Novel Security Models for IoT-Fog-Cloud Architectures in a Real-World Environment.

A Dissertation

Presented in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

by

Mohammed Ahmed Aleisa

Approved by

Major Professor: Frederick Sheldon, Ph.D.

Committee Members: Abdullah Abuhussein, Ph.D.; Robert Rinker, Ph.D.; Xiaogang Ma, Ph.D.

Department Administrator: Terence Soule, Ph.D.

May 2022

Abstract

The emergence of the Internet of things (IoT) has generated demand for computation performed at the ‘edge’ of the network. With companies being increasingly challenged to collect and send data collected from IoT devices to the cloud, this increases the need for fog computing. Fog computing is an intermediate computing layer that has emerged to address the latency issues of cloud-based Internet of things (IoT) environments. As a result, new forms of security and privacy threats are emerging. These threats are mainly due to the huge number of sensors, as well as the enormous amount of data generated in IoT environments that needs to be processed in real time. These sensors send data to the cloud through the fog computing layer, creating an additional layer of vulnerabilities. In addition, the cloud by nature is vulnerable because cloud services can be located in different geographical locations and provided by multiple service providers. Moreover, cloud services can be hybrid and public, which exposes them to risks due to their infinite number of anonymous users. This research proposed two architectures of cloud-based IoT environments and three analysis methods. The two proposed architectures are evaluated based on the three analysis methods to show the efficacy of the fog layer in different experiments in a real-world environment by examining performance metrics on the cloud and fog layers using different numbers of IoT devices. To overcome the security challenges between the IoT layer and fog layer and, thus, meet the security requirements, this research also proposed a fine-grained data access control model based on the attribute-based encryption of the IoT–Fog–Cloud architecture to limit the access to sensor data and meet the authorization requirements. In addition, this research proposed a blockchain-based certificate model for the IoT–Fog–Cloud architecture to authenticate IoT devices to fog devices and meet the authentication requirements. We evaluated the performance of the two proposed security models using AWS cloud metrics to determine their efficiency in real-life experiments of the IoT–Fog–Cloud architecture.

Acknowledgements

First and foremost, I must praise and thank my Almighty God (Allah) for his blessings. This dissertation could not be done without the Allah blessings. He alone helps me to proceed and succeed in my life. Then, I would like to thank my sponsors, the government of Saudi Arabia and Majmaah University for supporting me to pursue the Ph.D. degree.

Moreover, I would like to express my sincere appreciation to my respected major advisor Prof. Frederick Sheldon and Prof. Abdullah Abuhussein, for their invaluable support, motivation in my Ph.D. journey.

I would like to convey my acknowledgements to the rest of my dissertation committee members for their support: Prof. Abdullah Abuhussein, Prof. Robert Rinker, and Prof. Xiaogang Ma.

My appreciation also goes to all faculty members, and colleagues in the Department of Computer Science at University of Idaho.

Furthermore, I would like to express my gratitude to my family, parents (Aljawhara Almukhadab and Ahmed Aleisa), wife (Mona Altuwayjiri), son (Ahmed), my brothers and sisters for their trust, encouragement, support, help and love in this journey.

Dedication

This work is dedicated to my parents, Aljawhara Almukhadab and Ahmed Aleisa. I am not able to express my appreciation to them for all of the sacrifices that they have made on my behalf. Their invaluable unconditional and continuous love, care, and support since the first day of my life till where I am today and for the rest of my life is the real blessing that cannot be perceived by all other blessings. Without their prayers, no happiness could touch my heart. In memory of my father Ahmed Aleisa death, may Allah forgive him and have mercy on him, and give him strength and pardon him, and be generous to him. May Allah bless him and grant him the highest level of paradise. Also, this work is dedicated to my beloved family members including my brothers, my sisters, and my relatives.

This work is particularly dedicated to my dear wife, Mona Altuwayjiri, who has spent years supporting me, encouraging me, and taking care of our son Ahmed Aleisa. You have been a constant source of support and encouragement during the challenges of graduate school and difficulties of life.Your encouragements are the fuel that drives me straight to achieve my goals in this life.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

M. Aleisa, A. Abuhussein, and F. T. Sheldon, "Access Control in Fog Computing: Challenges and Research Agenda," *IEEE Access*, vol. 8, pp. 83986–83999, May 2020, DOI: 10.1109/ACCESS.2020.2992460

1.1 Problem Background

1.1.1 What is Fog Computing?

Fog computing is defined as an intermediate layer between the cloud and IoT devices [1]. Figure 1.1 presents a classic fog computing-aided IoT environment. Fog computing extends the cloud services to the edge of the network, near IoT devices, to reduce the latency and network congestion. Low latency is a desired quality in today's applications, such as emergency responses in the medical domain, and fog computing guarantees low latency by providing real-time processing capabilities for the transferred data [1]. According to Cisco [2], fog computing is the place where IoT data is analyzed near the IoT devices that generate and process data. A typical fog computing environment consists of nodes connected to IoT devices. These nodes are referred to as fog nodes.

Fog nodes can be deployed anywhere within the network connection. Fog devices can be any device that has computing, storage, and network connectivity. According to NIST [3], fog computing is an intermediate layer that allows global access to several IoT devices. The environment of fog computing enables the deployment of distributed applications and services [4], [5].

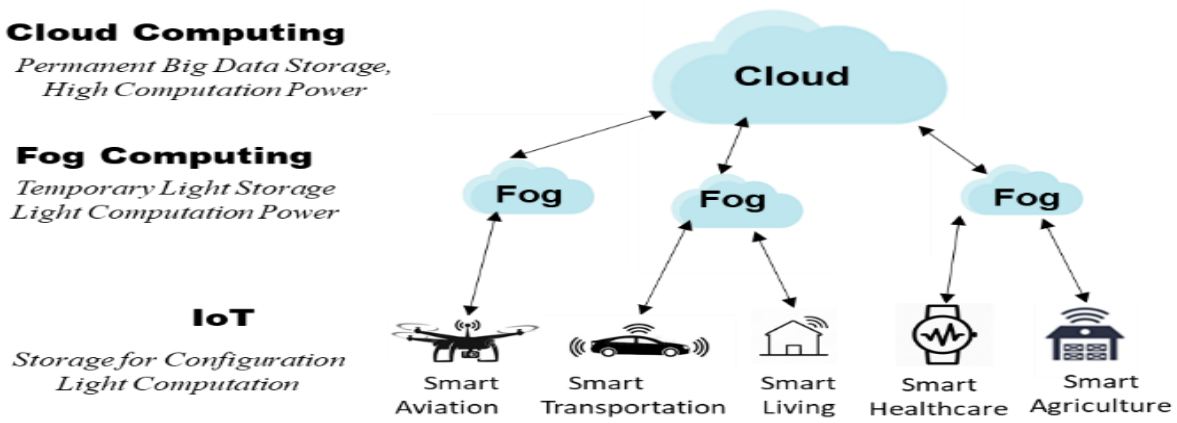


FIGURE 1.1 **Fog Computing Environment**

1.1.1.1 Why Fog Computing?

The fog computing layer between the cloud and IoT devices has valuable functionalities:

1) Move cloud content closer to IoT

By bringing cloud content closer to IoT devices, fog computing solves the delay issues in time-sensitive applications in which decisions must be made in a timely manner.

2) Save network bandwidth

Since not all data should be transferred to the cloud for processing, using a fog layer between the cloud and the IoT devices helps to save network bandwidth. In this case, fog computing can better handle managing and controlling data processing, transfer, privacy, and security. This will also reduce operating expense.

3) Bring storage closer to IoT

This functionality of fog computing is essential because it places temporary storage closer to IoT devices which have limited storage capability. Fog computing serves as a temporary storage location for the data aggregated from IoT devices, whereas the cloud stores the data permanently.

4) Bring computation power closer to IoT

In the cases in which data gathered from IoT devices require immediate processing, fog computing can serve as a processing facility that is located closer than the cloud. Fog computing, in this case, will take care of quick and small workloads. Big data analytics will still be handled by the cloud.

5) Protect IoT data

Although fog computing expands the cloud-fog-IoT architecture attack surface, this additional computing layer, with its storage and computing capability, can be utilized to host and run automated monitors to detect threats to IoT. It can also be used to fine grain AC to avoid over- and underexposure of authorization.

1.1.1.2 Fog Computing Layers

Several fog computing architectures have been proposed in [1], [6]–[9]. Commercial architectures of fog computing have also been developed for commercial fog devices [8]. The architecture of a standard fog computing environment consists of several layers:

1) Physical Layer

This layer represents all fog hardware devices that send and receive data to and from IoT devices. These devices can be virtual or physical devices, such as virtual and physical network routers.

2) Monitoring Layer

This layer is responsible for detecting and logging performance and security-related flaws in IoT devices and/or fog nodes. For example, this layer can select a fog node based on criteria such as throughput, congestion, etc., and detect malicious activities against fog nodes or IoT devices.

3) Processing Layer

This layer is responsible for analyzing and filtering the data collected from IoT devices. As the number of IoT devices increases, the amount of data also increases. Therefore, processing this enormous amount of data can be challenging. Fog nodes usually have light-to-medium-weight processing capability. Intensive processing is usually performed in the cloud.

4) Storage Layer

This layer is responsible for storing data generated from IoT devices. IoT devices have limited storage capability, so fog computing provides a temporary storage service for IoT data. Long-term storage and storing historical data are usually handled by the cloud.

5) Security Layer

This layer maintains the security objectives (i.e., confidentiality, integrity, and availability) in the fog nodes. The security layer is where all controls and measures are applied to detect and prevent threats, as well as to respond to security incidents. For example, encryption and decryption of data received and sent by fog devices is a security measure handled by the fog computing layer to maintain confidentiality and is considered a prevention technique. In another example, fog nodes may be used to balance the load directed to IoT devices based on throughput or congestion in cases of denial of service (DoS) attacks. The objective of this security measure is to maintain availability by responding to a DoS incident.

6) Application Layer

This layer includes the applications and protocols responsible for networking (such as routing) and load balancing (such as routing tables and Hypertext Transfer Protocol (HTTP) and MQ Telemetry Transport (MQTT) protocols) [10].

1.1.1.3 Fog-Aided IoT Process Phases

In fog-aided IoT environments [11], [12], a subscribed IoT node may request to join a fog network before it can collect and publish data. This model is known as the publishing/subscribing model. In another model, a fog-aided IoT network finds an IoT node and requests to add it to the network in order to collect and publish data. This is known as the request/response model. When the IoT node joins, it is assigned network resources and can then start communicating and operating as a component of the IoT environment. Figure 1.2 shows a three-phase process in a fog-aided IoT environment. The process assumes a fresh start of an unsubscribed node. Thus, initially, a node needs to be subscribed on-demand

before it can join the fog network and transfer data within the network (by aggregating and publishing data, for example).

1) Subscribe Phase

When a new IoT device wants to connect to a fog device, the fog device captures the new IoT device that needs services and registers it to the requested services.

2) Join Phase

Fog devices detect new IoT devices that request services, and each IoT device is asked to show its identity before joining the fog network. To avoid security issues from malicious IoT devices, the new IoT devices should be authenticated first. After that, the authenticated devices may request access to fog devices to obtain authorization for the services provided. This is where AC models are applied. When the authentication and authorization operations are complete, different groups of IoT devices become connected to the corresponding fog devices with access to network resources and service may begin.

3) Transfer Phase

In this phase, fog devices start aggregating data from IoT devices and/or send tasks to them. When the amount of data collected is huge, fog devices filter data received from IoT devices before processing it or sending it to the cloud. Load balancing strategies can be used to send workload to free fog nodes when a fog node is overwhelmed with tasks.

Fog computing receives data from IoT devices and then processes it or sends it to cloud storage. Fog computing can interact with all three types of cloud services (Software as a Service [SaaS], Platform as a Service [PaaS], and Infrastructure as a Service [IaaS]) [3]. To the best of our knowledge, there is no standard architecture for fog computing. Several commercial platforms, including ParaDrop and Cloudlet, have been proposed [13]. ParaDrop is a fog computing platform based on wireless routers using operating system-level virtualization. A cloudlet is a mobility enhanced small-scale cloud infrastructure that is located at the edge of the Internet and can act as a fog layer.

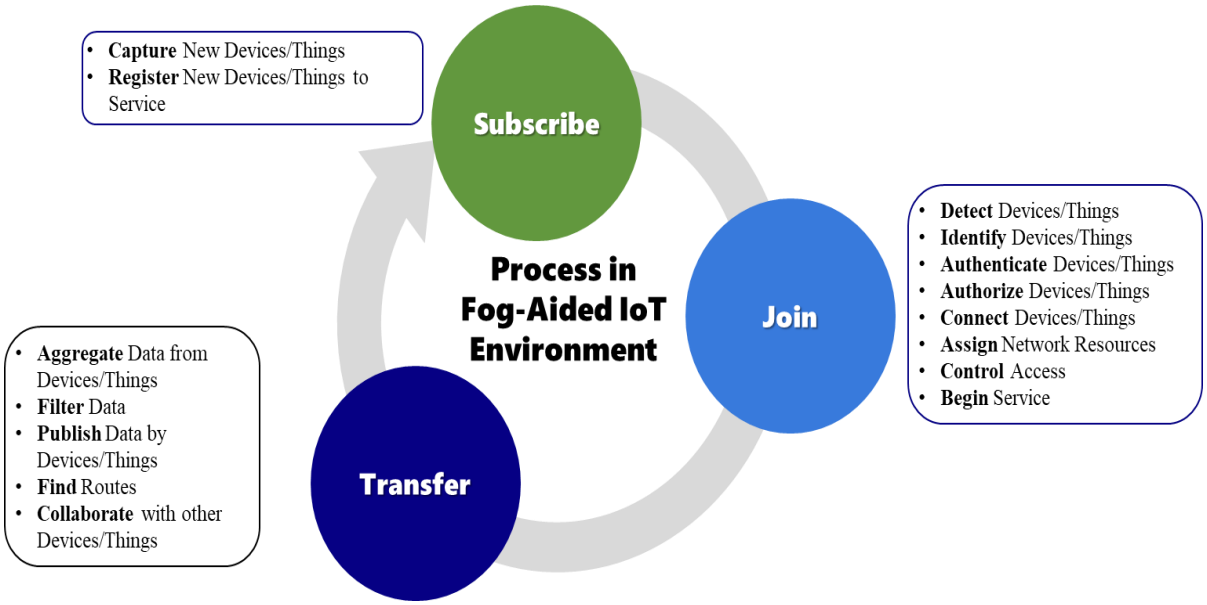


FIGURE 1.2 **The subscribe, join, and transfer phases in a fog-aided IoT environment with events occurring in each phase.**

There are several areas where fog computing can be utilized, such as smart cities, smart vehicles, smart grids, and mobile healthcare [14]. Figure 1.3 shows a taxonomy of fog computing applications. In the healthcare domain, data are generated by thousands of sensors that require low latency and real-time processing demand. Fog nodes can be a feature to support the scalability of patient monitoring.

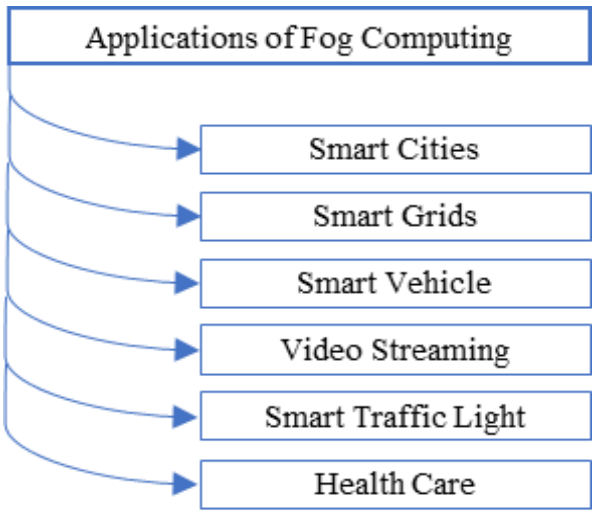


FIGURE 1.3 **Fog Computing Applications.**

1.1.1.4 Characteristics of Fog Computing

As the number of IoT devices increases, handling the data generated by IoT devices and transferring it to the cloud may turn out to be challenging. Therefore, fog computing emerged to address these challenges by processing the data at the edge of the network (close to the IoT devices), which results in reduced latency. Table 1.1 depicts the differences between fog computing and cloud computing in terms of the following common characteristics:

1) Latency

Some transfer delay between IoT devices and the cloud can be tolerated, depending on the requirements and the nature of the application. However, for medical applications or in case of emergency events, the data are very time-sensitive. The latency will be high in the cloud because the distance between IoT devices and the cloud is long. Thus, computing the data in the cloud will cause a high latency. However, fog computing reduces that latency by bringing data to the edge of the network and closer to end users to meet the high processing demand [15].

2) Scalability

As the number of IoT devices increases, it is difficult for the cloud to handle the heavy computation and bandwidth overhead of these devices. Fog computing can solve this issue by distributing serval fog nodes that can reduce the heavy computation and support hierarchal scalability when the number of IoT devices increases [15].

3) Location-Awareness

Since the cloud is far from IoT devices, sending location information may push heavy workloads toward the cloud when the number of IoT devices is high. Therefore, having fog nodes closer to the IoT devices to manage and control traffic sent to the cloud and to support geographic location becomes necessary [15].

4) Mobility

Fog computing supports the feature of mobility. Per Cisco, any device that has computing, storage, and network connectivity can be a fog node [2]. In fog computing, a fog node can be any mobile device, such as smart vehicles or smart phones, or any static device, such as traffic cameras in smart city devices [16].

Table 1.1 Characteristics that differ between fog computing and cloud computing.

Characteristics	Fog Computing	Cloud Computing
Architecture	Decentralized	Centralized
Latency	Low	High
Location Awareness	Yes	No
Mobility	Supported	Limited
Geographic location	Yes	No
Delay	Low	High
Scalability	High	Limited
Deployment	At the edge of the network	Network Core

5) Geographic Location

A fog computing layer may consist of a number of distributed fog nodes that are deployed in different locations [6]. As previously mentioned, fog computing supports the feature of geographic location, and distributed fog nodes can track the locations of IoT devices to support their mobility. The applications and services of fog computing are decentralized and can process and store data from end devices. Therefore, the massive amounts of data generated by IoT devices will be processed faster in decentralized fog computing than in centralized cloud computing.

6) Heterogeneity

The fog computing layer consists of two components: the physical node and the virtual node. Physical nodes include physical sensors and routers, while virtual nodes include virtual sensors and virtual load balancers. These physical or virtual nodes may have different operating systems and may be used to run

different applications. Therefore, heterogeneity in fog nodes is desirable to make these devices interoperable [6], [16].

7) Bandwidth

Fog computing can process the data created by IoT devices at the edge of the network, close to the end user, rather than sending it to the cloud. Therefore, fog computing efficiently saves the bandwidth by computing and storing the data locally. As the number of IoT devices increases, more data may be generated and collected. Therefore, an architecture of distributed fog nodes addresses this problem by computing the enormous amount of data locally instead of transmitting it to the cloud. This, in turn, reduces network traffic and saves bandwidth [16].

1.2 Problem Definition

First, the 2016 Dyn cyberattack that occurred on October 21, 2016 and disrupted internet across internet service across Europe and US. The 2016 Dyn cyberattack was a series of distributed denial-of-service attacks (DDoS attacks) that targeted Internet of Things-enabled (IoT) devices such cameras, residential gateways, and baby monitors. Many services were affected by the 2016 Dyn cyberattack [123]:

Airbnb	CrunchBase	HBO	Overstock.com	RuneScape
Amazon.com	DirecTV	Heroku	PayPal	SaneBox
Ancestry.com	The Elder Scrolls Online	HostGator	Pinterest	Seamless
The A.V. Club	Electronic Arts	iHeartRadio	Pixlr	Second Life
BBC	Etsy	Imgur	PlayStation Network	Shopify
The Boston Globe	FiveThirtyEight	Indiegogo	Qualtrics	Slack
Box	Fox News	Mashable	Quora	SoundCloud
Business Insider	The Guardian	National Hockey League	Reddit	Squarespace
CNN	GitHub	Netflix	Roblox	Spotify
Comcast	Grubhub	The New York Times	Ruby Lane	Starbucks

Second, although fog computing offers promising solutions to many of the performance and security problems of IoT, it is confronted with various security and privacy risks. For instance, while the fog computing is crucial for spreading risks across distributed fog node, it also has untoward effect of increasing the attack surface. What makes the situation even worse is that in fog computing devices interact with devices only. In other words, the fog nodes receives IoT data from sensors and sends it to the cloud and vice versa. This means, no human interaction is involved in the communication. Although this can be considered as an advantage because these interacting devices do not even have screens or an on-device user interface, which means smaller attack surface, yet this can lead do failures and/or targeted attacks that cannot be easily detected and deterred. There are plenty of other security and privacy issues in fog aided IoT that deserve our genuine attention.

In this dissertation, we aim to better understand fog aided IoT environments in order to pave the way for further research to address interesting confidentiality, integrity, and availability violations.

1.3 Motivation

First, there is a lack of real-life implementations of the many theoretical studies in research and academia. Although simulation-based experiments provide easy access to practical results about the performance of computing systems, observations and research outcomes may not be generalizable to all scenarios due to the variety in IoT platform providers and device manufacturers; their different implementations, service specifications, and configurations; and differences in network architectures and protocols. Therefore, in order to develop a profound and general insight into the tradeoffs involved in a particular system, it is important to use real IoT platforms built on top of a real-world network (i.e., Internet) when obtaining analytical results for the performance of fog aided IoT implementations. In addition, it would be interesting to explore the performance differences of fog implementations interacting with different commercial IoT platforms, such as Amazon IoT and Azure IoT.

Second, due to the diverseness that fog-aided IoT environments enjoys and the lack of consensus among practitioners and hobbyists on a standard fog computing implementation, there is a lack of

resources that show how to implement an efficient fog-aided IoT system. Most of the implementations available are either domain specific, complex, or too abstract to be useful in all scenarios.

Third, The IoT-Fog-Cloud field is rapidly evolving because security requirements and objectives are changed. Therefore, current solutions are not sufficient.

1.4 Contribution

First, this research proposes two architectures of cloud-based IoT environments in a real-world environment. In first architecture, we used a fog layer between IoT devices and the cloud, whereas in the second, IoT devices published data directly to the cloud. Second, we developed a methodology of each IoT devices to provide more accurate, consistent, and real results about the environment performance. Third, we proposed three analysis methods to perform the experiments on both architectures and evaluated them based on the three analysis methods. Through conducting several experiments based on the three analysis methods, we found that the first architecture outperforms the second architecture in terms of performance. Overall, with an increased number of IoT devices, the first architecture outperforms the second architecture. From a security standpoint, authenticating IoT devices at the fog layer provides more flexibility in adding sophistication to the authentication and authorization process. Therefore, to overcome the security challenges between the IoT layer and fog layer and, thus, meet the security requirements, authentication and authorization models must be proposed. Fourth, this research proposes a fine-grained data access control model based on the attribute-based encryption of the IoT–Fog–Cloud architecture to limit the access to sensor data and meet the authorization requirements. The encryption-based access control is fined grained and hard to break. However, the encryption-based access control increases the demand of resources utilization and heavy computation. To solve this, fog computing is used to outsource the heavy computation. Fifth, this research proposes a blockchain-based certificate model for the IoT–Fog–Cloud architecture to authenticate IoT devices to fog devices and meet the authentication requirements. The blockchain-based certificate model defines

as a decentralized. Therefore, blockchain supports fog decentralization and offers encryption and validation. Also, it is hard to break and can be traceable and irreversible. Although blockchain transactions take time to process and validate, the number of fog devices is not expected to be huge as the number of IoT devices. Therefore, the number of transactions will be also low.

1.5 Structure of the Dissertation

The dissertation has five remaining chapters: the second chapter is literature review; the third chapter is the empirical study of analyzing the performance of the two Cloud-Based IoT Implementations; the fourth chapter is the scientific experiments of examining performance in Fog-Aided Cloud-Centered IoT in a Real-World Environment.; the fifth chapter is the novel security models for IoT–Fog–Cloud architectures in a Real-World Environment. The sixth chapter is the main conclusions and future works.

Chapter 2: Literature Review

M. Aleisa, A. Abuhussein, and F. T. Sheldon, "Access Control in Fog Computing: Challenges and Research Agenda," *IEEE Access*, vol. 8, pp. 83986–83999, May 2020, DOI: 10.1109/ACCESS.2020.2992460

2.1 Introduction

Cloud computing provides IoT (Internet of things) environments with a facility for computation and storage. However, cloud computing requires a high latency due to its distance from the end user [17]. Additionally, the data generated from IoT devices takes time to be computed in the cloud. As the number of IoT devices increases, the amount of data generated will also increase. This huge amount of data aggregated from devices located far away from the cloud must be transferred with low latency. To solve this issue, fog computing emerged.

Fog computing serves as a middle layer between cloud and IoT devices to solve the problem of high data transfer latency. To meet the high processing demand, the huge number of sensors in IoT environments send data through fog nodes rather than directly to the cloud. Smart cities and smart grids are examples of systems in which fog computing can be found between the smart devices and the cloud [18], [19]. This additional layer (i.e., fog computing) can introduce new vulnerabilities since it expands the attack surface on which threats such as data loss and breaches can occur [20]. In addition, several threats, such as malicious fog nodes [21], man in the middle attacks [22], malicious insider threats [20], and denial of service attacks [20], arose in fog computing environments. For instance, in fog computing environments, attackers may seek infinite processing or storage in fog devices, which prevents users from accessing fog device(s) [23].

Access control (AC) is one of the crucial defense frontlines to maintain users' security and privacy, as well as to protect data and services from unauthorized access. Due to the increase in the number and type of threats, it is essential to have effective AC models in fog computing environments. Cloud computing and fog computing are being used in many domains to provide support to IoT

environments. This is known as cloud-fog-IoT architecture. To ensure the appropriateness of the AC strategies in the cloud-fog-IoT architecture, it is important to identify the requirements of the application for which this architecture is used. Application requirements in fog computing include, but are not limited to, scalability, mobility, and heterogeneity [6]. Thus, it is important to select AC models that meet these fog application requirements. Moreover, choosing one of the AC strategies over the others can have a negative impact. For example, it may significantly increase the computation overhead in fog nodes due to the multiple operations involved in AC models, such as file encryption, ciphertext decryption, and distribution of attributes [24]–[26]. On the other hand, using more than one AC model in fog nodes can cause additional heavy computation on fog nodes due to the heavy operations used in controlling access. Therefore, outsourcing part of the operations when implementing AC models for fog nodes becomes crucial. The aforementioned reasons demonstrate the need for dynamic and more efficient AC models. It is also important to appropriately select AC models to protect the cloud-fog-IoT architecture. In this paper, we survey AC models in fog computing, present their challenges, and identify gaps for future research.

The remainder of this paper is structured as follows: In sections 2 and 3, we discuss fog computing and AC comprehensively. In section 4, we present the state of the art in the field of AC in fog computing. In section 5, we discuss some security and privacy issues related to AC in fog computing. In section 6, gaps in the field are identified and discussed. Finally, we conclude this work in section 7.

2.2 Access Control Overview

AC is based on a data access policy (e.g., HIPAA [27]) that determines what privileges are granted to which roles within the various operational scenarios. In other words, the user should first be authenticated to access the system. Then, the user can request access to the system resources and be authorized by the system administrator [28]. There are multiple AC models, such as Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role-Based Access Control (RBAC), and Attribute-Based Access Control (ABAC) [28]. Figure 5 shows a taxonomy of AC models in fog

computing. Attribute-Based access control (ABAC) could be an appropriate model to deal with fog computing, as the owner of the data in the fog layer can define the AC policies for users to achieve the authorization [8]. Here, we summarize AC models:

2.2.1 Access Control Models:

1) Attribute-based Access Control (ABAC)

One of the most well-known AC models is attribute-based access control (ABAC) [28]. This model has three key elements: attributes, a policy model, and an architecture model. Attributes are features that define a user, a resource, or an environmental condition. In fog computing environments, there are four components that interact with fog devices: IoT devices, the data owner, users, and attribute authority. When fog nodes receive data from IoT devices for high processing demand, users may want to access the data. Therefore, a set of attributes will be provided to the users and resources to grant the authorization for access. Attributes can be a username, a job, a resources owner, and/or an environment's time or date created or last accessed. AC policies are defined by the owner of the data, which could be an organization or an individual. AC policies are rules specified by the owner of the data within the organization. These rules can be defined based on users' behavior [28].

2) Discretionary Access Control Model (DAC)

This AC model controls access based on the identity of the users who request the access. Any authorized entity can grant access rights, such as read, write, and/or view, to others. This model is less secure and known to cause management overhead in the environment of fog computing [28].

3) Role-based Access Control Model (RBAC)

In this model, AC is defined based on the role of the user in the organization, such as students, faculty, and staff in universities and colleges. Therefore, access rights are assigned to the roles instead of the users. Some users may have more than one role within the organization. In this situation, AC policies for each role are applied and may overlap. In this model, AC rights that the owner of data would grant to users are view, read, update, and/or write. When a user requests to access data, the user's role

is compared to the access policy that is predefined by the owner of the data and access is granted accordingly [28].

4) Access Policy Access Control Model (APAC)

A policy is a set of rules that is pre-defined by the data owner. The owner of the data can be an organization or an individual that sets up the policy for access to their resources. These rules may consist of authorized behaviors that are defined by the owner of the data and meet the data owner's security objectives. Each user has one or more identifying attributes. Access policies consist of attributes that define every user's accessibility to the resources. These attributes are written in access policy in multiple levels and may be connected by a logical expression such as AND or OR [28].

5) Identity-based Access Control (IBAC)

There are three elements that interact in AC: subject, object, and access rights. The subject is an active entity and can be a user or an application requesting access to a resource(s). The object is a passive entity and can be a resource for which access needs to be controlled. The access rights are the method by which a subject may access an object. The access rights consist of several operations, such as read, write, delete, and search. This model manages any access by a subject to an object through access rights. This model is based on the identity of the subject and an object identifier [29].

6) Task-based Access Control (TBAC)

In this model, a task is considered a subrole for a subject. When the task of a subject satisfies the roles involved in the task, the subject is granted access to an object [29].

7) Rule-based Access Control (RBAC)

In this model, rules are defined such that a subject can access an object through satisfying these rules. As in DAC, access control lists (ACL) are associated with each object and include access rights of a subject to gain access to an object. When a user tries to access a resource, the system checks the rules in the ACL for that resource. Then, if rules are satisfied, the user gains access to a resource. For example, students may access a course website only at a certain time of day [30].

8) Mandatory Access Control (MAC)

This model manages access based on comparing security labels with security clearances. The security labels are allocated to each object, such as a resource, and indicate how important the system resources are. The security labels consist of two components: (1) a classification component (e.g., top secret) and (2) a category component which declares the level to which the object is available. The subject, which is a user, has a classification and a category. When a user tries to access a resource using MAC, the system checks the classifications and category of the user and compares it to the security labels of the resource. Then, if the classification and the category of the user match the security labels of a resource, access is permitted. Otherwise, access is denied [30].

2.2.2 Access Control Requirements in Fog Computing Environments

Although we thoroughly surveyed the AC models, some of the AC models mentioned earlier are not used for fog computing. Thus, we identified the requirements for adopting and applying AC in fog computing. The requirements necessary to maintain efficiency in fog computing are as follows:

1) AC models use operations such as building access policy, which may cause computational overhead on the side of the IoT device. Since IoT devices have limited resources, the computational overhead can be taken care of by the closest fog node [8].

2) AC models should support the creation, deletion, and revocation of an AC policy. For example, what techniques should be used to update the system when policy is revoked [8]? In fog-based environments, the emerging fog layer further exposes the user data and applications since it is an additional attack point. This necessitates the application of an AC model that enables policy creation, deletion, and revocation at the cloud, fog, and IoT device levels.

3) Since the IoT devices are resource-limited, it is essential to restrict some resources from being accessed when the number of IoT devices increases [8].

4) AC models should support the revocation of attributes. This is important (1) to prevent the user whose attribute is revoked from being able to decrypt the new encrypted data (i.e., backward security) and (2) to enable the newly subscribed user whose attribute is satisfied and valid to decrypt the newly published encrypted data (i.e., forward security) [31].

5) Since the fog layer is supposed to be close to the IoT devices to solve the latency issue, the time required to decide whether the access policy is satisfied should be low and reasonable. If the user's attribute satisfies the access policy, the policy decision's response time should be low. In addition, execution cost, networking cost, and deployment cost of the fog-based AC models should also be reduced since the fog layer is close to the IoT devices [18], [32].

6) As the number of IoT devices increase, the fog nodes will also increase. Therefore, multiple attribute authorities are needed to support scalability of fog nodes and IoT devices. Thus, AC models should support multiple attribute authorities [33].

7) Selecting an AC model in fog computing depends on the application requirements, which can impact the computation overhead. Some of the AC models mentioned earlier can be encryption-based, which results in additional operations (e.g., encryption and decryption). Thus, it is important to decide whether data encryption is required and mandated as an application requirement before selecting the AC model [6].

8) As the number of IoT devices increases, the data generated by these IoT devices will also increase. It is important to utilize an AC model that gives data owners more flexibility to underexpose/overexpose their data. Therefore, fine-grained AC becomes an essential requirement [34].

