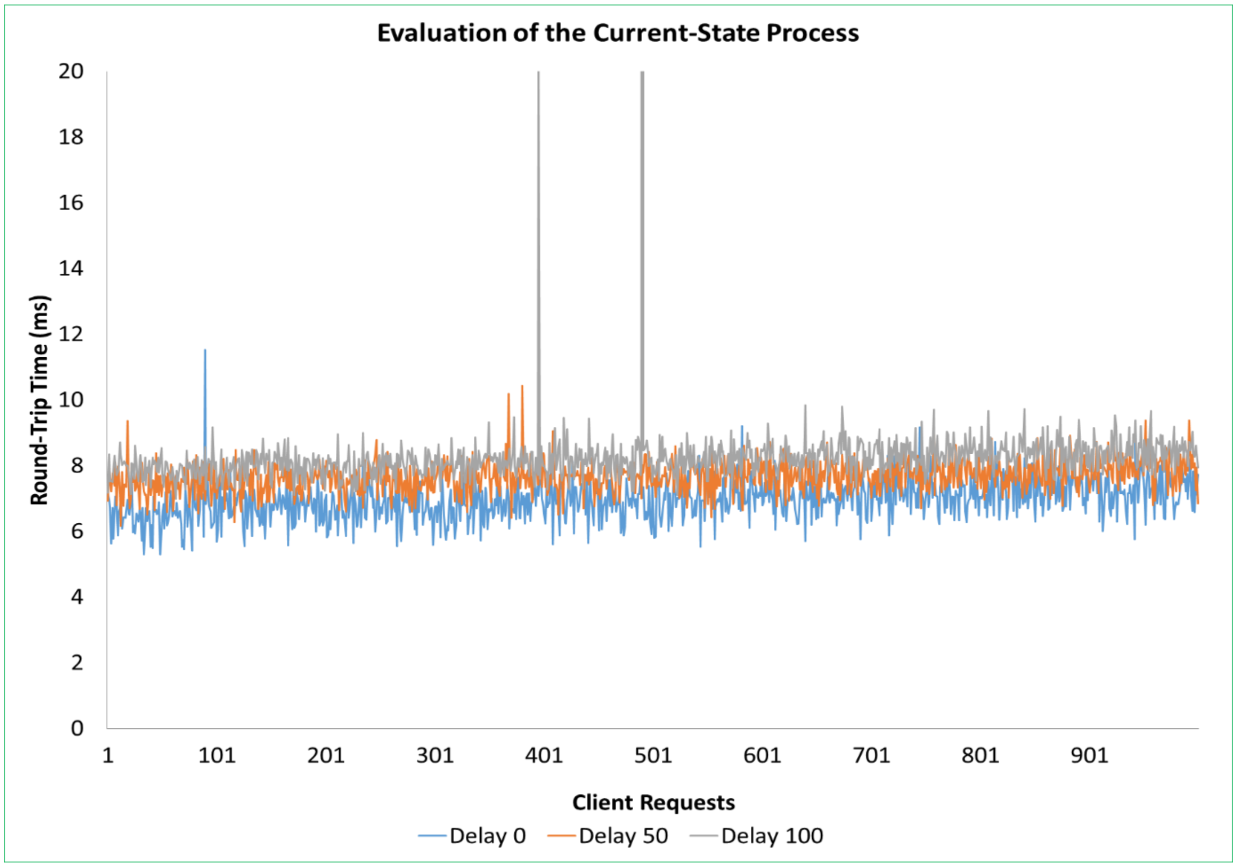


Figure 6.3 Results of the evaluation of the current-state process ("/state") of the Virtual Resources.

C. Database communication

The third part of Experiment 1 evaluates the communication performance between the View Virtual Resource and the database. This experiment introduces delays of 0, 50ms, and 100 ms in the requests from the View Virtual Resource to the database. Figure 6.4 presents the results of this evaluation. The y-axis of the graph represents the round-trip time in milliseconds measured from the Virtual Resource side. The x-axis of the graph represents the Virtual Resource's requests (1-1000). This graph shows that the delay intervals do not affect the response times of the database server. Overall, the results are between 4.8 ms and 7 ms.

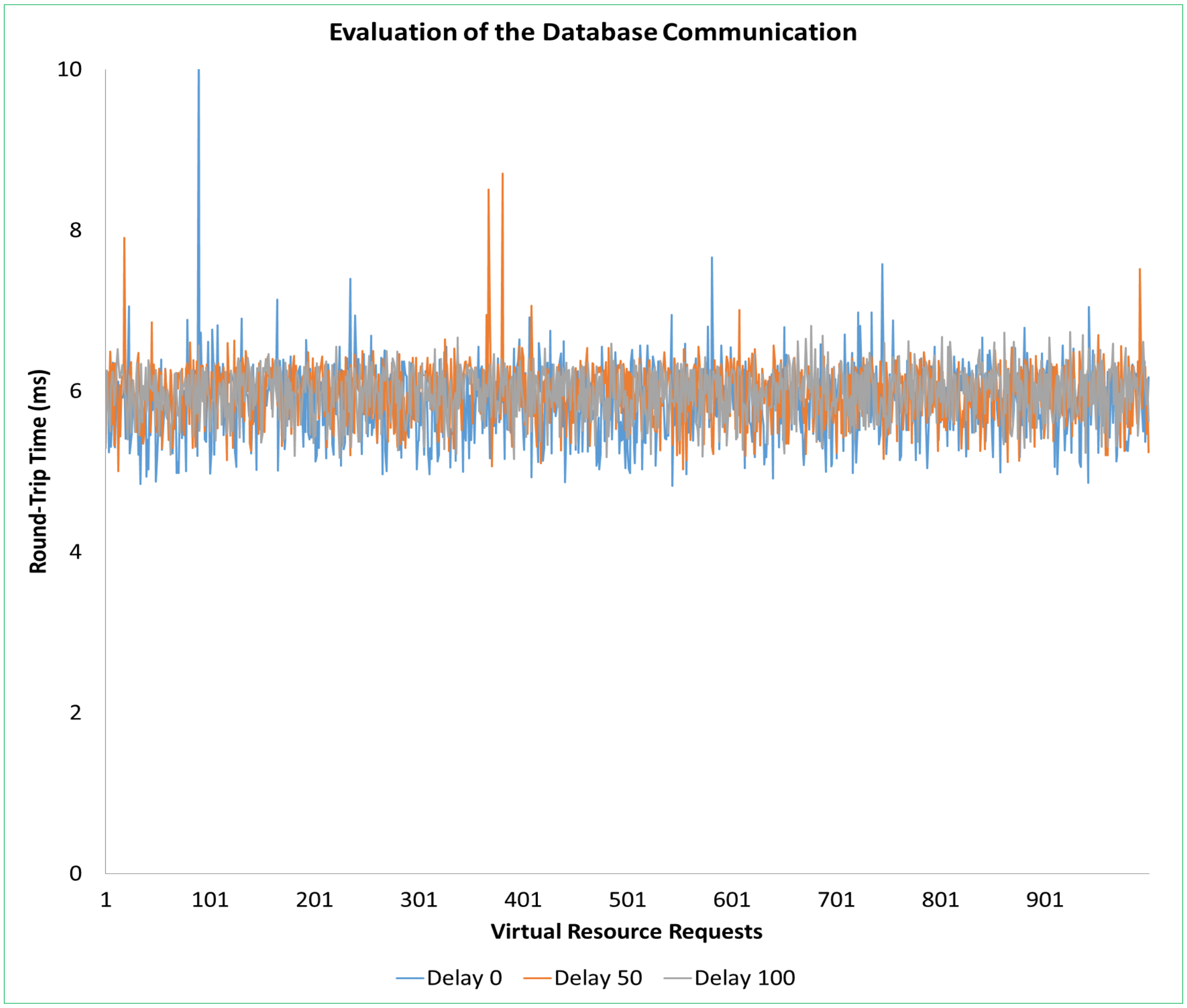


Figure 6.4 Results of the Communication between the Virtual Resource and the Database.

6.1.2 Summary:

This experiment shows that Virtual Resources defined as RESTful micro services and programmed in Go language, perform well when responding to requests from a more powerful computer. Additionally, this experiment shows that a Raspberry Pi computer can manage the connection to a database hosted in a Fog node.

6.2 Virtual Resources deployed on Edison Arduino boards

6.2.1 Experiment 2:

This experiment evaluates the communication performance between two Virtual Resources deployed on Edison Arduino boards. Figure 6.5 shows the setup of this experiment. The setup includes an IoT Things layer of two Edison Arduino boards connected to the university's Wi-Fi network. Both boards communicate via CoAP protocol. The first Edison board hosts an Atomic Virtual Resource. This Virtual Resource responds with a CoAP acknowledgment message, which includes its current state. The second Edison board hosts a state-less View Virtual Resource that sends 1000 CoAP POST requests to the Atomic Virtual Resource. This Virtual Resource waits for the acknowledgment of the current request before sending the next one.

Table 6-3 explains the characteristics of the Edison modules.

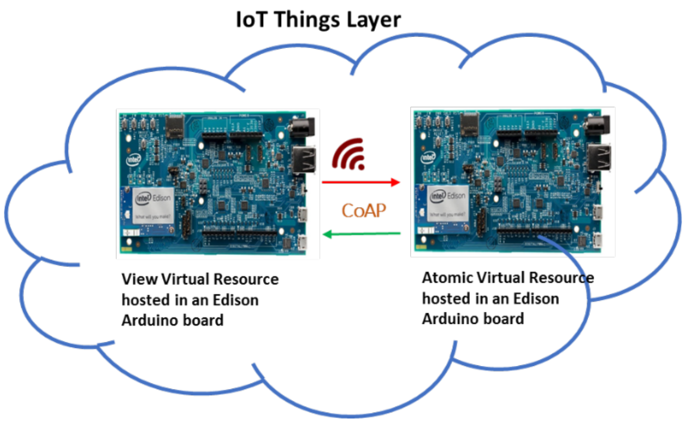

IoT Things Layer
CoAP
View Virtual Resource hosted in an Edison Arduino board
Atomic Virtual Resource hosted in an Edison Arduino board

Figure 6.5 Setup of experiment 2. An IoT Things layer formed by two Edison Arduino boards connected to the Wi-Fi network and communicating via CoAP protocol. Each Edison board hosts a Virtual Resource.

Table 6-3 Specification of Edison Arduino Board

Hardware Edison Arduino Board	Details
Operating System	Linux Yocto
CPU	500 MHz dual-core, dual threaded Intel Atom and a 100 MHz 32-bit Intel Quark microcontroller
RAM	4 GB

The payload of the requests is encrypted to make the data transmission secure. Due to the limited resources of the Edison boards, a synchronous AES encryption method with a key of 16 bytes is performed. The payload size of the requests is 8 bytes + 16 bytes of AES encryption.

Figures 6.6 to 6.9 show the results of this experiment. In the result graphs, the y-axis represents the round-trip time in milliseconds measured from the side of the View Virtual Resource. The x-axis represents the View Virtual Resource's requests (1-1000). This experiment introduces delays of 0, 50, 100, 150, 200, 250 and 300 ms in issuing each request. These delays are represented as the series of each graph. The delays help to evaluate the performance of the Virtual Resources under different loads.

This experiment shows that the seven-delay series have similar round-trip time values. The delay times do not have a significant impact on the performance of the Edison board as there is just one single Virtual Resource sending CoAP POST requests. In general, the round-trip time is between 4 and 11 milliseconds. Some picks appear in the graphs, but they can be attributed to the noise of the network, memory allocation or background processes of the Edison board.

The average round-trip time of each delay series is explained as follows. The 0ms-delay series has an average round-trip time of 6.45 ms. The 50ms-delay series has an average round-trip time of 6.46 ms. The 100ms-delay series has an average round-trip time of 8.02 ms. The 150ms-delay series has an average round-trip time of 6.46 ms. The 200ms-delay series has an average round-trip time of 7.93 ms. The 250m-delay series has an average round-trip time of 6.52 ms. Finally, the 300ms-delay series has an average round-trip time of 6.55 ms. These results indicate that the definition of Virtual Resources presented by this research has a good communication performance when deployed on Edison Arduino boards.

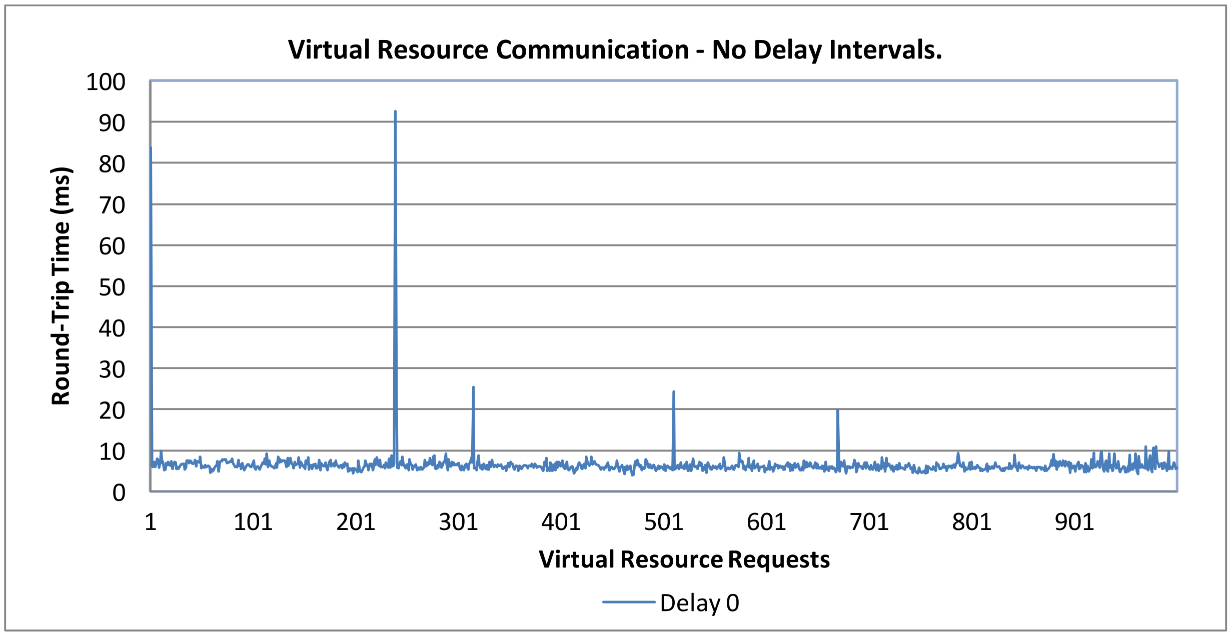


Figure 6.6 Results of Experiment 2. One View Virtual Resource is sending 1000 synchronous requests to one Atomic Virtual Resource. The payload is 8 bytes + 16 bytes AES synchronous encryption. No delay.

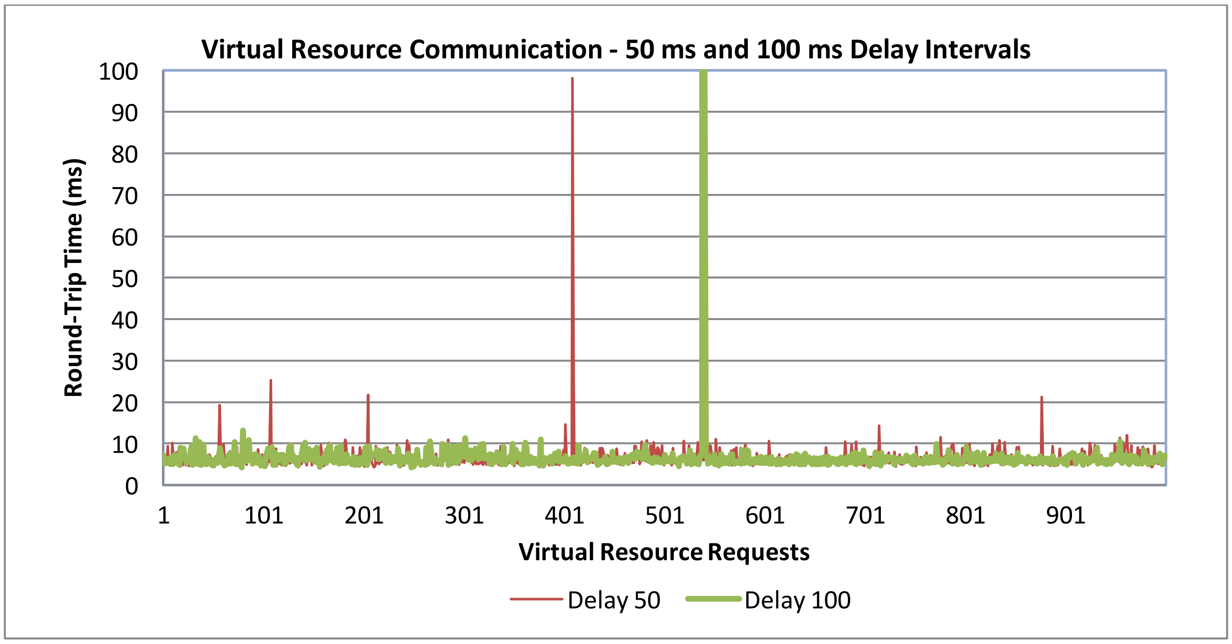


Figure 6.7 Results of Experiment 2. One View Virtual Resource is sending 1000 synchronous requests to one Atomic Virtual Resource. The payload is 8 bytes + 16 bytes AES synchronous encryption. 50ms and 100ms delays.

