

DESIGN AND IMPLEMENTATION OF AN IOT-BASED HOME AUTOMATION
SYSTEM UTILIZING FOG AND CLOUD COMPUTING PARADIGMS

by

Hüseyin Anıl Özmen

B.S., Computer Engineering, Boğaziçi University, 2015

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University

2018

ACKNOWLEDGEMENTS

First of all, I would like to express my greatest appreciation to my supervisor Prof. Cem Ersoy for his generosity in bringing my thesis to a good level. His experiences and sincere approach in all subjects are the most important factors in the completion of this thesis.

My co-advisor, Dr. Sinan Işık, was very helpful while I was writing the thesis and implementing the FOGHA system. He shared lots of open-minded ideas with me which form the milestones of the thesis.

I would like to thank all NETLAB members for their friendship. I would also like to thank Ahmet Cihat Baktır and Orhan Ermiş for their contribution to the thesis. Alper Alimoğlu, Barış Yamansavcılar, Can Tunca, and Serhan Daniş also deserve a genuine appreciation.

My family always supported me about all my decisions which deserves the deepest gratitude.

ABSTRACT

DESIGN AND IMPLEMENTATION OF AN IOT-BASED HOME AUTOMATION SYSTEM UTILIZING FOG AND CLOUD COMPUTING PARADIGMS

There are considerable amount of architectures and implementations for home automation systems in both industry and academia. Many of them target to inform home residents who can be anywhere in the world using Cloud technologies. In addition, some of them could be named as passive systems which only monitor events, environmental conditions and inform the owner to take actions. This traditional approach is changing in the way of “smart things” with their increased processing capabilities. They are becoming active appliances which can make decisions by analysing the current state of the other appliances in the environment.

In this study, we designed and implemented a well-structured home automation system, namely FOGHA, which enables both remote and local management of the home environment by separating tasks into layers and modules. Since direct communication between home appliances and Cloud servers is not desirable because of reliability, security and orchestration concerns, we divided the computation nodes into two main subsystems as Cloud and Fog. The latter subsystem is physically near to smart things and local orchestration of home environment is provided via Fog managers. In our system, Raspberry Pi (RPI) single board computers are used for this purpose in a distributed and fault-tolerant manner. Thus, they can be seen as the local brains of the overall system. While the gateway Fog manager additionally orchestrates other Fog managers and communicates with the Cloud, others collect data from smart things and manage them. Additional to the system design, we implemented a prototype of the FOGHA architecture and performed functional tests on it.

ÖZET

SİS VE BULUT HESAPLAMA YAKLAŞIMLARINI KULLANAN NESNELERİN İNTERNETİ TABANLI EV OTOMASYON SİSTEM TASARIMI VE UYGULAMASI

Endüstride ve akademik çevrelerde ev otomasyon sistemleri alanında önemli miktarda mimari ve uygulama bulunmaktadır. Bunların birçoğu, bulut teknolojilerini kullanarak herhangi bir yerde bulunan ev sakinlerini bilgilendirmeyi hedeflemektedir. Ayrıca, bazıları yalnızca olayları, çevre koşullarını izleyen ve ev sakinlerini bilgilendiren pasif sistemler olarak adlandırılabilir. Bu geleneksel yaklaşım akıllı nesnelerin gelişmesi ve onların veri işleme yeteneklerinin artmasıyla değişiyor. Onlar artık çevredeki diğer cihazların mevcut durumunu analiz ederek karar verebilen aktif cihazlar haline geliyorlar.