2.3 State of the art in Fog Access Control

Attribute-based encryption (ABE) has been extensively studied [35], [36] and used in several schemes. ABE is a type of public encryption that is dependent on the attributes of the users and resourced

accessed (i.e., APAC). Users who want to obtain access obtain secret keys that reflect their attributes, and the ciphertext has attributes that are encrypted according to policies created by data owner. Then, if the user's attributes satisfy the attributes embedded in the ciphertext, the user can decrypt the ciphertext and obtain access to the plaintext. In healthcare applications, a patient can define access policies by using logical expressions such as AND or OR and encrypt their personal health record according to the defined policies. The doctor can decrypt the personal health record if the doctor's attributes satisfy the access policies embedded in the ciphertext. There are two types of attribute-based encryption (ABE): Ciphertext-Policy Attribute-Based Encryption (CP-ABE) and Key Policy Attribute-Based Encryption (KP-ABE), as shown in Figure 2.1

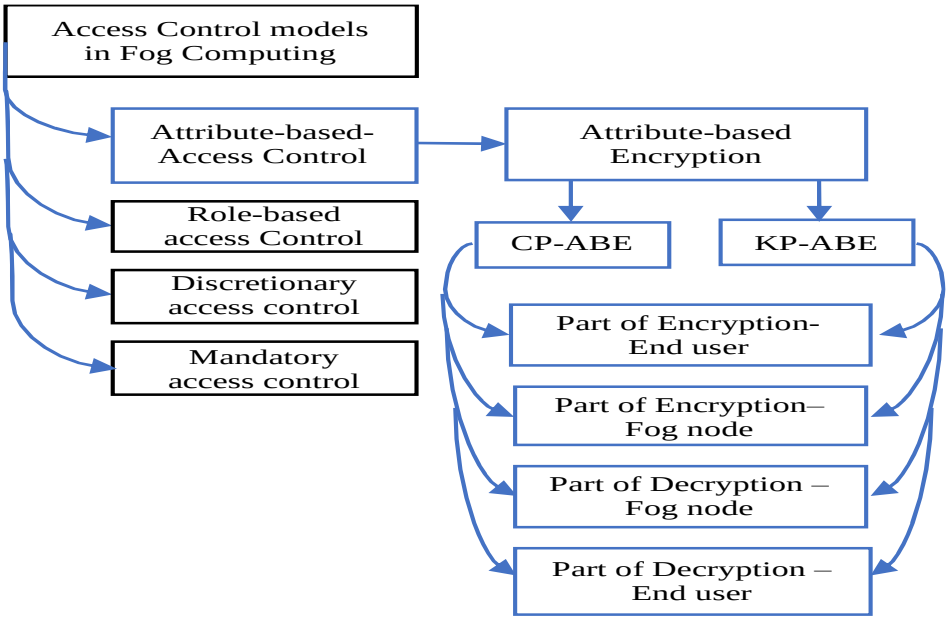


FIGURE 2.1 Access Control Models in Fog Computing

The existing schemes CP-ABE and KP-ABE have a number of operations to handle encryption and decryption, which cause a heavy computation overhead due to the resource constraints at the end users' side. Figure 4 shows a system model of applying ABE in fog computing. To decrease the heavy computation at the end users' side, serval works that outsource the encryption and decryption operations to the near fog nodes have been proposed, as shown in Figure 2.2 and Table 2.1.

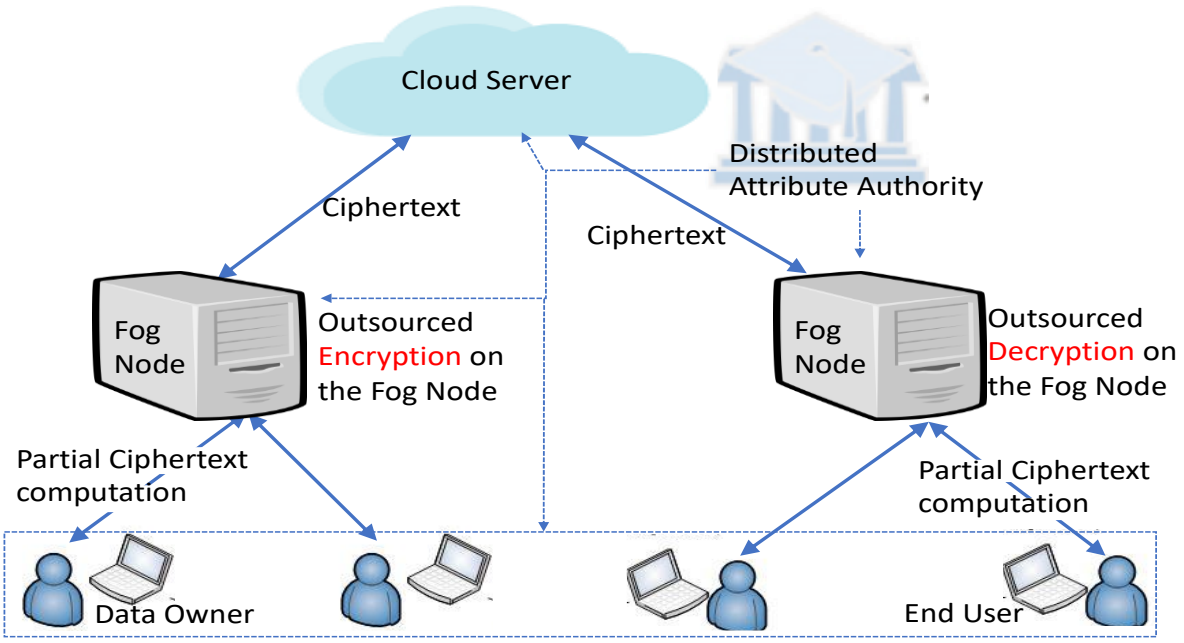


FIGURE 2.2 **Layers of Fog Computing.**

The authors of [25] proposed an AC CP-ABE scheme that outsources the heavy computation of encryption and decryption to fog nodes. This makes the number of attributes in access policy and secret keys independent from the encryption/decryption computation. This scheme uses the intermediate fog layer to reduce the computational overhead for the data owner or end users. As shown in Figure 5, this scheme has five types of entities: (1) a cloud service provider (CSP), (2) fog nodes, (3) the data owner, (4) the end user, and (5) the key authority. The authors assumed that CSP and fog nodes are trusted in the scheme. The data owner has files that need to be encrypted before being sent to the cloud. Each data owner is responsible for defining access policy and generating part of the ciphertext of the encrypted file before sending it to a fog node. A fog node is responsible for the creation of the other part of the ciphertext of the encrypted file. Then, the whole ciphertext is uploaded to the CSP. The key authority oversees user registration and the creation of secret keys for users. The secret keys reflect each user's attributes. The end user is the user on the other side who wants to gain access to the encrypted data stored in the cloud. If the end user's attribute set satisfies the access policy embedded in the ciphertext, the ciphertext will be uploaded from the cloud by the fog node. The fog node will then decrypt part of

the ciphertext. The other part of ciphertext is decrypted by the end user. Updated users are those whose attributes are updated by the key authority, and non-updated users are those whose attributes are not yet updated.

The authors in [37] proposed a framework that secures the sharing of personal health records (PHR) in a cloud computing environment. Their framework supports the scalability feature in cloud computing. The patients can encrypt their PHR files so that only authorized users can decrypt and access them. The presented framework classifies users according to security domains as (1) public domain and (2) private domain. The public domain includes the professional users who are managed distributively by multiple attribute authorities. The attribute authority can control several attributes for all users in the public domain, and each user should be able to reach more than one attribute authority to get his/her attributes. Multiple Authorities Attribute-Based Encryption (MA-ABE) is used by public domain users, such as physicians, so that the user (i.e., physician) attributes represent the professional role of the user in the healthcare domain. In the public domain, patients (i.e., owners) defines access policies for their PHR files based on the professional roles of the users in that domain. On the other hand, the private domain includes those close to the PHR owner, such as family members, and access rights are assigned by the PHR owner to all users in the private domain. KP-ABE is used with private domain users such as close friends or family members so that a patient can control secret keys and access rights for their PHR files. PHR files are encrypted using ABE, so a PHR owner can easily permit users from two domains to access the files. This framework tackled the key and attribute management issues for all users by dividing them into two types of security domains and supported to the scalability features.

The authors in [38] proposed a framework, Privacy Preserving Cipher Policy Attribute-Based Encryption (PP-CP-ABE), to secure data inquiry in mobile cloud computing. This framework protects the collected data from mobile devices. Therefore, outsourcing the heavy computation of encryption and decryption operations to CSPs can be achieved using PP-CP-ABE. Attribute-Based Data Storage (ABDS) is also proposed to decrease the data management overhead caused by CSPs. In the system

architecture, there are three service providers: (1) encryption service provider (ESP), (2) decryption service provider (DSP), and (3) storage service provider (SSP). The ESP presents the encryption service to data owners without disclosing the data, the DSP presents the decryption service to users without exposing the data, and the SSP stores the encrypted data. Mobile devices can outsource the heavy operations of encryption and decryption to ESP and DSP using PP-CP-ABE.

The authors in [39] proposed an ABE scheme that supports the outsourcing encryption of a host and the outsourcing decryption of a user. The host and users are using mobile devices that have limited computational power, and the ABE scheme requires several operations of encryption and decryption. Therefore, the proposed ABE scheme uses two semi-trusted proxies. The first semi-trusted proxy is used to outsource the computation of encryption operations, and the second semi-trusted proxy is used to outsource the heavy computation of decryption operations. Thus, the computational overhead of mobile devices can be reduced. In the encryption stage, the data owner encrypts part of the message and the proxy encrypts the remaining part. This occurs according to the access policy that is defined by a host with set of attributes. In the decryption stage, the proxy compares the predefined access policy by the host with the user's attributes. If the user's attributes satisfy the access policy embedded in the encrypted message, the proxy decrypts part of the ciphertext and transforms the ciphertext to ElGamal ciphertext style. The ElGamal ciphertext can then be decrypted by the user. This scheme can help to relieve the computation overhead in constrained devices.

The authors in [31] proposed a Data Access Control for Multi-Authority Cloud Storage (DAC-MACS) system. A multi-authority CP-ABE scheme with support for an attribute revocation method that achieves both forward and backward security was also proposed. The system model has several entities: (1) a global trusted Certificate Authority (CA), (2) multiple Attribute Authorities (AAs), (3) a cloud server, (4) data owners, and (5) users. The CA is responsible for registering all users and AAs in the system. Each user is assigned a global unique identity by the CA. Every AA is responsible for managing and distributing secret keys that reflect users' attributes or roles. The cloud server stores the owner's

data after it has been encrypted and allows users to access the data if the user's attributes satisfy access policies defined in the ciphertext. Data owners can define access policies and encrypt their data according to predefined access policies. Each user is allowed to decrypt the data stored in the cloud server if their secret keys issued by multiple AAs satisfy access policy embedded in the ciphertext.

The authors in [34] proposed a data AC scheme with the ciphertext update based on CP-ABE and an Attribute-Based Signature (ABS) in fog computing. This scheme has delegated most operations of encryption to the data owner, decryption to IoT devices, and signing to update the ciphertext to fog nodes. Therefore, fog nodes perform heavy computation.Since IoT devices are resource-constrained, they can outsource their heavy computation to fog nodes. This scheme consists of five entities: attribute authority, CSP, fog nodes, data owner, and end user, as shown in Figure 2. The data are first encrypted with a symmetric encryption algorithm by the data owner, who also then defines an access policy and update policy. The access policy is used for end users to decrypt the data when their attributes are satisfied, while the update policy is used for end users who intend to modify the ciphertext. In other words, the data owner specifies two policies: one for decryption and another for modification. Fog nodes play a role in encryption by partially encrypting the data according to access policy, while the data owner completes the encryption phase with the access and update policies and sends it to cloud. IoT devices are limited resources that are connected to fog nodes and used by end users who would like to access the stored encrypted data in the cloud. To access data, the end user's attribute set must satisfy the access policy in ciphertext. Then, the fog node plays a second role to partially decrypt the ciphertext and let the user perform the rest of the decryption to recover the data. Once the user obtains access to the data, he or she might wish to modify and re-encrypt it. A signature-based attribute is applied, and a fog node plays a third role by supporting the request of the user to update the ciphertext. The partial signature is created by a fog node and is used to generate the user's signature. The CSP verifies the signature from end users and renews the ciphertext if the user's attribute set satisfies the update policy defined by the data owner.

The authors of [40] proposed a Chosen-Ciphertext Attack (CCA) security model for ABE with outsourced decryption in fog computing. The CCA is an attack in which the cryptanalyst can collect information by obtaining the decryptions of the chosen ciphertext. Once the information is gathered, the adversary tries to use the collected information to retrieve the user's secret key, which is used to decrypt the ciphertext. This model outsources the decryption operation to fog devices and consists of six algorithms: key generation, key extraction, outsourced decryption key generation, encryption, outsourced decryption by fog devices, and decryption by IoT devices. Two formats of the ciphertext are presented: the original ciphertext, which is generated from an encryption algorithm, and transformed ciphertext, which is executed by the outsourced decryption in fog devices. The paper presented two cases in which the attacker might try to figure out the ciphertext: one with the original ciphertext, and the other with the transformed ciphertext. Each method has several phases that are explained in detail in the paper. One way to detect a CCA is to check the validity of the ciphertext. Since two decryptions are needed in an Outsourced Decryption-Attribute-Based encryption scheme, ciphertext transformation by proxy and transformed ciphertext decryption by the decryptor, two techniques are used. These techniques are (1) Asymmetric and Symmetric Encryption Schemes, proposed in [41], and (2) Identity-Based Encryption, proposed in [42]. For Identity-Based Encryption techniques, which support verifiability, the proxy checks the validity before transformation. The decryptor can also check the validity of the transformed ciphertext using the Asymmetric and Symmetric Encryption technique. Therefore, the two techniques (Asymmetric and Symmetric Encryption and Identity-Based Encryption) are applied on an ABE scheme. The authors showed the cost of the algorithms (KeyGen, Ext, OKGen, Enc, and TDec) with a collection of attributes. The proposed scheme supported outsourced decryption in which the heavy computation is outsourced to fog devices, as the IoT devices have limited resources. However, the scheme does not support outsourced encryption, ciphertext update, and attribute revocation. When the number of data owners is increased, it is difficult to compute the encryption operations on the limited IoT devices without supporting the outsourced encryption to release the computational overhead from IoT devices.

The authors in [43] proposed a protocol of encrypted key exchange based on CP-ABE to secure the communications between fog nodes. In this protocol, communications between fog nodes and the cloud are confidential. The system model consists of several entities: a cloud, a key generator server, fog nodes, and IoT devices. The cloud is responsible for defining the access structure and executing the encryption to produce the ciphertext. Fog nodes are deployed on the network and each one is associated with a set of attributes that are defined by the access policy in the ciphertext. If the fog node's attribute set satisfies the access policy defined by the cloud, the fog node can decrypt the ciphertext and obtain the shared key.

In [26], Sun et al. proposed an attribute-based searchable encryption scheme based on cloud-fog computing. The proposed framework integrated the ABE technology and searchable encryption technology to achieve search-based keywords with fine-grained AC simultaneously. The CP-ABE with multiple authorities was also proposed to manage attribute creation and secret key distribution. The scheme has six entities: central authorities, attribute authority, CSP, fog nodes, the data owner, and the end user. In their scheme, due to the limited resources available by end users and data owners, part of the encryption and decryption operations are outsourced to the attached fog nodes. Therefore, the high computation overhead on end users and data owners are reduced. Personal health records in hospitals are an example of an application of the proposed scheme. One of the limitations is that the keyword sets are taken from the actual encrypted file in the cloud, which introduces the possibility of a chosen-ciphertext attack.

In [24], the authors proposed a keyword search over encrypted data system in fog computing that supported a fine-grained AC using CP-ABE. The system also supported attribute updates by updating the user's secret key and attributes associated with the ciphertext. In addition, they provided a multiple keywords option in a single search query, which locates the data quickly and reduces the range of retrieved data. The system has five entities: a key generator center, a data owner, CSP, an end user, and a fog node. The system supported outsourcing encryption and decryption by moving part of the

computational overhead, including file encryption and decryption, from the data owner or end user to the chosen fog nodes. They presented a security analysis that prevented two types of attacks: Chosen Keyword Attack and Chosen Plaintext Attack. One of the limitations is that, when the number of fog nodes and end users increases, the single key generator center is not enough to manage the distribution of secret keys and the creation of attributes for fog nodes and end users.

2.4 Security and Privacy issues related to Access Control in Fog Computing

There are many security issues related to stored data in fog devices [44]. The ability to access and modify the user's data should only be permitted to authorized entities. Security and privacy requirements for several data services in fog computing, such as storage, sharing, and computation, are mentioned in [15]. AC ensures that only valid users are permitted to read, update, and/or send data within the fog. Thus, AC is used to prevent unauthorized access to data of any kind. Since fog computing is an extension of cloud computing, the security and privacy issues are inherited. Security and privacy issues that are relevant to AC in fog computing include:

2.4.1 Trust in Fog Nodes

Since end users attach to the nearest fog node for real-time processing of their data, the trust level should be measured by the fog node or IoT device layer [45], [46]. Trust between fog nodes and IoT devices is important. The fog node that provides a service to the end user's device should confirm the authentication of the device. The end user's device that requests a service from a fog node should also be able to confirm the authentication of the intended fog node. AC models can be applied to measure the trust level when designing a trust model in the fog computing environment. The challenge will be how to define the trust level in fog computing [44], [47]. To measure the trust level of fog nodes, several attributes of fog nodes can be defined. One of the AC models is ABE, which provides fine-grained AC. The two types of attribute-based AC are CP-ABE and KP-ABE. Another challenge will be to determine who can verify the trust level of a fog node. The trusted authority can be defined to design a trust model

in the fog computing environment. One of the roles of the attribute authority is to create a secret key that reflects fog or user attributes and manages these attributes.

2.4.2 Data Computation in Fog Nodes

End users can offload their data to the nearest fog node for computation. However, outsourcing data to the fog node can cause data breaches. For example, in a smart grid, the reading of the smart meter by a fog node can leak household data [47]. The proposed solution to prevent data breaches is to apply AC models such as CP-ABE when outsourcing the data to the nearest fog nodes to achieve fine-grained AC. Several schemes involving the outsourcing of end users' data to fog nodes have been proposed in the literature. Collectively, these schemes suggest that the data should first be encrypted before offloading it to the fog nodes. The fog node can then perform part of the encryption and decryption of the data to relieve the heavy computation from a wearable end user device. Another important service is search over encrypted data. Serval schemes use a searchable encryption technology to search over encrypted data. Search-based keyword schemes that extend to achieve fine-grained AC using CP-ABE have been proposed in the literature.

2.4.3 Rogue Fog Node

A rogue fog node is a node that is reached by a malicious user. It appears as a legitimate fog node to other fog nodes in the network. Thus, a rogue fog node encourages other fog nodes to connect to it, which causes data damage or false data in the fog layer. One of the features of fog computing is to provide reliability in the fog node layer; however, a rogue fog node can lead to an attack to end users' data. Fog nodes must be protected against a malicious fog node when the end user sends sensitive information to it. When a fog node is divided and sends the computation task to several other fog nodes in the network, if one of fog nodes is a rogue node, it injects false data to the other fog nodes. Therefore, the security and privacy of end user will be destroyed [47]. An ABE scheme, which is a type of attribute-based AC, can be applied to provide confidence in end users and fog nodes.

2.4.4 Fog Node Privacy

In fog computing, sensitive data can be disclosed because the fog node is closed to end users. When an end user offloads its task to the nearest fog node in the network, the location of the end user can be disclosed since location awareness is one of the features of fog computing. If an end user outsources its data to the nearest fog node, the fog node can indicate that the end user is close to that fog node. Once an end user outsources its data to several fog nodes in the network, the privacy of the user's location will be at risk [47]. AC can be a solution to address the issue of user privacy preservation and the security of fog nodes.

In a data breach, the user's information is disclosed and accessed by unauthorized users. When a fog node is performing its task of collecting computing end users' data, data breaches can occur on the end users' side or the fog side. Therefore, AC is needed to maintain the confidentiality of end users' data. One scenario that could occur is one of the fog nodes being reached by a malicious user, then acting as a legitimate fog node to the other fog nodes in the network. The malicious data from the attacker will then be delivered to the other fog devices in the network [48]. To solve the issue, there is a need for security mechanisms such as AC to protect fog devices and prevent malicious activities [8]. Another scenario that could put end users' privacy at risk is a fog node leaving the fog network permanently. In this case, entities interacting with the fog node that has left the network, such as the data owner, end user, or CSP, need to update their AC lists to avoid leftover access to a node that does not exist. AC should be applied to allow only authorized users to access the data—protecting the stored data in fog devices from unauthorized users is another challenge. It is also challenging to design a fine-grained AC system that supports scalability. When an end user's attributes are revoked, updating the user's attributes turns out to be challenging as the number of users increases.

2.4.5 Privacy Preserving in Fog Nodes

Fog computing, like any other computing model, is not immune to privacy issues, including those involving data privacy and location privacy [49], [50]. To solve the latency issue, fog nodes are close to the IoT devices, which facilitates the real-time processing capability in fog nodes. However,

data can be overexposed and revealed to the outside world because of this additional fog layer. The data generated by IoT devices will be computed by the nearest fog nodes, and false data injection attacks can occur when data are outsourced to the fog nodes [51]. Since fog nodes are close to IoT devices, location privacy is another issue. When IoT devices subscribe to a specific fog node for processing demand, it can be inferred that the subscribed IoT devices are close to that fog node and far away from other fog nodes [49], [50]. Therefore, a privacy-preserving guarantee must be achieved in fog computing. AC models can be a solution to address the issue of privacy preserving. Since some AC models, such as ABE, are encryption-based, the promise of data confidentiality can be met. In addition, fine-grained AC can help in limiting the access to data. This can also ensure preserving the privacy of data and users [52].

2.5 Discussion & Research Gaps

2.5.1 Discussion of Several Features in Different Schemes

There are several features that should be taken into consideration when designing AC models. These features are crucial to make the designed scheme more efficient and secure. The features are: (1) outsourcing encryption, (2) outsourcing decryption, (3) multiple authorities, (4) supportability in the fog computing environment, and (5) providing search-based keywords. Using AC models require several operations of encryption and decryption, which increases computation overhead. One of the desired features is to outsource part of the encryption and decryption when designing AC models. Therefore, to reduce the computation overhead, IoT devices can perform part of the encryption and decryption operations, with fog nodes performing the rest. There are several entities that interact with each other in AC models. One of these entities is the key authority, which is responsible for creating secret keys and distributing attributes to users. Thus, when the number of IoT devices increases, one key authority will not be enough to generate secret keys and distribute attributes to users. Designing AC models with multiple key authorities can significantly reduce the network congestion and improve the system's efficiency. In addition, multiple authorities can relieve the enormous effort required for the data owner

to handle the user attributes. Designing AC models to be used in the context of fog computing can decrease the latency issues of IoT devices and relieve the computation overhead of encryption and decryption operations from IoT devices. Since end users should satisfy the access policies of the ciphertext, fog nodes can compare and execute part of the decryption in a timely manner before outsourcing decryption to the end user.

Using searchable encryption (SE) technologies is another feature that could decrease the range of retrieved data from the cloud. As the amount of data increases, integrating SEs in AC models becomes desirable. Therefore, combining the features mentioned can crucially improve the design of AC models. These features are important to researchers due to their benefits in many domains, such as medical care. Designing AC models in medical care is challenging since the privacy and security of patients' data are extremely important.

Table 2.1 compares existing works that propose AC schemes according to the features they have (i.e., outsourcing encryption, outsourcing decryption, multiple authorities, supportability in a fog computing environment, and providing search-based keywords). All schemes in Table II use AC strategies that require heavy computation due to the operations of encryption and decryption. Schemes described in [24]–[26], [34], [38], [39], [52]–[55] support outsourcing encryption and decryption, and schemes in [33], [40], [56], [57] support only outsourcing the decryption operations. Outsourcing the heavy operations of encryption and decryption means that the computation overhead of the end users will be decreased. However, some of the schemes solve the latency issue, while other do not. Schemes in [24]–[26], [34], [52], [54], [55] provide outsourcing of encryption and decryption and solve the latency issue by introducing a fog computing layer between IoT devices and the cloud. The schemes presented in [38], [39], [53], [56], [57] offer outsourcing of encryption and decryption, except those in [40], [44], which only offer outsourcing of decryption. The latency for all of them can be high due to the use of serval cloud computing servers. Multiple authorities enhance the scalability of building a model and reduce the computation overhead on a single authority. Data owners and end users' attributes are

distributed by the attribute authority. As the number of end users and data owners increases, a single attribute authority will not be enough to handle the distribution of users' attributes. Few schemes used multiple attribute authorities to improve AC in their proposed schemes. Authors of [26], [31], [33], [37], [52], [54], [55] developed a scheme that is highly scalable by introducing several distributing attribute authorities in their work. Schemes in [31], [33], [37], [52], [54], [55] support only multiple attribute authorities and AC. The scheme presented in [26] supports multiple attribute authorities and all features in Table II. SE technologies have been well studied in the literature. One of the known SE technologies is search-based keywords, which gained considerable attention in cloud computing for several years. Some schemes presented in Table II provided search-based keywords in the context of cloud computing so that the designed model has both SE technologies and fine-grained AC. Few papers enhanced their schemes by deploying CP-ABE and SE technologies in a fog computing layer that can solve the latency issue. In such schemes, encryption and decryption operations are outsourced to fog nodes and the computation overhead in end users' devices will be reduced. For example, one paper introduced a scheme that uses CP-ABE and search-based keywords and deployed it in the fog using multiple attribute authorities. Additionally, scheme [24] proposed a fine-grained keyword search with outsourcing encryption and decryption in fog computing, while scheme [26] presented ABE and keyword search for personal health records in fog computing using multiple attribute authorities.

Table 2.1 Comparison of features in different AC schemes

Schemes Authors	Features						Citation
	Search based Keywords	Fog Computing	Multiple Authorities	Outsourcing Encryption	Outsourcing Decryption	Access Control	
Zhang [25]		●		●	●	●	59
Zhou [38]				●	●	●	226
Asim [39]				●	●	●	16
Mao [56]					●	●	63
Zuo [40]		●			●	●	64
Alrawais [43]		●				●	60
Li [37]			●			●	1033
Yang [31]			●			●	2
Wang [53]				●	●	●	29
Li [57]	●				●	●	120
Huang [34]		●		●	●	●	48
Sun [26]	●	●	●	●	●	●	3
Miao [24]	●	●		●	●	●	26
Vohra [33]		●	●		●	●	2
Fan [54]		●	●	●	●	●	32
Xu [55]		●	●	●	●	●	9
Xue [52]		●	●	●	●	●	21

2.5.2 Gaps and Future Research Directions

Fog computing and cloud computing are similar in nature and highly coupled. However, solutions built to address the cloud efficiency, security, and privacy issues cannot necessarily be applied to fog computing. As previously discussed, multiple operations of AC models can have a major impact on fog computing solutions. This section outlines the research gaps that need to be addressed to enhance AC models in fog computing:

1) Auditing Mechanism

Designing an AC model with an auditing mechanism is necessary in distributed fog nodes. Auditing, in this case, is important to periodically check users' attributes and make sure that the attributes are valid, and that the users' privileges are not outdated. Moreover, in highly scalable

environments, an auditing mechanism becomes important for the robustness of AC. Since cloud computing, IoT, and fog computing are highly coupled and scalable, there will be a massive number of joining and leaving nodes (e.g., fog, cloud, IoT). This mandates continuous and thorough auditing to maintain the confidentiality and integrity of the data and applications. The existence of nodes with more privileges than needed also increases the demand of computation power to handle operations like encryption and decryption, which negatively impacts the environment performance. Therefore, intelligent auditing mechanisms are needed to automatically search for policy violations and update access policies and users' attributes.

2) Fine-Grained Access Control

Proposing fine-grained AC is essential in widely distributed fog nodes. When a number of IoT devices connect to fog nodes, fog nodes apply one or more AC models to grant access to a number of authorized IoT devices. Fine-grained AC can be introduced in fog computing to limit the access to specific data, and each fog node can apply its own access policy for its own IoT devices. This is important, as it will give administrators more control and flexibility to securely and effectively overexpose and underexpose data. Designing fine-grained AC models for distributed fog nodes is necessary; thus far, however, little or no work has been done to tackle this challenge.

3) Covering More Features for Better Efficiency

Designing and implementing AC models that cover more features is important for the efficiency of the model. Therefore, we surveyed AC models and their supporting features to better understand how these features work and how to integrate them in a future AC model. More AC features can still be explored and integrated to build efficient AC in fog computing.

2.6 Conclusion

Fog computing is a new computing paradigm that provides real-time processing at the edge of the network, close to IoT devices. AC models can be applied in fog computing to preserve the privacy of IoT data and to protect the system and users' data. Several security and privacy issues in fog

computing can be solved using one or more AC model(s); however, the choice of an AC model is dependent on the application's requirements. In this paper, we thoroughly discussed fog computing and AC models. Then, we presented the state of the art in the field of fog computing AC. We also discussed some security and privacy issues relevant to AC in fog computing. Several features that that are known to produce efficient AC models in fog computing were discussed and research gaps were outlined.

In our future research, we plan to propose an AC model that supports more features for better efficiency and security. We also plan to investigate designing a fine-grained AC model for fog computing-aided environments.

Chapter 3: Performance Analysis of Two Cloud-Based IoT Implementations: Empirical Study

M. Aleisa, A. A. Hussein, F. Alsubaei and F. T. Sheldon, "Performance Analysis of Two Cloud-Based IoT Implementations: Empirical Study," *2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2020, pp. 276-280, DOI: 10.1109/CSCloud-EdgeCom49738.2020.00055

3.1 Introduction

Cloud computing is an attractive environment that provides computation and storage for the Internet of Things (IoT). However, high latency is an issue within cloud computing due to the distance between the cloud and IoT devices. Data from IoT devices requires a substantial amount of time to be processed and analyzed in the cloud. Moreover, as the number of IoT devices amount, the amount of data generated becomes enormous. Low latency is an essential factor in computing the enormous amount of data published from IoT devices. Therefore, fog computing appeared to provide light computation power and temporary light storage with low latency [58].

Fog computing is an intermediate layer between IoT devices and the cloud that is designed to solve the latency issue. As the number of computations increases, fog computing becomes essential to provide high-performance computation in a real-time manner because it allows computation to occur closer to IoT devices [59]. Measuring the performance of published data in IoT-Fog-Cloud environments is important to assess the outcomes of continuous efforts to improve. Different researchers have proposed and developed various benchmark metrics that measure the performance of IoT-Fog-Cloud environments [60]–[63]. For more accurate benchmarking, implementations that simulate real-life environments must be evaluated when measuring the message subscribe and publish performance of IoT-Fog-Cloud environments.

Amazon Web Services (AWS) provides an IoT service (AWS IoT) that offers secure communication between IoT devices such as sensors and actuators [64]. This service allows developers to collect and analyze an enormous amount of data from several devices. In addition, Amazon

CloudWatch [65] is used to monitor the resources of AWS in real time. CloudWatch provides a variety of metrics to measure the performance of cloud-based IoT environments [66].

This paper makes the following contributions. First, we built two experimental implementations of cloud-based IoT environments and measured their performance. We used AWS as a cloud provider in both implementations. The first implementation (a) applies a fog layer between IoT devices and the cloud. The second implementation (b) publishes data directly to the cloud without having a fog computing layer between the IoT sensors and the cloud. Second, we used AWS IoT metrics embedded in Amazon CloudWatch to measure the performance of the two implementations a and b (Fig. 3.1). Third, AWS metrics were used to compare the performance of implementations a and b over time.

The remainder of this paper is structured as follows: In section 2, we present the motivation for our work and discuss existing fog and IoT benchmarks. In section 3, we describe our implementations a and b in detail. In section 4, we provide a description of the AWS IoT metrics that were used to measure our implementations. We describe the empirical testing and discuss the results in section 5. The discussion, including challenges faced and future work, is presented in section 6. Finally, we conclude the paper in section 7.

3.2 Motivation

Several benchmark metrics have been proposed and used in the literature to measure the performance of IoT-Fog-Cloud environments [60]–[62], [67]. TPCx-IoT [68] is a TPC industrial benchmark to measure the performance of IoT systems.

The authors in [63] explained the details of TPCx-IoT and showed the performance of different industry configurations running HBase 1.2.0. IoTABench is another analytical IoT benchmark for big data [60]. A benchmark for an IoT system for big data that includes a smart meter use case is proposed in [60]. The smart meter use case includes eight node clusters operating the HP Vertical analytics platform version 7.0.0. Reference [61] presented Edge-Bench, which is a benchmark for edge computing platforms, and studied two platforms: Greengrass and Azure Edge. Comparing performance of the two

platforms showed that there is a high latency in the Azure Edge platform. The authors of [62] presented an edge computing AI benchmark called Edge AlBench. The Edge AIBench contains six components benchmarks and four application benchmarking frameworks.

MQTT, a well-known protocol produced by IBM [12],[69] can be a solution for communication between the IoT-Fog-Cloud layers [10]. Several researchers proposed different benchmarks for the MQTT protocol to measure the performance of IoT-Fog-Cloud environments. The authors of [67] studied two IoT platforms, ThingBoard and SiteWhere. The performance of two protocols, MQTT and HTTP, was evaluated on the ThingBoard and SiteWhere IoT platforms in order to compare these platforms. The performance evaluation showed that ThingBoard performs better than SiteWhere. In addition, the authors of [70] analyzed the process of message transmission using the MQTT protocol. The authors captured the delays and message loss of the transmitting messages with different service quality levels and payload sizes. The results showed that the delays are associated with message loss for different sizes of messages. It is important to simulate a real-life environment for cloud-based IoT environments to provide accurate benchmarking. Therefore, in this paper, we simulate a real-life environment of the two implementations a and b to measure the performance of subscribing and publishing messages. We conducted our analysis using several metrics to evaluate our implementations and to provide a better understanding of their performance.