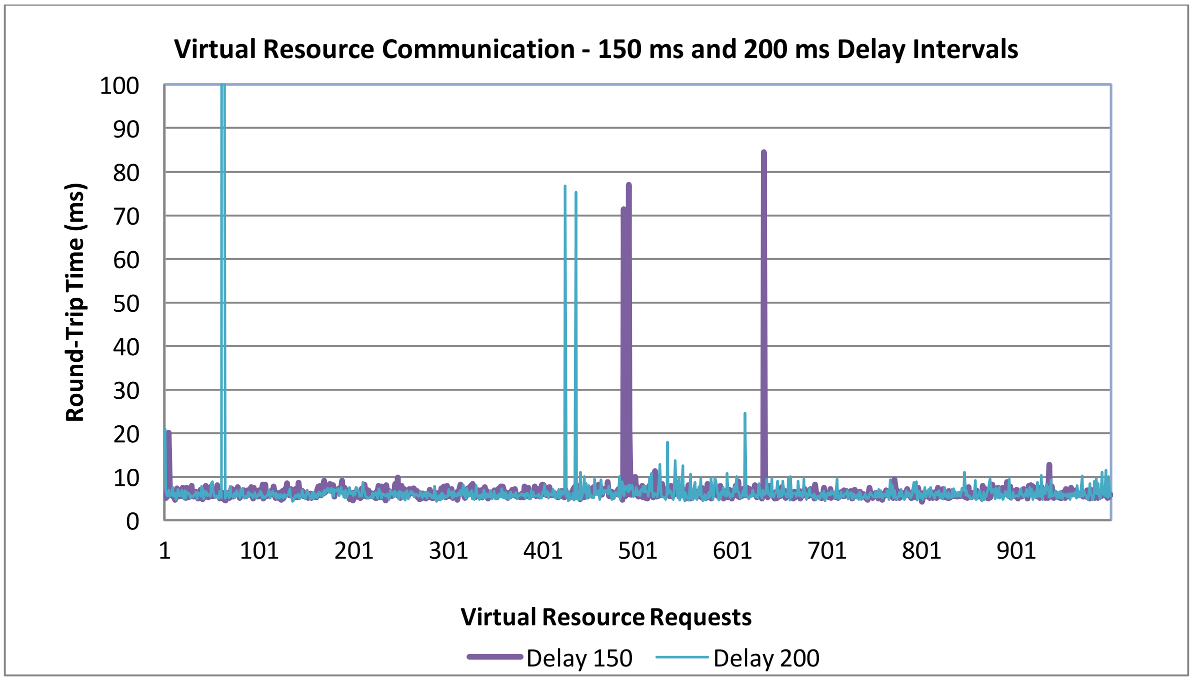


Figure 6.8 Results of Experiment 2. One View Virtual Resource is sending 1000 synchronous requests to one Atomic Virtual Resource. The payload is 8 bytes + 16 bytes AES synchronous encryption. 150ms and 200ms delays.

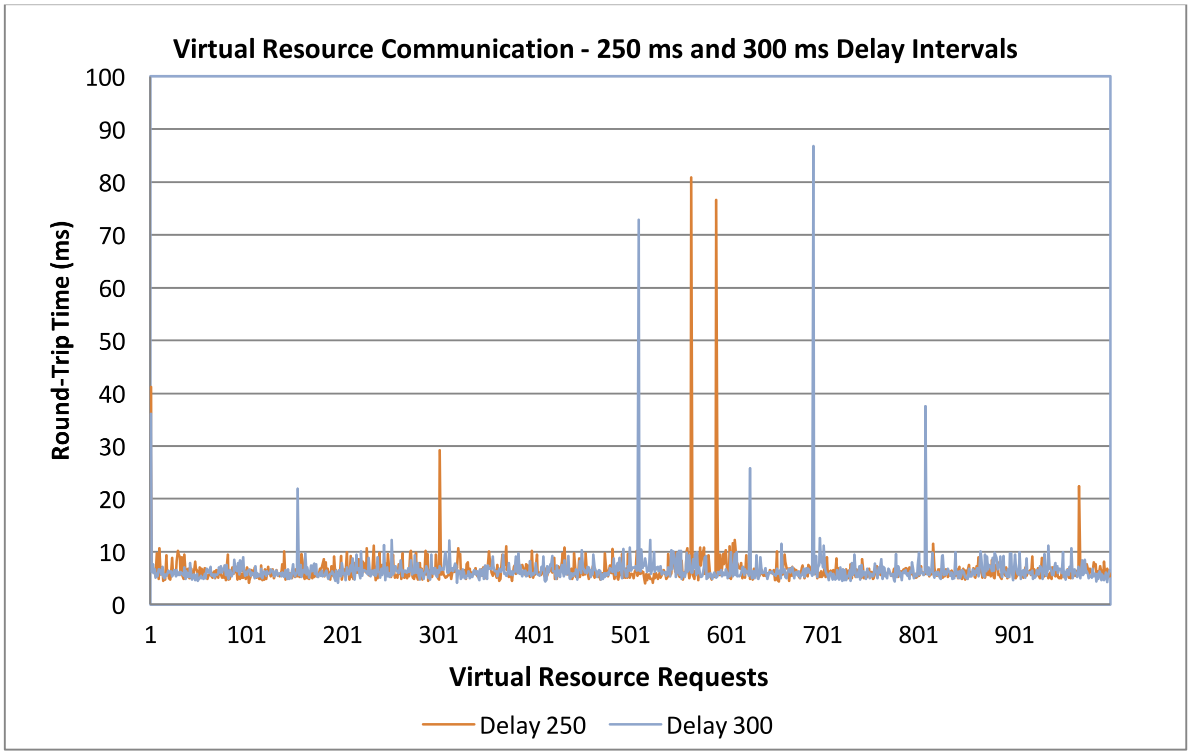


Figure 6.9 Results of Experiment 2. One View Virtual Resource is sending 1000 synchronous requests to one Atomic Virtual Resource. The payload is 8 bytes + 16 bytes AES synchronous encryption. 250ms and 300ms delays.

6.2.2 Experiment 3:

This experiment evaluates the performance of Virtual Resources responding to multiple requests. Virtual Resources are deployed on Edison Arduino boards. Like in Experiment 2 (Figure 6.5), the setup of this new experiment involves two Edison boards, which represent the IoT Things layer. The two boards are connected to the university's Wi-Fi network. Both boards communicate via CoAP protocol. The characteristics of the Edison boards are explained in Table 6-3 from Experiment 2.

The first Edison board hosts an Atomic Virtual Resource that responds with a CoAP acknowledgment message to all requests. This message includes the current state of the Virtual Resource. The second Edison board hosts ten View Virtual Resources. These Virtual Resources send 100 asynchronous CoAP POST requests each to the Atomic Virtual Resource. These Virtual Resources also wait for the acknowledgment of the current request before sending the next one. A synchronous AES encryption method with a key of 16 bytes is performed. The payload size of the requests is 8 bytes + 16 bytes of AES encryption.

This experiment introduces delays of 0, 50, 100, 150, 200, 250, and 300 ms in sending the requests. The delays help to evaluate the performance of the Virtual Resources under different stress levels. The results are split in seven graphs, one for each delay. The y-axis represents the round-trip time in milliseconds, measured from the side of the View Virtual Resource. The x-axis represents each View Virtual Resource's request (1-100).

Figures 6.10 to 6.16 show the performance of the Virtual Resource. The results of the seven graphs show high fluctuations on the response times. These fluctuations are obtained due to the ten clients sending requests concurrently and asynchronously. The ten View Virtual Resources are called as asynchronous routines in Go language.

In this experiment, we observed that sending concurrent CoAP POST requests affects the response times. Compared to Experiment 1, in which there was only one View Virtual Resource sending requests, the round-trip time sending concurrent requests has increased around three times.

The average round-trip time value of each delay series is explained as follows. The series with no delay has an average round-trip time of 21.12 ms. The 50ms-delay series has an average round-trip time of 20.28 ms. The 100ms-delay series has an average round-trip time of 23.44 ms. The 150ms-delay series has an average round-trip time of 23.41 ms. The 200ms-delay series has an average round-trip time of 15.39 ms. The 250ms-delay series has an average round-trip time of 19.91. Finally, the 300ms-delay series has an average round-trip time of 19.21 ms.

Even though the results show some high picks, such as the ones in the 0-delay graph that rise to more than 320 ms, the response times represent a good performance considering that the Edison Arduino Boards are IoT devices with limited computational capabilities.

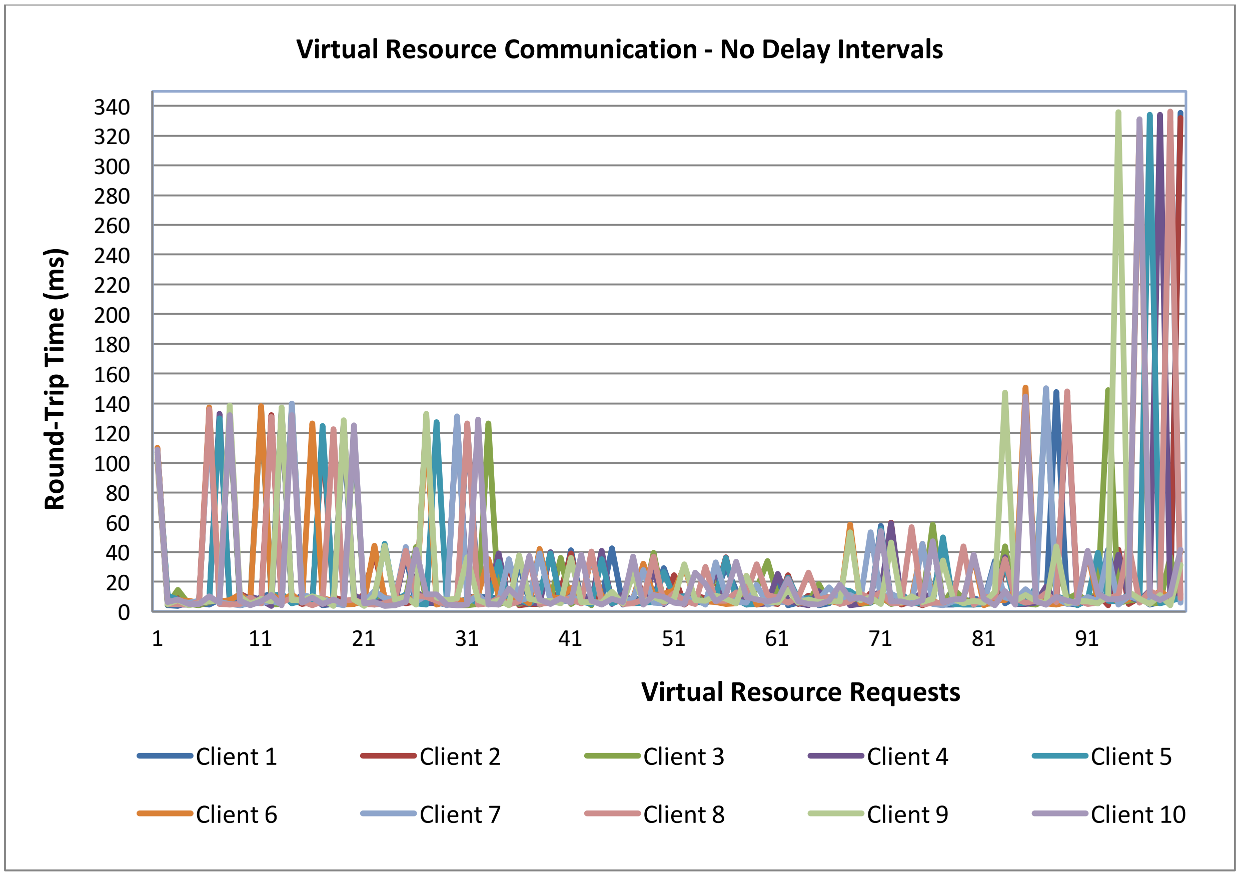


Figure 6.10 Results of Experiment 3. Ten View Virtual Resources are sending 100 CoAP POST requests to one Atomic Virtual Resource. The payload is 8 bytes + 16 bytes AES synchronous encryption. No delay time interval.

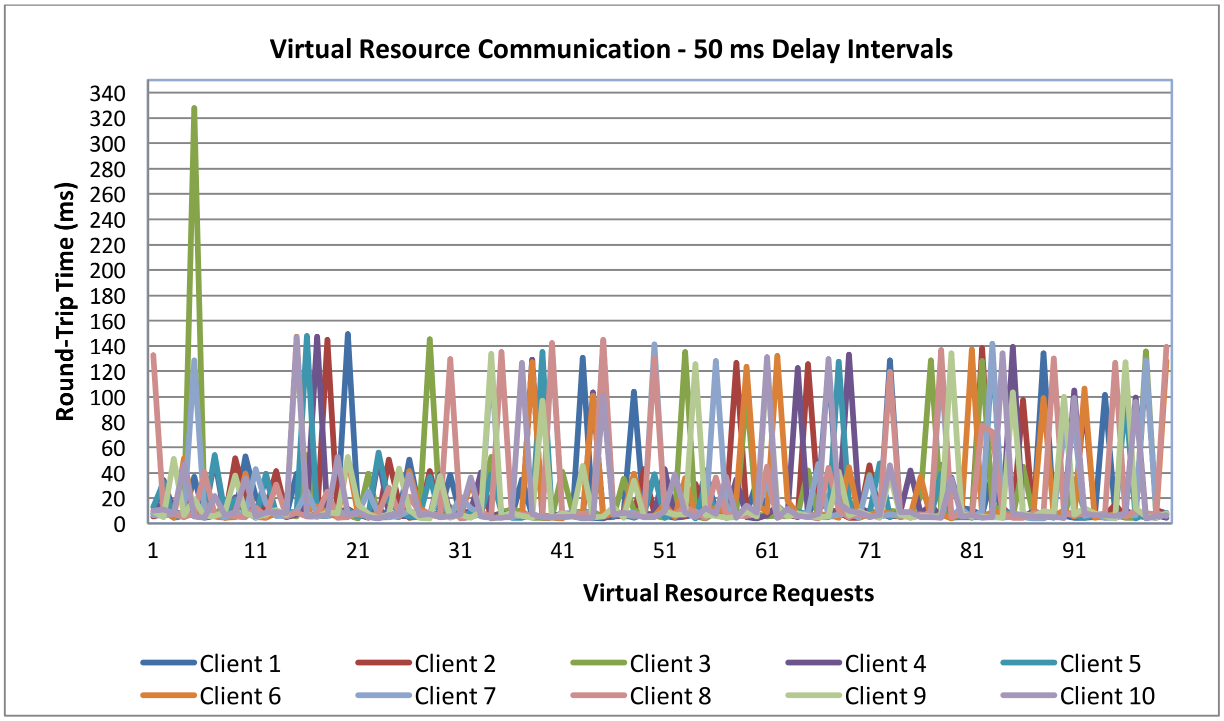


Figure 6.11 Results of Experiment 3. Ten View Virtual Resources are sending 100 CoAP POST requests to one Atomic Virtual Resource. The payload is 8 bytes + 16 bytes AES synchronous encryption. 50 ms delay.

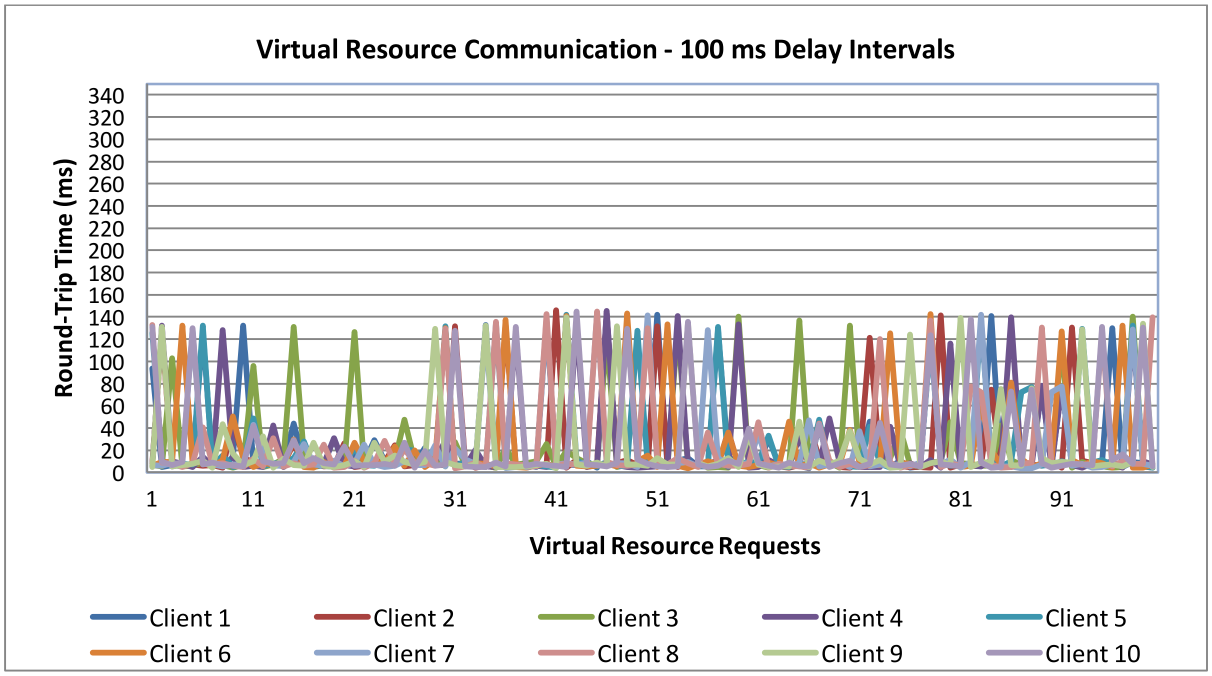


Figure 6.12 Results of Experiment 3. Ten View Virtual Resources are sending 100 CoAP POST requests to one Atomic Virtual Resource. The payload is 8 bytes + 16 bytes AES synchronous encryption. 100 ms delay.