Bu çalışmada, görevleri katmanlara ve modüllere ayırarak ev ortamının uzaktan veya yerel olarak yönetilmesini sağlayan, iyi yapılandırılmış bir ev otomasyon sistemi olan FOGHA tasarlanmış ve uygulaması yapılmıştır. Güvenilirlik, güvenlik ve yönetebilme kaygıları nedeniyle ev aletleri ve bulut sunucuları arasındaki doğrudan iletişim arzu edilen bir durum olmadığı için, hesaplama düğümleri bulut ve sis olmak üzere iki ana alt sisteme ayrılmıştır. Sis katmanı fiziksel olarak akıllı nesnelere yakındır ve ev ortamının yerelde yönetilmesi onlar tarafından sağlanmaktadır. Sistemimizde, Raspberry Pi (RPI) cihazları bu amaçla dağıtık ve hatalara toleranslı şekilde kullanılmaktadır. Bu nedenle, bu cihazlar genel sistemin yerel beyinleri olarak görülebilirler. Ağ geçidi sis katmanı yöneticisi, diğerlerine ek olarak sis katmanını yönetirken ve bulut sunucularla iletişim kurarken, diğer sis katmanı yöneticileri çevredeki akıllı nesnelerden veri toplar ve onları yönetir. Sistem tasarımına ilaveten, bir FOGHA mimarisi prototipi hazırladık ve üzerinde işlevsel testler yaptık.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
ÖZET	v
LIST OF FIGURES	x
LIST OF TABLES	xiii
LIST OF ACRONYMS/ABBREVIATIONS	xiv
1. INTRODUCTION	1
1.1. History of Home Automation	3
1.2. Future of Smart Homes	4
1.3. Objective of the Thesis	6
2. FUNDAMENTALS OF HOME AUTOMATION SYSTEMS	8
2.1. Division of Tasks and Services	8
2.2. Home Automation Requirements	9
2.2.1. System Level Requirements	9
2.2.2. Smart Devices' Requirements	11
2.2.3. Management Software Requirements	11
2.2.4. Latency Requirements	13
2.3. Home Automation System Components	14
2.3.1. Single Board Computers	15
2.3.1.1. Arduino	16
2.3.1.2. Beaglebone Black	17
2.3.1.3. Intel Edison	17
2.3.1.4. Udoo NEO	18
2.3.1.5. Raspberry Pi 3	18
2.3.2. Home Automation Frameworks	19
2.3.3. Infrastructure Protocols	19
2.3.3.1. Z-Wave	20
2.3.3.2. Bluetooth Low Energy	21
2.3.3.3. LTE-A (Long Term Evolution - Advanced)	21

2.3.3.4.	RFID	22
2.3.3.5.	IEEE 802.15.4	22
2.3.3.6.	6LowPAN	22
2.3.4.	Application Protocols	24
2.3.4.1.	Message Queue Telemetry Transport (MQTT)	24
2.3.4.2.	Extensible Messaging and Presence Protocol (XMPP)	24
2.3.4.3.	Constrained Application Protocol (CoAP)	25
2.3.4.4.	Advanced Message Queuing Protocol (AMQP)	25
2.3.4.5.	Data Distribution Service (DDS)	25
2.3.4.6.	Websocket	26
3.	LITERATURE SURVEY ON HOME AUTOMATION ARCHITECTURES	28
3.1.	Home Automation Domain	28
3.2.	Fog Computing Domain	32
4.	FOGHA HOME AUTOMATION ARCHITECTURE	35
4.1.	General Overview	35
4.2.	Fog Layer	38
4.2.1.	Data Processor Nodes: Fog Managers	40
4.2.1.1.	Operating System	41
4.2.1.2.	Software Development on Fog Managers	42
4.2.2.	Wireless Ad-Hoc Network: FOGHANET	43
4.2.3.	Software Containerization	45
4.2.3.1.	Docker Containers	47
4.2.3.2.	Docker Swarm Mode	48
4.2.4.	Software Stack of Fog Managers	50
4.2.4.1.	Node-Red Server	51
4.2.4.2.	InfluxDB Database	54
4.2.4.3.	Chronograf	55
4.2.4.4.	Redis In-Memory Cache	55
4.2.4.5.	MQTT Broker	57
4.2.5.	Management of Fog Layer	59
4.2.6.	Auxiliary Scripts	60

4.2.6.1.	Monitoring Raspberry Pi Boards	61
4.2.6.2.	Local Communication Mechanisms	61
4.3.	Things Layer	63
4.3.1.	Communication Protocols and Technologies	64
4.3.2.	Actuators	65
4.3.3.	Sensors	66
4.3.4.	Fog Layer and Things Layer Interaction	67
4.4.	Cloud Layer	69
4.4.1.	Cloud Managers	69
4.4.2.	Software Stack of Cloud Managers	70
4.4.3.	Fog - Cloud Layer Interaction	70
4.5.	User Interaction	71
4.5.1.	Web Browser Interface	72
4.5.2.	Android Application	72
4.6.	Integrating New Smart Devices and Applications to the FOGHA	73
4.7.	System Level Evaluation of the FOGHA Architecture	74
4.7.1.	Performance	74
4.7.2.	Reliability	75
4.7.3.	Scalability	76
4.7.4.	Security	76
4.7.5.	Availability	77
4.7.6.	Manageability	78
4.7.7.	Privacy	78
5.	FOGHA USE CASES	79
5.1.	Implemented Use Cases	79
5.1.1.	Controlling Home Appliances via GUI	79
5.1.2.	Disaster Monitoring and Reporting	81
5.1.3.	Location Based User Tracking	83
5.1.4.	Face Recognition in Fog Layer	83
5.1.5.	Video Streaming in Fog Layer	86
5.1.6.	Chat-bot and Natural Language Understanding	88