3.3 Implementation

In this section, we present two cloud-based IoT implementations. Fig. 3.1 depicts the implementations, a and b. These implementations are used in the following section to measure performance. In implementation a, two basic temperature and humidity sensors (DHT11 [71]) are connected to a Raspberry Pi 3 model B [72], which enables sensors to communicate over Wi-Fi since the sensors are not equipped with Wi-Fi boards. Sensors along with their Wi-Fi-enabling Raspberry Pi are considered to be the IoT layer that has light computation capabilities and light storage for configurations. Both sensors, with their respective Wi-Fi-enabling Raspberry Pis, are connected to

another Raspberry Pi 3 model B that serves as middle layer (i.e., fog node). An MQTT broker operates on top of the fog layer to enable message exchange using a publish/subscribe model. The fog layer is used for light computation and temporary storage. Data from the sensors that is received by the fog node is then forwarded to the cloud layer (i.e., AWS IoT). This cloud computing layer is used for high computation and permanent big data storage. The three layers are wirelessly connected using Wi-Fi. In implementation b, two DHT11 sensors are connected to a Raspberry Pi model B. This is the IoT layer of the implementation. Sensors are then directly connected to the cloud (AWS IoT) without a middle fog layer.

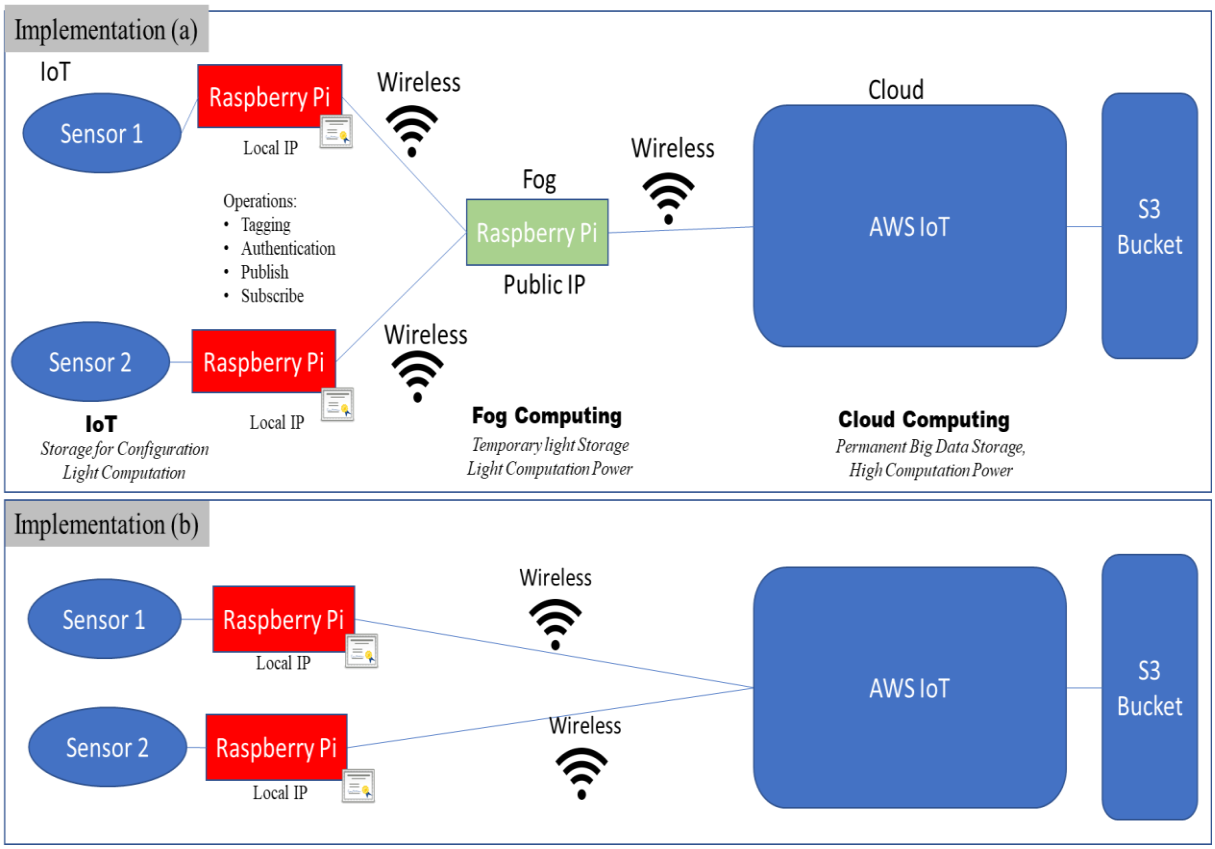


Figure 3.1 Cloud-Based IoT Implementations a and b

As mentioned above, in the IoT layer of implementations a and b, because the sensors do not come with Wi-Fi capability, the sensors are attached to the Raspberry Pi model B to enable Wi-Fi.

Arduino IDE software is installed on top of the Raspberry Pi and a C++ program is written to read the sensor's data and publish it to the fog layer. We installed another MQTT broker on the fog node (the green Raspberry Pi box in Fig. 3.1) to publish data to the cloud (AWS IoT). A python script is executed in fog to filter the data received from the IoT layer (the sensors) and publish it to the cloud (AWS IoT). In the cloud layer, the AWS message broker handles receiving the data sent from the fog node (the green Raspberry Pi box in Fig. 3.1). Similarly, in implementation b, we also attached the sensors to the Raspberry Pi to enable Wi-Fi. In addition, we installed Arduino IDE in Raspberry Pi and wrote a C++ program that reads the sensor's data and publishes it directly to the cloud (AWS IoT).

3.4 Description of AWS IoT metrics

Amazon CloudWatch provides a monitor to measure AWS IoT-based systems. It can process data and analyze it in real time. There are several AWS metrics [9] that can be taken in consideration to evaluate our cloud-based IoT implementations. These metrics are:

1) Connect.Success:

This metric is used to collect the number of successful connections from our IoT nodes or fog nodes to the AWS message broker.

2) Ping.Success

This metric collects the number of ping messages received by the AWS message broker. These ping messages are received from the fog node(s) in implementation a and from the IoT node(s) in implementation b.

3) Publishin.Success

This metric is used to collect the number of publish requests successfully processed by the AWS message broker. Like the ping messages, these messages are also received from the fog node(s) in implementation a and from the IoT node(s) in implementation b.

4) Publishout.Success

The Publishout.Success metric is used to collect the number of publish requests successfully made by the AWS message broker to the fog nodes in implementation a and to the IoT nodes in implementation b.

5) Subscribe.Success

The Subscribe.Success metric is used to collect the number of successful subscribe requests processed by the AWS message broker. These requests are made by the fog node in implementation a and made directly by the IoT devices in implementation b.

6) Publishin.Clienterror

Finally, the Publishin.Clienterror metric is used to collect the number of publish requests rejected because they did not meet the AWS IoT requirements.

7) Unsubscribe.Success

The Unsubscribe.Success metric collects the number of unsubscribed requests that were successfully processed by the AWS message broker. These unsubscribe requests are made by the fog node in implementation a and made directly by the IoT devices in implementation b.

8) Throttle.Exceeded

The Throttle.Exceeded metric is used to collect the number of requests that were throttled because the client (i.e., the IoT node or fog node) has sent too many messages and exceeded the allowed message rate.

9) Publishout.Throttle

The Publishout.Throttle metric is used to collect the number of publish requests that were throttled because the client (i.e., IoT node or fog node) exceeded the allowed message rate.

3.5 Empirical Testing

To test the differences in the two implementations, we analyzed the metrics provided by AWS IoT that are listed in section 4 and shown in Table 3.1. Fig. 3.2 shows the results of these metrics at

different durations (i.e., 1 second, 5 seconds, 10 seconds, 15 seconds, 30 seconds, 1 minute, 5 minutes, 15 minutes, and 1 hour). The metrics show that both implementations have similar results when run for less than 15 minutes. They also show that implementation a had a higher number of successful connections, especially after 15 minutes. Implementation a also showed better results in the number of subscribe requests.

Table 3.1 AWS metrics results for Two Cloud-Based IoT Implementations

AWS IoT metrics results – implementation *a* (imp1) & implementation *b* (imp2)																
AWS IoT metrics	**1 second**		**5 seconds**		**10 seconds**		**30 seconds**		**1 minute**		**5 minutes**		**15 minutes**		**1 hour**	
	Imp1	***Imp2***	***Imp1***	***Imp2***	***Imp 1***	***Imp2***	***Imp1***	**Imp2**	***Imp1***	***Imp2***	***Imp1***	***Imp2***	***Imp1***	***Imp2***	***Imp1***	***Imp2***
Connect. Success	3.135	2	3	2	3	2	3	2	3	2	6	2	9.005	3	29	3
Ping.Success	1	4	1	4	1	4	1	4	1	4	5	18	15	63	87	275
PublishIn.Success	40	30	40	30	40	30	40	30	40	30	248	150	708	450	1.42 k	1.8 k
PublishOut.Success	18	16	18	16	18	16	18	16	18	16	122	76	352	196	710	1.26 k
Subscribe. Success	2	1	2	1	2	1	2	1	2	1	2	2	6	2	10.025	2
Unsubscribe. Success	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Throttle.Exceeded	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Publishout.Throttle	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Publishin.Clienterror	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

However, implementation b outperformed implementation a in the number of processed publish requests by AWS IoT, the number of publish requests made by AWS IoT, and the number of successful pings after 1 hour. Both implementations had similar results in the number of unsubscribe requests processed by AWS IoT. It is important to note that three of the metrics (Throttle.exceeded, Publishout.throttle, and Publishin.clienterror) did not show any results, which indicates that our

implementations did not violate AWS IoT requirements and did not exceed the allowed rate of messages. Overall, implementation b (i.e., no fog layer) tends to outperform implementation a in some metrics, especially at or after 1 hour. However, implementation a has a better ability to scale up as the number of devices and messages increase due to its utilization of a fog computing node, as shown in Fig 3.2.

3.6 Discussion

Although in our experiment we demonstrated that adding a fog layer between the sensors and the cloud in cloud-based IoT environments did not substantially impact performance, these results are subject to some limitations. First, we noticed that implementation b outperformed implementation a after the first hour of running both implementations. We are not sure at what point between 15 minutes and 1 hour implementation b started to perform better. We also ran both of the experiments for only one hour; thus, we do not know whether implementation b would continue to perform better than a. The results we obtained, as well as the lack of results in some metrics, are likely due the small number of edge devices used and/or to running both experiments for only one hour.

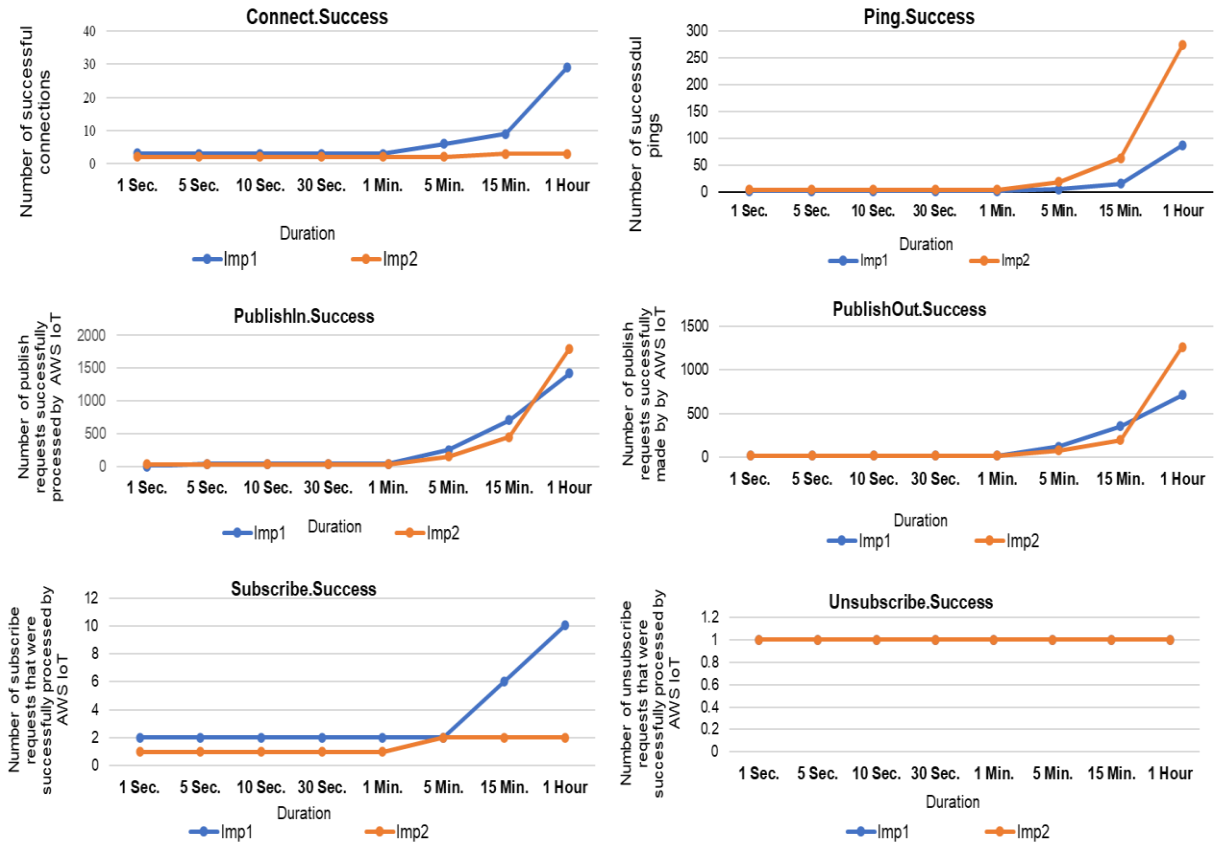


Figure 3.2 AWS metrics results for Implementations a (Imp1) and b (Imp2)

In this paper, we used only two sensors per implementation. Future studies may test whether the results we obtained hold true if we use more sensors. If 1000 sensors and 10 fog nodes were used, for example, it is likely that implementation a would outperform implementation b due to the hierarchical architecture in implementation a that makes it easier and less cumbersome for the IoT to manage the received messages from the MQQT brokers. In other words, instead of having the AWS IoT interacting with 1000 sensors, it will receive messages from 10 fog devices. This is important because it means that implementation a could also handle additional functionalities, such as access control, encryption, decryption, and filtering of messages. Also, although we utilized a fog layer in implementation a to better manage and control message exchange between the sensors and AWS IoT, the sensors and fog nodes were both operating using TCP on the same LAN. We plan to locate the fog node on a different network and perform the experiment again to determine whether that will affect the results. In addition,

our AWS IoT currently operates on Virginia datacenter. We will try to run our AWS IoT service on a different datacenter and determine whether that would impact our results.

This paper presents a work in progress on how adding a fog layer may impact the performance of cloud-based IoT systems. Such work is important because we are attempting to implement an access control model [73] that may need to execute some operations for the IoT devices, since sensors usually have low or no processing capability. This will improve the security of IoT environments, especially in critical applications such as medicine [74].

3.7 Conclusion

In this paper, we presented two cloud-based IoT implementations using a real-life platform that is used in industry. One of the implementations has a fog layer between the IoT devices (i.e., sensors) and the cloud, whereas in the other implementation, IoT devices were directly connected to the cloud. The purpose of these experiments was to better understand the impact of the additional fog layer on the performance of cloud-based IoT environments. We examined the performance of the two implementations and showed that adding a fog layer between the IoT devices and the cloud positively impacted the connect, message publish, ping, and subscribe metrics for the first 15 minutes. At the 1-hour time point, the implementation that did not have a fog layer performed better. We also noticed that both implementations gave consistent results (i.e., increasing, decreasing) using the AWS metrics, which indicates credible results. In the future, we plan to use the same metrics to evaluate the performance of IoT-Fog-Cloud environments that utilize encryption-based access control to determine how the encryption and decryption operations affect the performance of cloud-based IoT environments, with the aim of providing practical solutions.

Chapter 4: Examining Performance in Fog-Aided Cloud-Centered IoT in a Real-World Environment: Scientific Experiments

M. A. Aleisa, A. Abuhussein, F. S. Alsubaei, and F. T. Sheldon, "Examining the Performance of Fog-Aided, Cloud-Centered IoT in a Real-World Environment," *Sensors*, vol. 21, no. 21, p. 6950, Oct. 2021, doi: 10.3390/s21216950

4.1 Introduction

Cloud computing is an emerging technology that offers high computational power and permanent storage for the Internet of things (IoT). In cloud-based IoT environments, as the number of IoT devices increase, the amount of data generated from the IoT also increases. This causes a high latency due to the long distance between the IoT devices and the cloud. Exchanging data between IoT devices and the cloud increases the utilization of bandwidth and requires increasing resources as the number of IoT devices increases. In addition, operations such IoT device authentication and authorization, as well as encryption, add computation overhead on the cloud. This requires bringing the computation capability closer to the IoT devices and reserving the resource-demanding tasks for the cloud. Therefore, fog computing emerged to satisfy the demand for frequent computation, communication, and storage by the IoT layer [58], [75].

According to Cisco, fog computing is a layer of computing that extends the cloud, bringing it closer to the things that generate and process IoT data. Any device with computing, storage, and network connectivity can be a fog node, and fog layer nodes can be deployed anywhere with a network connection (e.g., on top of a traffic light, alongside a railway track, etc. [76]). Fog computing enjoys the following characteristics:

1) Low Latency: Network latency is defined as the time it takes for data or a request to travel from the source to the destination. In cloud-based IoT environments, latency is typically high due to the distance between the IoT devices and the cloud. This increases the cloud response time, especially as the number of IoT devices increases, making the cloud unable to support the real-time demand of IoT devices. Fog

computing reduces that latency by bringing data to the edge of the network and closer to end users to meet the high processing demand.

2) Higher Scalability: Scalability is the ability of a system to handle a growing amount of work by adding resources to the system. In cloud-based IoT environments, as the number of IoT devices increases, it becomes difficult for the cloud to handle the heavy computation and bandwidth overhead of the devices. Fog computing can solve this issue by distributing serval fog nodes that can reduce the heavy load on the cloud and support hierarchal scalability when the number of IoT devices increases.

3) Location Awareness: Location awareness refers to the ability of a device to passively or actively determine their location. This feature is important because it allows applications to provide services better suited to user and device location, thus lowering latency.

4) Mobility: Computing mobility is the ability to perform computing operations while a connected device is able to move, communicating from any location through a wireless channel. This includes mobility of IoT nodes as well as fog nodes in cloud-based IoT environments.

5) Decentralized Architecture: A decentralized network is a network of interconnected devices in which no single entity is the sole authority. Workloads in distributed architectures are distributed among several machines instead of relying on a single central server. This is an important feature of fog computing because applications and services on the fog can process and store data from any end devices, whether it is a fog node or a sensor (i.e., IoT node).

6) Heterogeneity: Heterogeneity in networking refers to a network that connects devices made by different manufacturers running different operating systems and uses multiple network architectures and protocols. Fog computing heterogeneity is a topological feature that is of particular importance in cloud-based IoT environments, as it enables devices to exchange information and to use the information that has been exchanged without restrictions.

7) Bandwidth Optimization: Optimization of network bandwidth refers to overall inbound and outbound bandwidth improvements on a network. This allows fog nodes to handle traffic from billions of devices to prevent congestion and latency problems. This is because the enormous amount of data collected from the IoT devices is can be processed locally instead of transmitting it to the cloud.

Despite the benefits offered by fog-aided IoT, researchers and practitioners are faced with challenges in the implementation and performance of fog-aided IoT systems.

First, there is a lack of real-life implementations of the many theoretical studies in research and academia. Although simulation-based experiments provide easy access to practical results about the performance of computing systems, observations and research outcomes may not be generalizable to all scenarios due to the variety in IoT platform providers and device manufacturers; their different implementations, service specifications, and configurations; and differences in network architectures and protocols. Therefore, in order to develop a profound and general insight into the tradeoffs involved in a particular system, it is important to use real IoT platforms built on top of a real-world network (i.e., Internet) when obtaining analytical results for the performance of fog-aided IoT implementations. In addition, it would be interesting to explore the performance differences of fog implementations interacting with different commercial IoT platforms, such as Amazon IoT and Azure IoT.

Second, due to the diverseness that fog-aided IoT environments enjoys and the lack of consensus among practitioners and hobbyists on a standard fog computing implementation, there is a lack of resources that show how to implement an efficient fog-aided IoT system. Most of the implementations available are either domain specific, complex, or too abstract to be useful in all scenarios.

Third, although fog computing offers promising solutions to many of the performance and security problems of the IoT [77], it is confronted with various security and privacy risks. For instance, while fog computing is crucial for spreading risks across distributed fog nodes, it also has the untoward effect of increasing the attack surface. This is exacerbated because fog computing devices interact with devices only; that is, the fog nodes receive IoT data from sensors and send it to the cloud and vice versa. This

means that no humans are involved in the communication. Although this can be considered an advantage because these interacting devices do not have screens or an on-device user interface, which reduces the attack surface, it can lead to failures and/or targeted attacks that cannot be easily detected and deterred. Other security and privacy issues in fog-aided IoT also deserve our attention. In this work, we aim to better understand fog-aided IoT environments in order to pave the way for further research to address interesting confidentiality, integrity, and availability violations. This paper makes the following contributions:

- We present two architectures of cloud-based IoT environments. The first architecture has a fog layer applied between the IoT devices and the cloud, whereas the second architecture publishes the data directly to the cloud without a fog layer.
- We used two benchmarks to measure the performance of the cloud-based IoT architectures. The first benchmark is Mosquitto message broker metrics, which are used to measure the performance of the IoT-Fog-cloud at the fog computing level. The second benchmark is AWS message broker metrics, which are used to measure the performance of the two architectures (IoT-cloud and IoT-fog-cloud) to show the impact of the additional middle layer (i.e., the fog layer) on cloud-centered IoT environments.
- We discuss some security and privacy implications of the two architectures presented in this paper, showing what triggers these implications and suggesting methods to address these implications.
- This work serves as a tutorial reference of fundamental fog computing concepts and aims to walk practitioners through different implementations of fog-aided IoT and to reveal tradeoffs that inform when to use each implementation based on one's objectives.

The remainder of this paper is structured as follows. In section 2, we present related work and discuss existing industrial and Message Queuing Telemetry Transport (MQTT) benchmarks. In section 3, we describe the experiment setup of the two architectures of cloud-based IoT environments in detail. In section 4, we provide a description of the AWS IoT metrics and Mosquitto message broker metrics

that were used to measure our two architectures. We describe the performance metrics in section 5 and discuss the results in section 6. The discussion, including challenges faced and future work, is presented in section 7. Finally, we conclude the paper.

4.2 Background

Today, IoT sensors are used everywhere and have become crucial to the operation of many domains of life. As shown in Fig 4.1, different kinds of sensors can be found in our homes, cars, workplaces, etc., and are sold independently (e.g., smoke sensor, light sensor, temperature sensor, motion sensor, proximity sensor, touch sensor, ultrasonic sensor, humidity sensor, IR sensor, pressure sensor, gyroscope sensor, etc.) or as an integral part of a sophisticated device such as a smartphone that may have dozens of sensors. These sensors are developed by major manufacturers and are deployed in many sectors, including healthcare, education, communication, transportation, and manufacturing.

Manufacturers have developed IoT platforms to help organizations build fully functional IoT environments. According to AT&T [78], an IoT platform is an end-to-end software framework that pulls together information from sensors, devices, networks, and software that work together to unlock valuable, actionable data. IoT platforms enable management and automation of connected devices within the IoT universe. There are several proprietary IoT platforms available, including AWS IoT Core [79], Microsoft Azure IoT [80], IBM IoT [81], and Google IoT Core [82], as well as opensource IoT platforms such as IoTivity [83], Zetta [84], Arduino IDE [85], DeviceHive [86], and openremote [87]. These IoT platforms usually reside and run on a virtual machine on the cloud that efficiently pulls, processes, and stores the data received from the massive number of IoT sensors.

Classical IoT environments are configured so that IoT sensors are directly connected to the IoT platform and the cloud. In modern architectures, a fog layer is introduced between the IoT sensors and the IoT platform (the cloud) for extra-efficient computation, communication, and storing. Case studies to show tradeoffs between the two implementations will be discussed extensively in this paper.

Devices in all layers of IoT environments (see fig. 4.1) communicate using different protocols. These protocols are [11]:

- MQTT is a lightweight many-to-many communication protocol for the IoT that is designed to be a publish/subscribe messaging transport protocol. MQTT is ideal for connecting remote devices with minimal memory consumption and network bandwidth. MQTT is used in a wide variety of domains, such as industry, health care, and transportation. Port 1883 is the default MQTT port, whereas port 8883 is the default MQTT port over TLS (i.e., secure-mqtt), and both are registered at the Internet Assigned Numbers Authority (IANA) for Secure MQTT.

- Constrained Application Protocol (CoAp) is a one-to-one User Datagram Protocol (UDP) protocol for transferring state information between client and server. Despite its ability to preserve resources, CoAP is best suited to a state transfer model. Since CoAp uses UDP, it does not guarantee the delivery of datagrams. In addition, CoAp is unencrypted. The default CoAP port registered at IANA is 5683.

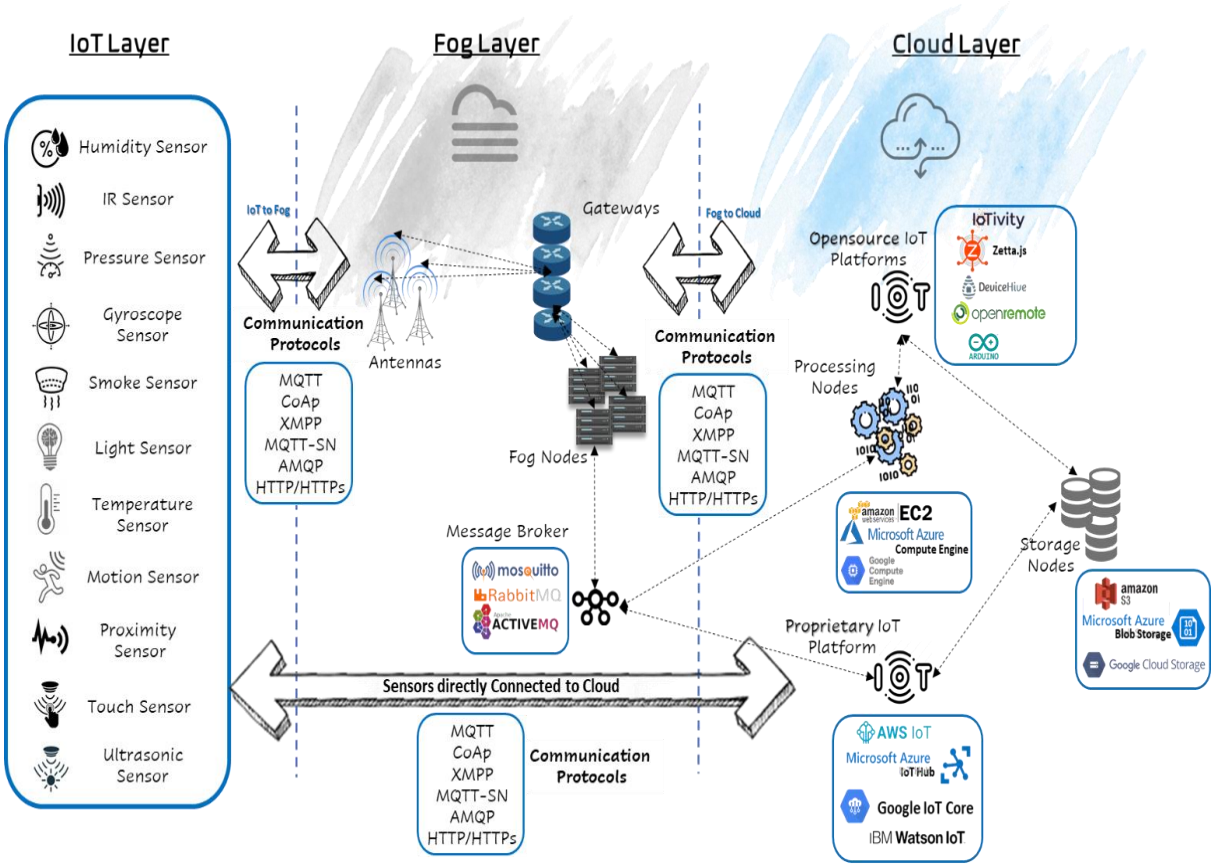
- Extensible Messaging and Presence Protocol (XMPP) is a secure and near-real-time communication protocol for message-oriented middleware based on XML that enables the exchange of structured but extensible data between any two or more devices over a network. XMPP is mainly used by instant messaging applications such as WhatsApp [88] and Telegram [89]. XMPP offers persistent decentralized connection between devices; thus, no central XMPP servers are needed to communicate. However, to establish a connection between two devices, one of the devices is considered an xmpp-client and communicates over port 5222, while the other is considered an xmpp-server and uses port 5269. XMPP can also use port 5280 for two-way communication. This is called xmpp-bosh, which means Bidirectional-streams Over Synchronous HTTP (BOSH). XMPP has been used in the literature to network IoT devices, such as in [90], [91]. Despite its features and potential, XMPP has some limitations. First, XMPP does not have a Quality of Service (QoS) mechanism. In addition, XMPP streams data in XML format, which introduces overhead due to the

text-based communication. These reasons, among others, make MQTT a more popular protocol for IoT, since it has a mechanism for QoS and uses lightweight binary-based communication.

- MQTT For Sensor Networks (MQTT-SN) [92] is considered a modified version of MQTT that is adapted to the attributes of a wireless connection, such as a lossy wireless network. It is designed specifically for wireless sensor networks with scale in mind. MQTT-SN was developed to support non-TCP networks like UDP. This is another advantage because it makes the communication lighter by eliminating the TCP handshakes.

- The Advanced Message Queuing Protocol [93] (AMQP) is an open standard application layer protocol for middleware. AMQP is designed with more advanced features that introduce more overhead than when using MQTT. These features include message orientation, queuing, routing, reliability, and security. The registered port number for AMQP at IANA is 5672, and for AMQPS (i.e., TLS/SSL encrypted AMQP), it is 5671. For more information on which protocol functions best for the IoT based on the messaging requirements, see [94].

Figure 4.1 Overview of cloud-aided IoT environments.



This work focuses on commercial IoT sensors (i.e., DHT11 temperature and humidity sensors) connected using a proprietary IoT platform (i.e., AWS IoT Core). We choose proprietary IoT platforms over the free opensource ones because they are popular among industries due to their faster time to market and lower initial cost.

Amazon Web Services (AWS) provides reliable, scalable, and inexpensive on-demand cloud computing services to individuals, companies, and governments around the world [64]. Customers can benefit from the cloud data centers distributed in the different locations in many ways, including the low cost (pay as you go) and the massive cloud infrastructure to perform experiments and deploy new applications. AWS offers many services; however, we will be using the following in this work: (1) AWS IoT, (2) Amazon S3, and (3) AWS CloudWatch. AWS IoT allows a secure communication and messaging exchange over MQTT for internet-connected devices such as sensors and micro-controllers

in real time [95]. Amazon CloudWatch [96] monitors devices and applications connected to AWS in real time using several metrics [95] to track the connected devices and measure the performance, security, and scalability, among other criteria.

The MQTT protocol is a widely used protocol and is supported by all IoT platforms as well as commercial sensors; therefore, we used MQTT as the communication protocol. In addition, we used a Raspberry Pi board to simulate fog nodes for the scenarios in which fog nodes are introduced. To enable the Raspberry Pi boards used in this experiment to communicate over MQTT, we used an MQTT broker software, Eclipse Mosquitto [97]. An MQTT broker is a server that receives all the messages from the IoT devices and publishes them to other devices. MQTT broker also has other benefits, including (1) supporting scalability with many IoT devices, (2) managing credentials and certificates that are used for authentication, (3) decreasing network strain on the cellular network without disclosing security, and (4) excluding the connection of insecure and vulnerable devices. Many MQTT brokers are available, including Eclipse Mosquitto [97], RabbitMQ [98], and ActiveMQ [99]. In this paper, we used Eclipse Mosquitto because it is the most popular and has ample resources for implementation. In addition, it is lightweight and suitable for use on all devices, from low-power single-board computers to full servers. An MQTT broker feature called SYS-Topics [100] is widely used to monitor the Mosquitto MQTT broker by providing metrics about Mosquitto and track the devices connected to it.

In this paper, we present numerical results based on an experiment that uses a real-world IoT platform, sensors, and network (not a simulation) to show the performance tradeoffs of various IoT implementations and discuss the results. Notably, this experiment in a real environment is vulnerable to real-life cyber and/or physical attacks as well as performance failures.

4.3 Related Works

The rapid adoption of cloud-based IoT environments in large scale and with intensive use has induced, among other factors, a growing need to simulate real-life environments to measure the security and performance of the IoT-Fog-Cloud environments to provide suitable support for the construction of

an efficient access control model [58]. Benchmarks are one of the ways to measure the security and performance of cloud-based IoT environments. There are several widely used general industry benchmarks that are adopted in many commercial solutions. The Standard Performance Evaluation Corporation (SPEC) provides benchmarks for a wide range of IT components, such as cloud, CPU, storage, power, and virtualization [101]. The Transaction Processing Performance Council (TPC) also offers a suite of widely used IT industry benchmarks [102]. TPCx-IoT is one of TPC benchmarks that measures the operating system, and data storage and management systems to provide the industry with performance metrics and other available metrics of IoT systems [63], [103]. Moreover, HP developed IoTABench, an IoT analytics benchmark for big data scenarios that is used to evaluate the performance and scalability of a big data platform [60]. The benchmark was demonstrated using a smart metering IoT use case and evaluated on the HP Vertica 7 analytics platform, which can handle data for an "electric utility with 40 million smart meters".

For MQTT, different benchmarks to measure the performance and security of cloud-based IoT environments have been proposed in the literature [104], [105]. Two IoT platforms, Things Board and Site Where, have been evaluated using different metrics [67]. In addition, evaluation of the message transmission process (i.e., Subscribe and Publish) of the MQTT protocol via wireless and wired clients was presented in [70]. The end-to-end delay and message loss when transmitting messages are analyzed with different quality of service levels and different payload sizes. The results of their experiment showed that end-to-end delay is related to the message loss with different sizes of payloads.