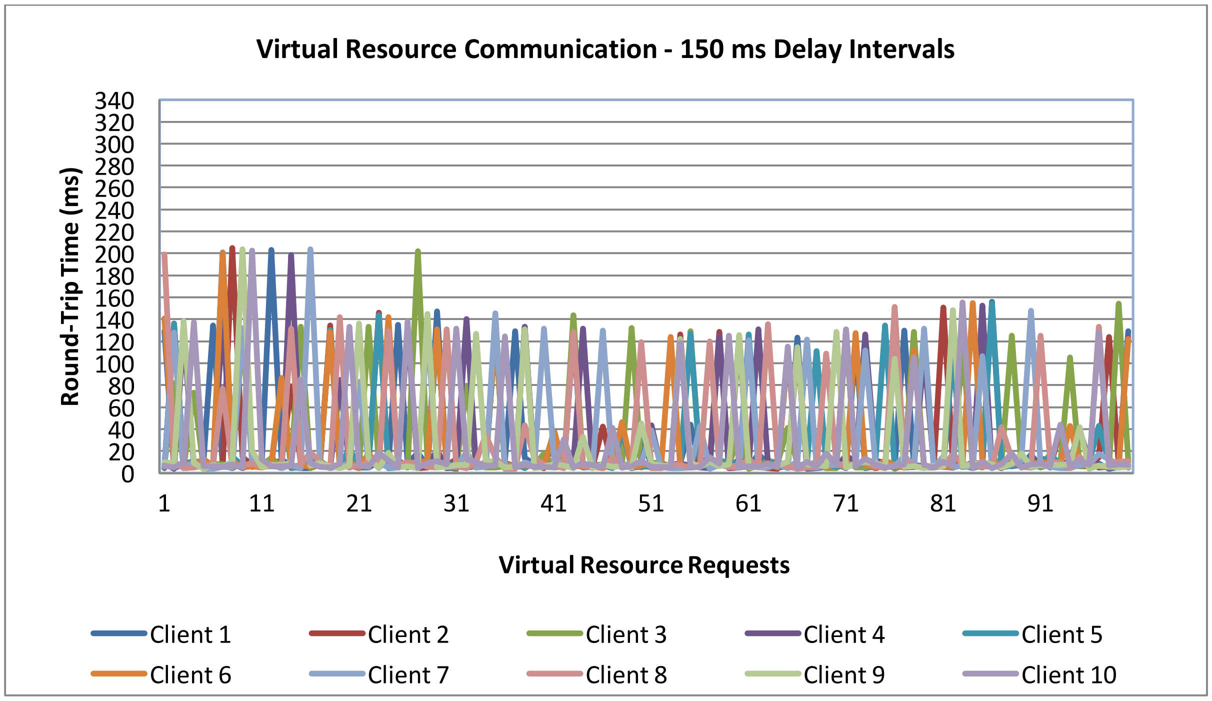


Figure 6.13 Results of Experiment 3. Ten View Virtual Resources are sending 100 CoAP POST requests to one Atomic Virtual Resource. The payload is 8 bytes + 16 bytes AES synchronous encryption. 150 ms delay.

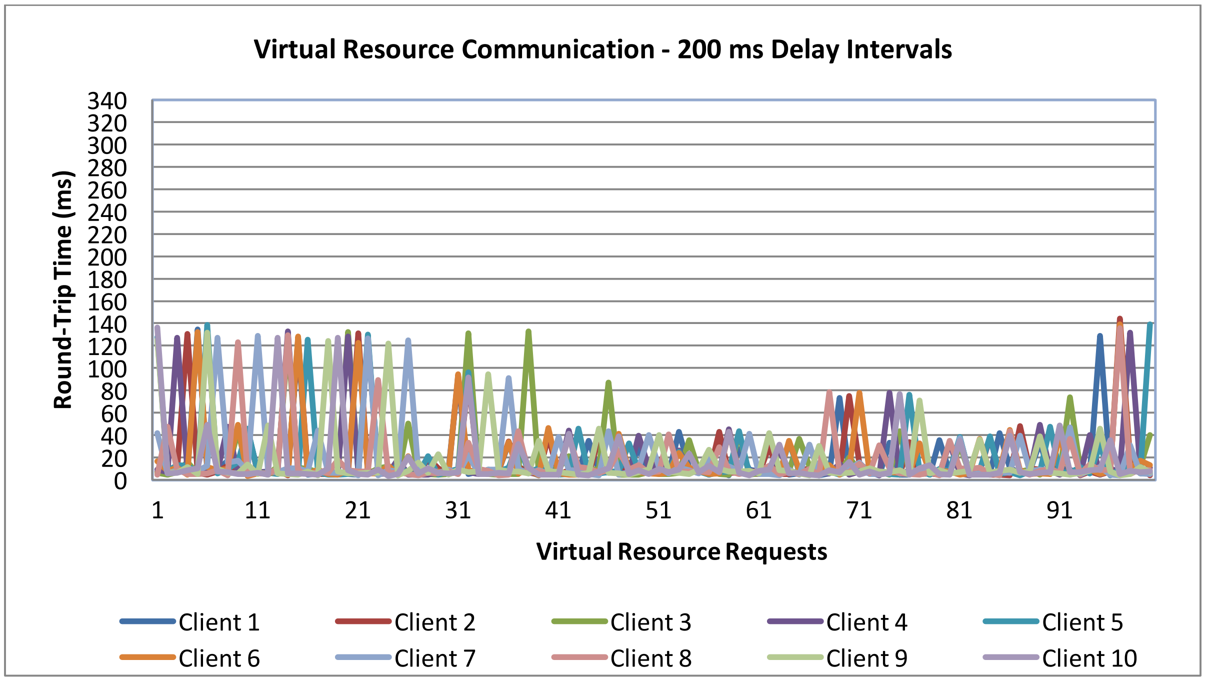


Figure 6.14 Results of Experiment 3. Ten View Virtual Resources are sending 100 CoAP POST requests to one Atomic Virtual Resource. The payload is 8 bytes + 16 bytes AES synchronous encryption. 200 ms delay.

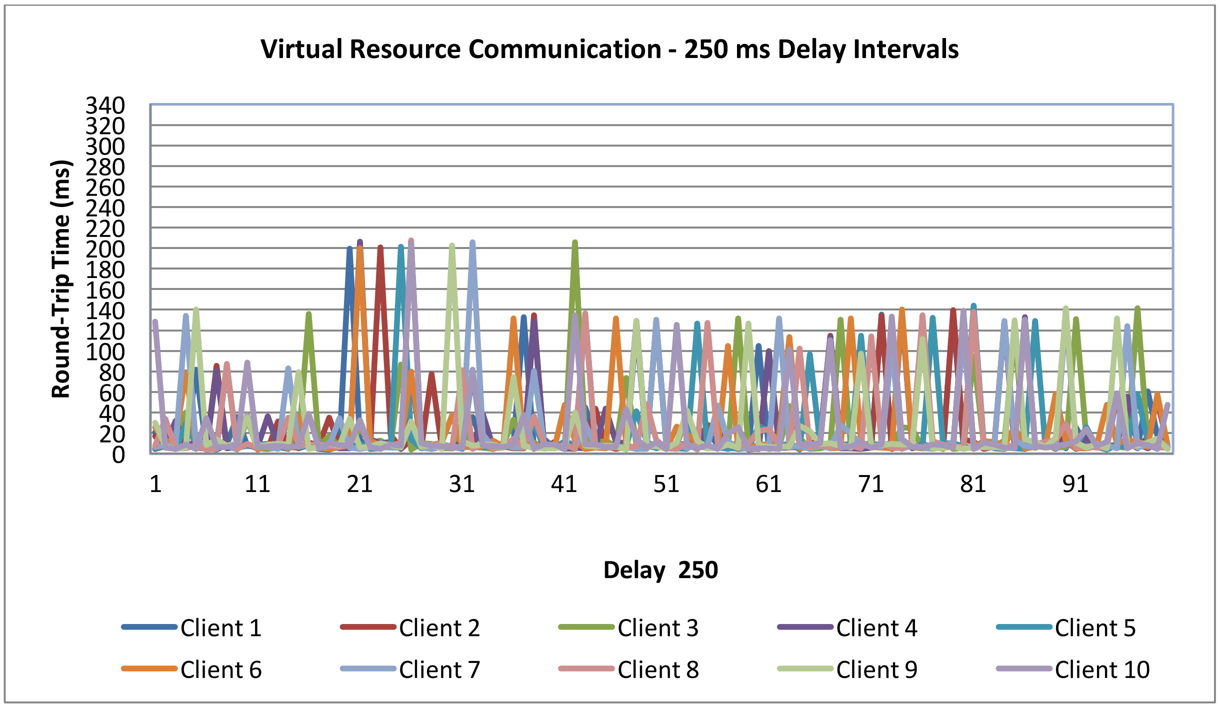


Figure 6.15 Results of Experiment 3. Ten View Virtual Resources are sending 100 CoAP POST requests to one Atomic Virtual Resource. The payload is 8 bytes + 16 bytes AES synchronous encryption. 250 ms delay.

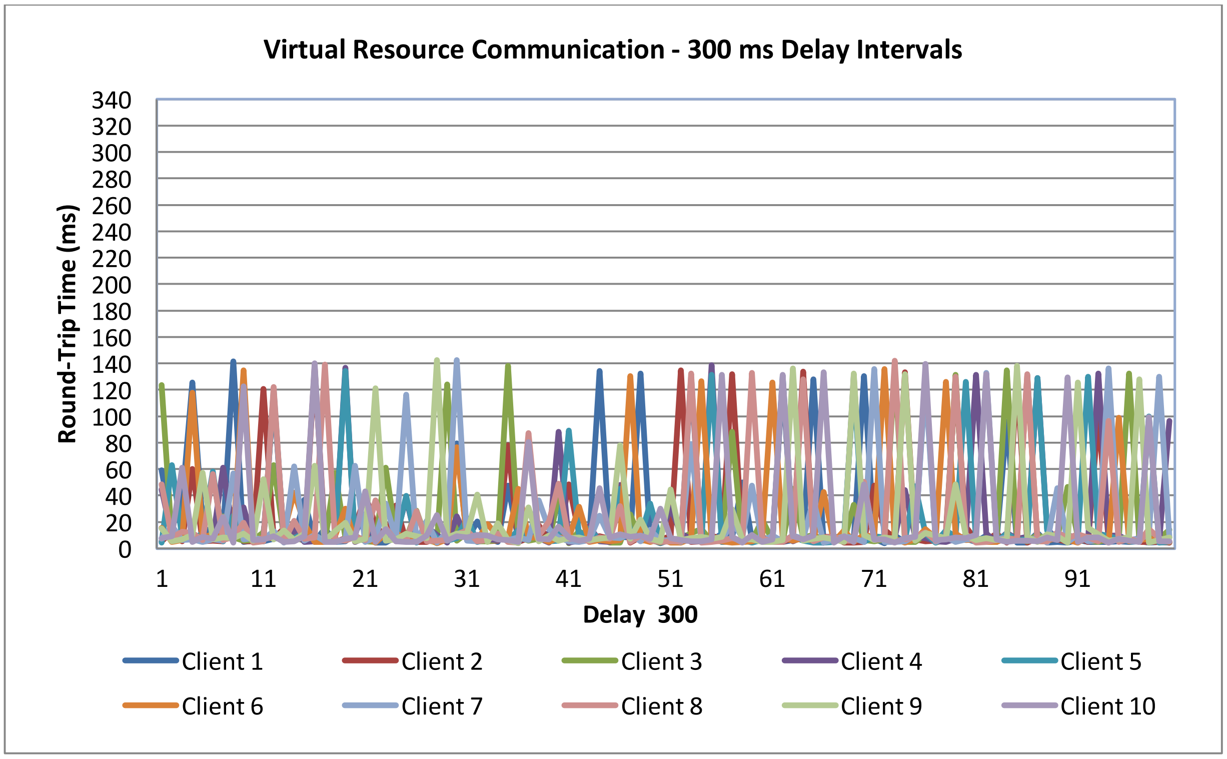


Figure 6.16 Results of Experiment 3. Ten View Virtual Resources are sending 100 CoAP POST requests to one Atomic Virtual Resource. The payload is 8 bytes + 16 bytes AES synchronous encryption. 300 ms delay.

6.2.3 Experiment 4:

This experiment evaluates the impact of different payload sizes in the performance of Virtual Resources deployed on Edison Arduino boards. The hardware setup of this experiment is the same as in Experiment 2 (Figure 6.5). The setup includes two Edison Arduino boards, which represent the IoT Things layer. Both boards are connected to the university's Wi-Fi network and communicate via CoAP protocol. The characteristics of the Edison Arduino boards are explained in Table 6-3 from Experiment 2.

The first Edison board hosts an Atomic Virtual Resource, which responds with a CoAP acknowledgment message to all requests. The second Edison board hosts one View Virtual Resource, which sends 1000 CoAP POST requests to the Atomic Virtual Resource. The payload size of the requests are between 16 bytes and 512 bytes. The payload is encrypted using an AES synchronous encryption method with a key of 16 bytes. The total payload size would be from 16 bytes to 512 bytes + 16-bytes AES encryption key.

Figures 6.17 to 6.19 present the results of this experiment. Each graph presents the results obtained by sending two different payload sizes. The y-axis represents the round-trip time in milliseconds, measured from the View Virtual Resource side. The x-axis represents each View Virtual Resource's request (1-1000).

Overall, the results indicate that the payload size does not affect the communication between the Virtual Resources. The round-trip time values are between 5 and 10 ms. There is a slight difference sending CoAP POST requests with different payload sizes, the average round-trip time values of each delay are explained as follows. The 16-bytes-payload series has an average round-trip time of 6.46 ms. The 32-bytes-payload series has an average round-trip time of 6.47 ms. The 64-bytes-payload series has an average round-trip time of 6.72 ms. The 128-bytes-payload series has an average round-trip time of 6.75 ms. The 256-bytes-series has an average round-trip time of 6.94 ms. Finally, the 51- bytes-series has an average round-trip time of 7.54 ms. The computational capabilities of the Edison board do not affect the data transmission.

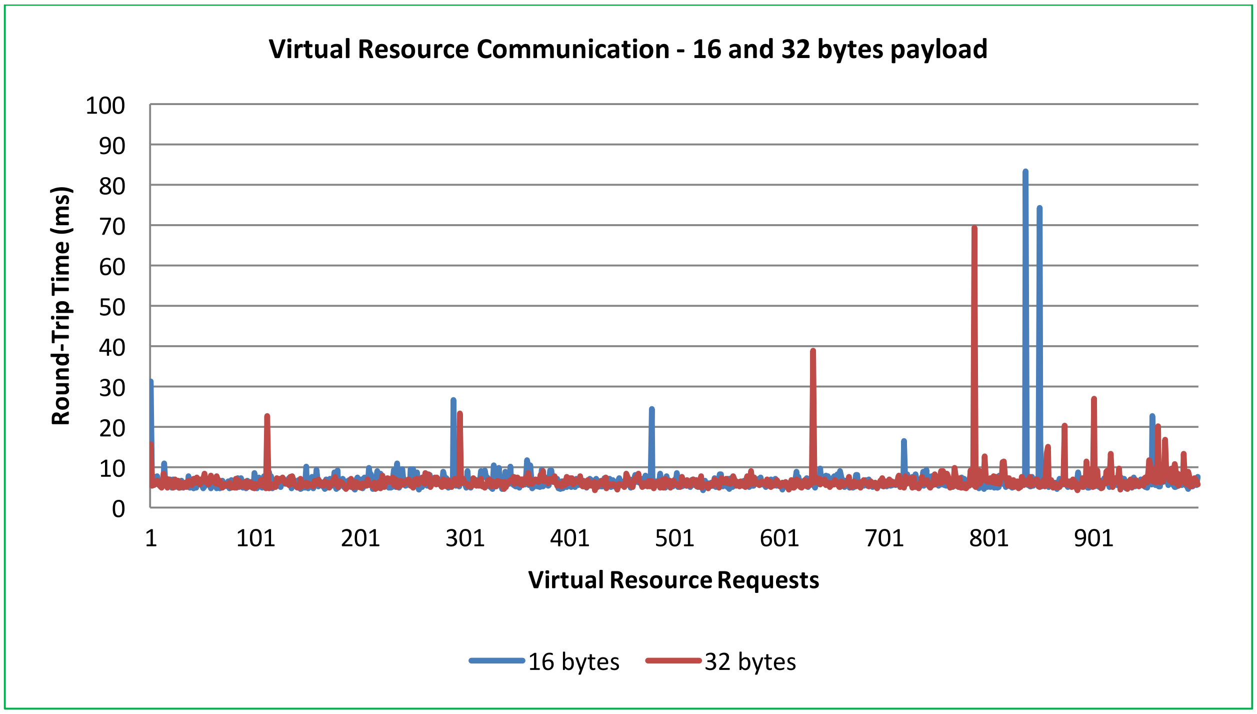


Figure 6.17 Results of Experiment 4. One View Virtual Resource sending 1000 requests to one Atomic Virtual Resource. Payload 16 bytes and 32 bytes + 16 bytes AES synchronous encryption.