Moreover, Azzam et al., in a recent survey, evaluated the performance of fog computing using performance metrics such as processing delay, processing costs, and processing power, and derived the performance gains obtained in comparison to a cloud computing-only approach [106]. In the healthcare sector, Alsubaei et al. evaluated security in the Internet of Medical Things (IoMT) [46]. In addition, Kafhali et al. evaluated the response time of accessing medical data stored in a fog-based IoMT implementation [107]. They also proposed a queuing model to predict the minimum number of computing

resources (both fog and cloud nodes) required to meet the Service Level Agreement (SLA) for response time. Another study in the healthcare field was presented by Vilela et al., who compared the performance of fog-based computing to the conventional cloud computing model in a healthcare real-time monitoring system [108]. EdgeBench is another benchmark for serverless edge computing platforms and is used to measure the performance of two edge computing platforms, Greengrass and Azure [61]. In addition, DeFog, a fog computing benchmark, was proposed to provide a standard methodology and facilitate the understanding of the target platform by collecting a catalogue of relevant metrics for a set of benchmarks [109]. However, most experiments in these studies were carried out using simulators that rely on provided generic metrics and/or focus on one domain, which do not represent real IoT-fog-cloud systems across different domains.

Hence, in this paper, we extend the previous works by implementing real-life experiments and analyzing performance metrics from a popular cloud provider (AWS) and IoT protocol (MQTT). We implemented two real-life architectures of cloud-based IoT environments to measure their performance. In addition, we used different numbers of IoT devices to increase the number of subscribers and publishers in order to understand how this impacts the results.

4.4 Experiment Setup

This section presents the hardware and software components used to set up our experiment. In this paper, we present several IoT-cloud implementations: (1) IoT-Cloud, (2) IoT-fog-cloud using bridge, and (3) IoT-fog-cloud using Python. The hardware and software configurations used in these implementations are discussed in the following subsections.

4.4.1 Hardware

As shown in Fig. 4.2, the experiment in this section involves two architectures. The first architecture presents IoT-fog-cloud, while the second presents IoT-cloud. In this section, we first describe

the devices that are used in the two architectures of cloud-based IoT environments, then explain the two architectures in detail. In this experiment, we used DHT11 devices [110] and Raspberry 3 Pi model B [111]. The DHT11 is a low-cost sensor device that is used to measure the temperature and humidity of the surrounding air. The purpose of the DHT11 sensor in this experiment was to generate real data for the experiment. The Raspberry Pi is a low-cost, single-board computer with built in WiFi and processing capability that is used in several domains, such as weather monitoring, smart homes, and smart health care. In this experiment, the purpose of the Raspberry Pi was to provide light computation capability to the DHT11 sensor data. In addition, it provided light storage for the DHT11 configurations. Moreover, the Raspberry Pi can be easily moved to different locations. A complete list of the hardware used in this experiment is available in Table 4.1

Table 4.1 Summary of the equipment used in the two architectures.

Equipment Name	Equipment Type	Quantity	Purpose
DHT11	temperature-humidity sensor	3	Generate real life data
Raspberry Pi	Version 3 Model B	4	Enable WiFi & provide huge processing power and storage
Micro SD Card	32GB ImageMate Plus 130 mb/s Read	4	Initial storage for the operating system and files
Monitor	HP	4	Provide a visual display
Keyboard & mice	HP	4	Useful for working on a Raspberry Pi
Power Supply/Adapter	CanaKit	4	Supply the power for the Raspberry Pi
HDMI Cable	onn	4	Connect the Raspberry Pi to a monitor

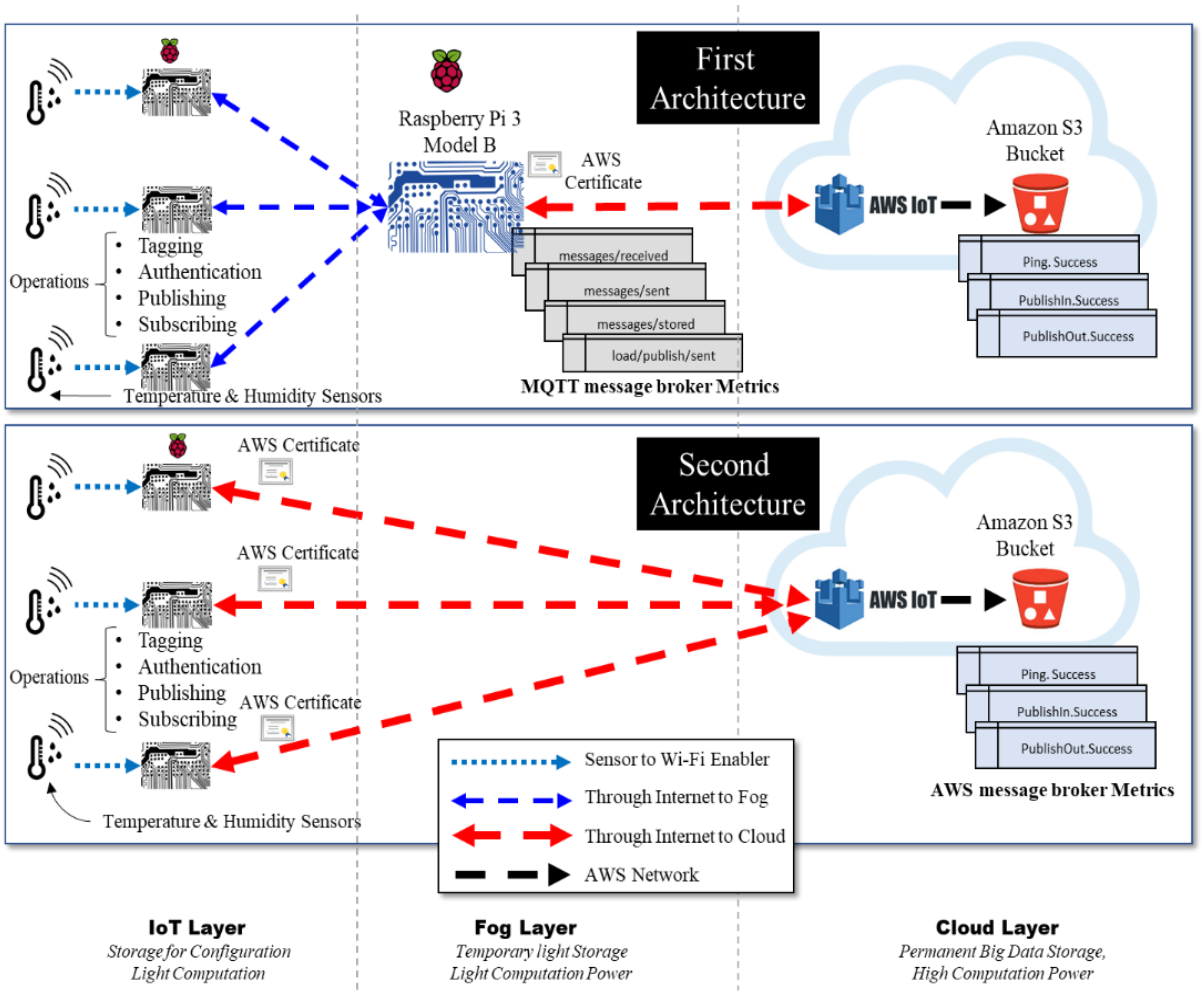


Figure 4.2 First architecture (IoT-Fog-Cloud) vs. second architecture (IoT-Cloud).

4.4.1.1 First Architecture

In the first architecture, each DHT11 sensor is connected to only one Raspberry Pi. The Raspberry Pi is used here to enable WiFi connectivity, since the DHT11 sensors are not equipped with network interfaces. Each sensor is connected to Raspberry Pi board (i.e., connectivity enabler) that is considered an IoT device in the IoT layer. Each IoT device is connected via WiFi to another Raspberry Pi board that acts as fog node in the fog layer. Communication between the IoT devices, the fog node, and the cloud uses the MQTT protocol. For this, an MQTT broker called Eclipse Mosquitto [97] is installed in the Raspberry Pi acting as a fog device. The Mosquitto MQTT broker exchanges all messages using the subscribe-publish model presented in [58]. The Mosquitto MQTT broker is also used to filter

all messages based on topics. A topic refers to an UTF-8 string that the broker (i.e., Mosquitto) uses to filter messages for each connected IoT device. Each data type (i.e., humidity, temperature) in our experiment is considered a separate topic. The data generated by the sensors and collected by the three IoT devices is transmitted over the Internet to the Raspberry Pi acting as the fog node. This Raspberry Pi, which contains the MQTT broker Mosquitto on it, is then connected over the Internet to the AWS cloud. The communication between the three layers is through the Internet. Fig. 4.3 shows the hardware used to implement the first architecture.

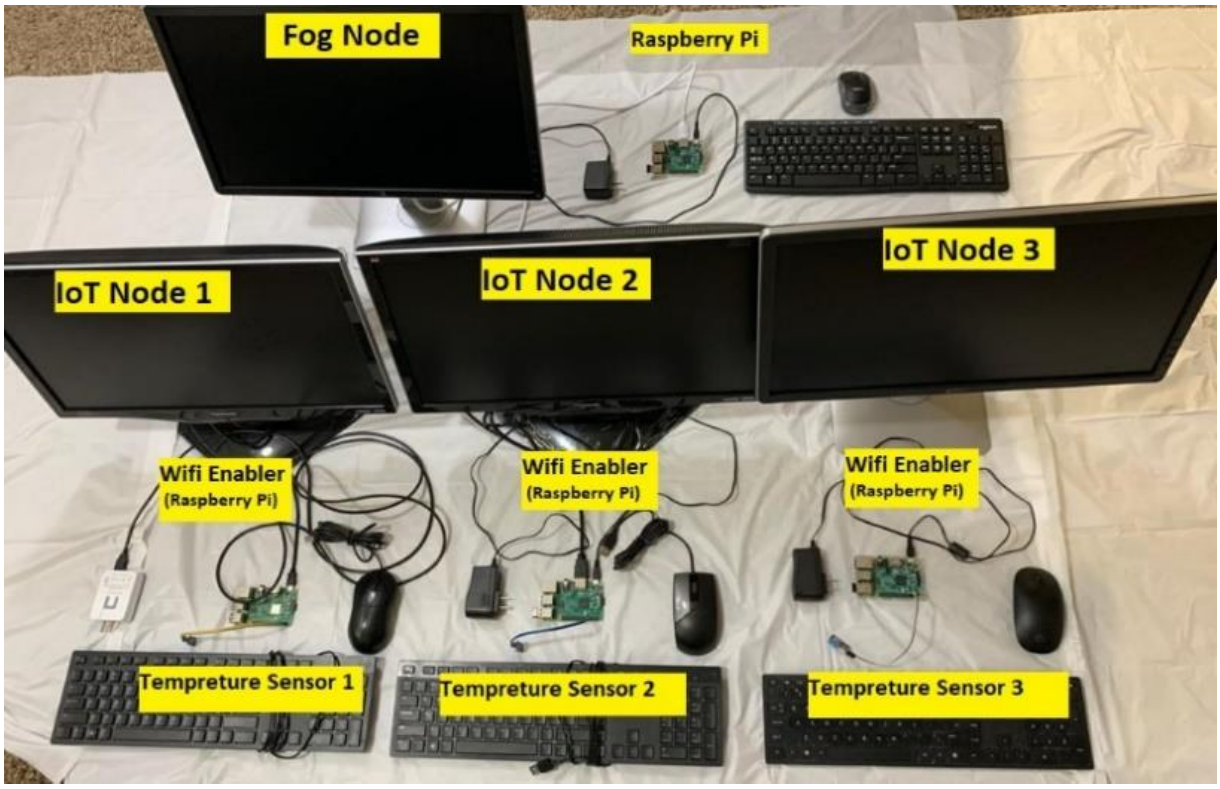


Figure 4.3 Hardware used in first architecture: IoT-Fog-Cloud.

4.4.1.2 Second Architecture

In the second architecture, all three IoT devices, consisting of a sensor and a Raspberry Pi board (i.e., connectivity enabler), are directly connected to the AWS cloud. Therefore, the huge real data received by device one, device two, or device three is forwarded wirelessly to the AWS cloud layer. The

hardware used in the second architecture is identical to that used in Fig. 4.3, except that no fog node is used in this architecture.

4.4.2 Software

We installed Arduino IDE [112] software on top of the three Raspberry Pi boards used to connect the IoT devices. Arduino IDE is a cross-platform application that is written in functions from C and C++ and is used to write and upload programs to Arduino-compatible boards like Raspberry Pi. We used Arduino IDE to read the data collected by the sensors (i.e., temperature and humidity) and then publish it over the Internet to the fog node. We used C++ scripts in Arduino IDE to perform the following operations on the IoT devices: tagging, authentication, publish, and subscribe.

4.4.2.1 First Architecture

In the first architecture, the algorithm gathers data from the IoT devices and forwards it to the fog. This involves authenticating the IoT devices to communicate with the MQTT broker on the fog device and publish data to it. The following two sections present two ways for fog nodes to communicate with the cloud.

Algorithm 1: Gather data generated from the IoT device and forward it to the fog node – First architecture

```
/* This algorithm authenticates the IoT device to the fog device, generates temperature and humidity data via
DHT11 sensors, and publishes them to the Mosquitto MQTT broker on the fog device
Define the type of DHT sensor, which is DHT11
Define the input/output pins of the Raspberry Pi to which the DHT11 is connected
Define an object of the sensor with two arguments: DHT pin and DHT type
Define the name of the network
Define the password of the network
Define the variable of MQTT broker
Define only two variables of humidity topic and temperature topic for each IoT device (DHT11 +
    Raspberry Pi) in each experiment
Create two instances of clients, one used to connect to the Internet and the other used to connect to
    the MQTT broker
Run MQTT connection, setup, loop
Function: MQTT connection
        Connect to Internet
        If (the connection is established) then
                Print "connected"
                Connect to MQTT broker on fog device
        Else (the connection isn't established) then
                        Try reconnecting to Internet
Function: setup
```

18: **Start** a serial communication at 9600 board rates
19: I**nitialize** the DHT11 sensor
20: **While** (MQTT connection is True):
21: **Read** humidity
22: **Read** temperature
23: **Print** humidity
24: **Print** temperature
25: **Publish** humidity topic with its read value to MQTT broker on fog device
26: **Publish** temperature topic with its read value to MQTT broker on fog device
27: **end while**

4.4.2.1.1 Bridging

In our fog-aided IoT implementation, the MQTT broker (i.e., Mosquitto) is installed on top of a Raspberry Pi board that serves as a fog node. In such cases, the MQTT broker needs to be very close to where the sensors are deployed. The Mosquitto MQTT broker has a built-in capability that allows the received data to be sent directly to the cloud (AWS IoT Core) by specifying the address of the AWS IoT core service used. This operation is called bridging. (Please see [113] for more information.) The following algorithm illustrates the connection of IoT devices to the AWS IoT core using a bridge connection.

Algorithm 2: Bridge every message received from the IoT devices based on the topics of the messages to the AWS broker on the cloud

/* This algorithm authenticates the IoT device to the MQTT broker on the fog device, bridges to the data received from the sensors via the MQTT broker, filters them based on topics, authenticates to the AWS via certificates, and then publishes the filtered data to the AWS IoT core service
1: **Define** the variable of endpoint of Amazon Web Service with port number 8883
2: **Determine** which topics of the messages to bridge to AWS
3: **Define** the version of the protocol to be used between the MQTT broker and the AWS broker
4: **Create** one instance of client to be used over the MQTT protocol
5: **Define** the name of the bridge connection
6: **Start** Connection
7: **Configure** the bridge using SSL/TLS support
8: **Define** bridge_cafile to hold the path of Amazon Root CA certificate
9: **Define** bridge_certfile to hold the path of Amazon certificate
10: **Define** bridge_keyfile to hold the path of Private key

4.4.2.1.2 Python Script

Using the same Mosquitto MQTT broker, we developed a Python script on the fog device to replace the Mosquitto built-in bridging capability to simultaneously authenticate the MQTT broker and

AWS, receive the data from the sensors, and filter and publish the data received to the AWS IoT core service. This Python script provides more flexibility for future improvements in security and performance. The following algorithm depicts the operations implemented in our Python script. This algorithm is used to authenticate IoT devices to fog devices, gather the temperature and humidity data from the DHT11 sensors, and publish the data to the Mosquitto MQTT broker on the fog device. The algorithm also authenticates the fog node to interact with the AWS IoT Core (i.e., cloud). Afterward, the algorithm filters the collected data and publishes it to the cloud.

Algorithm 3: Receive data from the IoT devices and forward it to the cloud

/* This algorithm authenticates the IoT device to the MQTT broker on the fog device, subscribes to the data received from the sensors via the MQTT broker, filters the data based on topics, authenticates to the AWS via certificates, and then publishes the filtered data to the AWS IoT core service

1: **Define** two variables (humidity topic and temperature topic) for each IoT device in each experiment
2: **Define** the variables of MQTT broker and MQTT port
3: **Create** two instances of clients, one used for the MQTT broker and the other used for the AWS broker
4: **Connect** the first client to the MQTT broker using the IP address of the fog device and MQTT port
5: **Create a loop_start()** method to start a new thread for the first client
6: **Set** the transport layer security (TLS) for the second client using the three paths of AWS certificates and the current version of MQTT protocol
7: **Connect** the second client to AWS broker using AWS Endpoint and AWS port
8: **Create** a **loop_start ()** method to start a new thread for the second client
9: **While** True **do**
10: **Define** a connection function
11: **Subscribe** for all topics in each IoT device
12: **Print** "connected" when the connection is established
13: **Print** "error" when the connection is disconnected
14: **Define** a message function
15: **Get** topic of the message
16: **Get** payload of the message
17: **Print** topic of the message
18: **Print** payload of the message
19: **Publish** topic and payload of the message to AWS cloud
20: **Make** the first client execute the two functions: (1) connection, and (2) message
21: **Make** the second client execute the message function to publish the recovered data to AWS cloud
22: **end while**

4.4.2.2 Second Architecture

In the second architecture, Arduino IDE is installed on top of the three Raspberry Pi boards used to connect the IoT devices. Arduino IDE is used to read the data collected by the sensors (i.e., temperature and humidity) and then publish it over the Internet directly to the cloud. This operation requires

authentication to the cloud before the data can be published. The process to authenticate the IoT device (i.e., DHT11 + Raspberry Pi) to access the cloud (AWS IoT Core) using the AWS certificate and then to publish the data generated by the sensors to the cloud is illustrated in the Algorithm 4.

Algorithm 4: Gather data generated from the IoT device and forward it to the cloud – Second architecture

```
/* This algorithm authenticates the IoT device to the AWS cloud via certificates, generates temperature and
humidity data via the DHT11 sensors, and publishes them to the AWS cloud
Define the type of DHT sensor, which is DHT11
Define the input/output pins of the Raspberry Pi to which the DHT11 is connected
Define an object of the sensor with two arguments, DHT pin and DHT type
Define the name of the network
Define the password of the network
Define the variable of endpoint of Amazon Web Service
Define only two variables (humidity topic and temperature topic) for each IoT device (DHT11 +
Raspberry Pi) in each experiment
Create a client to connect to AWS using AWS endpoint and port number 8883
Run connection, setup, loop
Function: connection
        Connect to Internet
        If (the connection is established) then
                Print "connected"
        Else (the connection isn't established) then
                Try reconnecting to Internet
Function: setup
        Start a serial communication at 9600 board rates
        Initialize the DHT11 sensor
        Run a connection function
        Convert the AWS certificates to .der Format
        Open certificate
        If (certificate is existing)
                Load certificate
        Else
                Print "certificate is not existing"
        Open Amazon Root CA certificate
        If (Amazon Root CA certificate is existing)
                Load Amazon Root CA certificate
        Else
                Print "Amazon Root CA certificate is not existing"
        Open private key
        If (private key is existing)
                Load private key
        Else
                Print "Private key is not existing"
While (connection is True):
        Read humidity
        Read temperature
        Print humidity
```

40: **Print** temperature
41: **Publish** humidity topic with its read value to MQTT broker on fog device
42: **Publish** temperature topic with its read value to MQTT broker on fog device
43: **end while**

To monitor the Mosquitto MQTT broker, the following algorithm illustrates how $SYS-Topics is used to provide metrics.

Algorithm 5: Monitoring script on fog device

/* This algorithm connects to the MQTT broker, subscribes to SYS-Topics via the MQTT broker to monitor and provide benchmark metrics of the Mosquitto broker, and publishes the results based on these metrics

1: **Create** one instance of clients to connect to Mosquitto broker
2: **Connect** the client to MQTT broker using the IP address of fog device and MQTT port
3: **Define** a connection function
4: **Print** "connected" when the connection is established
5: **Print** "error" when the connection is disconnected
6: **Subscribe** to $SYS/# topics to monitor the mosquito MQTT broker (print the metrics results of Mosquitto broker on fog device
7: **Make** the client execute the connection function
8: **Create** a **loop forever()** method for the client to remain monitoring the Mosquitto

4.5 Descriptions of Metrics

Many metrics can be used to measure the performance in IoT systems. The following subsections describe the metrics that we utilized to measure the performance in the cloud and fog layers.

4.5.1 Cloud Layer: AWS IoT Metrics

Many metrics can be used to measure the performance of cloud-based IoT systems from the cloud layer. Since we used the AWS IoT as the cloud service provider for our experiments, we used Amazon CloudWatch to measure the performance. Amazon CloudWatch has been used in the literature to monitor performance [114]–[116]. Amazon CloudWatch processes and analyzes data in real time and provides the following metrics to measure our two architectures of cloud-based IoT environments:

1) Connect.Success:

This metric is used to collect the number of successful connections from our IoT nodes or fog nodes to the AWS message broker.

2) Ping.Success:

This metric is used to collect the number of ping messages received by the AWS message broker. These ping messages are received from the fog node(s) in the first architecture and from the IoT node(s) in the second architecture.

3) PublishIn.Success:

This metric is used to collect the number of publish requests successfully processed by the AWS message broker. Like the ping messages, these messages are received from the fog node(s) in the first architecture and from the IoT node(s) in the second architecture.

4) PublishOut.Success:

This metric is used to collect the number of publish requests successfully made by the AWS message broker to the fog nodes in the first architecture and to the IoT nodes in the second architecture.

5) Subscribe.Success:

This metric is used to collect the number of successful subscribe requests processed by the AWS message broker. These requests are made by the fog node in the first architecture and made directly by the IoT devices in the second architecture.

6) PublishIn.Clienterror:

This metric is used to collect the number of publish requests rejected because they did not meet the AWS IoT requirements.

7) Unsubscribe.Success:

This metric is used to collect the number of unsubscribe requests that were successfully processed by the AWS message broker. These unsubscribe requests are made by the fog node in the first architecture and made directly by the IoT devices in the second architecture.

8) Throttle.Exceeded:

This metric is used to collect the number of requests that were throttled because the client (i.e., the IoT node or fog node) has sent too many messages and exceeded the allowed message rate.

9) PublishOut.Throttle:

This metric is used to collect the number of publish requests that were throttled because the client (i.e., IoT node or fog node) exceeded the allowed message rate.

4.5.2 Fog Layer: Eclipse Mosquitto Broker Metrics

Likewise, many metrics can be used to measure the performance of the fog layer. MQTT has been used in the literature as a lightweight protocol to communicate between messages [104], [105], [117]. Since we used the MQTT protocol for the message broker on the fog node, we utilized Eclipse Mosquitto [97] in our experiments. Eclipse Mosquitto [97] provides some metrics as $SYS topics, which are described as follows:

1) $SYS/broker/uptime:

This metric is used to measure the amount of time in seconds the broker has been online.

2) $SYS/broker/load/messages/received:

This metric is used to measure the moving average of the number of all types of MQTT messages received by the broker over different time intervals.

3) $SYS/broker/load/messages/sent:

This metric is used to measure the moving average of the number of all types of MQTT messages sent by the broker over different time intervals.

4) $SYS/broker/load/publish/received:

This metric is used to measure the moving average of the number of publish messages received by the broker over different time intervals.

5) $SYS/broker/load/publish/sent:

This metric is used to measure the moving average of the number of publish messages sent by the broker over different time intervals.

6) $SYS/broker/load/bytes/received:

This metric is used to measure the moving average of the number of bytes received by the broker over different time intervals.

7) $SYS/broker/load/bytes/sent:

This metric is used to measure the moving average of the number of bytes sent by the broker over different time intervals.

8) $SYS/broker/load/sockets:

This metric is used to measure the moving average of the number of socket connections opened to the broker over different time intervals.

9) $SYS/broker/load/connections:

This metric is used to measure the moving average of the number of CONNECT packets received by the broker over different time intervals.

10) $SYS/broker/messages/stored:

This metric is used to measure the number of messages currently held in the message store. This includes retained messages and messages queued for durable clients.

11) $SYS/broker/store/messages/bytes:

This metric is used to measure the number of bytes currently held by message payloads in the message store. This includes retained messages and messages queued for durable clients.

12) $SYS/broker/subscriptions/count:

This metric is used to measure the total number of subscriptions active on the broker.

13) $SYS/broker/heap/current:

This metric is used to measure the current size of the heap memory in use by Mosquitto.

14) $SYS/broker/messages/received:

This metric is used to measure the total number of messages of any type received since the broker started.

15) $SYS/broker/messages/sent:

This metric is used to measure the total number of messages of any type sent since the broker started.

16) $SYS/broker/publish/messages/received:

This metric is used to measure the total number of PUBLISH messages received since the broker started.

17) $SYS/broker/publish/messages/sent:

This metric is used to measure the total number of PUBLISH messages sent since the broker started.

18) $SYS/broker/bytes/received:

This metric is used to measure the total number of bytes received since the broker started.

19) $SYS/broker/bytes/sent:

This metric is used to measure the total number of bytes sent since the broker started.

20) $SYS/broker/publish/bytes/received:

This metric is used to measure the total number of PUBLISH bytes received since the broker started.

21) $SYS/broker/publish/bytes/sent:

This metric is used to measure the total number of PUBLISH bytes sent since the broker started.

4.6 Analysis Methods

We used two benchmark metrics to analyze the performance of the two IoT architectures implemented in this paper. In the two architectures of cloud-based IoT environments, we set the number of subscribes and publishes to two for each device. This is because the IoT devices (i.e., the DHT11 sensors) generate two types of data: (1) temperature and (2) humidity. Therefore, as the number of sensor devices increase, the number of subscribes and publishes should also increase. This provides more accurate, consistent, and real results about the environment performance and scalability. In this section, we present the methods that we used to perform the experiment on both architectures.

4.6.1 Architecture 1 vs. Architecture 2

The performance of the two architectures of cloud-based IoT environments was analyzed using the AWS benchmark, and the results obtained were compared. The experiment was to be performed using a different number of IoT devices each time (1, 2, or 3 IoT devices) to compare and analyze the results in order to show the impact of the fog layer in the first architecture on IoT environments. Fig. 4.2 shows the experiment setup and metrics applied to measure the performance of the two cloud-based IoT environments. AWS, like any IoT platform provider, requires that any device be authenticated before communicating with it. AWS uses certificates to authenticate devices. As shown in Fig. 4.2, the location where the certificate is stored is different in the two architectures due the structure of the environments. Since the first architecture has a fog layer between AWS and the IoT devices, the certificate is stored in the fog device (i.e., the Raspberry Pi board serving as the fog layer). In the second architecture, however, since the IoT devices are directly connected to AWS, the certificates are stored in each IoT device.

4.6.2 Architecture 1 Implementation: Python Script vs. Bridging

The first architecture of the IoT-fog-cloud layer was implemented using two different experiments, as shown in Fig 4.4 The first experiment is based on a Python script shown in Algorithm 3 that manually receives, filters, and forwards the messages to the cloud. The second experiment (Algorithm 2) bridges all of the messages received from the IoT devices based on their topics to AWS. The purpose of using the bridge in the first architecture is to connect two brokers, the MQTT message broker “mosquitto” and the AWS message broker, to exchange messages based on the different topics and to validate the results of the first architecture. AWS benchmarks were used to analyze the performance of the first architecture in two different implementations. Although bridging offers faster implementation, the Python script provides more flexibility to add features to optimize performance. The objective of this analysis is to show the impact of both implementations.

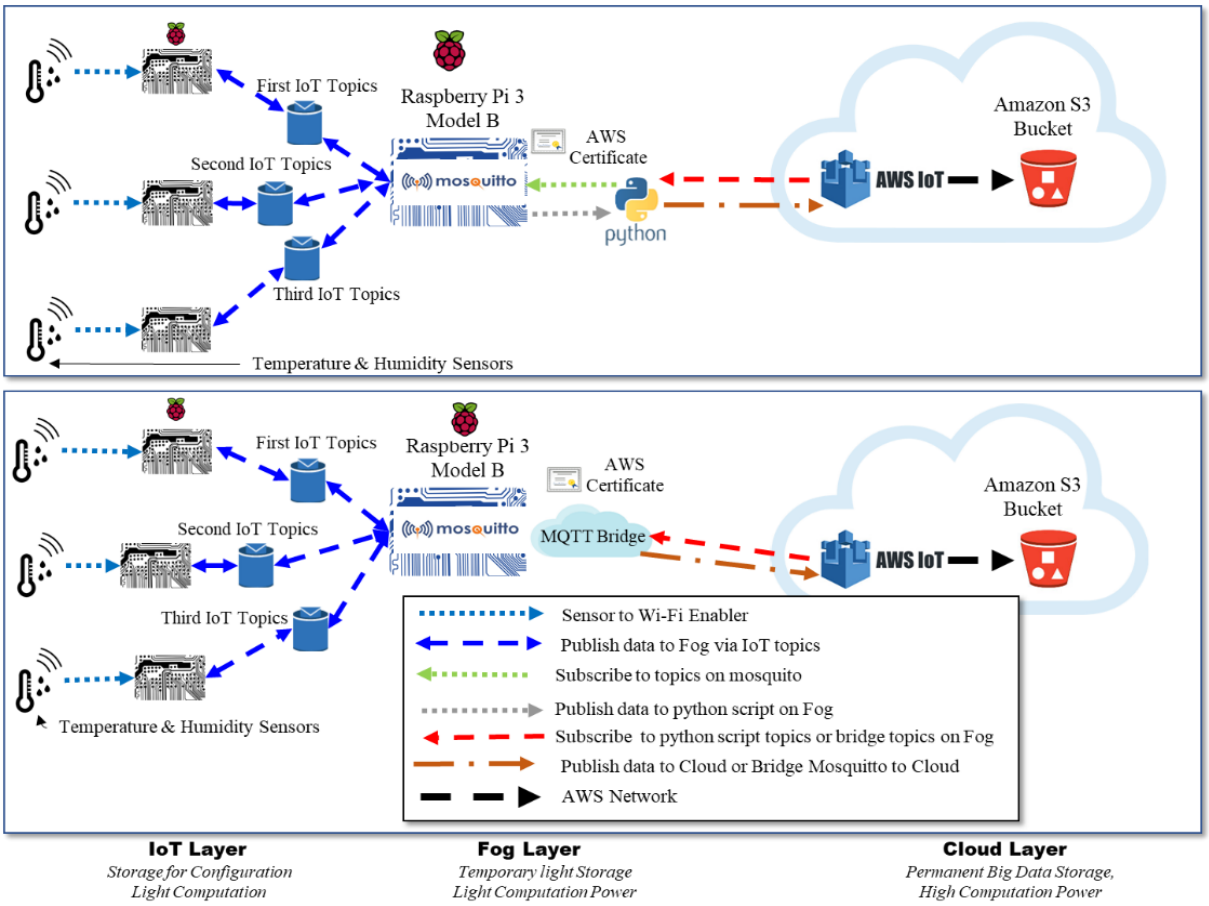


Figure 4.4 IoT-Fog-Cloud Architecture using two methods: Python script and MQTT bridging.

4.6.3 Architecture 1 Measurement: Mosquitto Metrics vs. AWS Metrics

The performance of the first architecture (i.e., IoT-fog-cloud) will be measured from both the fog side and the cloud side. The fog-side metrics (i.e., Mosquitto Broker Metrics) were measured over different durations (30 seconds, 1 minute, 5 minutes, 15 minutes, 30 minutes, 45 minutes, and 1 hour), while the cloud-side metrics (i.e., AWS metrics) were measured over 30 seconds, 1 minute, 5 minutes, 15 minutes, and 1 hour. (Due to the AWS platform constraints, it was difficult to unify the experiment durations.) Fig. 4.5 presents the different metrics used to measure the performance of the IoT environment that uses a fog layer from two sides. Fig 4.5 (a) presents the metrics applied to measure performance of the IoT environment at the fog layer, whereas Fig 4.5 (b) presents the metrics applied to measure the

performance of the IoT environment at the cloud layer. This was done to ensure that the performance of the IoT environment remains consistent in the fog and cloud layers using different setups.

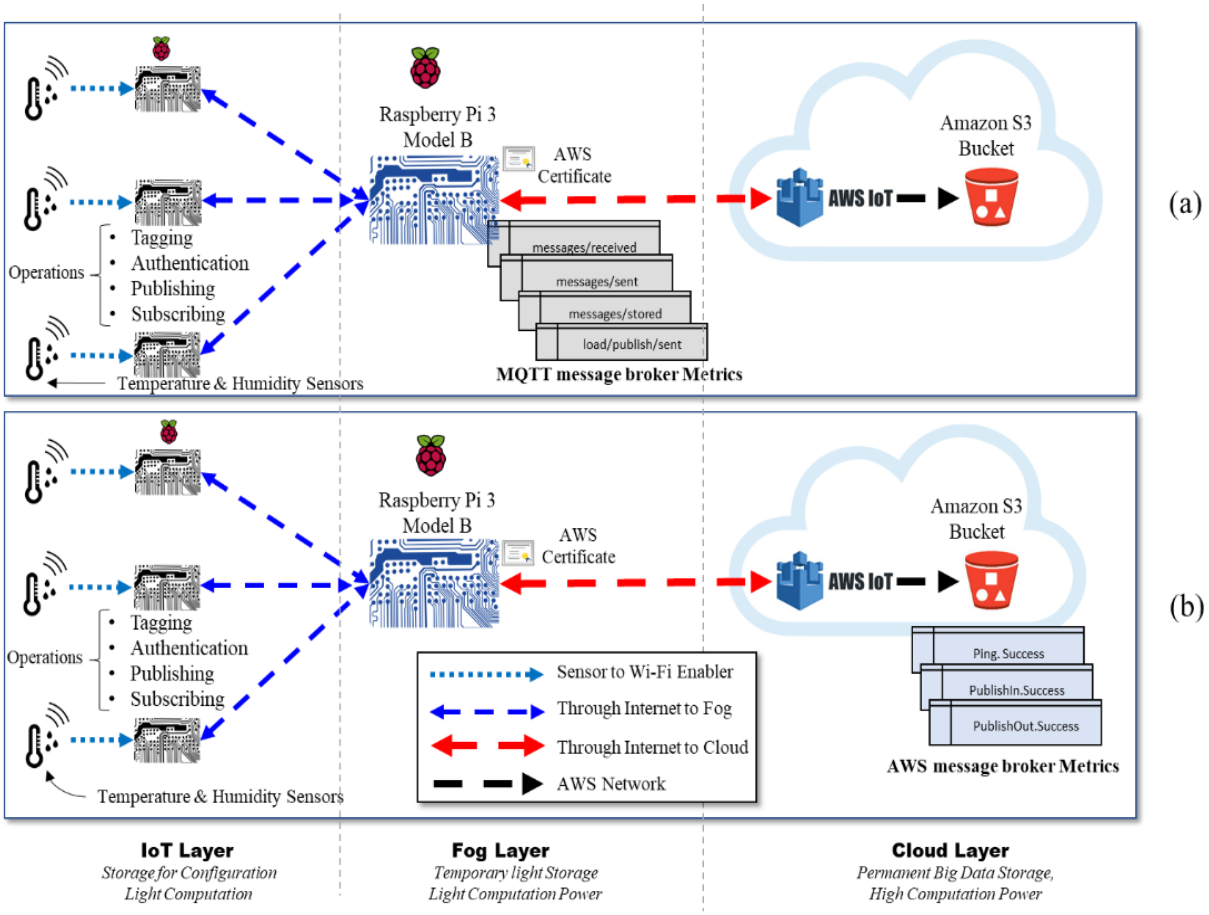


Figure 4.5 (a) Metrics applied in the fog layer vs. (b) metrics applied in the cloud layer.

4.7 Results and Description of Experiments

In this section, the results of the two architectures of cloud-based IoT environments will be analyzed using the two benchmarks metrics (Mosquitto and AWS) based on the selected architecture using one, two, or three IoT devices with increasing numbers of subscribing and publishing requests. In the following sections, the benchmark metrics of each architecture are described in detail, along with observations.

4.7.1 Results: Description of the three experiments of the first architecture with one, two, or three IoT devices using AWS benchmark metrics (cloud layer)

The Amazon CloudWatch monitor has a variety of metrics to analyze the cloud layer that were used in the experiments on the first architecture, as shown in Tables 4.2 and 4.3. The AWS metrics are: (1) Connect.Success, (2) Ping.Success, (3) PublishIn.Success, (4) PublishOut.Success, (5) Subscribe.Success, (6) Unsubscribe.Success, (7) PublishIn.Clienterror, (8) Throttle.Exceeded, and (9) PublishOut.Throttle. These metrics are explained in detail in section 5.1, and the metrics were collected for one hour, starting from 30 seconds, as shown in Tables 4.2 and 4.3. Before analyzing the results obtained using the AWS metrics, there are a number of assumptions that should be taken in consideration. First, the number of successful connections from either IoT devices or fog devices using the Connect.Success metric must be equal to the number of subscribe requests received from either fog devices or IoT devices using the Subscribe.Success metric because the loss of connections will lead to the loss of subscribe requests in each device and the subscriptions will be renewed automatically once the connection is reestablished. Second, the number of unsubscribe requests received from either IoT devices or fog devices using the Unsubscribe.Success metric must reflect the number of subscribing and publishing requests generated in each IoT device. In our experiments, there are three IoT devices, and each of them generates two types of data: temperature and humidity. Therefore, the number of IoT devices is three and the number of all subscribing and publishing requests is six. Third, the number of publish requests received from either fog devices or IoT devices using the Publishin.Success metric should be close to or the same as the number of publish requests made by AWS to either fog devices or IoT devices using the Publishout.Success metric.

The experiments using the first architecture, which includes a fog layer, were conducted with one, two, or three IoT devices for different time periods (30 seconds, 1 minutes, 5 minutes, 15 minutes, and 1 hour), as shown in Table 4.2 The first experiment used one IoT device attached to one fog device, and the results showed that the Connect.Success and Subscribe.Success numbers reflect the number of

subscribing and publishing requests for one IoT device with two subscribe and publish requests. This is because the connection is not disconnected and the subscribe request is not lost from either the IoT or the fog device. Moreover, there is only one IoT device connected to only one fog device, which is then authenticated to the AWS cloud, and no other devices interrupted them. In addition, the PublishIn.Success and PublishOut.Success numbers are expected and stable because the data came from the IoT device and were filtered based on topic on the fog device, then sent to the AWS cloud. Thus, it takes time for the data to be transferred between the three layers of IoT, fog, and cloud. In the second experiment, on the other hand, two IoT devices were attached to the fog device, as shown in Table 4.2 The results indicated that the Connect.Success and Subscribe.Success numbers were equal to the number of subscribing and publishing requests for both IoT devices, since two IoT devices are linked to one fog device with four subscribing and publishing requests. Therefore, the PublishIn.Success and PublishOut.Success numbers are significantly increased with two IoT devices compared to the previous experiment using only one IoT device attached to one fog device. This is because two IoT devices are connected to only one fog device serving as the middle layer between the IoT devices and the AWS cloud that is used to authenticate to the AWS cloud. Thus, the fog device increased the number of messages published to the AWS cloud and reduced the number of IoT devices that need to be authenticated to the AWS cloud. The third experiment used three IoT devices connected to one fog device, as shown in Table 4.2 The Connect.Success and Subscribe.Success numbers still reflected the actual number of subscribing and publishing requests for all three IoT devices, even though there was only one fog device. Thus, the PublishIn.Success and PublishOut.Success numbers were significantly increased due to the high volume of published messages from the three IoT devices. Therefore, the fog device made a significantly impact in filtering and transferring the actual volume of data from the three IoT devices to the AWS cloud and decreasing the number of authentications required for the AWS cloud.

4.7.2 Results: Description of the three experiments on the second architecture with one, two, or three IoT devices using AWS benchmark metrics (cloud layer)

In Table 4.2, it can be seen that for one IoT device, the Connect.Success and Subscribe.Success numbers are low because there is only one IoT device with two subscribe and publish requests authenticated to the AWS, and no other IoT devices are attached with it. Thus, the connection and subscription are established one time. In addition, the PublishIn.Success and PublishOut.Success numbers are consistent and high due to the single authenticated IoT device. With two IoT devices with four subscribe and publish requests, however, the Connect.Success and Subscribe.Success numbers increase because there are two IoT devices attempting to authenticate to the AWS cloud at one time, which increases the number of connections and subscriptions. The Publishin.Success and PublishOut.Success numbers have not substantially changed compared to the previous experiment even though the number of subscribe and publish requests increased with the second IoT device added because the second IoT device tried to authenticate to the AWS cloud while the first IoT device tried to publish messages to the AWS cloud; as a result, publish messages were lost in each period. The third experiment, using three IoT devices with six subscribe and publish requests, shows that the number of connections and subscriptions increases while the PublishIn.Success and PublishOut.Success numbers become low. This is because the three IoT devices tried to authenticate to the

AWS cloud at once, causing a huge number of connections and subscriptions request.

Table 4.2 AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using bridge – First Architecture vs. Second Architecture

AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using bridge – First Architecture						AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) – Second Architecture					
Number of IoT devices	1					Number of IoT devices	1				
Number of subscribing & publishing	2					Number of subscribing & publishing	2				
AWS IoT metrics in minutes (m)	0.5	1	5	15	60	AWS IoT metrics in minutes (m)	0.5	1	5	15	60
Connect.Success	2	2	2	2	2	Connect.Success	2	2	2	2	2
Ping.Success	2	2	8	29	120	Ping.Success	5	5	25	75	297
PublishIn.Success	44	44	206	586	2360	PublishIn.Success	60	60	300	900	3580
PublishOut.Success	44	44	206	586	2360	PublishOut.Success	60	60	300	900	3580
Subscribe.Success	2	2	2	2	2	Subscribe.Success	2	2	2	2	2
Unsubscribe.Success	2	2	2	2	2	Unsubscribe.Success	2	2	2	2	2

AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using bridge – First Architecture						AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) – Second Architecture					
Number of IoT devices	2					Number of IoT devices	2				
Number of subscribing & publishing	4					Number of subscribing & publishing	4				
AWS IoT metrics in minutes	0.5	1	5	15	60	AWS IoT metrics in minutes	0.5	1	5	15	60
Connect.Success	4	4	4	4	4	Connect.Success	6	6	35	111	432
Ping.Success	2	2	8	29	120	Ping.Success	1	1	8	20	85
PublishIn.Success	70	70	350	1110	4590	PublishIn.Success	62	62	322	965	3850
PublishOut.Success	70	70	350	1110	4590	PublishOut.Success	51	51	322	965	3850
Subscribe.Success	4	4	4	4	4	Subscribe.Success	6	6	35	111	432
Unsubscribe.Success	4	4	4	4	4	Unsubscribe.Success	4	4	4	4	4
AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using bridge – First Architecture						AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) – Second Architecture					
Number of IoT devices	3					Number of IoT devices	3				
Number of subscribing & publishing	6					Number of subscribing & publishing	6				
AWS IoT metrics in minutes	0.5	1	5	15	60	AWS IoT metrics in minutes	0.5	1	5	15	60
Connect.Success	6	6	6	6	6	Connect.Success	12	12	76	228	893
Ping.Success	2	2	9	29	119	Ping.Success	2	1	5	14	60
PublishIn.Success	115	115	548	1660	6510	PublishIn.Success	68	68	359	1050	4240
PublishOut.Success	115	115	548	1660	6510	PublishOut.Success	58	68	359	1050	4240
Subscribe.Success	6	6	6	6	6	Subscribe.Success	12	12	76	228	893
Unsubscribe.Success	6	6	6	6	6	Unsubscribe.Success	6	6	6	6	6

Table 4.3 AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using bridge vs. Python – First Architecture

AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using Python script – First Architecture						AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using bridge – First Architecture					
Number of IoT devices	1					Number of IoT devices	1				
Number of subscribing & publishing	2					Number of subscribing & publishing	2				
AWS IoT metrics in minutes	0.5	1	5	15	60	AWS IoT metrics in minutes	0.5	1	5	15	60
Connect.Success	2	2	2	2	2	Connect.Success	2	2	2	2	2
Ping.Success	2	2	10	30	119	Ping.Success	2	2	8	29	120
IoT PublishIn.Success	46	46	206	590	2360	IoT PublishIn.Success	44	44	206	586	2360
IoT PublishOut.Success	46	46	206	590	2360	IoT PublishOut.Success	44	44	206	586	2360
Subscribe.Success	2	2	2	2	2	Subscribe.Success	2	2	2	2	2
IoT Unsubscribe.Success	2	2	2	2	2	IoT Unsubscribe.Success	2	2	2	2	2

AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using Python script – First Architecture						AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using bridge – First Architecture					
Number of IoT devices	2					Number of IoT devices	2				
Number of subscribing & publishing	4					Number of subscribing & publishing	4				
AWS IoT metrics in minutes	0.5	1	5	15	60	AWS IoT metrics in minutes	0.5	1	5	15	60
Connect.Success	4	4	4	4	4	Connect.Success	4	4	4	4	4
Ping.Success	2	2	9	29	118	Ping.Success	2	2	8	29	120
IoT PublishIn.Success	76	76	359	1170	4680	IoT PublishIn.Success	70	70	350	1110	4590
IoT PublishOut.Success	76	76	359	1170	4680	IoT PublishOut.Success	70	70	350	1110	4590
Subscribe.Success	4	4	4	4	4	Subscribe.Success	4	4	4	4	4
IoT Unsubscribe.Success	4	4	4	4	4	IoT Unsubscribe.Success	4	4	4	4	4

AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using Python script – First Architecture						AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using bridge – First Architecture					
Number of IoT devices	3					Number of IoT devices	3				
Number of subscribing & publishing	6					Number of subscribing & publishing	6				
AWS IoT metrics in minutes	0.5	1	5	15	60	AWS IoT metrics in minutes	0.5	1	5	15	60
Connect.Success	6	6	6	6	6	Connect.Success	6	6	6	6	6
Ping.Success	1	2	10	30	118	Ping.Success	2	2	9	29	119
IoT PublishIn.Success	117	117	534	1680	6650	IoT PublishIn.Success	115	115	548	1660	6510
IoT PublishOut.Success	117	117	534	1680	6650	IoT PublishOut.Success	115	115	548	1660	6510
Subscribe.Success	6	6	6	6	6	Subscribe.Success	6	6	6	6	6
IoT Unsubscribe.Success	6	6	6	6	6	IoT Unsubscribe.Success	6	6	6	6	6

4.7.3 Results: Description of the three experiments of the first architecture with one, two, or three IoT devices using Mosquitto benchmark metrics (fog layer)

The MQTT Mosquitto broker using SYS-Topics has several metrics that were used in the experiments on the first architecture from the fog layer, as shown in Tables 4.4, 4.5, and 4.6. The Mosquitto broker metrics are: (1) load/messages/received, (2) load/messages/sent, (3) load/publish/received, (4) load/publish/sent, (5) load/bytes/received, (6) load/bytes/sent, (7) load/sockets, (8) load/connections, (9) messages/stored, (10) store/messages/bytes, (11) subscriptions/count, (12) heap/current, (13) messages/received, (14) messages/sent, (15) publish/messages/received, (16) publish/messages/sent, (17) bytes/received, (18) bytes/sent, (19) publish/bytes/received, and (20) publish/bytes/sent. These metrics are described in detail in section 5.2 and were obtained over one hour,

starting from 30 seconds, as shown in Tables 4.4, 4.5, and 4.6. Before analyzing the Mosquitto broker metrics on the fog device, a number of hypotheses should be discussed. First, the number of subscribe requests received from each IoT device using the subscription/count metric must reflect the number of subscribe and publish requests established in each IoT device. Second, the number of publish messages received from each IoT device in the first architecture using the publish/messages/received metric should be equal to the number of publish messages received from the fog device using the Publishin.Success metric. Third, the number of publish messages sent by the Mosquitto broker on the fog device using the publish/messages/sent metric should increase as the number of IoT devices increases due to the broker capabilities and the high computational power of the fog device to process and publish many messages. The experiments on the first architecture, which has a fog device between the IoT device(s) and the AWS cloud, is performed using one, two, or three IoT devices for different periods of 30 seconds, 1 minute, 5 minutes, 15 minutes, 30 minutes, 45 minutes, and 1 hour. The first experiment used one IoT device attached to the fog device connected to the AWS cloud, as shown in Table 4.4. The results showed that the number of subscribe requests using the subscriptions/count metric on the fog device reflects the number of subscribe and publish requests of the single IoT device with two subscribe and publish requests. This is because the fog device is subscribed to the topics of each message received from IoT devices and filters them based on the topics of the messages, then publishes them to the AWS cloud. Therefore, the message is identified by its topic in each layer. In addition, the number of publish messages made by the fog device (publish/messages/sent) is high even though there is only one IoT device with two subscribe and publish requests. This is because the fog device has high capabilities to filter and publish many messages since it is located close to the IoT device that is generating the real data. Moreover, the size of the heap memory used by Mosquitto on the fog device is stable over different durations from 30 seconds to 1 hour. The second experiment was conducted using two IoT devices with four subscribe and publish requests attached to the fog device, which connects to the AWS cloud, as shown in Table 4.5. The results showed that the number of subscribe requests increased since the number of IoT devices increased to two. In addition, the number of publish messages made by the fog device

(publish/messages/sent) significantly increased because two IoT devices with four subscribe and publish requested were attached. The third experiment was conducted using three IoT devices with six subscribe and publish requests attached to a fog device and then to the AWS cloud, as shown in Table 4.6. The results demonstrated that the number of subscribe requests was six when the number of IoT devices increased to three. In addition, the number of publish messages made by the fog device (publish/messages/sent) became significantly higher when the number of IoT devices increased to three.

Table 4.4 Mosquitto message broker metrics on fog layer using bridge – First Architecture– One IoT device

Number of IoT devices	One IoT device						
Number of subscribing & publishing	Two subscribing & two publishing						
Mosquitto message broker metrics as $SYS topics	***30 (seconds)***	***1 (minutes)***	***5 (minutes)***	***15 (minutes)***	***30 (minutes)***	***45 (minutes)***	***1 (hours)***
$SYS/broker/load/messages/received 1(m)	25.31	36.26	27.43	46.25	47.81	25.96	44.70
$SYS/broker/load/messages/sent 1(m)	137.87	177.35	179.52	215.36	199.56	188.81	213.35
$SYS/broker/load/publish/received 1(m)	22.68	33.07	23.95	43.88	45.38	23.43	42.69
$SYS/broker/load/publish/sent 1(m)	135.23	174.15	176.03	213.00	197.14	186.28	211.34
$SYS/broker/load/bytes/received 1(m)	802.93	1135.98	849.26	1478.98	1524.90	791.00	1439.38
$SYS/broker/load/bytes/sent 1(m)	5315.03	6898.23	7230.76	8637.79	8029.54	7694.46	8610.21
$SYS/broker/load/sockets 1(m)	1.05	0.61	0.92	0.09	0.12	0.94	0.12
$SYS/broker/load/connections 1(m)	1.05	0.61	0.92	0.09	0.12	0.94	0.12
$SYS/broker/subscriptions/count	2	2	2	2	2	2	2
$SYS/broker/heap/current	30224	30232	30360	30148	30084	29744	30132
$SYS/broker/messages/received	37	65	225	663	1285	1885	2515
$SYS/broker/messages/sent	202	329	1144	3250	6356	9416	12552
$SYS/broker/publish/messages/received	32	58	209	625	1213	1787	2375
$SYS/broker/publish/messages/sent	197	322	1128	3212	6284	9310	12412
$SYS/broker/bytes/received	1154	2029	7149	21219	41165	60374	80633
$SYS/broker/bytes/sent	7730	12711	45329	129895	255029	378636	505171
$SYS/broker/publish/bytes/received	160	290	1045	3125	6065	8935	11875
$SYS/broker/publish/bytes/sent	806	1400	5505	16551	33192	49748	66818

Table 4.5 Mosquitto message broker metrics on fog layer using bridge – First Architecture– Two IoT devices

Number of IoT devices	Two IoT devices						
Number of subscribing & publishing	Four subscribing & four publishing						
Mosquitto message broker metrics as $SYS topics	***30 (seconds)***	***1 (minutes)***	***5 (minutes)***	***15 (minutes)***	***30 (minutes)***	***45 (minutes)***	***1 (hours)***
$SYS/broker/load/messages/received 1(m)	59.48	79.02	109.81	119.86	109.03	123.12	114.70
$SYS/broker/load/messages/sent 1(m)	175.38	226.02	296.83	312.57	296.21	307.21	298.85
$SYS/broker/load/publish/received 1(m)	38.21	53.18	73.62	79.78	74.34	82.05	78.49
$SYS/broker/load/publish/sent 1(m)	154.11	200.18	260.63	272.49	261.52	266.13	262.63
$SYS/broker/load/bytes/received 1(m)	2124.60	2767.93	3916.20	4291.54	3854.92	4393.78	4074.59
$SYS/broker/load/bytes/sent 1(m)	5981.40	7807.80	10344.23	10803.61	10424.09	10568.27	10457.24
$SYS/broker/load/sockets 1(m)	19.69	23.40	33.62	37.64	32.03	38.43	34.27
$SYS/broker/load/connections 1(m)	19.69	23.40	33.64	37.65	32.12	38.43	34.27
$SYS/broker/subscriptions/count	4	4	4	4	4	4	4
$SYS/broker/heap/current	30268	30268	30252	30268	30252	30260	30204
$SYS/broker/messages/received	84	143	593	1747	3433	5194	6954
$SYS/broker/messages/sent	253	416	1632	4686	9154	13704	18242
$SYS/broker/publish/messages/received	53	94	398	1157	2283	3451	4618
$SYS/broker/publish/messages/sent	222	367	1437	4096	8004	11961	15906

$SYS/broker/bytes/received	2976	5015	21111	62781	123353	186822	250274
$SYS/broker/bytes/sent	8609	14285	56570	162328	318376	476257	633619
$SYS/broker/publish/bytes/received	265	470	1990	5785	11415	17255	23090
$SYS/broker/publish/bytes/sent	942	1650	7159	21442	42979	65025	87076

Table 4.6 Mosquitto message broker metrics on fog layer using bridge – First Architecture– Three IoT devices

Number of IoT devices	Three IoT devices						
Number of subscribing & publishing	Six subscribing & six publishing						
Mosquitto message broker metrics as $SYS topics	***30 (seconds)***	***1 (minutes)***	***5 (minutes)***	***15 (minutes)***	***30 (minutes)***	***45 (minutes)***	***1 (hours)***
$SYS/broker/load/messages/received 1(m)	90.09	129.50	172.49	163.15	163.81	172.50	179.13
$SYS/broker/load/messages/sent 1(m)	207.44	279.24	359.50	357.54	353.60	352.41	369.72
$SYS/broker/load/publish/received 1(m)	55.43	81.41	109.62	99.23	98.21	107.07	115.52
$SYS/broker/load/publish/sent 1(m)	172.78	231.14	296.63	293.61	287.99	286.97	306.10
$SYS/broker/load/bytes/received 1(m)	3329.39	4744.07	6384.79	6124.10	6170.40	6480.58	6627.83
$SYS/broker/load/bytes/sent 1(m)	6672.01	8948.99	11686.44	11697.09	11528.00	11394.30	12088.13
$SYS/broker/load/sockets 1(m)	33.08	45.51	60.27	61.48	63.11	63.81	61.52
$SYS/broker/load/connections 1(m)	33.08	45.51	60.29	61.49	63.11	63.82	61.63
$SYS/broker/subscriptions/count	6	6	6	6	6	6	6
$SYS/broker/heap/current	30376	30320	30268	30312	30376	30240	30252
$SYS/broker/messages/received	129	228	951	2689	5310	7915	10503
$SYS/broker/messages/sent	299	505	1992	5639	11077	16495	21823
$SYS/broker/publish/messages/received	79	142	603	1699	3333	4973	6612
$SYS/broker/publish/messages/sent	249	419	1644	4649	9100	13553	17932
$SYS/broker/bytes/received	4736	8349	35210	100003	197975	295130	391395
$SYS/broker/bytes/sent	9603	16202	64301	183194	359880	536549	710168
$SYS/broker/publish/bytes/received	395	710	3015	8495	16665	24865	33060
$SYS/broker/publish/bytes/sent	1082	1920	8315	24546	49094	73862	98359

4.8 Evaluation of Results

4.8.1 First Architecture vs. Second Architecture

The first experiment of each of the two architectures was performed by connecting one IoT device to the cloud. We used two subscribes and two publishes to send the data. The results show that the number of subscribes and publishes are the same for both architectures and matches the defined number of subscribes and publishes for one IoT device. However, the number of published messages (PublishIn.Success and PublishOut.Success in Figs. 4.6.a-4.7.a) in the first architecture is slightly less than the number of published messages in the second architecture. This is because there is an additional hop (i.e., fog layer) in the middle of the first architecture that processes the messages before transmitting them to the AWS cloud; as a result, the messages take additional time to be delivered to the AWS cloud.

In contrast, the second architecture does not have a fog layer and the messages are forwarded directly to the AWS cloud, so the number of published messages is slightly higher. Moreover, the IoT devices must be authenticated to the AWS before publishing messages. Connect.Success and Subscribe.Success in Figs. 4.8.a-4.9.a show that when using one IoT device, authentication did not impact these two metrics in either architecture.

The second experiment was conducted using two IoT devices and four subscribes and publishes. The results show that the number of subscribes and publishes matches the defined number of subscribes and publishes for two IoT devices only in the first architecture, as shown in Figs. 4.8.b-4.9.b (Connect.Success and Subscribe.Success); the number of subscribes and publishes was significantly increased in the second architecture. This is because in the second architecture, the two IoT devices need to be authenticated separately to the AWS cloud, whereas in the first architecture, only one device (the fog node) needs to be authenticated because the IoT devices are authenticated to the fog node in a different, simpler process. Additionally, this high number of connect and subscribe requests in the second architecture causes a loss in the number of messages published to the AWS cloud. This is because while one of the IoT devices is subscribed and publishing, the other device remains trying to connect, as shown in Figs. 4.6.b-4.7.b (PublishIn.Success and PublishOut.Success). Notably, the rate of published messages in the first architecture is much better than in the second architecture. This is because in the first architecture, there was no failure to subscribe and the fog node was always able to publish messages successfully to the AWS cloud, as shown in Figs. 4.8.b-4.9.b (Connect.Success and Subscribe.Success).

In the third experiment, we used three IoT devices with six subscribes and publishes. The results show that the number of subscribes and publishes matches the defined number of subscribes and publishes (PublishIn.Success and PublishOut.Success in Figs. 4.6.c-4.7.c) in the first architecture. In the second architecture, however, the number of subscribes and publishes is significantly higher. This is because the fog node in the first architecture authenticates three IoT devices and the AWS cloud authenticates only the fog node. In contrast, in the second architecture, the AWS cloud authenticates three

IoT devices separately, which increases the number of subscribes and publishes. This also negatively affects the number of published messages in the second architecture due to the time spent by the IoT devices that are not connected trying to connect, as shown in Figs. 4.6.c-4.7.c (PublishIn.Success and PublishOut.Success). In contrast, in the first architecture, there is no loss in the published messages, as there is no sign of failure in the number of subscribes and publishes, as shown in Figs. 4.8.c-4.9.c (Connect.Success and Subscribe.Success).

In this experiment, we found that the performance of the second architecture was better than that of the first architecture when using one IoT device. However, when using more than one IoT device, the first architecture outperforms the second architecture in terms of performance. Moreover, when using more than one IoT device, the resource utilization in the first architecture was better than in the second architecture because all of the IoT devices were able to successfully connect to the fog node simultaneously. Overall, with an increased number of IoT devices, the first architecture outperforms the second architecture.

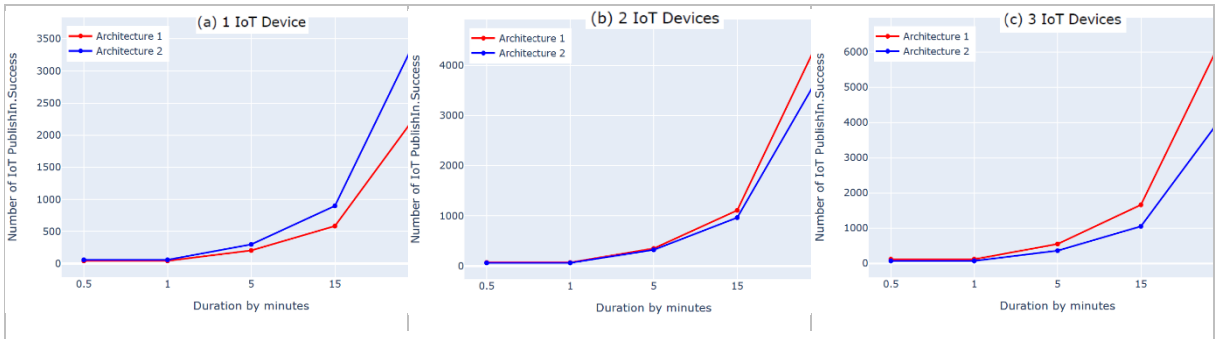


Figure 4.6 AWS IoT message broker PublishIn.Success metric with 1, 2, and 3 IoT devices on N. Virginia datacenter (cloud layer)

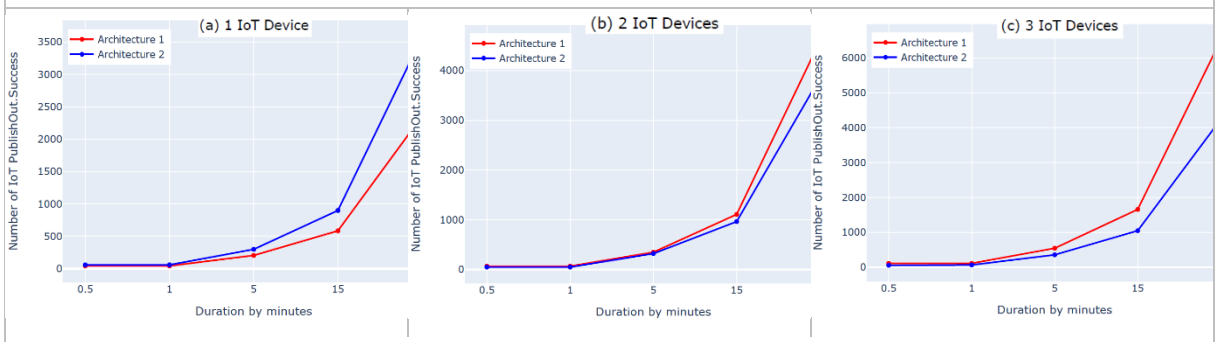


Figure 4.7 AWS IoT message broker PublishOut.Success metric with 1, 2, and 3 IoT devices on N. Virginia datacenter (cloud layer)

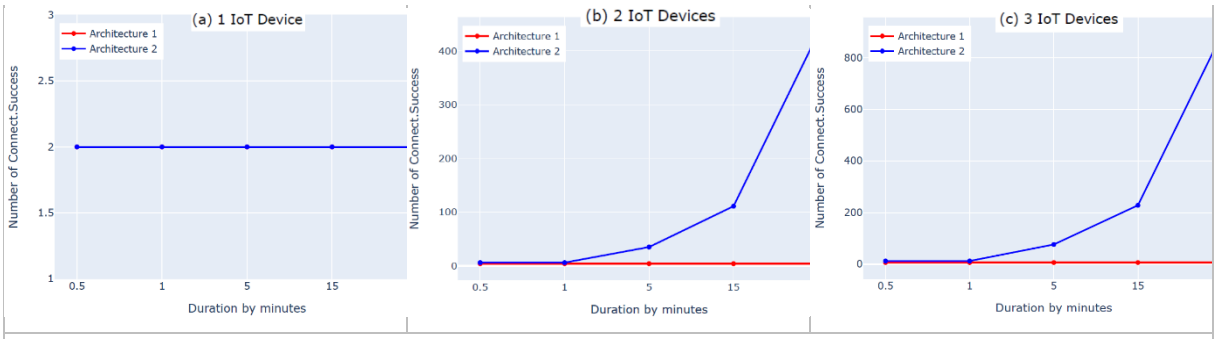


Figure 4.8 AWS IoT message broker Connect.Success metric with 1, 2, and 3 IoT devices on N. Virginia datacenter (cloud layer)

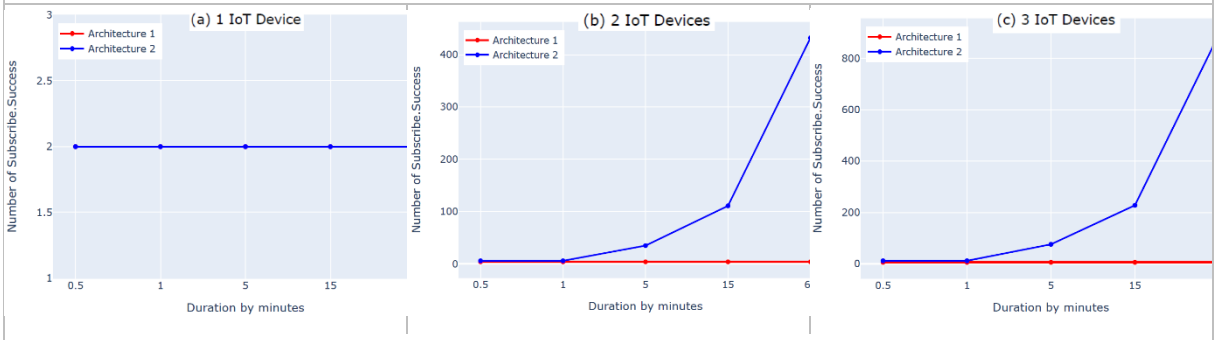


Figure 4.9 AWS IoT message broker Subscribe.Success metric with 1, 2, and 3 IoT devices on N. Virginia datacenter (cloud layer)

4.8.2 Architecture 1 Implementation: Python Script vs. Bridging

In this experiment, we evaluated the first architecture (i.e., IoT-fog-cloud) using two different implementations (Python Script vs. bridging) using AWS metrics. Three experiments were performed using one, two, or three IoT devices. Using two subscribes and two publishes per device, both implementations showed insignificant differences in performance using the AWS benchmark, as shown in Figs. 4.10, 4.11,4.12, and 4.13. Comparing the performance results of the same architecture using two different implementations shows that the results of our experiment are accurate.