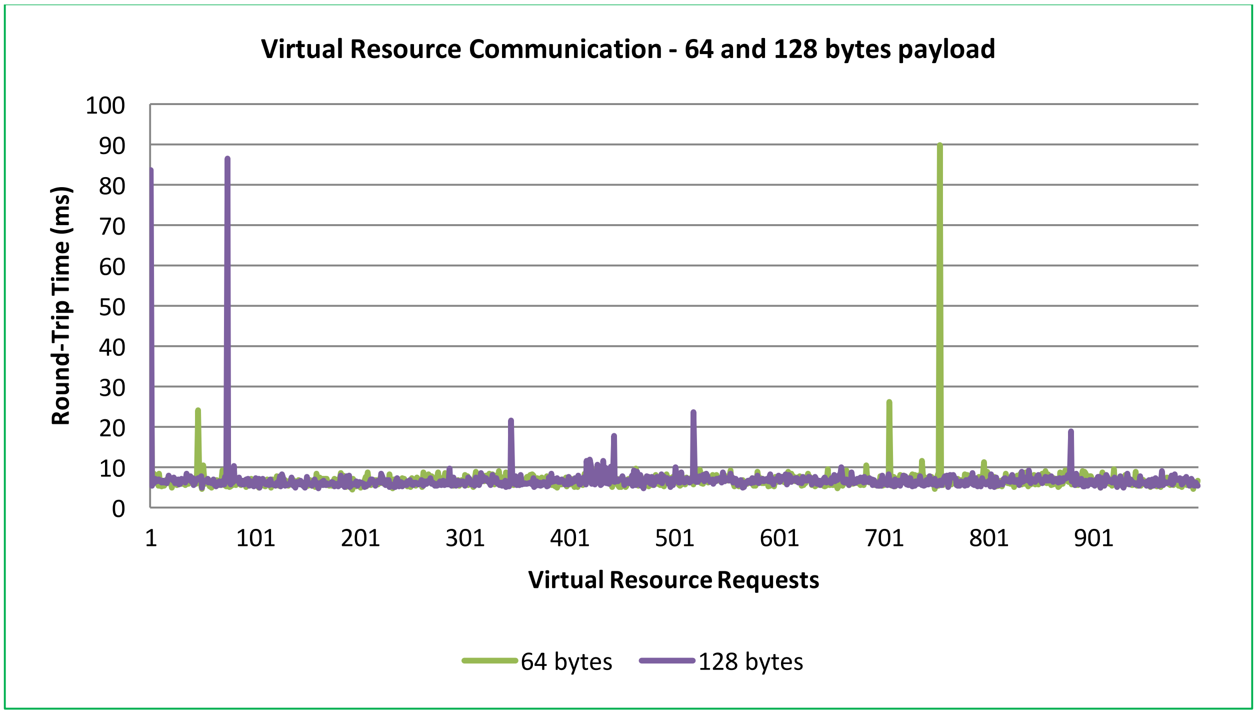


Figure 6.18 Results of Experiment 4. One View Virtual Resource sending 1000 requests to one Atomic Virtual Resource. Payload 64 bytes and 128 bytes + 16 bytes AES synchronous encryption.

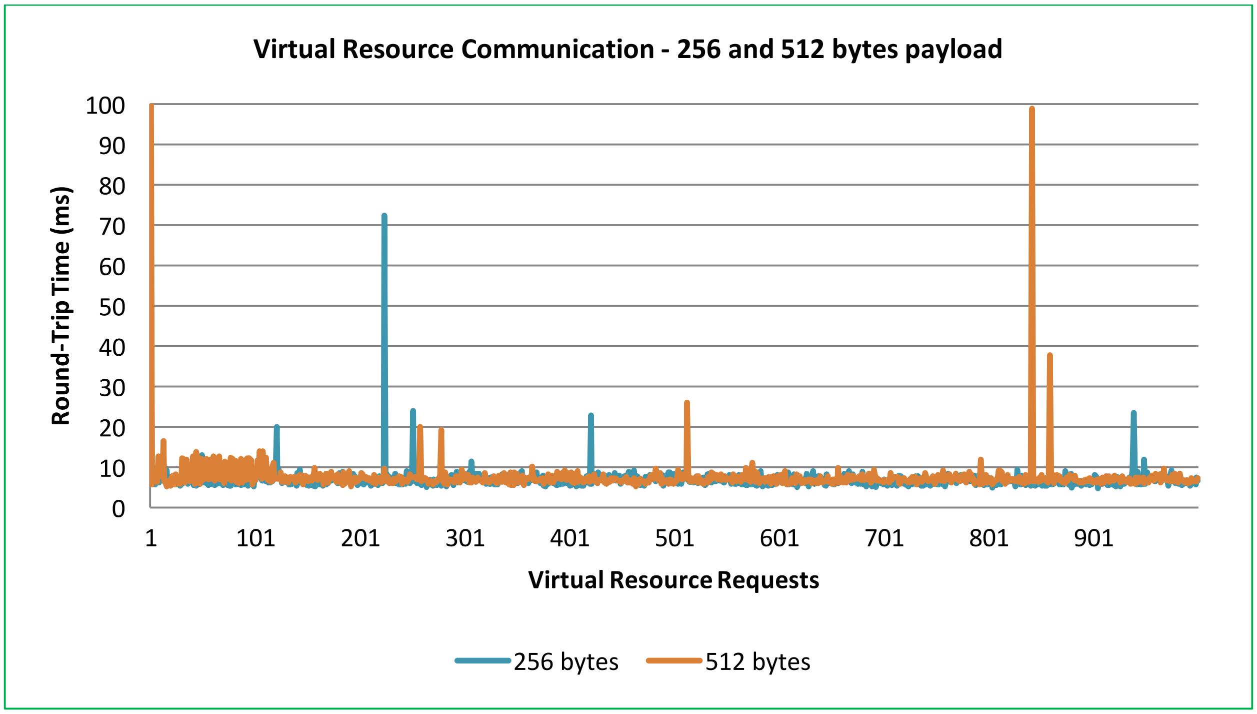


Figure 6.19 Results of Experiment 4. One View Virtual Resource sending 1000 requests to one Atomic Virtual Resource. Payload 256 bytes and 512 bytes + 16 bytes AES synchronous encryption.

6.2.4 Summary:

These experiments show that Virtual Resources defined as REST micro services, programmed in Go language and communicated via CoAP protocol can be successfully deployed on different IoT devices. Also, these experiments show that the definition of Virtual Resources presented by this work can successfully handle concurrent access with different payload sizes.

6.3 Provisioning and multi-tenant access to Virtual Resources using Blockchain

6.3.1 Experiment 5:

This experiment evaluates the communication performance between a Virtual Resource and a Multichain blockchain cluster. Figure 6.20 shows the setup of this experiment. The setup includes a cluster of three machines running Multichain blockchain. One machine runs a python API that handles read and write operations to the blockchain cluster. The Multichain cluster represents the Fog layer and stores the configuration of Virtual Resources. The experiment setup also includes one Edison Arduino board, which represents the IoT Things layer. The Edison and Multichain cluster are connected to the university's Wi-Fi network and communicate via HTPP protocol. The characteristics of the Multichain nodes are specified in Table 6-4.

The Edison board hosts one View Virtual Resource that sends 1000 HTTP POST requests to the python blockchain API. Each request asks for permission to write blocks. The payload of the requests is 712 bytes. The payload includes 256 bytes encrypted using an AES encryption method with a key of 16 bytes.

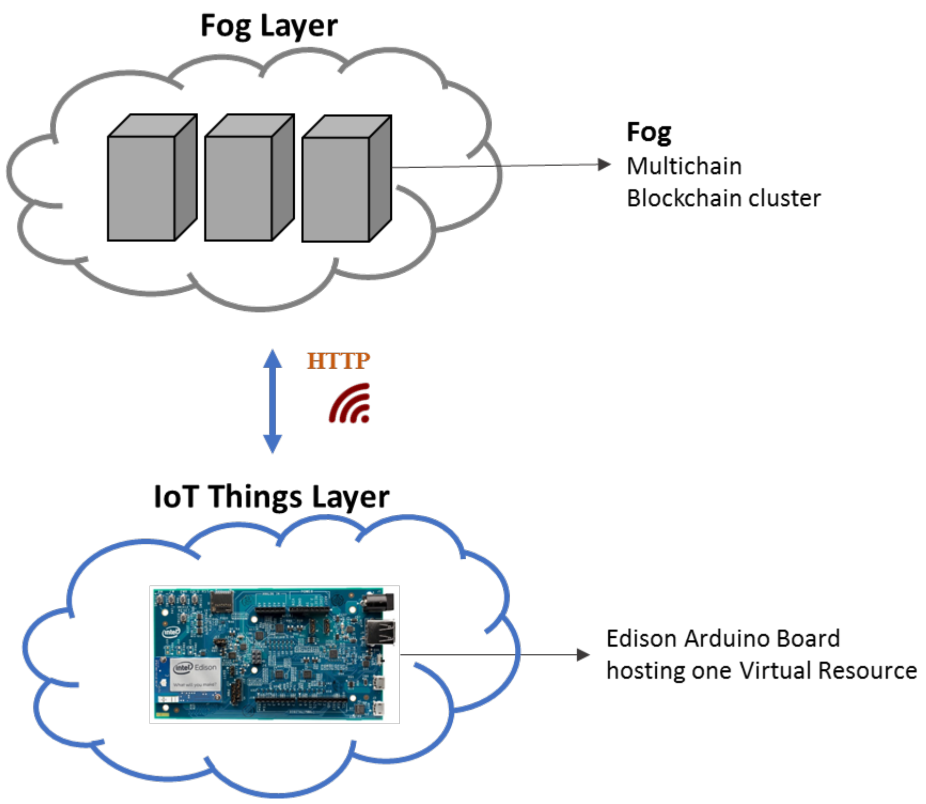


Figure 6.20 Setup of experiment 5. One Edison module connected to the Wi-Fi network and communicating via HTTP protocol with a Multichain blockchain cluster hosted in a Fog layer.

Table 6-4 Specification of the Multichain blockchain nodes.

Hardware Edison Arduino Board	Details
Operating System	Linux Debian 8.5 (Jessie)
CPU	Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
RAM	14 GB

Figures 6.21 to 6.24 summarize the results of this experiment. In the graphs, the y-axis represents the round-trip time in milliseconds measured from the side of the View Virtual Resource. The x-axis represents each View Virtual Resource request (1-1000). This experiment introduces delays between 0 and 300 ms before issuing each request to the blockchain cluster. The delays are useful to evaluate the performance of the Virtual Resources under different load scenarios. Each series of the graphs represents the delay intervals.

In this experiment, the average communication time between the Virtual Resource and the blockchain cluster is between 10 to 40 ms. The arrival rate variation of the requests makes the round-trip time slightly decrease. Hence, this private blockchain cluster does not represent a bottleneck. The traffic and the network card determine the cluster performance.

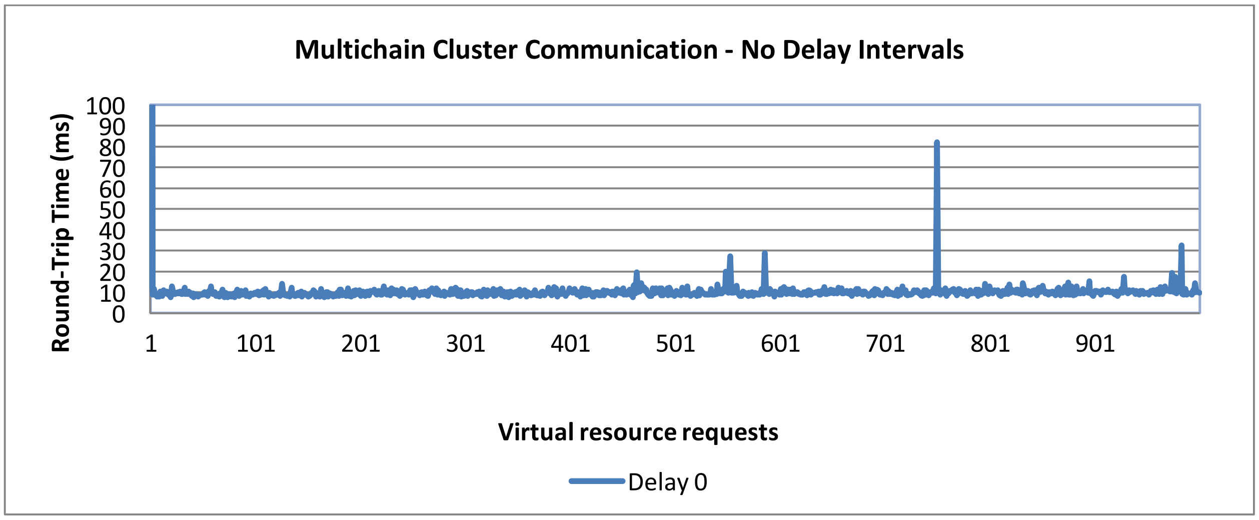


Figure 6.21 Results of Experiment 5. One View Virtual Resource is sending 1000 synchronous requests to the Multichain cluster. The payload is 712 bytes. No delay.

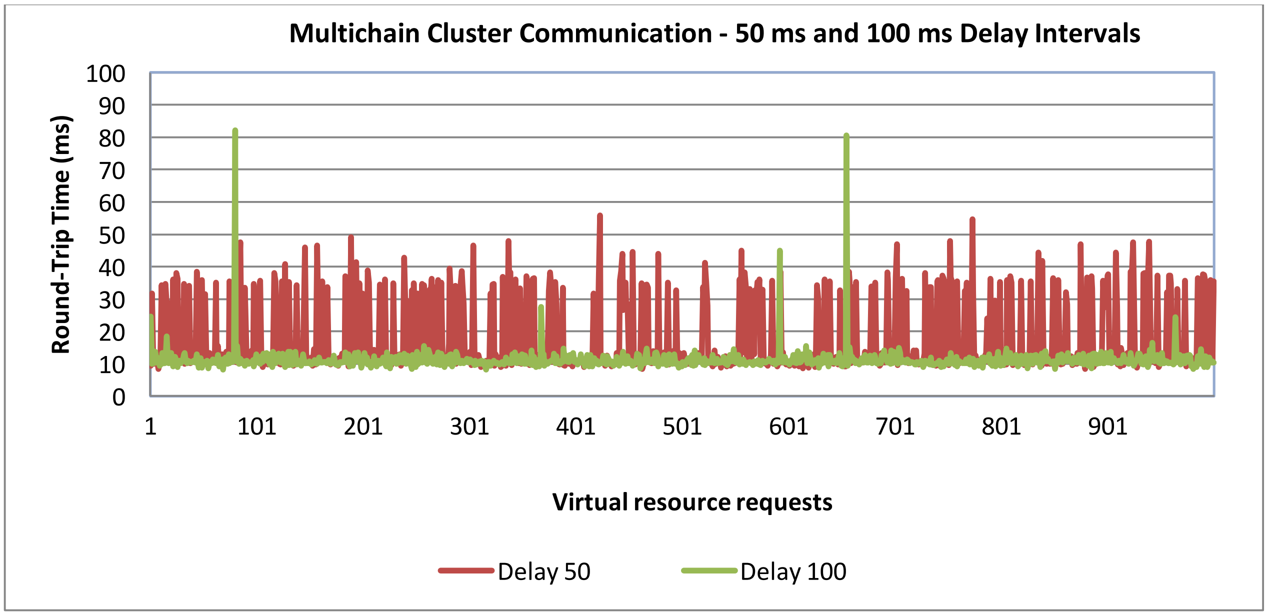


Figure 6.22 Results of Experiment 5. One View Virtual Resource is sending 1000 synchronous requests to the Multichain cluster. The payload is 712 bytes. 50 ms and 100 ms delays.