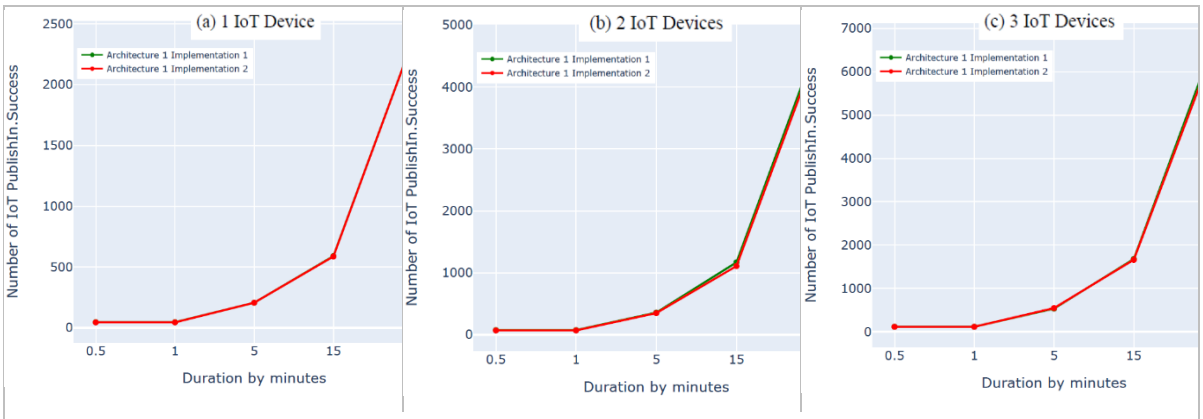


Figure 4.10 AWS IoT message broker PublishIn.Success metric with 1, 2, and 3 IoT devices on N. Virginia datacenter (cloud layer)

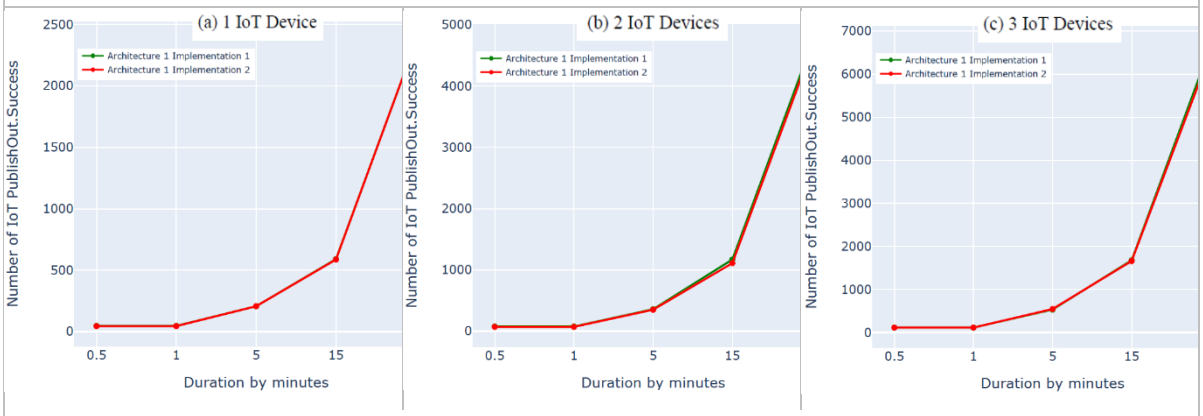


Figure 4.11 AWS IoT message broker PublishOut.Success metric with 1, 2, and 3 IoT devices on N. Virginia datacenter (cloud layer)

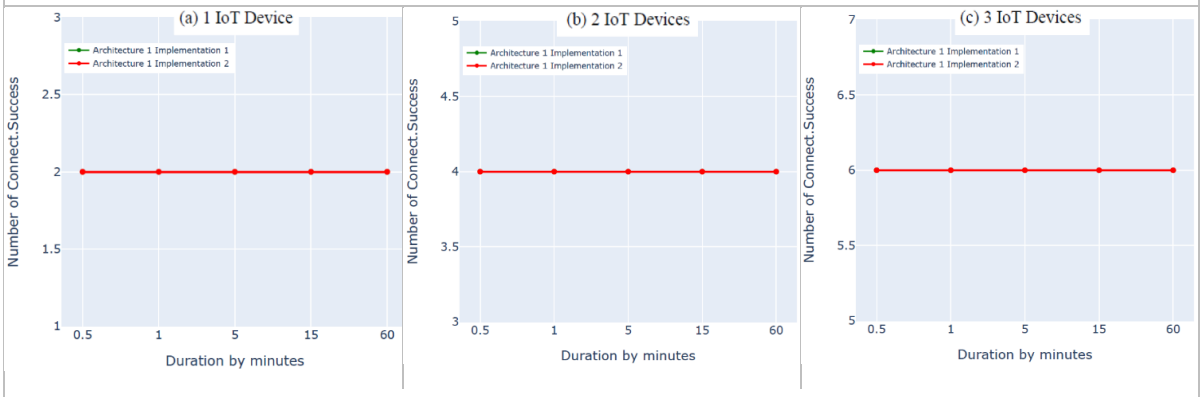


Figure 4.12 AWS IoT message broker Connect.Success metric with 1, 2, and 3 IoT devices on N. Virginia datacenter (cloud layer)

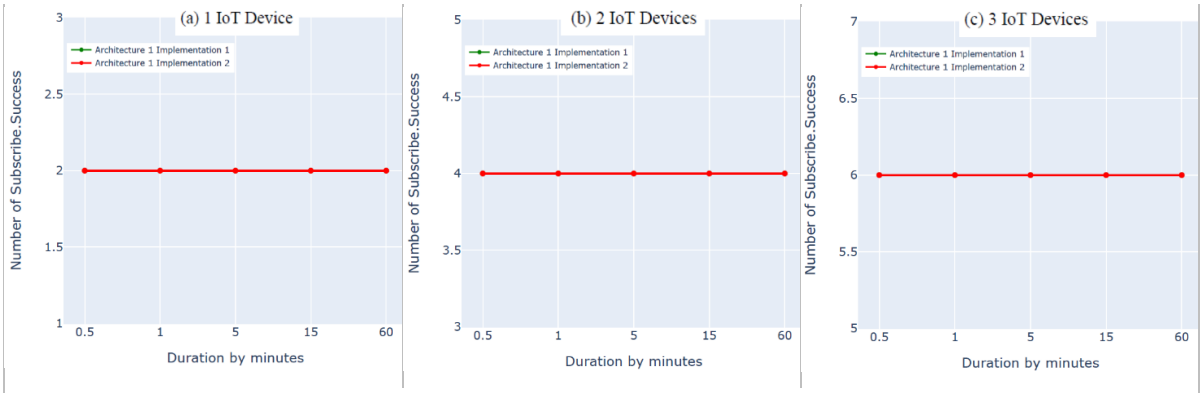


Figure 4.13 AWS IoT message broker Subscribe.Success metric with 1, 2, and 3 IoT devices on N. Virginia datacenter (cloud layer)

4.8.3 Architecture 1 Measurement: Mosquitto Metrics vs. AWS Metrics

In this section, we evaluated the first architecture of a cloud-based IoT environment using AWS and Mosquitto benchmarks. This is to prove that we have monitored the first architecture from two sides, the cloud layer and the fog layer, as shown in Fig. 4.14

The first experiment is conducted using one IoT device with two subscribes and two publishes. The results show that the number of subscribes and publishes are the same for both the AWS (i.e., Subscribe.Success and Connect.Success) and Mosquitto (i.e., $SYS/broker/subscriptions/count) benchmarks and reflect the defined number of subscribes and publishes for one IoT device, as shown in Figs. 4.8.a, 4.9.a and 4.14.9. This experiment was implemented using the subscribe-publish mode presented in [58]. In the architecture with three layers (IoT, fog, and cloud), the fog node subscribes to all of the message topics sent from the IoT device(s). These messages are then processed and published to the AWS cloud. Again, the cloud layer (i.e., the AWS cloud) subscribed to all of the message topics sent from the fog device. This is important, as it ensures that the IoT device(s) are connected and able to send data to the fog and then to the cloud without loss.

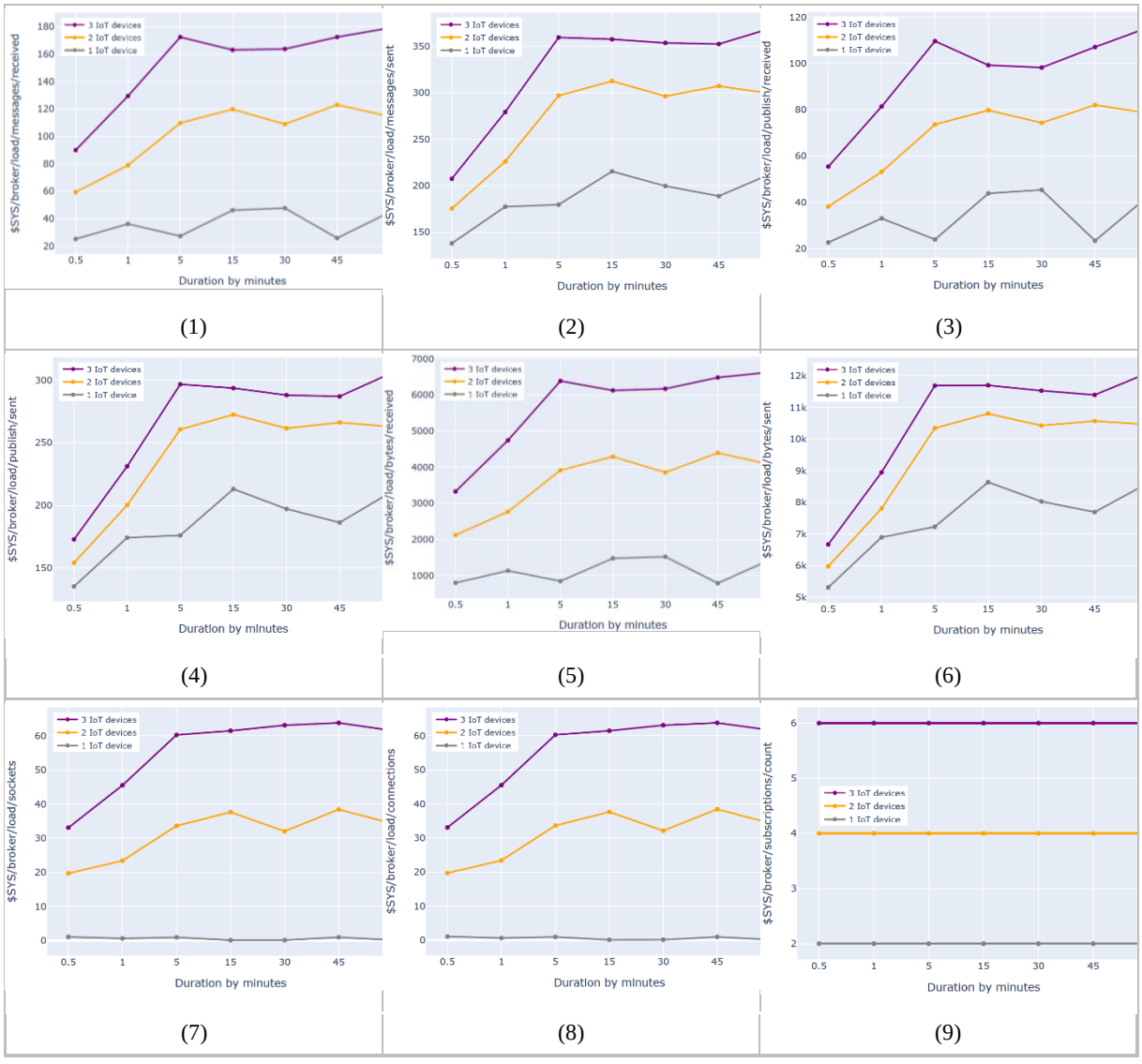
The number of published messages on the fog device using the Mosquitto metric (publish/messages/received) is equal to the number of published messages on the AWS cloud using the AWS metric (PublishIn.Success), as shown in Figs. 4.14.13 and 4.6.a. This is because the messages

generated from one IoT device were received by both the fog and the AWS cloud and no messages were lost during transmission. However, the number of messages processed by the Mosquitto broker ($SYS/broker/publish/messages/sent) on the fog device is significantly higher than the number of messages processed by the AWS broker (PublishOut.Success) on the AWS cloud, as shown in Figs. 4.14.14 and 4.7.a. This is because the fog device is closer to the IoT devices, which reduces the latency of transmitting all of the messages to the AWS cloud. In addition, fog devices play an important role in decreasing the computation overhead caused by the IoT devices due to its close proximity. It also reduces AWS resource consumption by bringing computation closer to the IoT devices. Therefore, the memory consumption ($SYS/broker/heap/current) on the fog node remains stable even when the number of messages processed increases, as shown in Fig. 4.14.10.

The second and third experiments of the first architecture were performed using two or three IoT devices with four and six subscribes and publishes, respectively. The results show that the number of subscribes and publishes remains identical using the Mosquitto metric ($SYS/broker/subscriptions/count) and the AWS metrics (Subscribe.Success and Connect.Success) and reflects the defined number of subscribes and publishes, as shown in Figs. 4.8-b,4.8-c, 4.9-b,4.9-c and 4.14.9. In addition, the number of published messages on the fog device remains equal to the number of published messages on the AWS cloud, as shown in Fig. 4.14.13 and 4.6-b, and 4.6-c. However, we noticed that when we increased the number of IoT devices, the number of messages made and processed by the Mosquitto broker on the fog device significantly increased without affecting the performance, as shown in Fig. 4.14.14.

Overall, in the first experiment with one IoT device, the number of messages processed by the fog device (i.e., received from the IoT device and sent to the cloud) is higher than the number of messages received by the AWS cloud. Similarly, as the number of IoT devices increased (i.e., using two and three IoT devices), the number of messages made by the Mosquitto broker on fog device remains higher than the number of messages made by the the AWS broker on the AWS cloud. This is because the messages are transmitted through the three layers of the environment (i.e., IoT, fog, and cloud); thus, it takes more

time for the messages to be delivered to the AWS cloud due to the additional intermediate fog layer. Moreover, since the fog device is located close to the IoT devices, message latency decreased and the processing of the messages on the fog increased. Thus, using fog computing is very beneficial when connecting more than one IoT device to the cloud. Overall, as the number of IoT devices increases, the process ability of the fog device in the first architecture outperforms that of the AWS cloud.



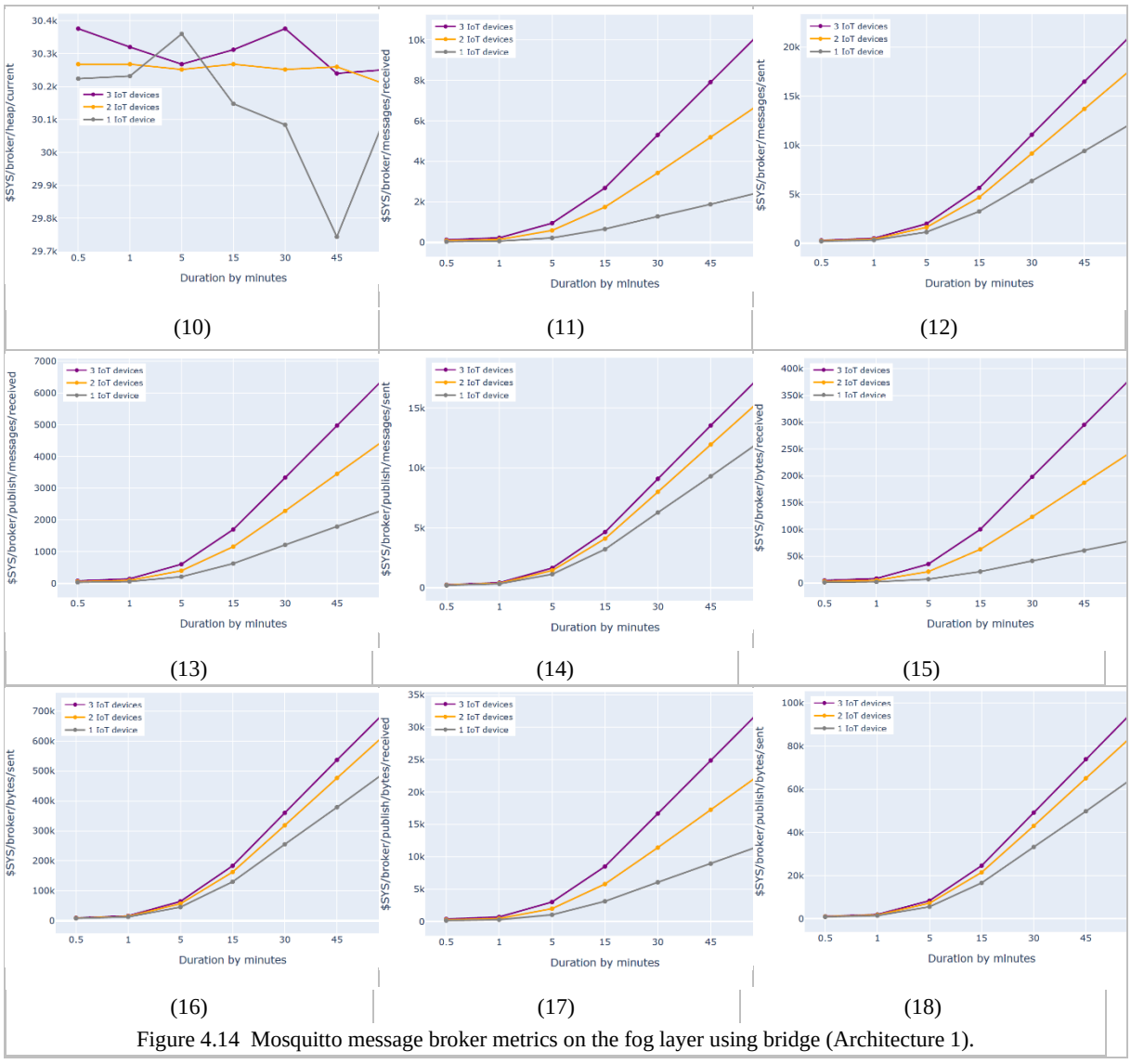


Figure 4.14 Mosquitto message broker metrics on the fog layer using bridge (Architecture 1).

4.9 Threats to validity

The experiments in this paper were implemented using real commercial sensors interacting with a real-world cloud through a commercial IoT service. The environment is susceptible to real attacks and simulates a real-life IoT environment. We used Raspberry Pi boards to enable the sensors to send collected data through the Internet to the cloud or the fog node. We also used a Raspberry Pi board as a fog node, the intermediate layer between the sensors and the cloud in the first architecture. The first architecture consists of three layers (IoT, fog, and cloud), whereas the second architecture consists of only two layers (IoT and cloud). Every layer operates on a separate network and all networks were

connected to the Internet to simulate real-life implementations. In both architectures, up to three sensors were used to capture temperature and humidity data and send it to the cloud.

Each experiment was performed for an hour, during which the data captured by the sensors was sent to the cloud (i.e., the AWS N. Virginia datacenter) and stored there. It is of interest to investigate whether the results can be applied to other cloud service providers or to other datacenters in different geographical areas. In addition, three sensors were used to conduct the experiments by either connecting the sensors to the cloud directly or connecting the sensors through a fog node to the cloud. Further experiments are needed to determine how many sensors a fog node can handle before performance is impact.

Moreover, our results indicate that the first architecture outperforms the second architecture in publishing the data captured by the sensors to the cloud. This is attributed to the fact that in the second architecture, all of the sensors need a certificate for authentication, whereas in the first architecture, only one certificate for authentication is needed and is placed on the fog node. Notably, this leaves the communication between the IoT layer and the fog layer without a proper authentication mechanism. We plan to utilize the computation and storage capability of the fog node to implement an authentication and authorization model for the sensors interacting with the fog nodes in the future.

4.10 Discussion and Limitations

Fog computing evolved to support cloud-based IoT environments in many ways. It is known for its ability to lower communication latency, optimize communication bandwidth, and enable higher scalability and heterogeneity of networks. In addition to these advantages, fog computing enjoys many valuable characteristics, such as fog node mobility and location awareness, in addition to computational ability that IoT devices lack. In this paper, we demonstrated that fog computing has a substantial impact on cloud-based IoT environments in reducing latency and improving communication performance. In addition to these benefits, fog computing has a great deal of advantages that were not extensively discussed in our results.

First, fog computing-aided IoT environments are known for their higher scalability, since every group of IoT devices is connected to a fog node. This hierarchical structure enables better management, tracking, and monitoring of IoT devices. In addition to this, compared to environments in which IoT devices are connected directly to the cloud, fog-aided IoT environments improve resource utilization. This is attributed to savings in processing capability that is wasted when the cloud authenticates a massive number of IoT devices. At first sight, one might say that the same processing time and resources will be consumed at the fog node to authenticate IoT devices. This is true; however, with a fog node, consumers of cloud services will not be overcharged for service that was wasted on the cloud to authenticate a significant number of IoT devices, especially because providers charge based on consumption.

From a security standpoint, fog computing spreads risks across distributed fog nodes in fog-aided IoT environments. In addition, authenticating IoT devices at the fog layer provides more flexibility in adding sophistication to the authentication and authorization process, such as encryption-based access control. Although the convenience of having a fog layer with semi-heavyweight computation capability has a higher capital cost, in the long run, in saves ample resources, time, and money.

4.11 Conclusion and Future Works

In this paper, we proposed two architectures of cloud-based IoT environments using a real environment. In one architecture, we used a fog layer between IoT devices and the cloud, whereas in the second, IoT devices published data directly to the cloud. In order to validate our results, we also examined two ways of implementing fog-aided IoT-cloud environments: (1) bridging and (2) using a Python script to forward the data to the cloud. For each architecture, we conducted several experiments and increased the number of IoT devices as well as the number of subscribes and publishes in each experiment. To evaluate performance in the experiments, we used two sets of benchmark metrics: (1) AWS message broker metrics and (2) Mosquitto message broker metrics. Performance was evaluated based on the following analysis methods. First, we compared the performance for the first and second architecture. Second, we compared the performance for the two implementations of the fog-aided IoT environment

(i.e., Python script vs. bridging). Finally, to validate our results, the performance of the first architecture was analyzed using Mosquitto metrics vs. AWS metrics. The results showed that the performance in the IoT-cloud with a fog layer is significantly better than without the fog layer as the number of IoT devices and the number of subscribes and publishes increases. The results also showed that using Python script or bridging fog-aided IoT-cloud environments performs the same. The results of our third analysis method showed that as the number of IoT devices increases, the processability of the fog device in the fog-aided IoT-cloud architectures outperforms that of the AWS cloud. This work aimed to educate readers on different methods to implement IoT-cloud environments and compares the performance for each. It also guides researchers by providing different ways to implement fog-aided IoT-cloud systems.

In the future, we plan to extend this work by using the same implementations presented in this paper to analyze the performance using different cloud providers, protocols, fog devices, and IoT devices. The presented implementations will also be used to investigate security and address issues in fog-aided IoT-cloud systems.

Chapter 5: Novel Security Models for IoT–Fog–Cloud Architectures in a Real-World Environment

M. A. Aleisa, A. Abuhussein, F. S. Alsubaei, and F. T. Sheldon, " Novel Security Models for IoT-Fog-Cloud Architectures in a Real-World Environment," Sensors, vol. 21, no. 21, p. 6950, Mar. 2022.

(Under Review)

5.1 Introduction

The Internet of Things (IoT) ecosystem includes multitudinous devices connected to the Internet [118] with a variety of capabilities, such as sensing, processing, and communicating. The number of IoT devices is expected to rise to over 75 billion in 2025 [119], [120] , driving a parallel rise in the already massive amount of data that must be locally processed at the edges of networks to reduce latency and save network bandwidth. Cloud computing offers high computation power and storage for thousands of IoT devices [121]. However, due to the geographic centralization of cloud computing data centers, the large volume of data generated by the distributed IoT devices will not be processed in a timely manner, which will increase the latency between IoT devices and the cloud especially as the number of IoT devices continues to grow. To overcome these challenges, fog computing has emerged to deal with high processing demand and temporary storage. Fog computing acts as an intermediate layer between the cloud and IoT devices [58], [75], [122], solving the data transmission latency between them.

Despite the benefits of fog computing for IoT devices and the cloud, there are several security issues between the IoT–Fog–Cloud layers. For example, the Dyn cyberattack (October 21, 2016) disrupted Internet service across Europe and US [123] through a series of distributed denial-of-service (DDoS) attacks that targeted IoT-enabled devices such as cameras, residential gateways, and baby monitors. Many services were affected by this cyberattack, including businesses like Amazon, Comcast, PayPal, and Netflix and news networks like Fox News and CNN.

Further significant threats to IoT devices include eavesdropping [20], [43], [118], [121] and unauthorized access [43], [124], [125], which can lead to device failure. Because there is no human

interaction involved in the communication between these devices and because they have extensive operating times, it is difficult to monitor and detect their security issues. Therefore, it is essential to build a security model that meets the security requirements of the IoT–Fog–Cloud architecture by applying authentication and authorization between the IoT–Fog–Cloud layers.

Because there is a lack of real-life implementations of cloud-based IoT environments, as highlighted in our earlier work [58], we proposed two architectures of cloud-based IoT environments and three analysis methods using a real-world environment [75], [122]. In the first architecture, we used a fog layer between IoT devices and the cloud, whereas in the second, IoT devices published data directly to the cloud [75], [122]. We conducted several experiments and evaluated our results of the methodologies and the three analysis methods [122], finding that the first architecture outperforms the second. This was attributed to the fact that in the second architecture, all of the sensors require a certificate for authentication whereas, in the first, only one authentication certificate is needed and is placed on the fog node. Notably, this left the communication between the IoT layer and fog layer without proper authentication and authorization models [122].

To fill the gap in security requirements between IoT layer and fog layer and overcome the limitations and challenges presented by these security issues [58], [75], [122], this paper makes the following contributions:

- We propose a fine-grained data access control model based on the attribute-based encryption (ABE) of the IoT–Fog–Cloud architecture to limit access to sensor data to meet the authorization aim.
- We propose a blockchain-based certificate model of the IoT–Fog–Cloud architecture to authenticate IoT devices to fog devices to meet the authentication aim.
- We evaluate the performance of the security model (fine-grained data access control and blockchain-based certificate) using AWS message broker metrics for a real-life scenario of the IoT–Fog–Cloud architecture.

• We compare the performance of the IoT–Fog–Cloud architecture with and without our security model using AWS message broker metrics and present its efficiency and feasibility.

The remainder of this paper is structured as follows: In section 2, we present the authentication model, a blockchain-based certificate of the IoT–Fog–Cloud architecture. In section 3, we present the authorization model, a fine-grained data access control model based on the ABE of the IoT–Fog–Cloud architecture. In section 4, we detail the setup of the IoT–Fog–Cloud architecture experiments. In section 5, we explain the analysis methods used to evaluate the IoT–Fog–Cloud architecture with the two security models. In section 6, we evaluate the performance of the IoT–Fog–Cloud architecture with the two security models based on the analysis methods. Finally, section 7 concludes the paper.

5.2 Proposed Authentication Model: Blockchain-based Certificate

To fill the gap of the security requirements between the IoT layer and fog layer[58], [75], [122], we propose a blockchain-based certificate model of the IoT–Fog–Cloud architecture to authenticate the IoT devices to fog devices and achieve the authentication aim of this study. Figure 5.1 presents the operations comprising the model, which are as follows: 1) the IoT devices make a connection request to the fog devices; 2) the fog devices distribute a valid certificate to the IoT devices; 3) the handshake mechanism using TLS cryptographic protocol is established between IoT devices and fog devices; 4) encrypted communication is established between the IoT devices and fog devices; 5) because the fog devices are expected to be limited, the blockchain technology is applied to a set of fog devices within their geographical location; and 6) each fog device inside the blockchain has a copy of transactions, such as the distributed IoT devices certificates. Figure 5 shows how the blockchain-based certificate model was applied to the IoT–Fog–Cloud architecture using a real-life environment.

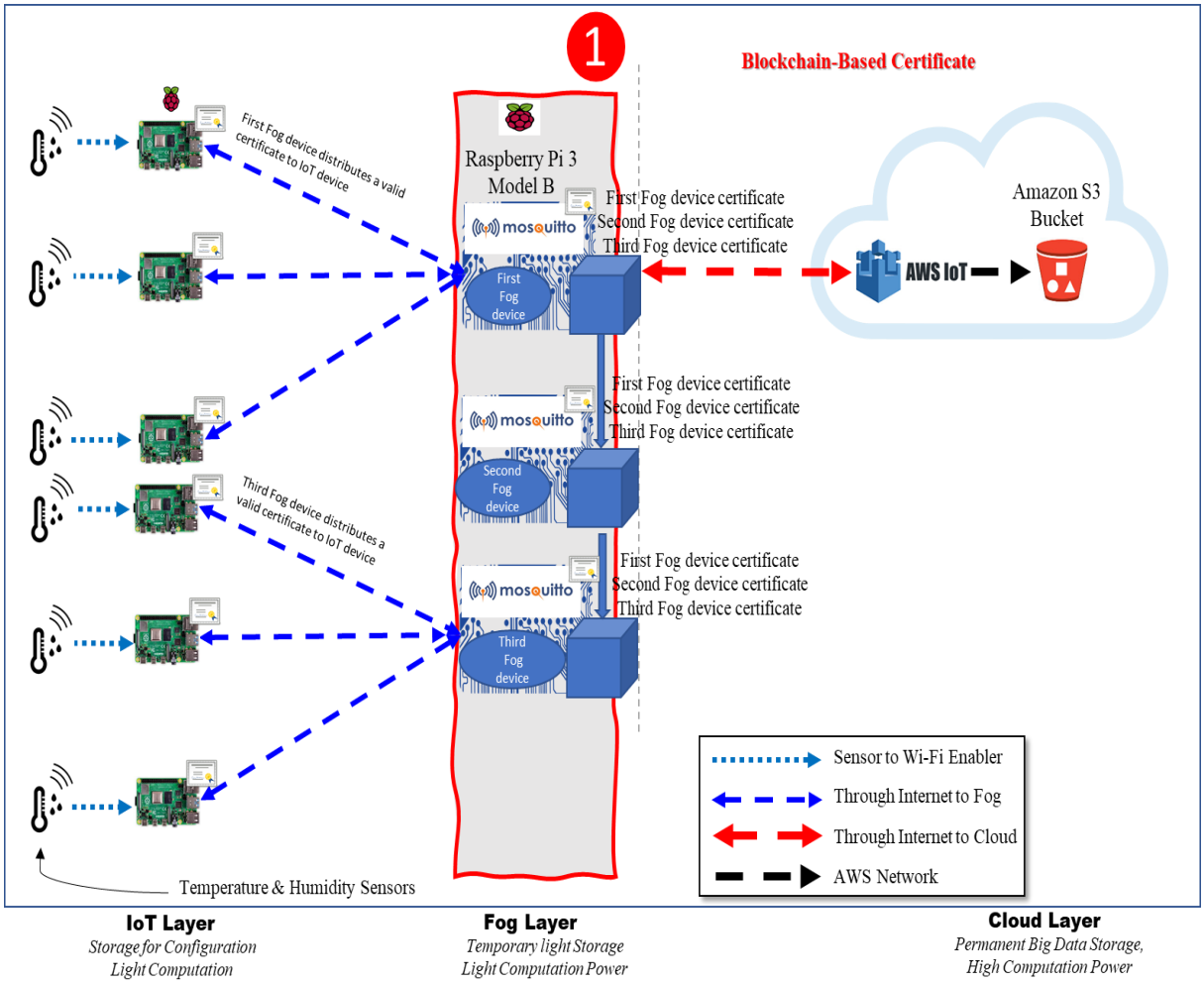


Figure 5.1 Blockchain-based certificate model applied in IoT–Fog–Cloud architecture.

5.3 Proposed Authorization Model: Attribute-based Encryption for Access Control

To fill the gap in the security requirements between the IoT and fog layers [58], [75], [122], we propose a fine-grained data access control model based on the ABE of the IoT–Fog–Cloud architecture to limit access to sensor data and achieve the authorization aim of this study.

Figure 5.2 illustrates the several operations comprising the model, which are as follows: 1) attributes are generated for each sensor data type; 2) keys containing a set of attributes or corresponding to attributes are generated; 3) the generated sensor data type value are encrypted with the corresponding key that contains its attribute; 4) the encrypted message is published to the fog device using a secure

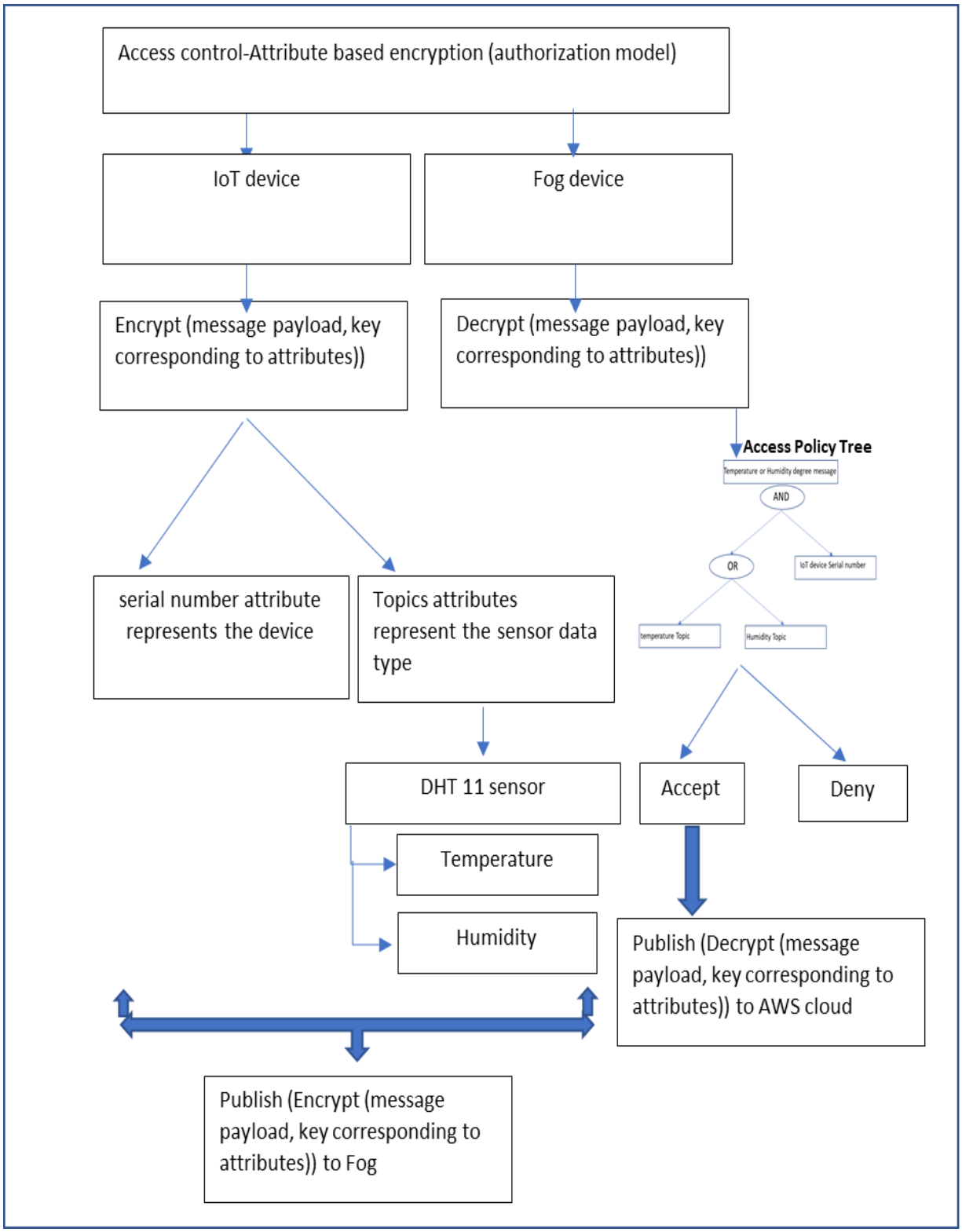


Figure 5.2 Access control model for the IoT–Fog–Cloud architecture.

communication channel; 5) the access tree that specifies the policy of the set of attributes is generated; 6) the ciphertext is decrypted if the key containing a set of attributes satisfies the access policy tree; and 7) the decrypted message is published to the AWS cloud. The model is designed such that each data type in an IoT device is associated with attributes, which represent the topic of each sensor data type. For example, the DHT11 sensor attached to Raspberry Pi is considered an IoT device [75], [122] and generates two types of data: 1) temperature and 2) humidity. The temperature value is encrypted according to the key that contains a set of attributes, and then published to the fog device.

The fog device then generates an access policy tree according to the attributes of each data type in the IoT device, an example of which is presented in Figure 5.3. Once the fog device receives the ciphertext from the IoT device, it decrypts it if the ciphertext key that contains the attributes satisfies the access policy tree. Otherwise, it will decline the decryption request. Then, the temperature degree value will be published to the AWS cloud. Figure 5.4 shows how the ABE for the access control model was applied to the IoT–Fog–Cloud architecture.

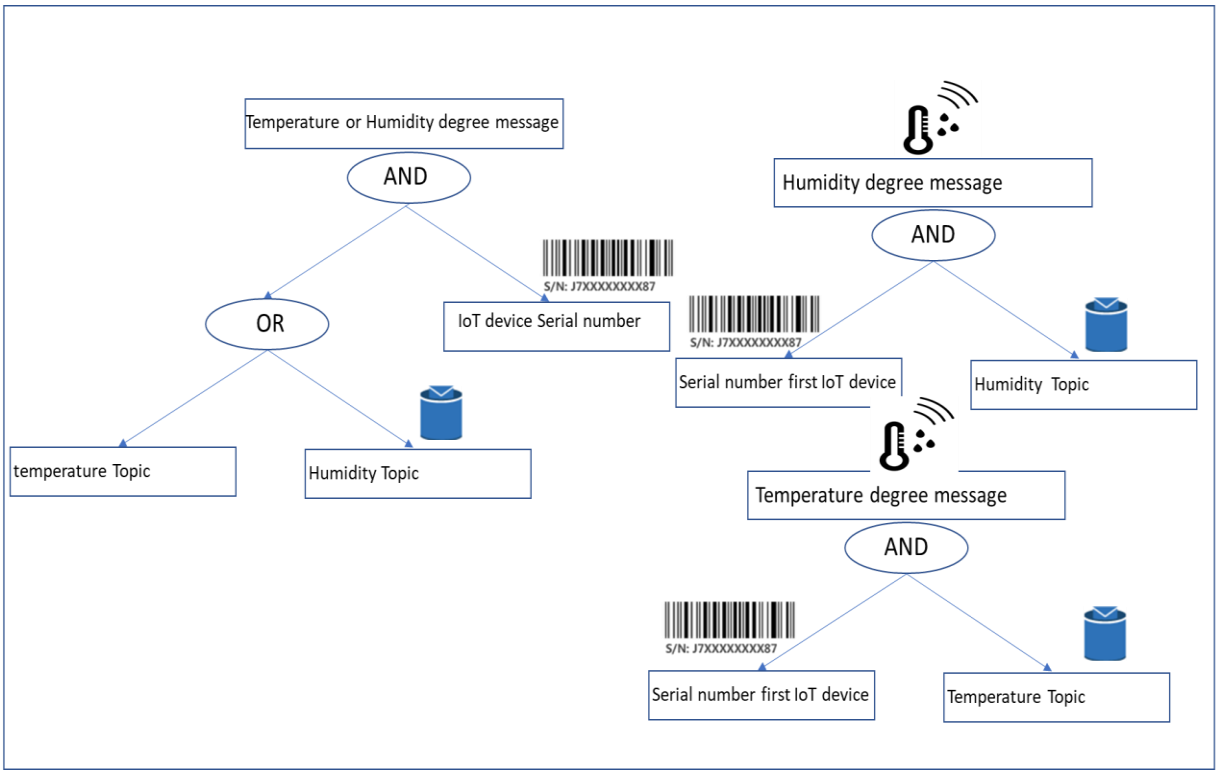


Figure 5.3 Access policy tree for access control model for the IoT–Fog–Cloud architecture.

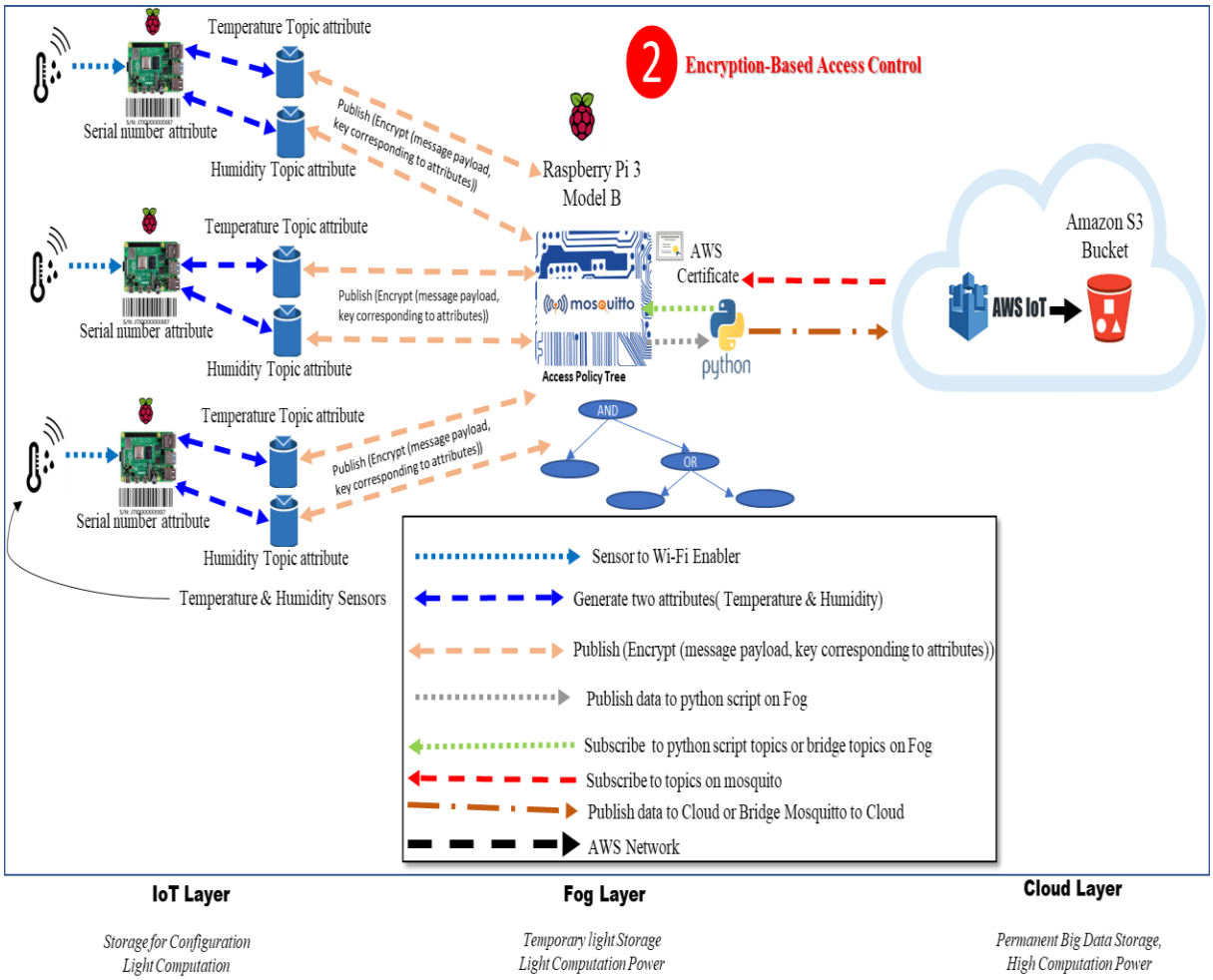


Figure 5.4 Access control model applied to the IoT–Fog–Cloud architecture.

5.4 Experiment Setup

In this paper, we propose a security model of the IoT–Fog–Cloud architecture [122] to meet its security requirements [43], [121]. The following subsections present the hardware and software configurations used for this security model and our IoT–Fog–Cloud architecture experiment.

5.4.1 Hardware

The experiment discussed in this section involved the first architecture [122] of IoT–Fog–Cloud as shown in Figure 5.4 We first describe the devices that were used in the IoT–Fog–Cloud architecture

and then explain it in detail. For the experiment, we used DHT11 devices [110] and a Raspberry Pi 3 Model B [126]. DHT11 is a low-cost sensor that measures the temperature and humidity of the surrounding air and was used in this experiment to generate real data. The Raspberry Pi is a low-cost, single-board computer with built-in WiFi and processing capabilities that is used across several domains, such as weather monitoring, smart homes, and smart health care. In this experiment, the purpose of the Raspberry Pi was to provide light computational capabilities for the DHT11 sensor data. It also provided light storage for the DHT11 configurations. Moreover, the Raspberry Pi can be easily moved to different locations. A complete list of the hardware used in this experiment is available in Table 5.1

Table 5.1 Summary of the equipment used in the IoT-Fog-Cloud architecture.

Equipment Name	Equipment Type	Quantity	Purpose
DHT11	Temperature–humidity sensor	3	Generate real-life data
Raspberry Pi	Version 3 Model B	4	Enable WiFi and provide substantial processing power and storage
Micro SD Card	32GB ImageMate Plus 130 mb/s Read	4	Initial storage for the operating system and files
Monitor	HP	4	Provide a visual display
Keyboard & mice	HP	4	Facilitate working on a Raspberry Pi
Power Supply/Adapter	CanaKit	4	Supply the power for the Raspberry Pi
HDMI Cable	onn	4	Connect the Raspberry Pi to a monitor

In the IoT–Fog–Cloud architecture, each DHT11 sensor was connected to only one Raspberry Pi board (i.e., connectivity enabler), which is considered an IoT device in the IoT layer. The Raspberry Pi was used here to enable WiFi connectivity, as the DHT11 sensors are not equipped with network interfaces. Each IoT device was connected via WiFi to another Raspberry Pi board that acted as a fog node in the fog layer. Communication between the IoT devices, the fog nodes, and the cloud used the MQTT protocol, for which an MQTT broker called Eclipse Mosquitto [127] was installed in the Raspberry Pi, acting as a fog device. The Mosquitto MQTT broker exchanged all messages using the subscribe–publish model presented in [58] and filtered all messages based on topics, which are UTF-8

strings used by the broker specifically for this task. Each data type in our experiment (i.e., humidity, temperature) was considered a separate topic. The data generated by the sensors and collected by the three IoT devices were transmitted over the Internet to the Raspberry Pi acting as the fog node. This Raspberry Pi, with the MQTT broker Mosquitto, was then connected to the AWS cloud over the Internet. The communication between the three layers occurred through the Internet. Figure 6 shows the hardware used to implement the IoT–Fog–Cloud architecture.

5.4.2 Software

We installed Python on the three IoT devices (DHT sensor + Raspberry Pi) and the fog device. Then, we installed the Circuit Python DHT Library on the three IoT devices to allow communication between the DHT11 sensor and Raspberry Pi. Next, we installed the cryptography library on the three IoT devices and fog device to perform the security operations. Algorithm 1 illustrates the authentication and authorization operations from IoT devices (DHT11 sensor + Raspberry Pi) to the fog device, and Algorithm 2 illustrates the authentication and authorization operations from the fog device to the IoT devices (DHT11 sensor + Raspberry Pi) and from the fog device to the AWS cloud.

Algorithm 1: Gather data generated from the IoT device and forward it to the fog device – IoT–Fog–Cloud architecture- /* This algorithm provides authentication and authorization operations from IoT devices (DHT11 sensor + Raspberry Pi) to fog devices.

```
44:  Import board
45:  Import adafruit_dht
46:  Import paho.mqtt.client as mqtt
47:  From Crypto.Cipher import ABS
48:  Import base64
49:  Define the type of DHT sensor, which is DHT11
50:  Define the input/output pins of the Raspberry Pi to which the DHT11 is connected
51:  Define a Python library (adafruit_dht.DHT11) to read the DHT series of humidity and temperature
     sensors on a Raspberry Pi with one argument, DHT pin connected
52:  Define the key length which must be either 16, 24, or 32 bytes long
53:  Define only two variables (humidity topic and temperature topic) for each IoT device (DHT11 +
     Raspberry Pi) in each experiment
54:  Define the variable of MQTT broker
55:  Define the variable of MQTT port
56:  While True do
57:      Define a connection function
58:          Connect to Internet
59:          If (the connection is established) then
60:              Print "connected"
61:          Else (the connection isn't established) then
```

```
62:                         Try reconnecting to Internet
63:                 Define a message function
64:                         Read humidity degree from Raspberry Pi serial port using (dhtDevice.humidity)
65:                         Read temperature degree from Raspberry Pi serial port using
        (dhtDevice.temperature)
66:                         Print humidity degree
67:                         Print temperature degree
68:                         Generate keys containing a set of attributes for each sensor data type in each IoT
        device
69:                         Generate a first key containing a set of attributes for temperature sensor in each
        IoT device
70:                         Generate a second key containing a set of attributes for humidity sensor in each
        IoT device
71:                         Create the cipher config for first key (temperature sensor)
72:                         Create the cipher config for second key (humidity sensor)
73:                         Use the cipher of the first key to encrypt the humidity degree message using
        cipher.encrypt
74:                         Use the cipher of second key to encrypt the temperature degree message using
        cipher.encrypt
75:                         Encode the cipher and humidity degree message using the base64 module
76:                         Encode the cipher and temperature degree message using the base64 module
77:                         Print the encrypted message
78:                         Publish humidity topic with its encrypted message to fog device
79:                         Publish temperature topic with its encrypted message to fog device
80:         end while
81:         Create a client to connect to fog device
82:         Make the client run connect, and message function
83:         Set the transport layer security (TLS) for the client using fog device certificates and the current
            version of MQTT protocol
84:         Connect the client to the MQTT broker using the IP address of the fog device and MQTT port
            1883
85:         Create a loop_start() method to start a new thread for the client
```

Algorithm 2: Gather data generated from the fog device and forward it to the cloud – IoT–Fog–Cloud architecture- /* This algorithm provides authentication and authorization operations from fog device to IoT devices (DHT11 sensor + Raspberry Pi) & from fog device to AWS cloud

```
1:          Import sys
2:          Import ssl
3:          Import adafruit_dht
4:          Import paho.mqtt.client as mqtt
5:          From Crypto.Cipher import ABS
6:          Import base64
7:          Define the key length which must be either 16, 24, or 32 bytes long
8:          Define the variable of MQTT broker
9:          Define the variable of MQTT port
10:         While True do
11:                 Define a connection function
12:                         Subscribe for all topics in each IoT devices
13:                         Connect to Internet
```

14: **If** (the connection is established) **then**
15: **Print** "connected"
16: **Else** (the connection isn't established) **then**
17: **Try** reconnecting to Internet
18: **Define** a message function
19: **Define** keys containing a set of attributes for each sensor data type in each IoT device based on access policy
20: **Define** a first key containing a set of attributes for temperature sensor in each IoT device
21: **Define** a second key containing a set of attributes for humidity sensor in each IoT device
22: **Create** the cipher config for first key (temperature sensor)
23: **Create** the cipher config for second key (humidity sensor)
24: **Decode** the encrypted message using the base64 module
25: **Use** the cipher of the first key to decrypt the humidity degree message using cipher.decrypt
26: **Use** the cipher of second key to decrypt the temperature degree message using cipher.decrypt
27: **Print** the decrypted message
28: **Publish** humidity topic with its decrypted message to AWS cloud
29: **Publish** temperature topic with its decrypted message to AWS cloud
30: **end while**
31: **Create** two instances of clients, one used for the MQTT broker and the other used for the AWS broker
32: **Make** the first client run connect, and message function
33: **Connect** the first client to the MQTT broker using the IP address of the fog device and MQTT port
34: **Create a loop_start()** method to start a new thread for the first client
35: **Set** the transport layer security (TLS) for the second client using the three paths of AWS certificates and the current version of MQTT protocol
36: **Connect** the second client to AWS broker using AWS Endpoint and AWS port
37: **Create a loop_start ()** method to start a new thread for the second client

5.5 Analysis Methods

We used AWS benchmark metrics to analyze the performance of our security model for the IoT–Fog–Cloud architecture proposed in our paper [75], [122]. We set the number of subscribers and publishers to two for each device because the IoT devices (i.e., the DHT11 sensors) generated two types of data, namely (1) temperature and (2) humidity data; therefore, as the number of sensor devices increases, the numbers of subscribers and publishers should also increase. This provides more accurate, consistent, and real results about the environment's performance and scalability. In this section, we present the methods used to perform the experiments on the security model for the IoT–Fog–Cloud architecture.

5.5.1 IoT–Fog–Cloud architecture with blockchain-based certificate model versus without blockchain-based certificate model

The performance of the IoT–Fog–Cloud architecture with the blockchain-based certificate model was analyzed and compared with the architecture without the blockchain-based certificate model, which was presented in the previous work [122], using AWS metrics. The experiment was conducted using different numbers of IoT devices (1, 2, or 3) two times, once with the blockchain-based certificate model and once without. The results were compared and analyzed to show the impacts of our blockchain-based certificate model on the proposed IoT–Fog–Cloud architecture [75], [122]. Because the AWS cloud provider requires a certificate to authenticate any device, and because the fog device is the next layer, the certificate was paced in the fog device (i.e., the Raspberry Pi board serving as the fog layer). This left the communication between the IoT layer and the fog layer without a proper authentication model [122]; this gap was filled by the proposed security model. The fog device distributed the certificates to the IoT devices after a connection request was made by IoT devices. The fog device certificate was paced in each IoT device to allow them to be authenticated to the fog device. The objective of this method was to illustrate the impact of the blockchain-based certificate model on the IoT–Fog–Cloud architecture and that the performance of the model remains identical when using different numbers of IoT devices.

5.5.2 IoT–Fog–Cloud architecture with access control model versus without access control model

The performance of the IoT–Fog–Cloud architecture with the access control model was evaluated by comparison with the architecture without the model [122]. The experiment was performed using one, two, and three IoT devices. The experiment was performed twice, once with the access control model and once without. The results were compared and evaluated to show the impact of our access control model on the proposed IoT–Fog–Cloud architecture [122]. Because the IoT devices were authenticated to the fog device using a blockchain-based certificate model, the sensor data needed to be unavailable to the other IoT devices and have limited access. Therefore, the access control model

proposed in this paper fills the authorization requirement gap between the IoT layer and fog layer. The objective of the analysis method was to show the impact of the access control model on the IoT–Fog–Cloud architecture and how the performance changed using different numbers of IoT devices.

5.6 Evaluation of Results

5.6.1 IoT–Fog–Cloud architecture with blockchain-based certificate model versus without blockchain-based certificate model

In this section, we evaluate the IoT–Fog–Cloud architecture with our blockchain-based certificate model using AWS cloud metrics, as shown in Table 5.2.

Table 5.2 AWS cloud message broker metrics results on N. Virginia datacenter (cloud layer) IoT–Fog–Cloud architecture with blockchain-based certificate model versus without blockchain-based certificate model.

AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using Python script – **IoT–Fog–Cloud architecture without blockchain-based certificate model**						AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using Python script – **IoT–Fog–Cloud architecture with blockchain-based certificate model**					
Number of IoT devices	1					Number of IoT devices	1				
Number of subscribing & publishing	2					Number of subscribing & publishing	2				
AWS IoT metrics in minutes (m)	0.5	1	5	15	60	AWS IoT metrics in minutes (m)	0.5	1	5	15	60
Connect.Success	2	2	2	2	2	Connect.Success	2	2	2	2	2
Ping.Success	2	2	8	29	120	Ping.Success	2	2	8	29	120
PublishIn.Success	44	44	206	586	2360	PublishIn.Success	44	44	206	586	2360
PublishOut.Success	44	44	206	586	2360	PublishOut.Success	44	44	206	586	2360
Subscribe.Success	2	2	2	2	2	Subscribe.Success	2	2	2	2	2
Unsubscribe.Success	2	2	2	2	2	Unsubscribe.Success	2	2	2	2	2
Number of IoT devices	2					Number of IoT devices	2				
Number of subscribing & publishing	4					Number of subscribing & publishing	4				
AWS IoT metrics in minutes	0.5	1	5	15	60	AWS IoT metrics in minutes	0.5	1	5	15	60
Connect.Success	4	4	4	4	4	Connect.Success	4	4	4	4	4
Ping.Success	2	2	8	29	120	Ping.Success	2	2	8	29	120
PublishIn.Success	70	70	350	1110	4590	PublishIn.Success	70	70	350	1110	4590
PublishOut.Success	70	70	350	1110	4590	PublishOut.Success	70	70	350	1110	4590
Subscribe.Success	4	4	4	4	4	Subscribe.Success	4	4	4	4	4

Unsubscribe.Success	4	4	4	4	4	Unsubscribe.Success	4	4	4	4	4
Number of IoT devices			3			Number of IoT devices			3		
Number of subscribing & publishing			6			Number of subscribing & publishing			6		
AWS IoT metrics in minutes	0.5	1	5	15	60	AWS IoT metrics in minutes	0.5	1	5	15	60
Connect.Success	6	6	6	6	6	Connect.Success	6	6	6	6	6
Ping.Success	2	2	9	29	119	Ping.Success	2	2	9	29	119
PublishIn.Success	115	115	548	1660	6510	PublishIn.Success	115	115	548	1660	6510
PublishOut.Success	115	115	548	1660	6510	PublishOut.Success	115	115	548	1660	6510
Subscribe.Success	6	6	6	6	6	Subscribe.Success	6	6	6	6	6
Unsubscribe.Success	6	6	6	6	6	Unsubscribe.Success	6	6	6	6	6

The first experiment of the IoT–Fog–Cloud architecture was performed using one IoT device. We ran the first experiment twice simultaneously, one without our blockchain-based certificate model and the other with the model. We used two subscribes and two publishes because the IoT device generated two types of data: (1) temperature and (2) humidity data. The results show that the number of subscribes and publishes (i.e., Subscribe.Success and Connect.Success) for the IoT–Fog–Cloud architecture with and without the blockchain-based certificate model were the same and reflect the defined number of subscribes and publishes for one IoT device. This is because the connection of neither experiment (with vs. without the security model) was disconnected and, thus, the subscribe request was not lost. Although the first experiment with the security model had a certificate in the authentication process of the IoT–Fog–Cloud layers, it did not affect the number of subscribes and publishes. Furthermore, the number of published messages (PublishIn.Success and PublishOut.Success) for the IoT–Fog–Cloud architecture with the blockchain-based certificate model remained the same as that of the architecture without the model.

The second and third experiments of the IoT–Fog–Cloud architecture were performed using two and three IoT devices with four and six subscribes and publishes, respectively. The results show that the number of subscribes and publishes (i.e., Subscribe.Success and Connect.Success) were the same for the IoT–Fog–Cloud architecture with and without the blockchain-based certificate model and reflect the defined number of subscribes and publishes for two or three IoT devices. This is because when the fog

device distributed the certificates to the two or three IoT devices, those devices were authenticated simultaneously to the fog device. Therefore, there was no sign of failure in the number of connects and subscribes because the two or three IoT devices remained authenticated to the fog device and started publishing messages. Moreover, the number of published messages (PublishIn.Success and PublishOut.Success) for the IoT–Fog–Cloud architecture with and without the blockchain-based certificate model also remained identical using two or three IoT devices; there was no loss in the number of published messages, as there was no sign of failure in the number of connects and subscribes (Connect.Success and Subscribe.Success) because the two or three IoT devices remain authenticated to fog device and start publishing messages at the same time.

Overall, we found that the performance of the IoT–Fog–Cloud architecture with and without the blockchain-based certificate model was the same when using one, two, or three IoT devices. Thus, there was no delay in the number of published messages for the IoT–Fog–Cloud architecture with the blockchain-based certificate model, as shown in table 3. This is because the first layer of security requirements, authentication, was proposed and added to the IoT–Fog–Cloud architecture, and it did not affect its performance. This means that the IoT–Fog–Cloud architecture had improved performance and security simultaneously.

5.6.2 IoT–Fog–Cloud architecture with access control model versus without access control model

In this section, we evaluate the IoT–Fog–Cloud architecture with our access control model using AWS cloud metrics, as shown in Table 5.3.

Table 5.3 AWS cloud message broker metrics results on N. Virginia datacenter (cloud layer) IoT–Fog–Cloud architecture with access control model versus without access control model.

AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using Python script – **IoT–Fog–Cloud architecture without access control model**						AWS IoT message broker metrics on N. Virginia datacenter (cloud layer) using Python script – **IoT–Fog–Cloud architecture with access control model**					
Number of IoT devices	1					Number of IoT devices	1				
Number of subscribing & publishing	2					Number of subscribing & publishing	2				
AWS IoT metrics in minutes (m)	0.5	1	5	15	60	AWS IoT metrics in minutes (m)	0.5	1	5	15	60

Connect.Success	2	2	2	2	2	Connect.Success	2	2	2	2	2
Ping.Success	2	2	8	29	120	Ping.Success	2	2	8	29	120
PublishIn.Success	44	44	206	586	2360	PublishIn.Success	42	42	204	584	2358
PublishOut.Success	44	44	206	586	2360	PublishOut.Success	42	42	204	584	2358
Subscribe.Success	2	2	2	2	2	Subscribe.Success	2	2	2	2	2
Unsubscribe.Success	2	2	2	2	2	Unsubscribe.Success	2	2	2	2	2
Number of IoT devices			2			Number of IoT devices			2		
Number of subscribing & publishing			4			Number of subscribing & publishing			4		
AWS IoT metrics in minutes	0.5	1	5	15	60	AWS IoT metrics in minutes	0.5	1	5	15	60
Connect.Success	4	4	4	4	4	Connect.Success	4	4	4	4	4
Ping.Success	2	2	8	29	120	Ping.Success	2	2	8	29	120
PublishIn.Success	70	70	350	1110	4590	PublishIn.Success	68	68	348	1108	4588
PublishOut.Success	70	70	350	1110	4590	PublishOut.Success	68	68	348	1108	4588
Subscribe.Success	4	4	4	4	4	Subscribe.Success	4	4	4	4	4
Unsubscribe.Success	4	4	4	4	4	Unsubscribe.Success	4	4	4	4	4
Number of IoT devices			3			Number of IoT devices			3		
Number of subscribing & publishing			6			Number of subscribing & publishing			6		
AWS IoT metrics in minutes	0.5	1	5	15	60	AWS IoT metrics in minutes	0.5	1	5	15	60
Connect.Success	6	6	6	6	6	Connect.Success	6	6	6	6	6
Ping.Success	2	2	9	29	119	Ping.Success	2	2	9	29	119
PublishIn.Success	115	115	548	1660	6510	PublishIn.Success	113	113	546	1658	6508
PublishOut.Success	115	115	548	1660	6510	PublishOut.Success	113	113	546	1658	6508
Subscribe.Success	6	6	6	6	6	Subscribe.Success	6	6	6	6	6
Unsubscribe.Success	6	6	6	6	6	Unsubscribe.Success	6	6	6	6	6

The first experiment of the IoT–Fog–Cloud architecture was conducted using one IoT device. We ran this experiment twice simultaneously, one instance without our access control model and the other with the model. We used two subscribes and two publishes because the IoT device generated two types of data: (1) temperature and (2) humidity data. The results show that the number of subscribes and publishes (i.e., Subscribe.Success and Connect.Success) for the IoT–Fog–Cloud architecture with and without the access control model were the same and reflect the defined number of subscribes and publishes for one IoT device. This is because the connection of neither experiment (with vs. without the access control model) was disconnected so the subscribe request was not lost. Although the first experiment with the security model had a certificate in the authentication process of the IoT–Fog-Cloud

layers, it did not affect the number of subscribes and publishes. In contrast, the number of published messages (PublishIn.Success and PublishOut.Success) for the IoT–Fog–Cloud architecture with the access control model was slightly less than that of the architecture without the model. This is because the IoT device performed some security operations that took one second for each sensor data type. Each sensor data type (i.e., temperature data and humidity data) took one second to generate keys containing a set of attributes and encrypt the generated sensor data type value with the corresponding key containing its attribute.

The second experiment of the IoT–Fog–Cloud architecture was performed by connecting two IoT devices and making four subscribe and publish requests. This experiment was also run twice simultaneously, with and without our access control model. The results show that the numbers of subscribe and publish requests (i.e., Subscribe.Success and Connect.Success) for the IoT–Fog–Cloud architecture with and without the access control model were the same for two IoT devices and match the defined number of subscribe and publish requests for one IoT device. However, the number of published messages (PublishIn.Success and PublishOut.Success) for the architecture with the access control model was slightly less than that of the architecture without the model. This is because each of the two IoT devices performed security operations, which took one second for each sensor data type (i.e., temperature data or humidity data) for each of the two IoT devices. Each sensor data type (i.e., temperature data or humidity data) of each of the two IoT devices took one second to generate a key containing a set of attributes and encrypt the generated sensor data type value with the corresponding key containing its attribute.

Overall, we found that the performance of the IoT–Fog–Cloud architecture without the access control model was slightly better than that of the architecture with the model when using one, two, or three IoT devices. However, when using one, two, or three IoT devices, the number of published messages was delayed two seconds when using the access control model, as shown in table 3. This is because the second layer of security requirements, authorization, was proposed and added to the IoT–

Fog–Cloud architecture. Therefore, this makes the IoT–Fog–Cloud architecture has a better performance and security at the same time.

5.7 Conclusion

In this paper, we proposed a blockchain-based certificate model and a fine-grained data access control model based on ABE for the IoT–Fog–Cloud architecture using a real environment. The two proposed models meet the authentication and authorization security requirements of the architecture. We conducted several experiments and increased the numbers of IoT devices, subscribes, and publishes in each experiment. We used AWS cloud metrics to evaluate the performance of the models based on the following analysis methods. First, we compared the performance of the IoT–Fog–Cloud architecture with and without the blockchain-based certificate model. Second, we compared the performance of the architecture with and without the access control model. The results showed that the performance of the IoT–Fog–Cloud architecture with and without the blockchain-based certificate model was the same when using one, two, or three IoT devices. Furthermore, the performance of the IoT–Fog–Cloud architecture without the access control model was slightly better than that of the architecture with the model when using one, two, or three IoT devices. This work aimed to improve the performance and security of the IoT–Fog–Cloud architecture.

Chapter 6: Conclusion

6.1 Main Conclusion and Future Works

The fog layer provides a substantial benefit in cloud-based IoT applications because it can serve as an aggregation layer and brings the computations near the IoT devices. However, it is important to ensure performance in such applications, as they usually communicate frequently and authenticate with the cloud. This can cause performance and availability issues, which can be dangerous in critical applications such as the those used in the healthcare sector. Therefore, this research proposed two architectures of cloud-based IoT environments and three analysis methods. The two proposed architectures are evaluated based on the three analysis methods to show the efficacy of the fog layer in different experiments in a real-world environment by examining performance metrics on the cloud and fog layers using different numbers of IoT devices. In the first architecture, we used a fog layer between IoT devices and the cloud, whereas in the second, IoT devices published data directly to the cloud. In order to validate our results, we also examined two ways of implementing fog-aided IoT–cloud environments, namely (1) bridging and (2) using a Python script to forward the data to the cloud. For each architecture, we conducted several experiments and increased the number of IoT devices as well as the number of subscribe and publish in each experiment. To evaluate the experimental performance, we used two sets of benchmark metrics, namely (1) AWS message broker metrics and (2) Mosquitto message broker metrics. The performance was evaluated based on the following analysis methods. First, we compared the performance of the first and second architectures. Second, we compared the performance of the two implementation frameworks of the fog-aided IoT environments (i.e., Python script vs. bridging). Finally, to validate our results, the performance of the first architecture was analyzed using Mosquitto metrics vs. AWS metrics. The results showed that the performance in the IoT–cloud with a fog layer was significantly better than without the fog layer, as the number of IoT devices and the number of subscribe and publish commands increased. The results also showed that the use of a Python script or fog-aided IoT–cloud environment resulted in the same performance. The results of our

third analysis showed that as the number of IoT devices increased, the processability of the fog device in the fog-aided IoT–cloud architectures outperformed that of the AWS cloud.