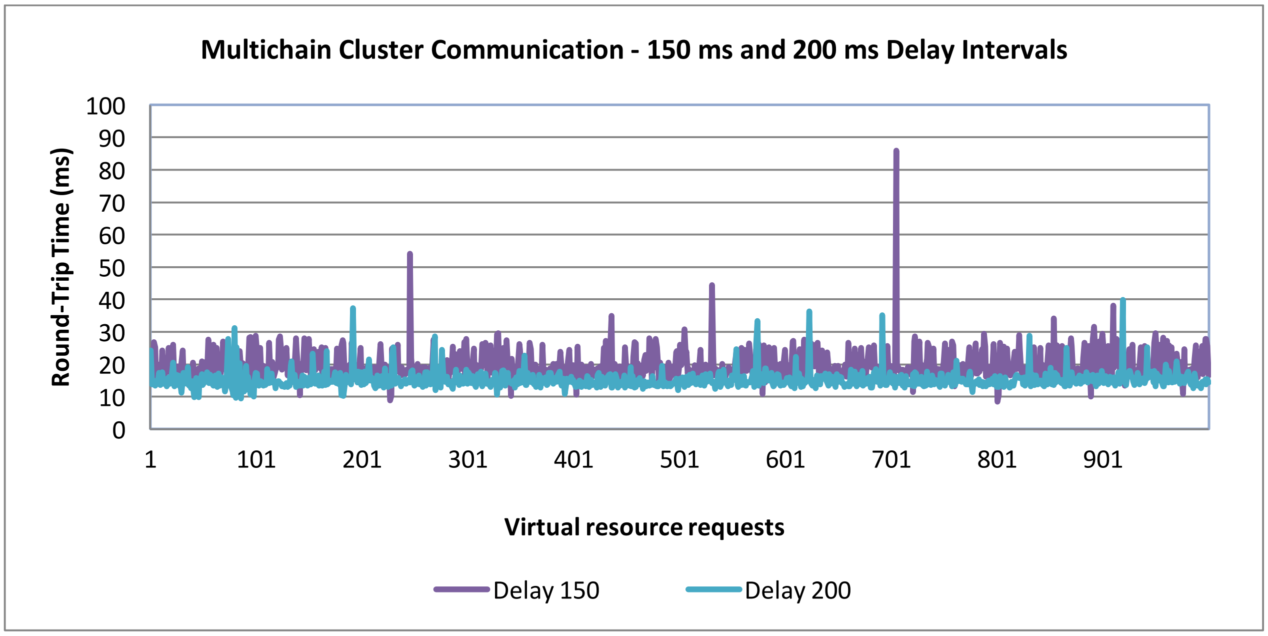


Figure 6.23 Results of Experiment 5. One View Virtual Resource is sending 1000 synchronous requests to the Multichain cluster. The payload is 712 bytes. 150 ms and 200 ms delays.

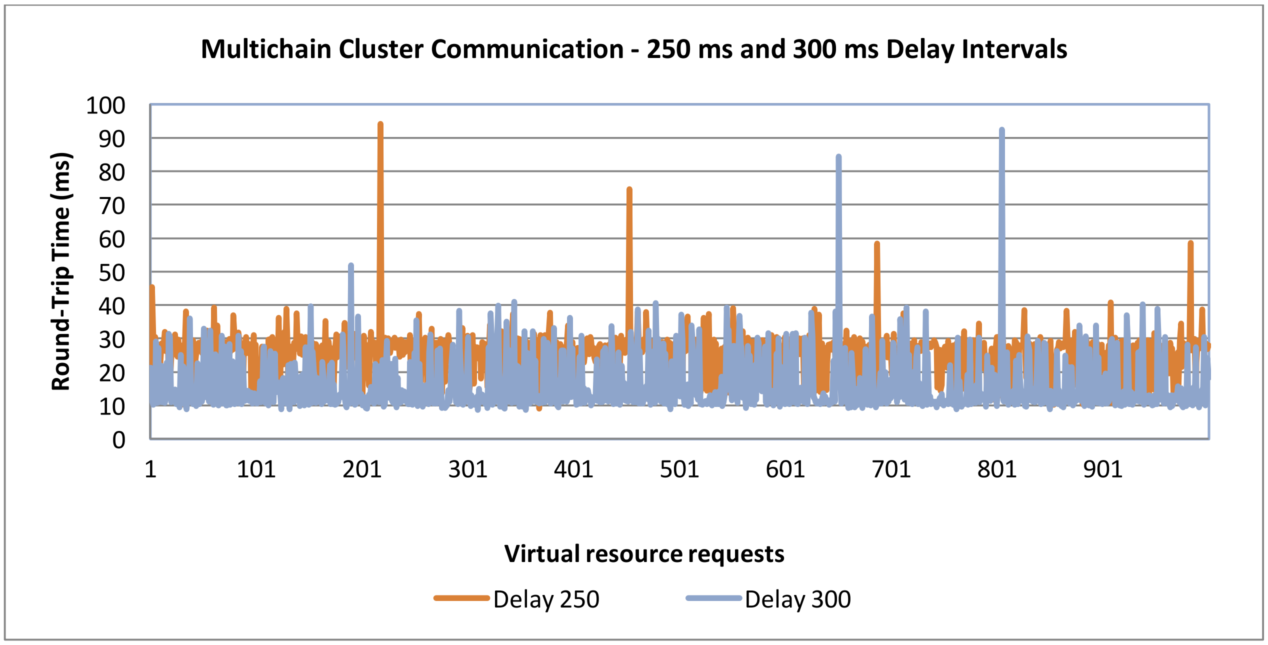


Figure 6.24 Results of Experiment 5. One View Virtual Resource is sending 1000 synchronous requests to the Multichain cluster. The payload is 712 bytes. 250 ms and 300 ms delays.

6.3.2 Experiment 6:

This experiment evaluates the impact of ten concurrent requests in the performance of the Multichain blockchain cluster. The characteristics of the Multichain cluster nodes are explained in Table 6-4 from Experiment 5. The hardware setup of this experiment is the same as in Experiment 5 (Figure 6.20). The setup includes a cluster of three machines running Multichain blockchain, one of them runs a python API that handles the operations to the blockchain cluster. The Multichain cluster represents the Fog layer. An Edison Arduino board in the IoT Things layer is also part of the experiment. The Multichain cluster and the Edison board are connected to the university's Wi-Fi network and communicate to each other via HTPP protocol.

The Edison board hosts ten View Virtual Resources that request for writing operations to the python API. These Virtual Resources send 100 HTTP POST requests each. The payload of all requests is 712 bytes. The payload includes 256 bytes encrypted using an AES encryption method with a key of 16 bytes.

Figures 6.25 to 6.31 show the results of this experiment. The graphs show the performance of the Multichain cluster. The series of each graph represent the delay intervals. The y-axis represents the RTT in milliseconds measured from the side of the View Virtual Resource. The x-axis represents each View Virtual Resource request (1-100). This experiment introduces delays of 0, 50, 100, 150, 200, 250 and 300 milliseconds in issuing the requests to the blockchain API.

The results indicate that the blockchain cluster is affected by the request concurrency. Overall, the concurrent requests increase de round-trip time values, between 20 ms and 50 ms. Additionally, increasing the delay time causes the peaks to decrease.

The average round-trip time of each delay series is explained as follows. The series with no delay has an average round-trip time of 59 ms. The 50ms-delay series has an average round-trip time of 34.74 ms. The 100ms-delay series has an average round-trip time of 25.96 ms. The 150ms-delay series has an average round-trip time of 37.93 ms. The 200ms-delay series has an average round-trip time of 20.78 ms. The 250ms-delay series has an average round-trip time of 24.13 ms. Finally, the 300ms-delay series has an average round-trip time of 24.16 ms.

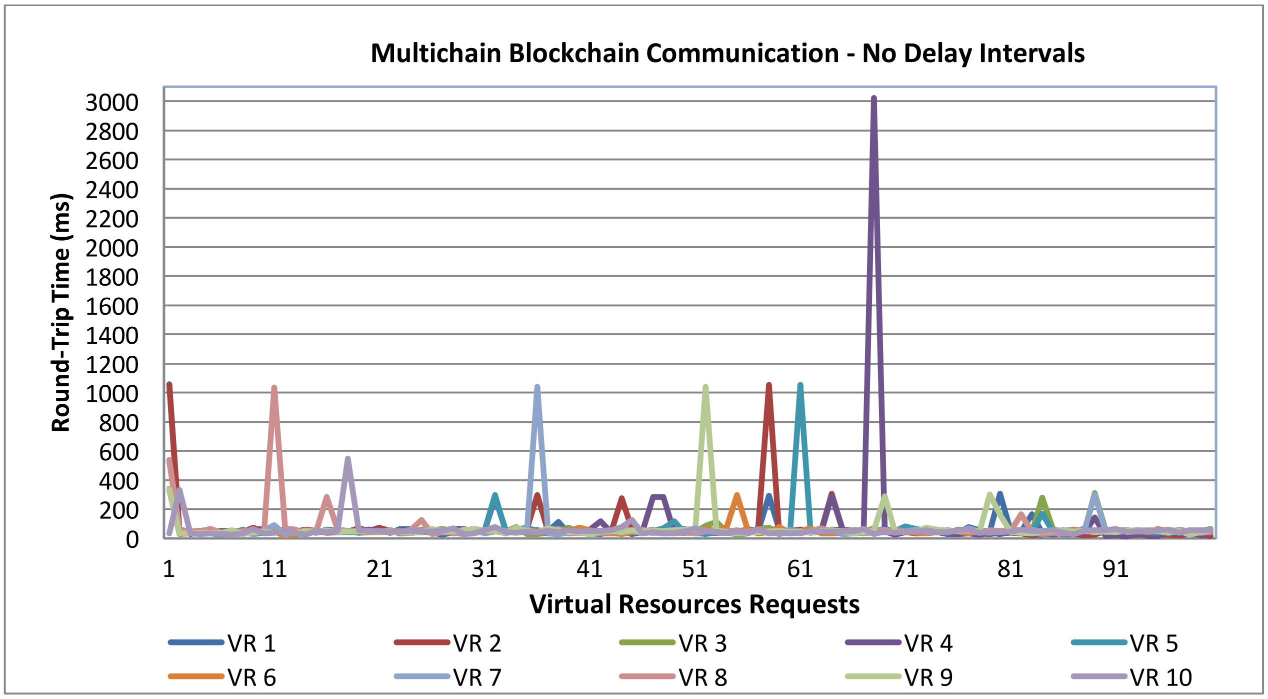


Figure 6.25 Results of Experiment 6. Ten View Virtual Resources are sending 100 asynchronous requests to the Multichain cluster. The payload is 712 bytes. No delay.

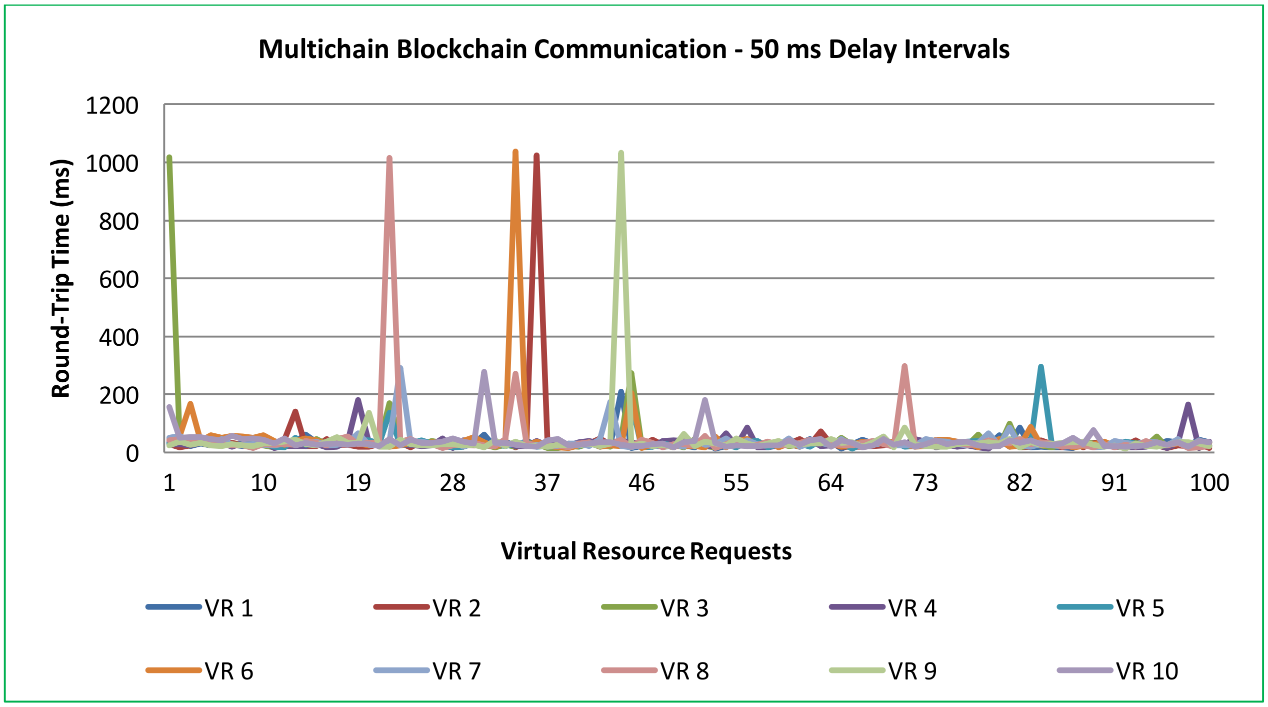


Figure 6.26 Results of Experiment 6. Ten View Virtual Resources are sending 100 asynchronous requests to the Multichain cluster. The payload is 712 bytes. 50 ms delay.

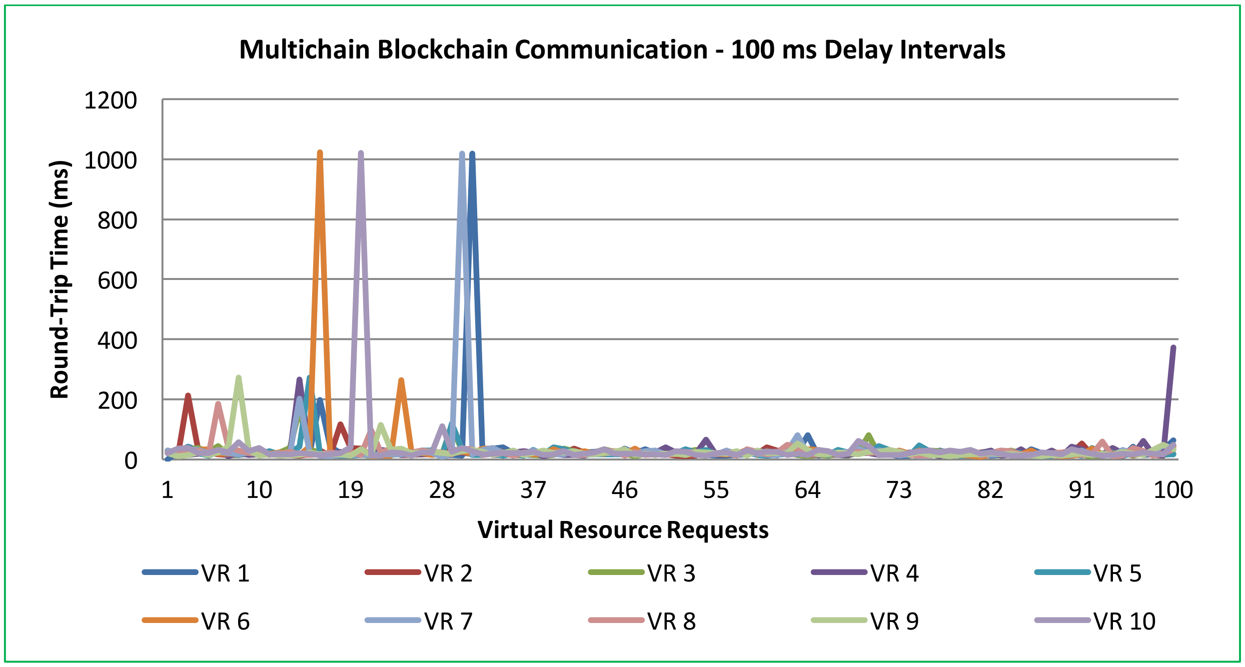


Figure 6.27 Results of Experiment 6. Ten View Virtual Resources are sending 100 asynchronous requests to the Multichain cluster. The payload is 712 bytes. 100 ms delay.

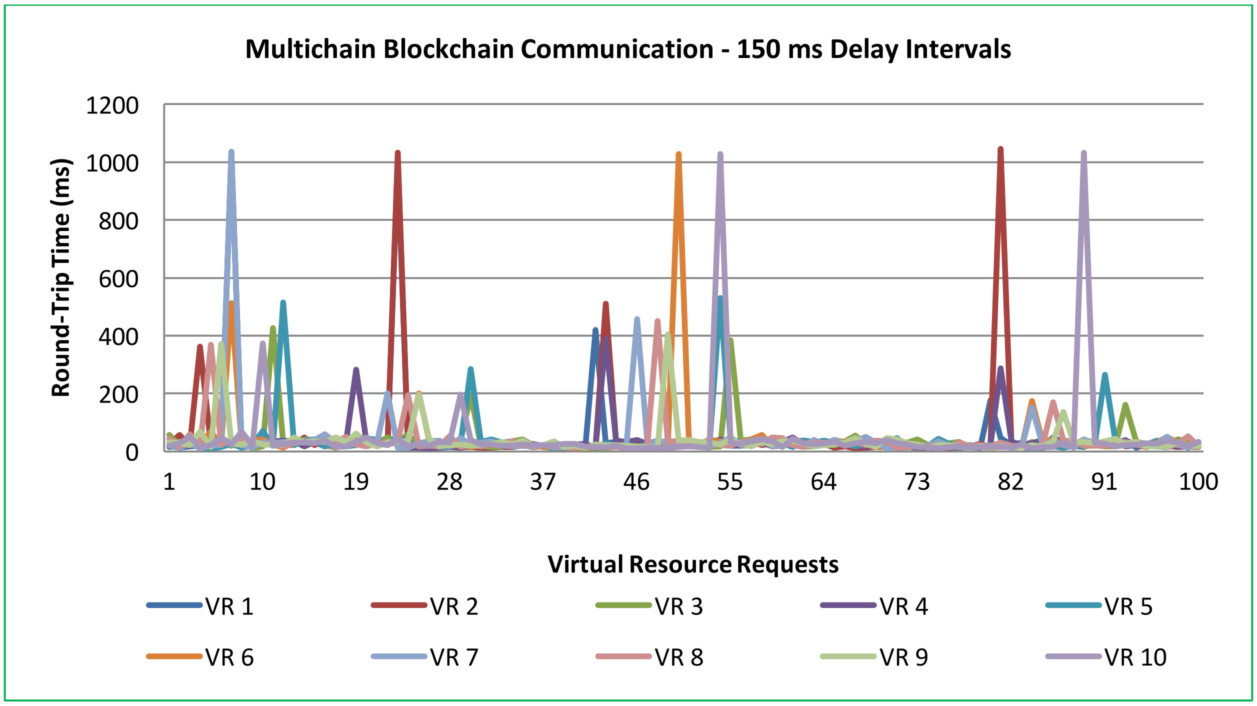


Figure 6.28 Results of Experiment 6. Ten View Virtual Resources are sending 100 asynchronous requests to the Multichain cluster. The payload is 712 bytes. 150 ms delay.

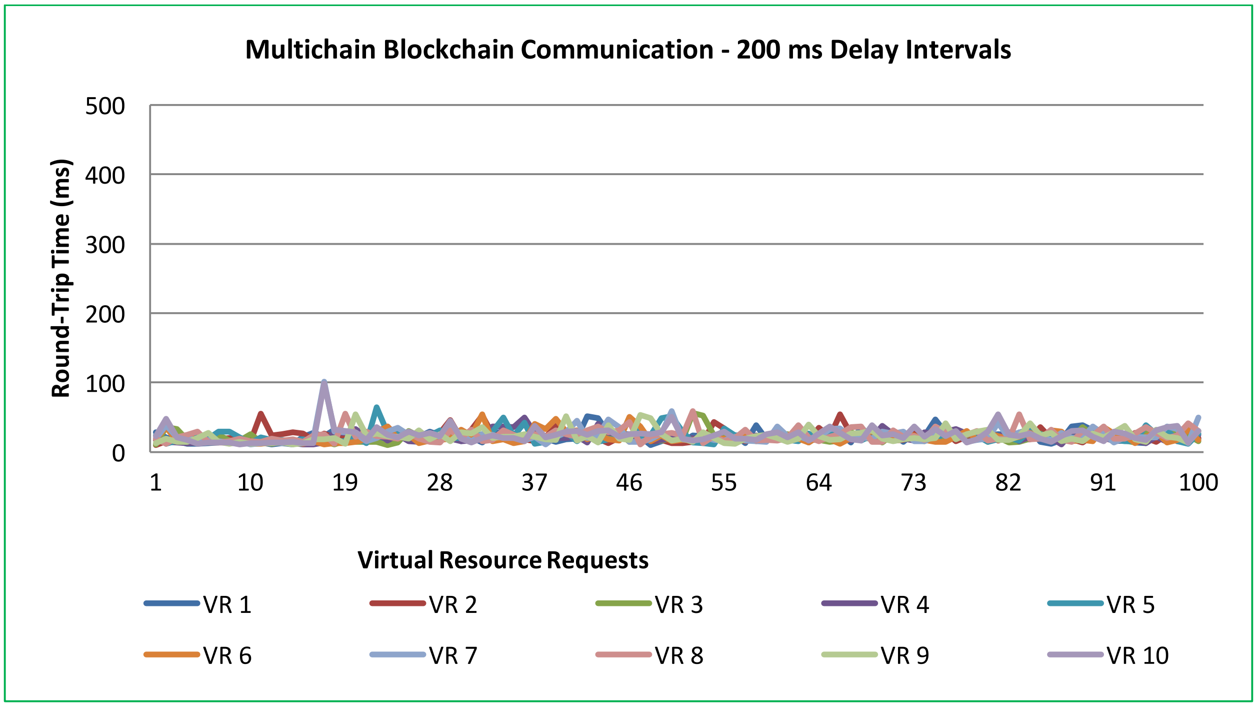


Figure 6.29 Results of Experiment 6. Ten View Virtual Resources are sending 100 asynchronous requests to the Multichain cluster. The payload is 712 bytes. 200 ms delay.

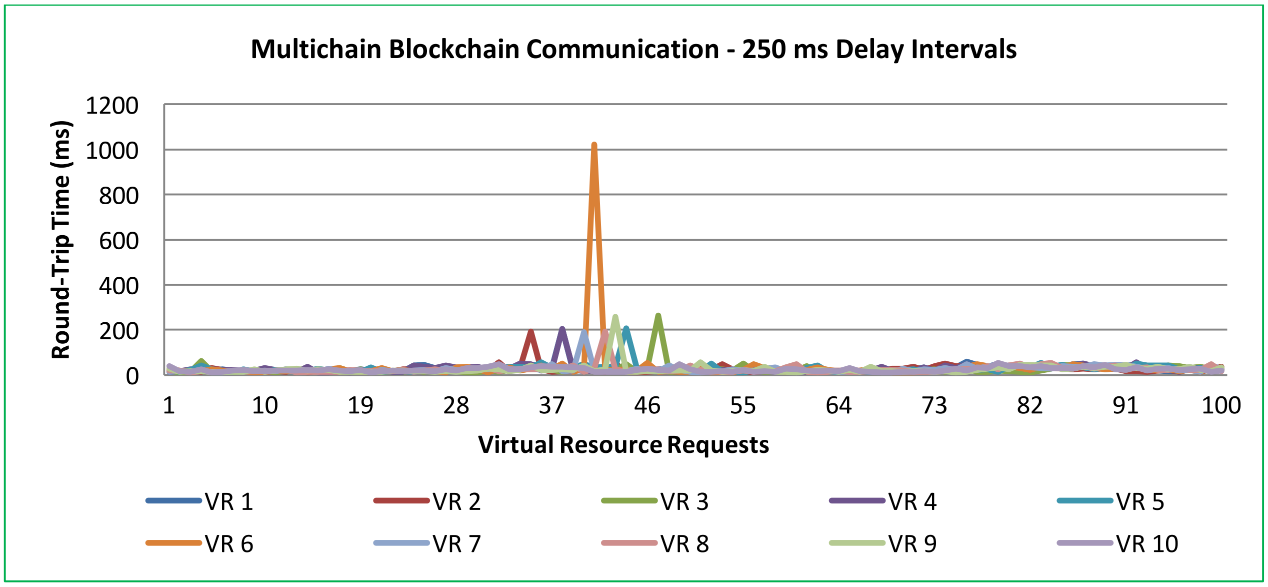


Figure 6.30 Results of Experiment 6. Ten View Virtual Resources are sending 250 asynchronous requests to the Multichain cluster. The payload is 712 bytes. 250 ms delay.

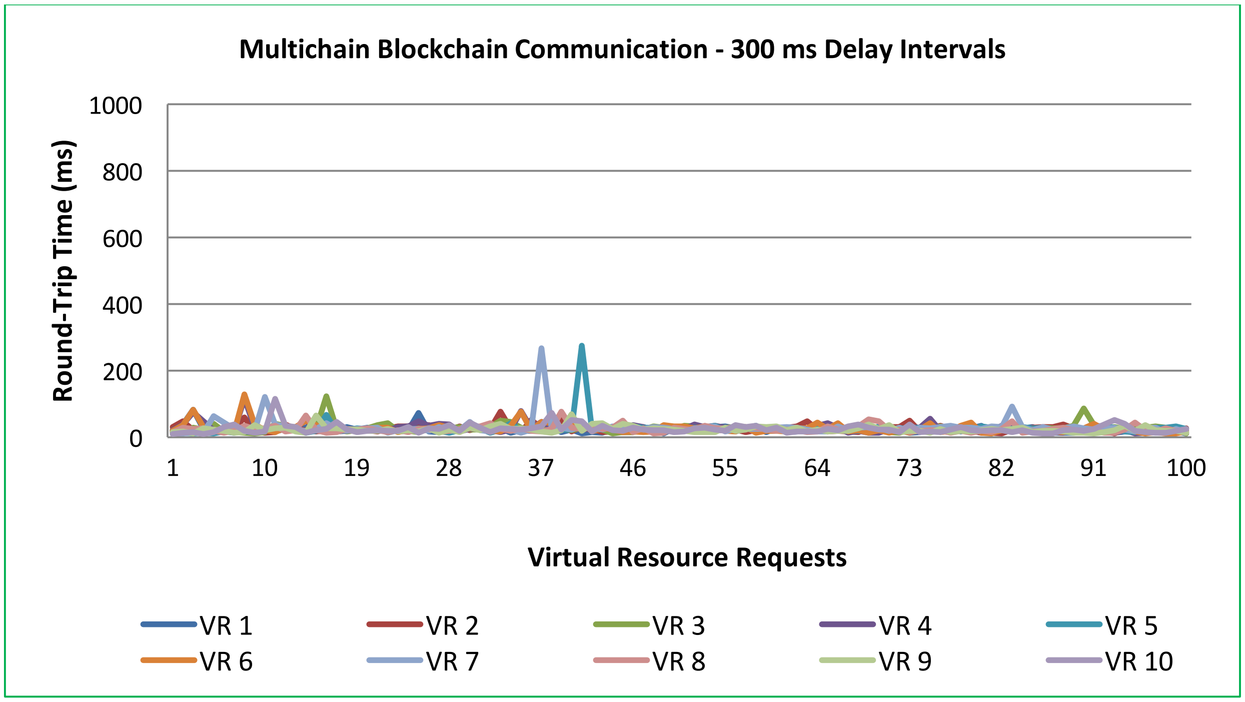


Figure 6.31 Results of Experiment 6. Ten View Virtual Resources are sending 100 asynchronous requests to the Multichain cluster. The payload is 712 bytes. 300 ms delay.

6.3.3 Experiment 7:

The following experiment evaluates the communication performance between a Virtual Resource and the Blockchain as a Service (BaaS) IBM Bluemix. Figure 6.32 shows the setup of this experiment. The experiment includes an IBM Bluemix free service account (http://www.ibm.com/blockchain/) and one Edison Arduino board connected to the Wi-Fi network. The communication between the blockchain service and the Edison board is implemented following the HTTP protocol.

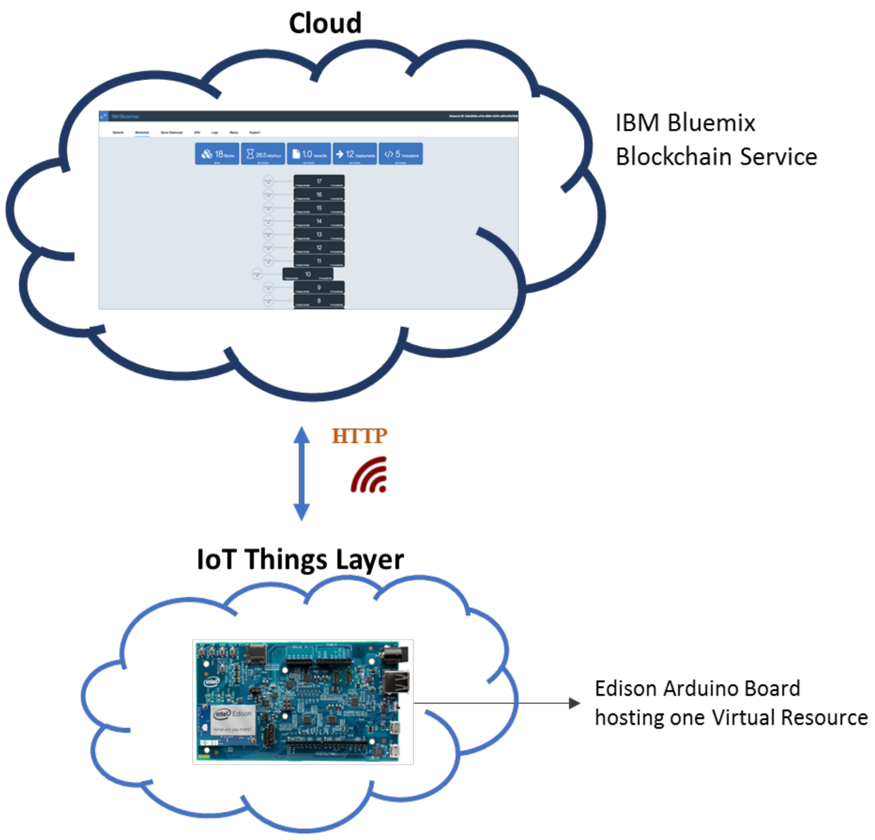

Cloud
IBM Bluemix Blockchain Service
HTTP
IoT Things Layer
Edison Arduino Board hosting one Virtual Resource

Figure 6.32 Setup of experiment 7. One Edison Arduino board connected to the Wi-Fi network and communicating via HTTP protocol to the IBM blockchain service in the Cloud.

The Edison Arduino board hosts ten View Virtual Resources that send 100 HTTP POST requests each to the blockchain service in the Cloud. The purpose of these requests is to perform a write operation in the blockchain. The payload size of all requests is 712 bytes, which includes 256 bytes encrypted using an AES encryption method with a key of 16 bytes.

Figures 6.33 to 6.39 present the results of this experiment. The graphs show the performance of IBM Bluemix blockchain service. Each graph represents the results of the ten Virtual Resources sending requests in certain delay interval. The y-axis represents the RTT in milliseconds measured from the side of the View Virtual Resource. The x-axis represents each View Virtual Resource request (1-100). Delay times of 0, 50, 100, 150, 200, 250 and 300 ms have been introduced to evaluate the performance of the blockchain cluster under different request loads.

The variation in the arrival rate of the requests does not lead to a better communication performance. Having the blockchain as a service in the Cloud means a significant impact on the performance. As Chapter 2 reviews, the latency caused by engaging Cloud services from IoT devices explains the high values of the round-trip time.

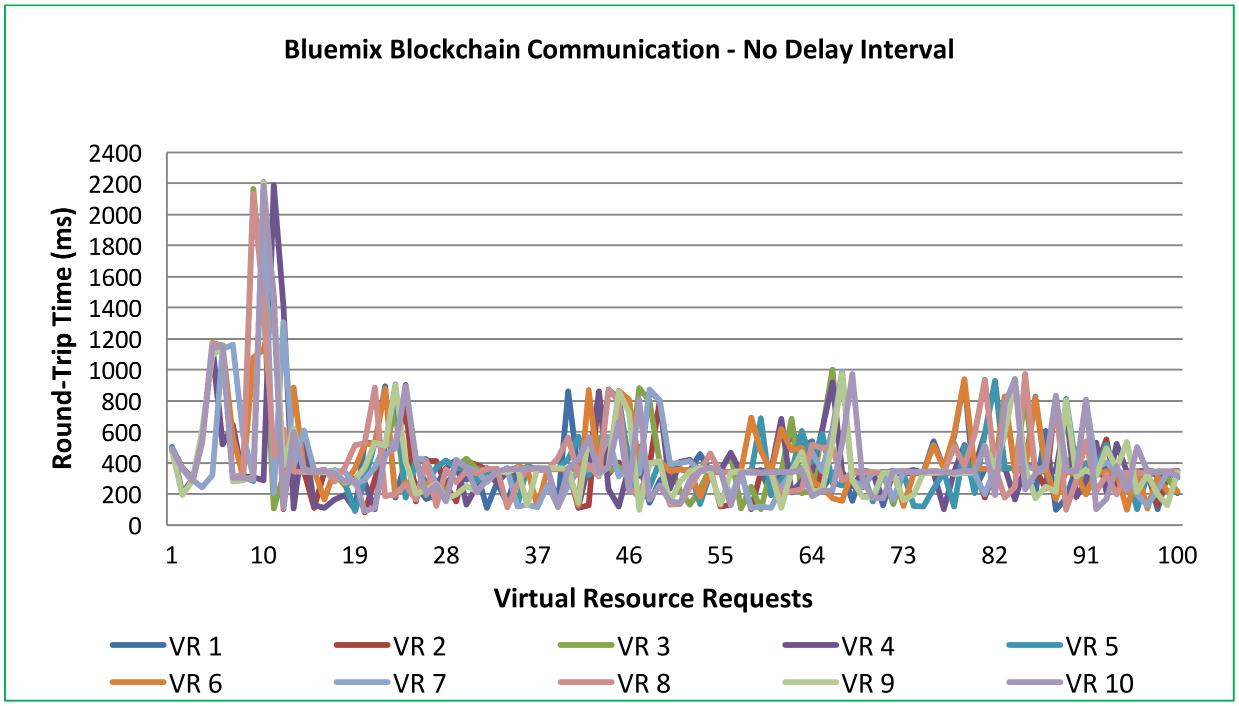


Figure 6.33 Results of Experiment 7. Ten View Virtual Resources are sending 100 asynchronous requests to the IBM Bluemix blockchain service in the Cloud. The payload is 712 bytes. No delay.