To overcome the security challenges between the IoT layer and fog layer and, thus, meet the security requirements, this research also proposed a blockchain-based certificate model and a fine-grained data access control model based on ABE for the IoT–Fog–Cloud architecture using a real environment. The two proposed models meet the authentication and authorization security requirements of the architecture. We conducted several experiments and increased the numbers of IoT devices, subscribes, and publishes in each experiment. We used AWS cloud metrics to evaluate the performance of the models based on the following analysis methods. First, we compared the performance of the IoT–Fog–Cloud architecture with and without the blockchain-based certificate model. Second, we compared the performance of the architecture with and without the access control model. The results showed that the performance of the IoT–Fog–Cloud architecture with and without the blockchainbased certificate model was the same when using one, two, or three IoT devices. Furthermore, the performance of the IoT–Fog–Cloud architecture without the access control model was slightly better than that of the architecture with the model when using one, two, or three IoT devices. This work aimed to improve the performance and security of the IoT–Fog–Cloud architecture.

For future works, we plan to apply the two proposed architecture of cloud-based IoT environment to different types of sensors attached to the IoT devices and different cloud service providers and evaluate the performance and security. Also, we plan to apply the two proposed architecture to the medical domains and evaluate the performance and security.

6.2 list of publications

6.2.1 As a first author

[1] Paper title: Access Control in Fog Computing: Challenges and Research Agenda

Description: Journal paper published online by IEEE Access

Impact Factor: 3.367

Reference: M. A. Aleisa, A. Abuhussein and F. T. Sheldon, "Access Control in Fog Computing: Challenges and Research Agenda," in *IEEE Access*, vol. 8, pp. 83986-83999, 2020, doi: 10.1109/ACCESS.2020.2992460.

[2] Paper title: Performance Analysis of Two Cloud-Based IoT Implementations: Empirical Study

Description: Conference paper in the 7th International Conference on Cyber Security and Cloud Computing (CSCloud) and the 6th International Conference on Edge Computing and Scalable Cloud (EdgeCom). published online by IEEE.

Reference: M. Aleisa, A. A. Hussein, F. Alsubaei and F. T. Sheldon, "Performance Analysis of Two Cloud-Based IoT Implementations: Empirical Study," *2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2020, pp. 276-280, DOI: 10.1109/CSCloud-EdgeCom49738.2020.00055

[3] Paper title: Examining Performance in Fog-Aided Cloud-Centered IoT in a Real-World Environment

Description: Journal paper published online by MDPI sensors

Impact Factor: 3.576

Reference: M. A. Aleisa, A. Abuhussein, F. S. Alsubaei, and F. T. Sheldon, “Examining the Performance of Fog-Aided, Cloud-Centered IoT in a Real-World Environment,” *Sensors*, vol. 21, no. 21, p. 6950, Oct. 2021, doi: 10.3390/s21216950

[4] Paper title: Novel Security Models for IoT-Fog-Cloud Architectures in a Real-World Environment

Description: Journal paper submitted online to MDPI sensors

Impact Factor: 3.576

Reference: M. A. Aleisa, A. Abuhussein, F. S. Alsubaei, and F. T. Sheldon, "Novel Security Models for IoT-Fog-Cloud Architectures in a Real-World Environment," *Sensors*, vol. 21, no. 21, p. 6950, Mar. 2022.

References

[1] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, Sep. 2017, doi: 10.1002/spe.2509.

[2] Fog Computing. (2015). *Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are*. [Online]. Available: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computingoverview.pdf.

[3] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. Goren, and C. Mahmoudi, "Fog computing conceptual model," National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 500-325, Mar. 2018. doi: 10.6028/NIST.SP.500-325.

[4] A. Abuhussein, H. Bedi, and S. Shiva, "Evaluating security and privacy in cloud computing services: A Stakeholder's perspective," in *2012 International Conference for Internet Technology and Secured Transactions*, Dec. 2012, pp. 388–395.

[5] A. Abuhussein, F. Alsubaei, and S. Shiva, "Toward an Effective Requirement Engineering Approach for Cloud Applications," in *Software Engineering in the Era of Cloud Computing*, M. Ramachandran and Z. Mahmood, Eds. Cham: Springer International Publishing, 2020, pp. 29–50. doi: 10.1007/978-3-030-33624-0_2.

[6] R. K. Naha *et al.*, "Fog Computing: Survey of Trends, Architectures, Requirements, and Research Directions," *IEEE Access*, vol. 6, pp. 47980–48009, 2018, doi: 10.1109/ACCESS.2018.2866491.

[7] R. K. Naha, S. Garg, and A. Chan, "Fog Computing Architecture: Survey and Challenges," *arXiv:1811.09047 [cs]*, Nov. 2018, Accessed: Jan. 28, 2020. [Online]. Available: http://arxiv.org/abs/1811.09047

[8] P. Zhang, J. K. Liu, F. R. Yu, M. Sookhak, M. H. Au, and X. Luo, "A Survey on Access Control in Fog Computing," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 144–149, Feb. 2018, doi: 10.1109/MCOM.2018.1700333.

[9] M. Aazam and E.-N. Huh, "Fog Computing Micro Datacenter Based Dynamic Resource Estimation and Pricing Model for IoT," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, Gwangiu, South Korea, Mar. 2015, pp. 687–694. doi: 10.1109/AINA.2015.254.

[10] J. Dizdarevic, F. Carpio, A. Jukan, and X. Masip-Bruin, “Survey of Communication Protocols for Internet-of-Things and Related Challenges of Fog and Cloud Computing Integration,” *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–29, Feb. 2019, doi: 10.1145/3292674.

[11] F. Alsubaei, A. Abuhussein, and S. Shiva, “An Overview of Enabling Technologies for the Internet of Things,” in *Internet of Things A to Z*, John Wiley & Sons, Ltd, 2018, pp. 77–112. doi: 10.1002/9781119456735.ch3.

[12] Q. Wang, D. Chen, N. Zhang, Z. Ding, and Z. Qin, “PCP: A Privacy-Preserving Content-Based Publish–Subscribe Scheme With Differential Privacy in Fog Computing,” *IEEE Access*, vol. 5, pp. 17962–17974, 2017, doi: 10.1109/ACCESS.2017.2748956.

[13] S. Yi, Z. Hao, Z. Qin, and Q. Li, “Fog Computing: Platform and Applications,” in *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, Washington DC, DC, USA, Nov. 2015, pp. 73–78. doi: 10.1109/HotWeb.2015.22.

[14] A. Ahmed *et al.*, “Fog Computing Applications: Taxonomy and Requirements,” *arXiv:1907.11621 [cs]*, Jul. 2019, Accessed: Jan. 28, 2020. [Online]. Available: http://arxiv.org/abs/1907.11621

[15] Y. Guan, J. Shao, G. Wei, and M. Xie, “Data Security and Privacy in Fog Computing,” *IEEE Network*, vol. 32, no. 5, pp. 106–111, Sep. 2018, doi: 10.1109/MNET.2018.1700250.

[16] P. Hu, S. Dhelim, H. Ning, and T. Qiu, “Survey on fog computing: architecture, key technologies, applications and open issues,” *Journal of Network and Computer Applications*, vol. 98, pp. 27–42, Nov. 2017, doi: 10.1016/j.jnca.2017.09.002.

[17] Z. Wan, “Cloud Computing infrastructure for latency sensitive applications,” in *2010 IEEE 12th International Conference on Communication Technology*, Nanjing, China, Nov. 2010, pp. 1399–1402. doi: 10.1109/ICCT.2010.5689022.

[18] B. Tang, Z. Chen, G. Hefferman, T. Wei, H. He, and Q. Yang ‘‘A hierarchical distributed fog computing architecture for big data analysis in smart cities,’’ in *Proc. ASE BigData SocialInformatics*, 2015, pp. 1–6.

[19] F. Y. Okay and S. Ozdemir, “A fog computing based smart grid model,” in *2016 International Symposium on Networks, Computers and Communications (ISNCC)*, Yasmine Hammamet, Tunisia, May 2016, pp. 1–6. doi: 10.1109/ISNCC.2016.7746062.

[20] S. Khan, S. Parkinson, and Y. Qin, “Fog computing security: a review of current applications and security solutions,” *J Cloud Comp*, vol. 6, no. 1, p. 19, Dec. 2017, doi: 10.1186/s13677-017-0090-3.

[21] K. Lee, D. Kim, D. Ha, U. Rajput, and H. Oh, “On security and privacy issues of fog computing supported Internet of Things environment,” in *2015 6th International Conference on*

the Network of the Future (NOF), Montreal, QC, Canada, Sep. 2015, pp. 1–3. doi: 10.1109/NOF.2015.7333287.

[22] I. Stojmenovic and S. Wen, ‘‘The fog computing paradigm: Scenarios and security issues,’’ in *Proc. Federated Conf. Comput. Sci. Inf. Syst.*, Sep. 2014, pp. 1–8.

[23] G. Rahman and C. C. Wen, “Fog Computing, Applications, Security and Challenges, Review,” *IJET*, vol. 7, no. 3, p. 1615, Jul. 2018, doi: 10.14419/ijet.v7i3.12612.

[24] Y. Miao, J. Ma, X. Liu, J. Weng, H. Li, and H. Li, “Lightweight Fine-Grained Search Over Encrypted Data in Fog Computing,” *IEEE Trans. Serv. Comput.*, vol. 12, no. 5, pp. 772–785, Sep. 2019, doi: 10.1109/TSC.2018.2823309.

[25] P. Zhang, Z. Chen, J. K. Liu, K. Liang, and H. Liu, “An efficient access control scheme with outsourcing capability and attribute update for fog computing,” *Future Generation Computer Systems*, vol. 78, pp. 753–762, Jan. 2018, doi: 10.1016/j.future.2016.12.015.

[26] J. Sun, X. Wang, S. Wang, and L. Ren, “A searchable personal health records framework with fine-grained access control in cloud-fog computing,” *PLoS ONE*, vol. 13, no. 11, p. e0207543, Nov. 2018, doi: 10.1371/journal.pone.0207543.

[27] “Health Information Privacy | HHS.gov.” https://www.hhs.gov/hipaa/index.html (accessed Mar. 14, 2020).

[28] W. Stallings and L. Brown, “Computer Security: Principles and Practice, 4th Edition.” /content/one-dot-com/one-dot-com/us/en/higher-education/program.html (accessed Jan. 28, 2020).

[29] M. Mammass and F. Ghadi, “Access Control models: State of the art and comparative study,” in *2014 Second World Conference on Complex Systems (WCCS)*, Agadir, Morocco, Nov. 2014, pp. 431–435. doi: 10.1109/ICoCS.2014.7060973.

[30] N. Smyth, *Security+ Essentials*. eBookFrenzy, 2010. [Online]. Available: http://www.lulu.com/shop/neil-smyth/security-essentials/ebook/product-12565906.html

[31] K. Yang, X. Jia, K. Ren, and B. Zhang, “DAC-MACS: Effective data access control for multi-authority cloud storage systems,” in *2013 Proceedings IEEE INFOCOM*, Turin, Italy, Apr. 2013, pp. 2895–2903. doi: 10.1109/INFCOM.2013.6567100.

[32] R. Mahmud, R. Kotagiri, and R. Buyya, “Fog Computing: A Taxonomy, Survey and Future Directions,” in *Internet of Everything*, B. Di Martino, K.-C. Li, L. T. Yang, and A. Esposito, Eds. Singapore: Springer Singapore, 2018, pp. 103–130. doi: 10.1007/978-981-10-5861-5_5.

[33] K. Vohra and M. Dave, “Multi-Authority Attribute Based Data Access Control in Fog Computing,” *Procedia Computer Science*, vol. 132, pp. 1449–1457, 2018, doi: 10.1016/j.procs.2018.05.078.

[34] Q. Huang, Y. Yang, and L. Wang, “Secure Data Access Control With Ciphertext Update and Computation Outsourcing in Fog Computing for Internet of Things,” *IEEE Access*, vol. 5, pp. 12941–12950, 2017, doi: 10.1109/ACCESS.2017.2727054.

[35] F. Alsubaei, A. Abuhussein, and S. Shiva, “Ontology-Based Security Recommendation for the Internet of Medical Things,” *IEEE Access*, vol. 7, pp. 48948–48960, 2019, doi: 10.1109/ACCESS.2019.2910087.

[36] A. Abuhussein, S. Shiva, and F. T. Sheldon, “CSSR: Cloud Services Security Recommender,” in *2016 IEEE World Congress on Services (SERVICES)*, Jun. 2016, pp. 48–55. doi: 10.1109/SERVICES.2016.13.

[37] M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou, “Scalable and Secure Sharing of Personal Health Records in Cloud Computing Using Attribute-Based Encryption,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 131–143, Jan. 2013, doi: 10.1109/TPDS.2012.97.

[38] Z. Zhou and D. Huang, “Efficient and secure data storage operations for mobile cloud computing,” p. 9.

[39] M. R. Asim, M. Petkovic, and T. Ignatenko, “Attribute-based encryption with encryption and decryption outsourcing,” 2014, pp. 21–28. doi: 10.4225/75/57b65cc3343d0.

[40] C. Zuo, J. Shao, G. Wei, M. Xie, and M. Ji, “CCA-secure ABE with outsourced decryption for fog computing,” *Future Generation Computer Systems*, vol. 78, pp. 730–738, Jan. 2018, doi: 10.1016/j.future.2016.10.028.

[41] E. Fujisaki and T. Okamoto, “Secure Integration of Asymmetric and Symmetric Encryption Schemes,” *J Cryptol*, vol. 26, no. 1, pp. 80–101, Jan. 2013, doi: 10.1007/s00145-011-9114-1.

[42] R. Canetti, S. Halevi, and J. Katz, “Chosen-Ciphertext Security from Identity-Based Encryption,” *Springer, Berlin, Heidelberg*, vol. 3027, p. 16, 2004, doi: https://doi.org/10.1007/978-3-540-24676-3_13.

[43] A. Alrawais, A. Alhothaily, C. Hu, X. Xing, and X. Cheng, “An Attribute-Based Encryption Scheme to Secure Fog Communications,” *IEEE Access*, vol. 5, pp. 9131–9138, 2017, doi: 10.1109/ACCESS.2017.2705076.

[44] M. Mukherjee *et al.*, “Security and Privacy in Fog Computing: Challenges,” *IEEE Access*, vol. 5, pp. 19293–19304, 2017, doi: 10.1109/ACCESS.2017.2749422.

[45] F. Alsubaei, A. Abuhussein, and S. Shiva, “Security and Privacy in the Internet of Medical Things: Taxonomy and Risk Assessment,” in *2017 IEEE 42nd Conference on Local Computer Networks Workshops (LCN Workshops)*, Oct. 2017, pp. 112–120. doi: 10.1109/LCN.Workshops.2017.72.

[46] F. Alsubaei, A. Abuhussein, V. Shandilya, and S. Shiva, “IoMT-SAF: Internet of Medical Things Security Assessment Framework,” *Internet of Things*, vol. 8, p. 100123, Dec. 2019, doi: 10.1016/j.iot.2019.100123.

[47] S. Yi, Z. Qin, and Q. Li, ‘‘Security and privacy issues of fog computing: A survey,’’ in *Proc. Int. Conf. Wireless Algorithms, Syst., Appl*., 2015, pp. 685–695.

[48] I. Stojmenovic, S. Wen, X. Huang, and H. Luan, “An overview of Fog computing and its security issues,” *Concurrency and Computation: Practice and Experience*, vol. 28, no. 10, pp. 2991–3005, Jul. 2016, doi: 10.1002/cpe.3485.

[49] N. Abubaker, L. Dervishi, and E. Ayday, “Privacy-preserving fog computing paradigm,” in *2017 IEEE Conference on Communications and Network Security (CNS)*, Las Vegas, NV, Oct. 2017, pp. 502–509. doi: 10.1109/CNS.2017.8228709.

[50] M. A. Ferrag, A. Derhab, L. Maglaras, M. Mukherjee, and H. Janicke, “Privacy-preserving Schemes for Fog-based IoT Applications: Threat models, Solutions, and Challenges,” in *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, El Oued, Oct. 2018, pp. 37–42. doi: 10.1109/SaCoNeT.2018.8585538.

[51] Y. Zhang *et al.*, “Privacy-Preserving Data Aggregation against False Data Injection Attacks in Fog Computing,” *Sensors*, vol. 18, no. 8, p. 2659, Aug. 2018, doi: 10.3390/s18082659.

[52] K. Xue, J. Hong, Y. Ma, D. S. L. Wei, P. Hong, and N. Yu, “Fog-Aided Verifiable Privacy Preserving Access Control for Latency-Sensitive Data Sharing in Vehicular Cloud Computing,” *IEEE Network*, vol. 32, no. 3, pp. 7–13, May 2018, doi: 10.1109/MNET.2018.1700341.

[53] H. Wang, D. He, J. Shen, Z. Zheng, C. Zhao, and M. Zhao, “Verifiable outsourced ciphertext-policy attribute-based encryption in cloud computing,” *Soft Comput*, vol. 21, no. 24, pp. 7325–7335, Dec. 2017, doi: 10.1007/s00500-016-2271-2.

[54] K. Fan, J. Wang, X. Wang, H. Li, and Y. Yang, “A Secure and Verifiable Outsourced Access Control Scheme in Fog-Cloud Computing,” *Sensors*, vol. 17, no. 7, p. 1695, Jul. 2017, doi: 10.3390/s17071695.

[55] Q. Xu, C. Tan, Z. Fan, W. Zhu, Y. Xiao, and F. Cheng, “Secure Data Access Control for Fog Computing Based on Multi-Authority Attribute-Based Signcryption with Computation Outsourcing and Attribute Revocation,” *Sensors*, vol. 18, no. 5, p. 1609, May 2018, doi: 10.3390/s18051609.

[56] X. Mao, J. Lai, Q. Mei, K. Chen, and J. Weng, “Generic and Efficient Constructions of Attribute-Based Encryption with Verifiable Outsourced Decryption,” *IEEE Trans. Dependable*

and Secure Comput., vol. 13, no. 5, pp. 533–546, Sep. 2016, doi: 10.1109/TDSC.2015.2423669.

[57] J. Li, X. Lin, Y. Zhang, and J. Han, "KSF-OABE: Outsourced Attribute-Based Encryption with Keyword Search Function for Cloud Storage," *IEEE Trans. Serv. Comput.*, vol. 10, no. 5, pp. 715–725, Sep. 2017, doi: 10.1109/TSC.2016.2542813.

[58] M. A. Aleisa, A. Abuhussein, and F. T. Sheldon, "Access Control in Fog Computing: Challenges and Research Agenda," *IEEE Access*, vol. 8, pp. 83986–83999, May 2020, doi: 10.1109/ACCESS.2020.2992460.

[59] F. Alsubaei, A. Abuhussein, and S. Shiva, "An Overview of Enabling Technologies for the Internet of Things," in *Internet of Things A to Z*, Wiley-IEEE Press, 2018, pp. 77–112. doi: 10.1002/9781119456735.ch3.

[60] M. Arlitt, M. Marwah, G. Bellala, A. Shah, J. Healey, and B. Vandiver, "IoTAbench: an Internet of Things Analytics Benchmark," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering - ICPE '15*, Austin, Texas, USA, 2015, pp. 133–144. doi: 10.1145/2668930.2688055.

[61] A. Das, S. Patterson, and M. Wittie, "EdgeBench: Benchmarking Edge Computing Platforms," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, Zurich, Dec. 2018, pp. 175–180. doi: 10.1109/UCC-Companion.2018.00053.

[62] T. Hao *et al.*, "Edge AIBench: Towards Comprehensive End-to-End Edge Computing Benchmarking," in *Benchmarking, Measuring, and Optimizing*, vol. 11459, C. Zheng and J. Zhan, Eds. Cham: Springer International Publishing, 2019, pp. 23–30. doi: 10.1007/978-3-030-32813-9_3.

[63] M. Poess, R. Nambiar, K. Kulkarni, C. Narasimhadevara, T. Rabl, and H.-A. Jacobsen, "Analysis of TPCx-IoT: The First Industry Standard Benchmark for IoT Gateway Systems," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, Paris, Apr. 2018, pp. 1519–1530. doi: 10.1109/ICDE.2018.00170.

[64] "AWS IoT - Amazon Web Services," *Amazon Web Services, Inc.* https://aws.amazon.com/iot/ (accessed Nov. 20, 2020).

[65] Amazon Web Services, Inc. *Amazon CloudWatch - User Guide*. (2020). Accessed: May 15, 2020. [Online]. Available: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/acw-ug.pdf.

[66] Amazon Web Services, Inc., "*AWS IoT - Developer Guide*." Accessed: May 15, 2020. [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/iot-dg.pdf

[67] A. A. Ismail, H. S. Hamza, and A. M. Kotb, “Performance Evaluation of Open Source IoT Platforms,” in *2018 IEEE Global Conference on Internet of Things (GCIoT)*, Alexandria, Egypt, Dec. 2018, pp. 1–5. doi: 10.1109/GCIoT.2018.8620130.

[68] Transaction Processing Performance Council (TPC). *TPC Express Benchmark IoT (TPCx-IoT) Standard Specification Version 1.0.5*. (2020). Accessed: May 15, 2020. [Online]. Available: http://www.tpc.org/tpc_documents_current_versions/pdf/tpcx-iot_v1.0.5.pdf

[69] B. Boyd et al., *Building Realtime Mobile Solutions with MQTT and IBM MessageSight*. Poughkeepsie, NY, USA: IBM Redbooks, 2014. Accessed: May 15, 2020. [Online]. Available: http://www.redbooks.ibm.com/redbooks/pdfs/sg248228.pdf

[70] Shinho Lee, Hyeonwoo Kim, Dong-kweon Hong, and Hongtaek Ju, “Correlation analysis of MQTT loss and delay according to QoS level,” in *The International Conference on Information Networking 2013 (ICOIN)*, Bangkok, Jan. 2013, pp. 714–717. doi: 10.1109/ICOIN.2013.6496715.

[71] A. Industries, “DHT11 basic temperature-humidity sensor + extras.” https://www.adafruit.com/product/386 (accessed May 15, 2020).

[72] Raspberry Pi Foundation. “Buy a Raspberry Pi 3 Model B – Raspberry Pi.” Accessed: May 15, 2020. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[73] K. Albulayhi, A. Abuhussein, F. Alsubaei, and F. T. Sheldon, “Fine-Grained Access Control in the Era of Cloud Computing: An Analytical Review,” in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan. 2020, pp. 0748–0755. doi: 10.1109/CCWC47524.2020.9031179.

[74] F. Alsubaei, A. Abuhussein, V. Shandilya, and S. Shiva, “IoMT-SAF: Internet of Medical Things Security Assessment Framework,” *Internet of Things*, vol. 8, p. 100123, Dec. 2019, doi: 10.1016/j.iot.2019.100123.

[75] M. Aleisa, A. A. Hussein, F. Alsubaei, and F. T. Sheldon, “Performance Analysis of Two Cloud-Based IoT Implementations: Empirical Study,” in *2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, New York, NY, USA, Aug. 2020, pp. 276–280. doi: 10.1109/CSCloud-EdgeCom49738.2020.00055.

[76] Fog Computing. (2015). *Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are*. [Online]. Available: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computingoverview.pdf

[77] J. Ni, K. Zhang, X. Lin, and X. Shen, "Securing Fog Computing for Internet of Things Applications: Challenges and Solutions," *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, pp. 601–628, Firstquarter 2018, doi: 10.1109/COMST.2017.2762345.

[78] "What is an IoT Platform & What Role Does it Play In Your Business?," *AT&T Business*. /content/attbusiness/en/learn/research-reports/whats-an-iot-platform-and-what-role-does-it-play.html (accessed Feb. 13, 2021).

[79] "AWS IoT Core Overview - Amazon Web Services," *Amazon Web Services, Inc.* https://aws.amazon.com/iot-core/ (accessed Feb. 13, 2021).

[80] "Azure IoT – Internet of Things Platform | Microsoft Azure." https://azure.microsoft.com/en-us/overview/iot/ (accessed Feb. 13, 2021).

[81] "Internet of Things | IBM." https://www.ibm.com/cloud/internet-of-things (accessed Feb. 13, 2021).

[82] "Cloud IoT Core," *Google Cloud*. https://cloud.google.com/iot-core (accessed Feb. 13, 2021).

[83] "Home | IoTivity." https://iotivity.org/ (accessed Feb. 13, 2021).

[84] "Zetta - An API-First Internet of Things (IoT) Platform - Free and Open Source Software," *Zetta - An API-First Internet of Things (IoT) Platform - Free and Open Source Software*. https://www.zettajs.org/ (accessed Feb. 13, 2021).

[85] "What is Arduino?" https://www.arduino.cc/en/Guide/Introduction (accessed Feb. 13, 2021).

[86] "DeviceHive - Open Source IoT Data Platform with the wide range of integration options." https://devicehive.com/ (accessed Feb. 13, 2021).

[87] "OpenRemote | The 100% Open Source IoT Platform," *OpenRemote*. https://openremote.io/ (accessed Feb. 13, 2021).

[88] "WhatsApp," *WhatsApp.com*. https://www.whatsapp.com/?lang=en (accessed Feb. 13, 2021).

[89] "Telegram – a new era of messaging," *Telegram*. https://telegram.org/?setln=en (accessed Feb. 14, 2021).

[90] H. Wang, D. Xiong, P. Wang, and Y. Liu, "A Lightweight XMPP Publish/Subscribe Scheme for Resource-Constrained IoT Devices," *IEEE Access*, vol. 5, pp. 16393–16405, 2017, doi: 10.1109/ACCESS.2017.2742020.

[91] S. Bendel, T. Springer, D. Schuster, A. Schill, R. Ackermann, and M. Ameling, "A service infrastructure for the Internet of Things based on XMPP," in *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, San Diego, CA, Mar. 2013, pp. 385–388. doi: 10.1109/PerComW.2013.6529522.

[92] A. Stanford-Clark and H. L. Truong, "MQTT For Sensor Networks (MQTT-SN) Protocol Specification," p. 28, 1999.

[93] "Home | AMQP." https://www.amqp.org/ (accessed Feb. 28, 2021).

[94] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in *2017 IEEE International Systems Engineering Symposium (ISSE)*, Oct. 2017, pp. 1–7. doi: 10.1109/SysEng.2017.8088251.

[95] Amazon Web Services, Inc., "*AWS IoT - Developer Guide*." Accessed: May 15, 2020. [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/iot-dg.pdf

[96] Amazon Web Services, Inc. *Amazon CloudWatch - User Guide*. (2020). Accessed: May 15, 2020. [Online]. Available: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/acw-ug.pdf

[97] "Eclipse Mosquitto," *Eclipse Mosquitto*, Jan. 08, 2020. https://mosquitto.org/ (accessed Nov. 20, 2020).

[98] "Messaging that just works — RabbitMQ." https://www.rabbitmq.com/ (accessed Feb. 13, 2021).

[99] "ActiveMQ." https://activemq.apache.org/ (accessed Feb. 13, 2021).

[100] "Mosquitto man page," *Eclipse Mosquitto*, Apr. 17, 2019. https://mosquitto.org/man/mosquitto-8.html (accessed Nov. 21, 2020).

[101] "SPEC - Standard Performance Evaluation Corporation." https://www.spec.org/ (accessed Dec. 12, 2020).

[102] "TPC-Homepage." http://tpc.org/ (accessed Dec. 12, 2020).

[103] "tpcx-iot_v1.0.5.pdf." Accessed: Nov. 20, 2020. [Online]. Available: http://tpc.org/tpc_documents_current_versions/pdf/tpcx-iot_v1.0.5.pdf

[104] E. Vanneback, "Using the Mosquitto implementation in an embedded environment," p. 56.

[105] S. Maksuti, O. Schluga, G. Settanni, M. Tauber, and J. Delsing, "Self-Adaptation Applied to MQTT via a Generic Autonomic Management Framework," in *2019 IEEE International Conference on Industrial Technology (ICIT)*, Melbourne, Australia, Feb. 2019, pp. 1179–1185. doi: 10.1109/ICIT.2019.8754937.

[106] M. Aazam, S. Zeadally, and K. A. Harras, "Fog Computing Architecture, Evaluation, and Future Research Directions," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 46–52, May 2018, doi: 10.1109/MCOM.2018.1700707.

[107] S. E. Kafhali, K. Salah, and S. B. Alla, "Performance Evaluation of IoT-Fog-Cloud Deployment for Healthcare Services," in *2018 4th International Conference on Cloud Computing Technologies and Applications (Cloudtech)*, Nov. 2018, pp. 1–6. doi: 10.1109/CloudTech.2018.8713355.

[108] P. H. Vilela, J. J. P. C. Rodrigues, P. Solic, K. Saleem, and V. Furtado, "Performance evaluation of a Fog-assisted IoT solution for e-Health applications," *Future Generation Computer Systems*, vol. 97, pp. 379–386, Aug. 2019, doi: 10.1016/j.future.2019.02.055.

[109] J. McChesney, N. Wang, A. Tanwer, E. de Lara, and B. Varghese, "DeFog: Fog Computing Benchmarks," *arXiv:1907.10890 [cs]*, Jul. 2019, Accessed: Dec. 12, 2020. [Online]. Available: http://arxiv.org/abs/1907.10890

[110] A. Industries, "DHT11 basic temperature-humidity sensor + extras." https://www.adafruit.com/product/386 (accessed Nov. 20, 2020).

[111] Raspberry Pi Foundation. "Buy a Raspberry Pi 3 Model B – Raspberry Pi." Accessed: May 15, 2020. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[112] "Software." https://www.arduino.cc/en/software (accessed Feb. 13, 2021).

[113] "How to Bridge Mosquitto MQTT Broker to AWS IoT," *Amazon Web Services*, May 04, 2020. https://aws.amazon.com/blogs/iot/how-to-bridge-mosquitto-mqtt-broker-to-aws-iot/ (accessed Feb. 13, 2021).

[114] M. N. Birje and C. Bulla, "Commercial and Open Source Cloud Monitoring Tools: A Review," in *Advances in Decision Sciences, Image Processing, Security and Computer Vision*, vol. 3, S. C. Satapathy, K. S. Raju, K. Shyamala, D. R. Krishna, and M. N. Favorskaya, Eds. Cham: Springer International Publishing, 2020, pp. 480–490. doi: 10.1007/978-3-030-24322-7_59.

[115] M. S. Aslanpour, S. S. Gill, and A. N. Toosi, "Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research," *Internet of Things*, vol. 12, p. 100273, Dec. 2020, doi: 10.1016/j.iot.2020.100273.

[116] A. Stephen, S. Benedict, and R. P. A. Kumar, "Monitoring IaaS using various cloud monitors," *Cluster Comput*, vol. 22, no. S5, pp. 12459–12471, Sep. 2019, doi: 10.1007/s10586-017-1657-y.

[117] P. Jutadhamakorn, T. Pillavas, V. Visoottiviseth, R. Takano, J. Haga, and D. Kobayashi, "A Scalable and Low-Cost MQTT Broker Clustering System," p. 5, 2017.

[118] G. Nebbione and M. C. Calzarossa, "Security of IoT Application Layer Protocols: Challenges and Findings," *Future Internet*, vol. 12, no. 3, p. 55, Mar. 2020, doi: 10.3390/fi12030055.

[119] K. Yunana, A. A. Alfa, S. Misra, R. Damasevicius, R. Maskeliunas, and J. Oluranti, "Internet of Things: Applications, Adoptions and Components - A Conceptual Overview," in *Hybrid Intelligent Systems*, Cham, 2021, pp. 494–504. doi: 10.1007/978-3-030-73050-5_50.

[120] W. Zhou, Y. Jia, A. Peng, Y. Zhang, and P. Liu, "The Effect of IoT New Features on Security and Privacy: New Threats, Existing Solutions, and Challenges Yet to Be Solved," *IEEE*

Internet of Things Journal, vol. 6, no. 2, pp. 1606–1616, Apr. 2019, doi: 10.1109/JIOT.2018.2847733.

[121] Y. I. Alzoubi, V. H. Osmanaj, A. Jaradat, and A. Al-Ahmad, "Fog computing security and privacy for the Internet of Thing applications: State-of-the-art," *SECURITY AND PRIVACY*, vol. 4, no. 2, p. e145, 2021, doi: 10.1002/spy2.145.

[122] M. A. Aleisa, A. Abuhussein, F. S. Alsubaei, and F. T. Sheldon, "Examining the Performance of Fog-Aided, Cloud-Centered IoT in a Real-World Environment," *Sensors*, vol. 21, no. 21, p. 6950, Jan. 2021, doi: 10.3390/s21216950.

[123] "The DDoS Attack on Dyn's DNS Infrastructure." https://www.thousandeyes.com/blog/dyn-dns-ddos-attack/ (accessed Feb. 13, 2022).

[124] "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications." https://ieeexplore.ieee.org/abstract/document/7879243/ (accessed Feb. 13, 2022).

[125] A. A.-N. Patwary *et al.*, "Authentication, Access Control, Privacy, Threats and Trust Management Towards Securing Fog Computing Environments: A Review," *arXiv:2003.00395 [cs]*, Feb. 2020, Accessed: Feb. 13, 2022. [Online]. Available: http://arxiv.org/abs/2003.00395

[126] Raspberry Pi Foundation. "Buy a Raspberry Pi 3 Model B – Raspberry Pi." Accessed: May 15, 2020. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[127] "Eclipse Mosquitto," *Eclipse Mosquitto*, Jan. 08, 2018. https://mosquitto.org/ (accessed Feb. 13, 2022).

Appendix A: Awards and Honors

During my study at the University of Idaho, I have published three journal papers with high impact factors and one conference paper. I have been eligible for funding the first journal paper from the University of Idaho Open Access Publishing Fund (OAPF) for the year of 2020. Also, I have been awarded for funding the second journal paper from the University of Idaho Open Access Publishing Fund (OAPF) for the year of 2021. In addition, I have received a Travel Award for my conference paper from the University of Idaho Graduate and Professional Student Association (GPSA) for the year of 2020. Moreover, I have been invited to be a reviewer for some journals such as IEEE Access, and Wiley.