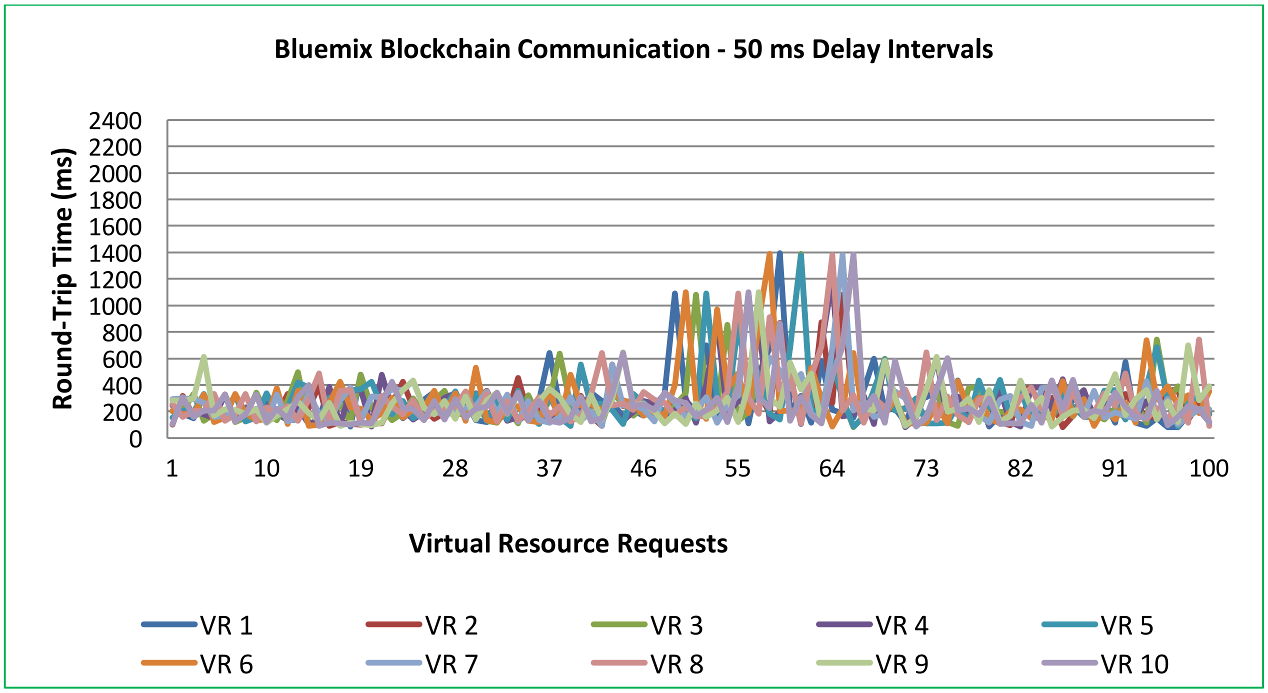


Figure 6.34 Results of Experiment 7. Ten View Virtual Resources are sending 100 asynchronous requests to the IBM Bluemix blockchain service in the Cloud. The payload is 712 bytes. 50 ms delay.

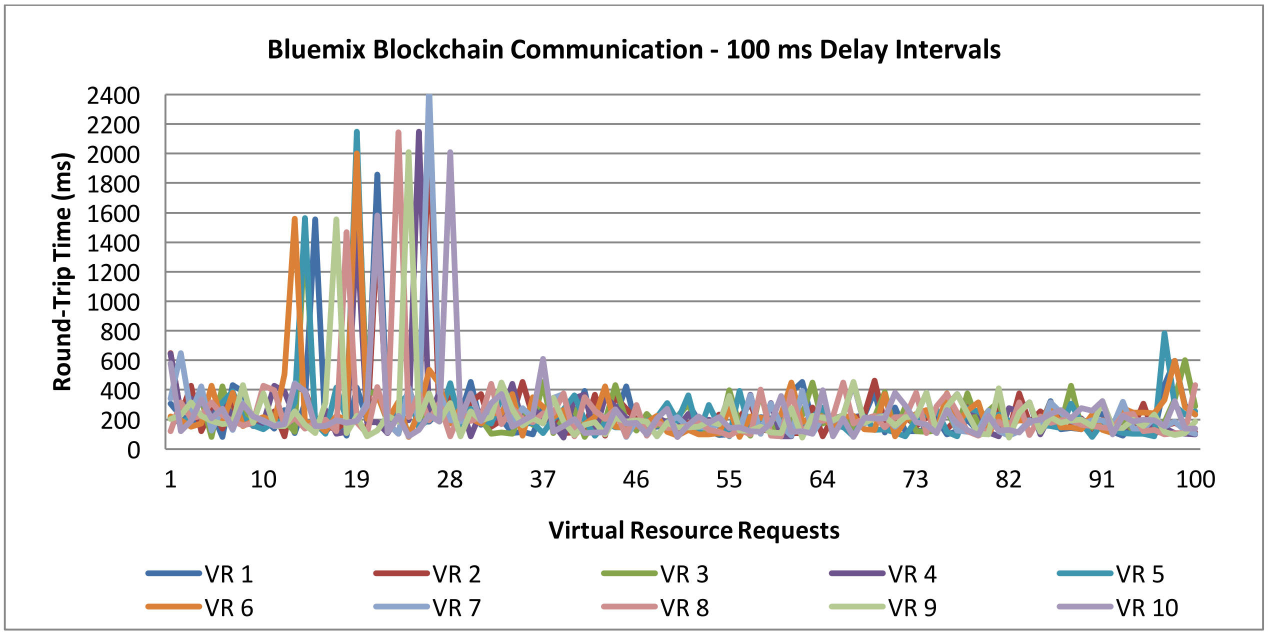


Figure 6.35 Results of Experiment 7. Ten View Virtual Resources are sending 100 asynchronous requests to the IBM Bluemix blockchain service in the Cloud. The payload is 712 bytes. 100 ms delay.

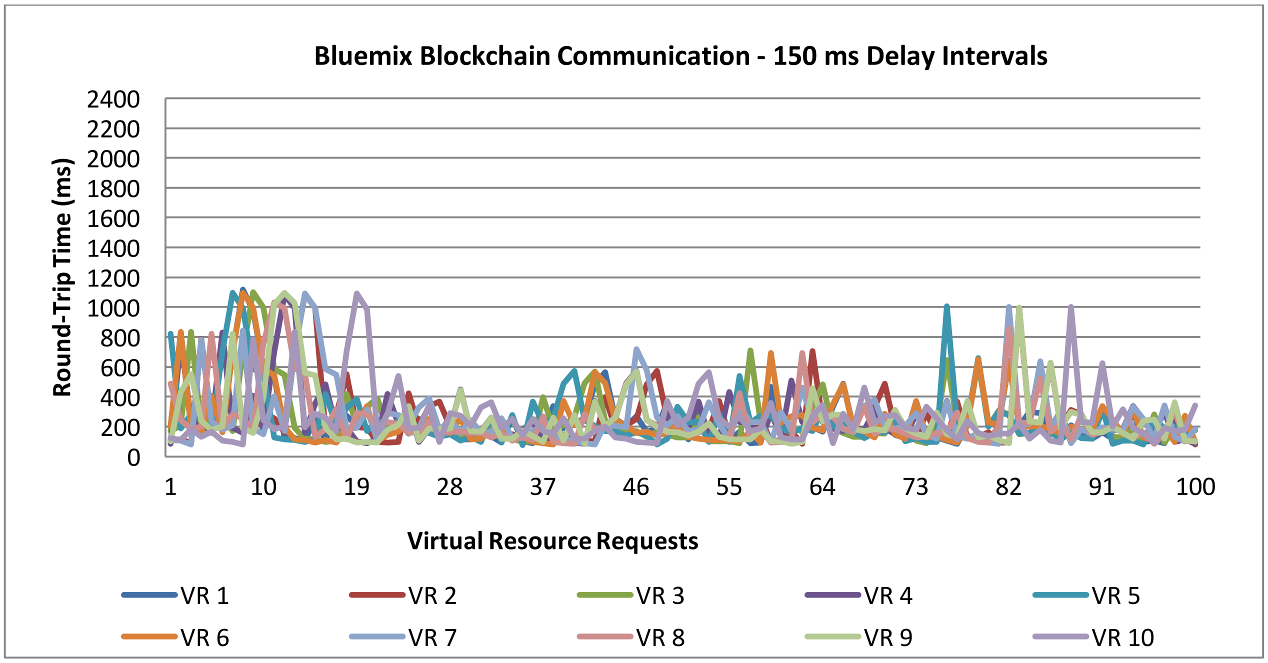


Figure 6.36 Results of Experiment 7. Ten View Virtual Resources are sending 100 asynchronous requests to the IBM Bluemix blockchain service in the Cloud. The payload is 712 bytes. 150 ms delay.

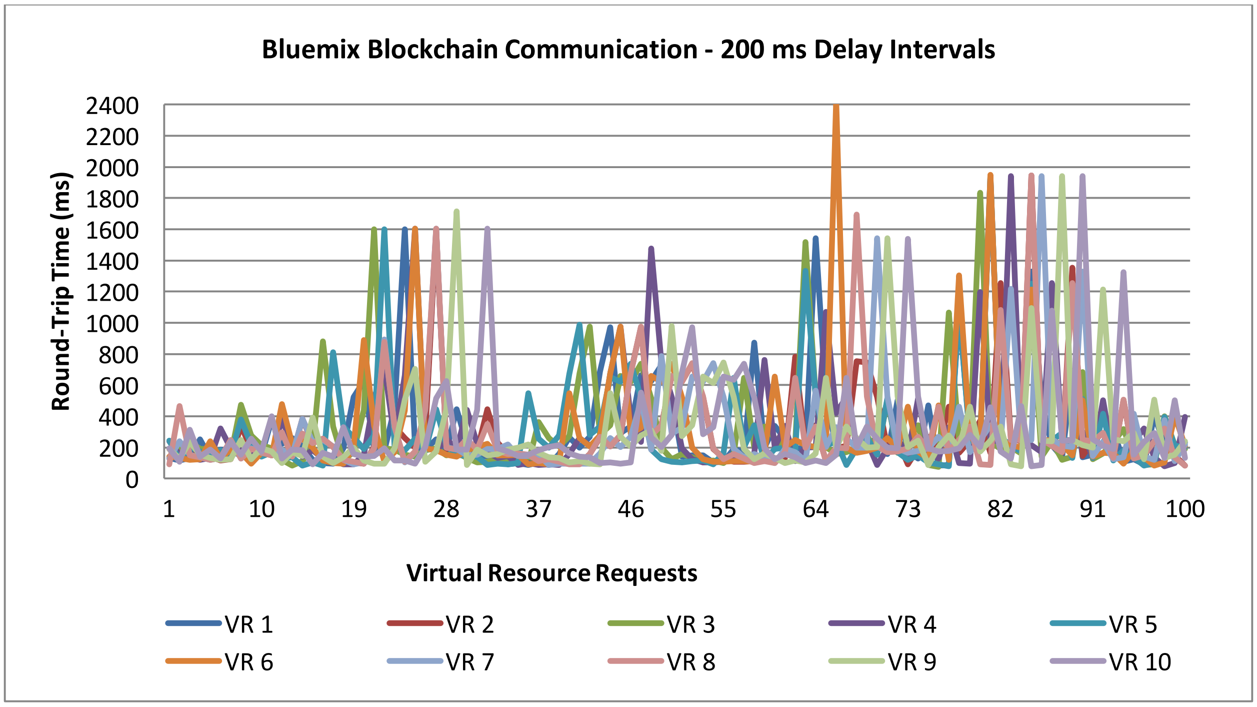


Figure 6.37 Results of Experiment 7. Ten View Virtual Resources are sending 100 asynchronous requests to the IBM Bluemix blockchain service in the Cloud. The payload is 712 bytes. 200 ms delay.

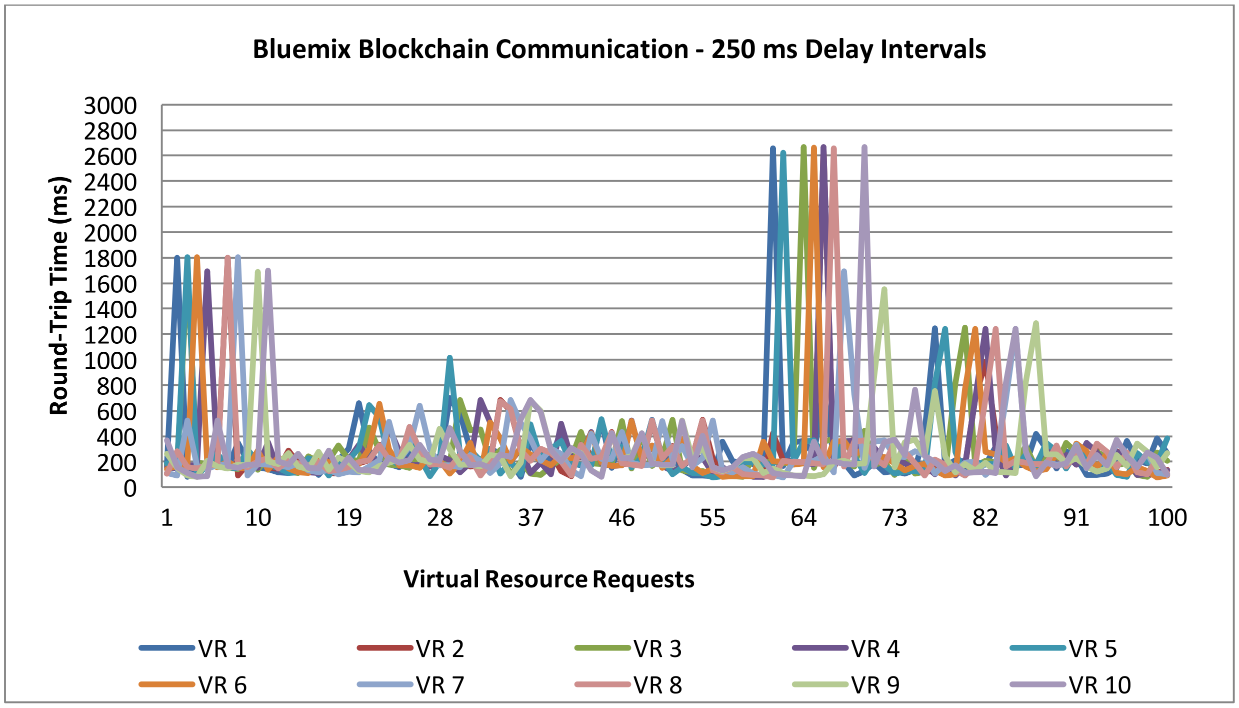


Figure 6.38 Results of Experiment 7. Ten View Virtual Resources are sending 100 asynchronous requests to the IBM Bluemix blockchain service in the Cloud. The payload is 712 bytes. 250 ms delay.

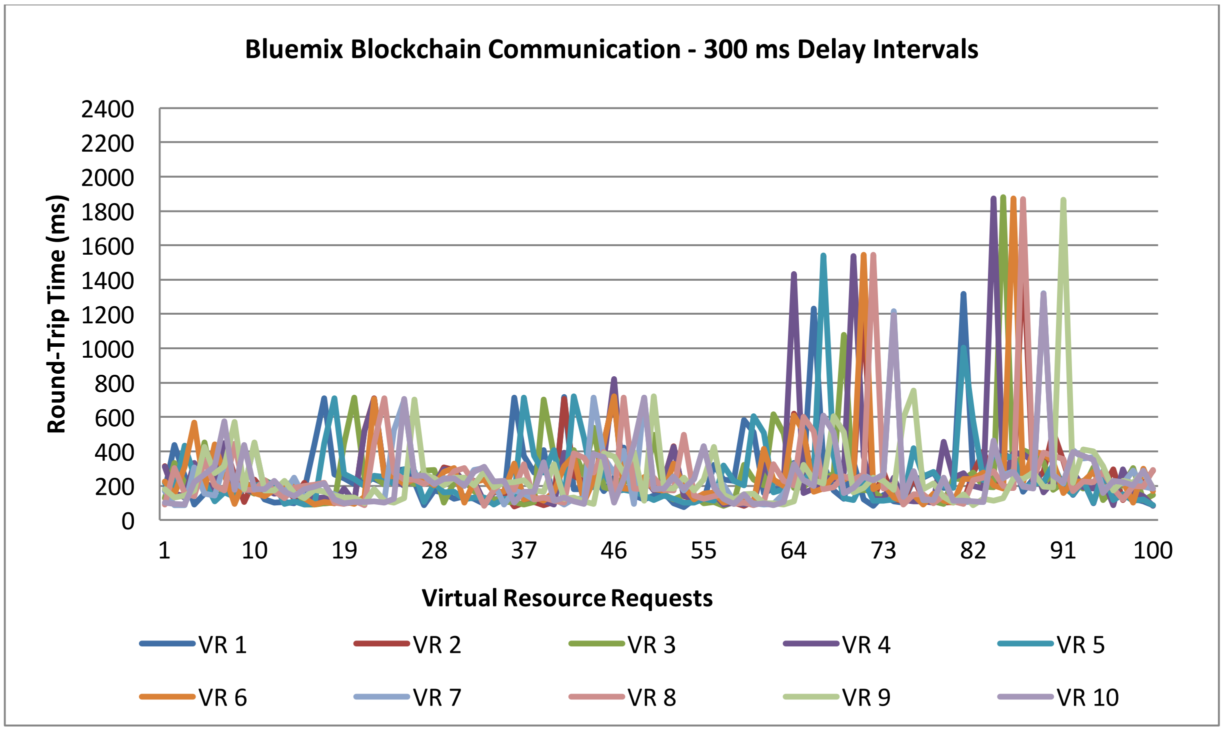


Figure 6.39 Results of Experiment 7. Ten View Virtual Resources are sending 100 asynchronous requests to the IBM Bluemix blockchain service in the Cloud. The payload is 712 bytes. 300 ms delay.

6.3.4 Summary:

Experiments 5 to 7 show that the definition of Virtual Resources presented by this work performs well when interacting with blockchains hosted in Fog nodes and the Cloud. These experiments indicate that permission-based blockchain technology can efficiently handle the duties of provisioning and multi-tenant access control in the IoT network.

Overall, the private blockchain Multichain has a better performance than the public blockchain service Bluemix. This result is obviously obtained due to the location of the Multichain cluster, which is closer to IoT devices. However, we can argue that the round robin process that Multichain uses to approve transactions demands less computational effort than the consensus process of Bluemix.

Bluemix as a Service can be used to store the configuration of high-level View Virtual Resources, which are required to be deployed on Fog nodes and accessed directly by third parties in the Cloud. On the other hand, private Blockchains like Multichain can be used to store the configuration of Atomic Virtual Resources, View Virtual Resources and Virtual Systems, which are required to be deployed on the IoT devices.

CHAPTER 7

CONTRIBUTION AND FUTURE WORK

7.1 Summary

Most of the Internet of Things (IoT) applications follow a Cloud-centric approach. Cloud-centric systems tend to isolate the "Things" due to the significant latency and bandwidth consumption necessary for the communication. Users do not interact with the constrained components but with virtualizations of them. This form of interaction is typical of Cloud-centric systems, which tend to ignore multi-tenancy as a direct manipulation of IoT devices is not supported.

This research evaluates existing technologies to develop a virtualization solution for IoT networks. The virtualization of IoT components introduces challenges in the provisioning and multi-tenancy services. This research proposes a definition of Virtual Resources deployed directly on IoT devices to handle those provisioning and multi-tenancy challenges. This work defines Virtual Resources as REST micro services and develop them using Go language following the CoAP protocol. Additionally, this research proposes blockchain to handle the provisioning of Virtual Resources and store the configuration of Virtual Systems for each tenant. Virtual Resources configured for each user (tenant) in a blockchain demonstrated to support provisioning and multi-tenancy.

The evaluations show that Virtual Resources can be deployed on different IoT platforms. Virtual Resources evidence a good performance when they are deployed on Raspberry Pi computers and Edison Arduino boards. The experiments with permission-based blockchains show that blocks are an efficient option to store the configuration of Virtual Resources and provision them on IoT devices. Also, these evaluations confirm that hosting applications at the edge of the IoT network notably reduces latency and bandwidth consumption. The decision-making over data becomes time-effective as the time the data takes to arrive at the processing unit (Fog layer) decreases.

7.2 Contributions

This research makes the following contributions.

7.2.1 Definition of Virtual Resources.

This work defines Virtual Resources as REST micro-services, which communicate via CoAP protocol. Virtual Resources expose two interfaces: "/.well-known/core" to discover the services of Virtual Resources and "/state" to get the current state of Virtual Resources. The CoAP methods that Virtual Resources implement are a simple mechanism to manipulate IoT components.

7.2.2 Provisioning of Virtual Resources on IoT devices.

This research explores two permission-based blockchain to handle the provisioning of Virtual Resources in the IoT network. Multichain, a private blockchain hosted in a Fog layer and IBM Bluemix blockchain as a Service hosted in the Cloud. The experiments show that it is possible to handle the provisioning of Virtual Resources storing encrypted configurations for each tenant in the form of blocks.

7.2.3 Support for Multi-tenant Access in IoT networks.

Permission-based blockchain manages a registry of tenants to control the access and operations on the blocks. Tenants must be registered in the blockchain and have the correct key to decrypt the blocks. Multi-tenancy is guaranteed when different tenants can access, write, and deploy Virtual Resources' configurations for different IoT components simultaneously.

7.3 Future Work

Virtual Resources are expected to be evaluated and improved in the following aspects.

7.3.1 Evaluation of Virtual Resources

A future work is focused on testing the performance of Virtual Resources on different IoT devices like Intel Genuino boards. New experiments, will monitor disconnections and will include the context in which Virtual Resources work.

Additionally, autonomy features will be added to Virtual Resources such as self-monitoring.

7.3.2 Evaluation of Blockchain

A future work will evaluate other private blockchain technologies in a Fog environment, for example, Hyperledger, Etherium, and Eris. Future work also includes the study of smart contracts in blockchain to manage events in the IoT network.

With the implementation of private blockchains in a Fog layer, it becomes possible to build CoAP APIs to interact with the blockchains. A future work includes the development of a CoAP API in Go language to communicate Virtual Resources with blockchains.

Cyber currencies will be explored as a mechanism to handle access to IoT networks and monetize services. A future work proposes to monetize the tasks performed in the Fog and IoT networks.

REFERENCES

[1] K. Ashton, "That 'Internet of Things' Thing - RFID Journal," *RFiD J.*, vol. 22, no. 7, pp. 97–114, 2009.

[2] Z. Pang *et al.*, "Design of a terminal solution for integration of in-home health care devices and services towards the Internet-of-Things," *Enterp. Inf. Syst.*, vol. 9, no. 1, pp. 86–116, 2015.

[3] G. Broil, M. Paolucci, M. Wagner, E. Rukzio, A. Schmidt, and H. Hußmann, "Perci: Pervasive service interaction with the internet of things," *IEEE Internet Comput.*, vol. 13, no. 6, pp. 74–81, 2009.

[4] M. Darianian and M. P. Michael, "Smart home mobile RFID-based internet-of-things systems and services," *Proc. - 2008 Int. Conf. Adv. Comput. Theory Eng. ICACTE 2008*, pp. 116–120, 2008.

[5] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 22–32, 2014.

[6] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Trans. Ind. Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[7] J. Rivera and R. Van der Muelen, "Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020," *Gartner*, 2013. [Online]. Available: https://www.gartner.com/doc/2625419/forecast-internet-things-worldwide-. [Accessed: 26-Nov-2015].

[8] A. Klubnikin, "Internet of Things: How Much Does it Cost to Build IoT Solution?" [Online]. Available: http://r-stylelab.com/company/blog/it-trends/internet-of-things-how-much-does-it-cost-to-build-iot-solution. [Accessed: 01-Nov-2016].

[9] 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.

[10] F. Li, M. Vögler, M. Claeßens, and S. Dustdar, "Efficient and Scalable IoT Service Delivery on Cloud.," *IEEE CLOUD*, pp. 740–747, 2013.

[11] S. Nastic, S. Sehic, M. Vögler, H. L. Truong, and S. Dustdar, "PatRICIA - A novel programming model for iot applications on cloud platforms," *Proc. - IEEE 6th Int. Conf. Serv. Comput. Appl. SOCA 2013*, pp. 53–60, 2013.

[12] F. Khodadadi, R. N. Calheiros, and R. Buyya, “A data-centric framework for development and deployment of Internet of Things applications in clouds,” *2015 IEEE Tenth Int. Conf. Intell. Sensors, Sens. Networks Inf. Process.*, no. April, pp. 1–6, 2015.

[13] M. Yuriyama and T. Kushida, “Sensor-cloud infrastructure physical sensor management with virtualized sensors on cloud computing,” *Proc. - 13th Int. Conf. Network-Based Inf. Syst. NBiS 2010*, pp. 1–8, 2010.

[14] 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.

[15] 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.

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

[17] L. M. Vaquero and L. Rodero-Merino, “Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 27–32, 2014.

[18] 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.

[19] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, “Fog Computing: A Platform for Internet of Things and Analytics,” *Big Data Internet Things A Roadmap Smart Environ.*, pp. 169–186, 2014.

[20] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, and H.-Y. Du, “Research on the architecture of the Internet of Things,” no. August, pp. 20–22, 2010.

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

[22] “Internet of Things - Cisco.” [Online]. Available: http://www.cisco.com/c/r/en/us/internet-of-everything-ioe/internet-of-things-iot/index.html. [Accessed: 23-Nov-2016].

[23] J. A. Stankovic, “Research Directions for the Internet of Things,” *Internet Things Journal, IEEE*, vol. 1, no. 1, pp. 3–9, 2014.

[24] J. Caldwell, “Ibm Point of View: Internet of Things Security,” 2015. [Online]. Available: http://toronto.ieee.ca/files/2016/02/TOR-IEEE-IBM-IoT-Jan-28-2016-Final.pdf. [Accessed: 26-Nov-2016].

[25] “What is Internet of Things | Microsoft.” [Online]. Available:

https://www.microsoft.com/en-ca/cloud-platform/internet-of-things. [Accessed: 23-Nov-2016].

[26] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” *Comput. Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[27] P. N. Howard, “Sketching out the Internet of Things trendline | Brookings Institution,” 2015. [Online]. Available: https://www.brookings.edu/blog/techtank/2015/06/09/sketching-out-the-internet-of-things-trendline/. [Accessed: 12-Nov-2016].

[28] T. Sánchez López, D. C. Ranasinghe, M. Harrison, and D. McFarlane, “Adding sense to the Internet of Things: An architecture framework for Smart Object systems,” *Pers. Ubiquitous Comput.*, vol. 16, no. 3, pp. 291–308, 2012.

[29] D. Rose, *Enchanted Objects: Innovation, Design, and the Future of Technology*. Simon and Schuster, 2014.

[30] P. Mell and T. Grance, “The NIST Definition of Cloud Computing (Draft) Recommendations of the National Institute of Standards and Technology,” 2011.

[31] 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.

[32] P. Parwekar, “From Internet of Things towards cloud of things,” *2011 2nd Int. Conf. Comput. Commun. Technol. ICCCT-2011*, pp. 329–333, 2011.

[33] M. Aazam and E. N. Huh, “Fog computing and smart gateway based communication for cloud of things,” *Proc. - 2014 Int. Conf. Futur. Internet Things Cloud, FiCloud 2014*, pp. 464–470, 2014.

[34] C. P. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. Hart, “Enabling multi-tenancy: An industrial experience report,” *IEEE Int. Conf. Softw. Maintenance, ICSM*, 2010.

[35] D. Jacobs and S. Aulbach, “Ruminations on Multi-Tenant Databases,” *BTW Proc.*, vol. 103, pp. 514–521, 2007.

[36] J. M. A. Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray, “Toward a multi-tenancy authorization system for cloud services,” *IEEE Secur. Priv.*, vol. 8, no. 6, pp. 48–55, 2010.

[37] R. Mietzner, T. Unger, R. Titze, and F. Leymann, “Combining different multi-tenancy patterns in service-oriented applications,” *Proc. - 13th IEEE Int. Enterp. Distrib. Object Comput. Conf. EDOC 2009*, pp. 131–140, 2009.

[38] S. Cherrier, Z. Movahedi, and Y. M. Ghamri-Doudane, “Multi-tenancy in decentralised

IoT," *IEEE World Forum Internet Things, WF-IoT 2015 - Proc.*, pp. 256–261, 2016.

[39] 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.

[40] X. Xu, "From cloud computing to cloud manufacturing," *Robot. Comput. Integr. Manuf.*, vol. 28, no. 1, pp. 75–86, 2012.

[41] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[42] Morreale and Anderson, *Software Defined Networking Design and Deployment*. 2012.

[43] J. Chen, X. Zheng, and C. Rong, "Survey on software-defined networking," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9106, no. 1, pp. 115–124, 2015.

[44] K. Kirkpatrick, "Software-defined Networking," *Commun. ACM*, vol. 56, no. 9, pp. 58–65, 2013.

[45] N. M. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Comput. Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[46] S. Distefano, G. Merlino, and A. Puliafito, "Sensing and actuation as a service: A new development for clouds," *Proc. - IEEE 11th Int. Symp. Netw. Comput. Appl. NCA 2012*, vol. 1, pp. 272–275, 2012.

[47] O. Terzo, P. Ruiu, E. Bucci, and F. Xhafa, "Data as a Service (DaaS) for sharing and processing of large data collections in the cloud," *Proc. - 2013 7th Int. Conf. Complex, Intelligent, Softw. Intensive Syst. CISIS 2013*, pp. 475–480, 2013.

[48] S. Alam, M. M. R. Chowdhury, and J. Noll, "SenaaS: An event-driven sensor virtualization approach for internet of things cloud," *2010 IEEE Int. Conf. Networked Embed. Syst. Enterp. Appl. NESEA 2010*, pp. 1–6, 2010.

[49] M. Jutila, "An Adaptive Edge Router Enabling Internet of Things," *IEEE Internet Things J.*, vol. 4662, no. c, pp. 1–1, 2016.

[50] A. R. Biswas and R. Giaffreda, "IoT and Cloud Convergence: Opportunities and Challenges," *2014 IEEE World Forum Internet Things*, pp. 375–376, 2014.

[51] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *Www.Bitcoin.Org*, p. 9, 2008.

[52] "Blockchain Info." [Online]. Available: https://blockchain.info/. [Accessed: 02-Nov-2016].

[53] V. Buterin, "On Public and Private Blockchains - Ethereum Blog," 2015. [Online]. Available: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/. [Accessed: 20-Nov-2016].

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

[55] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," *Proc. Symp. Oper. Syst. Des. Implement.*, no. February, pp. 1–14, 1999.

[56] W. Assumptions, "Scheduling: Introduction." [Online]. Available: http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched.pdf. [Accessed: 08-Oct-2016].

[57] 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].

[58] "IBM Blockchain." [Online]. Available: https://www.ibm.com/us-en/marketplace/cloud-based-blockchain-platform. [Accessed: 27-Nov-2016].

[59] V. Buterin, "A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM," 2014.

[60] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, Sep. 1997.

[61] E. Newcomer and G. Lomow, *Understanding SOA with Web services*. Addison-Wesley, 2005.

[62] D. Linthicum, "Chapter 1: Service Oriented Architecture (SOA)." [Online]. Available: https://msdn.microsoft.com/en-us/library/bb833022.aspx. [Accessed: 17-Nov-2016].

[63] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-based internet of things: Discovery, query, selection, and on-demand provisioning of web services," *IEEE Trans. Serv. Comput.*, vol. 3, no. 3, pp. 223–235, 2010.

[64] 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.

[65] A. P. Castellani, N. Bui, P. Casari, M. Rossi, Z. Shelby, and M. Zorzi, "Architecture and protocols for the internet of things: A case study," in *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops, PERCOM Workshops 2010*, 2010, pp. 678–683.

[66] I. Ishaq, J. Hoebeke, J. Rossey, E. De Poorter, I. Moerman, and P. Demeester, "Enabling the web of things: facilitating deployment, discovery and resource access to IoT objects using embedded web services," *Int. J. Web Grid Serv.*, vol. 10, no. 2/3, p. 218, 2014.

[67] R. T. Fielding, "Architectural Styles and the Design of Network-based Software

Architectures," University of California, Irvine, 2000.

[68] Y. Xue and R. Deters, "Resource Sharing in Mobile Cloud-computing with Coap," *Procedia Comput. Sci.*, vol. 63, no. Euspn, pp. 96–103, 2015.

[69] F. Khodadadi, A. V Dastjerdi, and R. Buyya, "Simurgh: A framework for effective discovery, programming, and integration of services exposed in IoT," *Recent Adv. Internet Things (RIoT), 2015 Int. Conf.*, no. April, pp. 1–6, 2015.

[70] A. Elmangoush, T. Magedanz, A. Blotny, and N. Blum, "Design of RESTful APIs for M2M services," *2012 16th Int. Conf. Intell. Next Gener. Networks, ICIN 2012*, pp. 50–56, 2012.

[71] 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.

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

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

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

[75] 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.

[76] 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].

[77] C. Esposito, S. Russo, and D. Di Crescenzo, "Performance assessment of OMG compliant data distribution middleware," *Parallel Distrib. Process. 2008. IPDPS 2008. IEEE Int. Symp.*, pp. 1–8, 2008.

[78] 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].

[79] V. Lampkin, "What is MQTT and how does it work with WebSphere MQ? (WebSphere and CICS Support Blog)," no. 43645, pp. 20–22, 2012.

[80] 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.

[81] 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.

[82] 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].

[83] 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].

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

[85] 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.

[86] 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.

[87] 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.

[88] N. Bui and M. Zorzi, "Health care applications: a solution based on the internet of things," *Proc. 4th Int. Symp. Appl. Sci. Biomed. Commun. Technol.*, p. 131, 2011.

[89] V. M. Rohokale, N. R. Prasad, and R. Prasad, "A cooperative Internet of Things (IoT) for rural healthcare monitoring and control," *Wirel. Commun. Veh. Technol. Inf. Theory Aerosp. Electron. Syst. Technol. (Wireless VITAE), 2011 2nd Int. Conf.*, pp. 1–6, 2011.

[90] A. Dohr, R. Modre-Opsrian, M. Drobics, D. Hayn, and G. Schreier, "The Internet of Things for Ambient Assisted Living," *2010 Seventh Int. Conf. Inf. Technol. New Gener.*, pp. 804–809, 2010.

[91] T. Fredrich, "What is REST?" [Online]. Available: http://www.restapitutorial.com/lessons/whatisrest.html. [Accessed: 27-Oct-2016].

[92] Z. Shelby, "Constrained RESTful Environments (CoRE) Link Format," pp. 1–22, 2012.

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

[94] "A Tour of Go." [Online]. Available: https://tour.golang.org/concurrency/1. [Accessed: 06-

Dec-2016].

[95] 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].

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

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

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

[99] A. Brasetvik, "Elasticsearch as a NoSQL Database," 2013. [Online]. Available: https://www.elastic.co/blog/found-elasticsearch-as-nosql. [Accessed: 15-Oct-2016].