5.1.7. Real Time Information Exchange	89
5.2. Supported Use Cases	91
6. PERFORMANCE EVALUATION	95
6.1. Comparison of Home Automation Frameworks	96
6.2. Scalability Tests in Fog Layer	98
6.3. Resource Utilization of Fog Managers	101
6.4. Comparison of Docker and Native Implementation	103
6.5. Network Performance Evaluation	103
6.6. Face Recognition Tests	106
7. CONCLUSION AND FUTURE WORK	115
REFERENCES	118

LIST OF FIGURES

Figure 4.1.	Layered System Architecture.	36
Figure 4.2.	Distribution of RPI Boards in a Smart Home Scenario	39
Figure 4.3.	The Physical Components of the Fog Layer.	40
Figure 4.4.	Data Transmission Methods at Fog Computing Layer.	43
Figure 4.5.	Docker Containers and Virtual Machines.	45
Figure 4.6.	Software Development and Deployment via Docker Containers. . .	47
Figure 4.7.	Orchestration of Containers and Services.	50
Figure 4.8.	Node-Red Development Environment	53
Figure 4.9.	Example Scenario: Communication with BLE Sensors.	54
Figure 4.10.	Visualization of the Resource Usage of the Fog Managers.	56
Figure 4.11.	Data Storage Mechanism.	57
Figure 4.12.	MQTT Communication Mechanism.	58
Figure 4.13.	Monitoring of RPI Boards.	62
Figure 4.14.	Host OS-Container Bridge.	63

Figure 4.15. Functionality of Actuators.	67
Figure 4.16. Interaction Between Fog and Things Layers.	68
Figure 4.17. The Data Forwarding Mechanism.	71
Figure 5.1. The Developed Smart Plug.	80
Figure 5.2. Disaster Monitoring with the FOGHA.	82
Figure 5.3. User Location Tracking Feature.	84
Figure 5.4. Face Recognition with OpenCV.	85
Figure 5.5. Face Recognition Algorithm.	87
Figure 5.6. Chatbot Implementation and Training of the Foghabot.	88
Figure 5.7. Natural Language Understanding Mechanism of FOGHA.	90
Figure 6.1. The Environment Used in the Performance Evaluation Tests.	96
Figure 6.2. CPU Usage of Frameworks	97
Figure 6.3. Memory Usage of Frameworks.	97
Figure 6.4. MQTT Benchmark Test.	99
Figure 6.5. The Effect of Number of MQTT Clients.	100
Figure 6.6. CPU Usage of the RPI boards per Container.	102

Figure 6.7.	The Effect of Distance for the MQTT messages.	107
Figure 6.8.	Face Recognition Task Analysis using RPI boards.	109
Figure 6.9.	Face Recognition Task Analysis using Cloud Nodes.	110
Figure 6.10.	Face Recognition Task Analysis Using Small Sized Image Files. . .	112
Figure 6.11.	Face Recognition Task Analysis Using Large Sized Image Files. . .	113

LIST OF TABLES

Table 2.1.	Fog - Cloud Comparison.	10
Table 2.2.	Home Automation Frameworks.	20
Table 2.3.	Overview of Infrastructure Communication Protocols.	23
Table 2.4.	Application Layer Protocols in IoT Domain.	27
Table 4.1.	Comparison of Containerization Software.	46
Table 4.2.	Comparison of Container Orchestration Tools.	49
Table 4.3.	Node-Red Features.	51
Table 4.4.	FOGHA Supported Protocols.	65
Table 5.1.	The Analysis of SenseHAT Sensors.	81
Table 6.1.	Docker Performance Evaluation with Sysbench.	104
Table 6.2.	Network Tests Using Iperf.	105
Table 6.3.	Effects of Distance Between Fog Managers.	105

LIST OF ACRONYMS/ABBREVIATIONS

AMQP	Advanced Message Queuing Protocol
AI	Artificial Intelligence
BLE	Bluetooth Low Energy
CN	Core Network
CoAP	Constrained Application Protocol
DARPA	Defense Advanced Research Projects Agency
DDS	Data Distribution Service
DIY	Do It Yourself
EABS	Event-Aware Backpressure Scheduling Scheme
EPC	Electronic Product Code
FOGHA	Fog Home Automation
FN	Fog Nodes
GPS	Global Positioning System
GPIO	General Purpose Input/Output
GPRS	General Packet Radio Service
GUI	Graphical User Interface
HADP	Home Automation Devices Protocol
HDMI	High Definition Multimedia Interface
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol Address
LBP	Local Binary Patterns
LBPH	Local Binary Patterns Histograms
MQTT	MQ Telemetry Transport
MTC	Machine Type Communications
M2M	Machine to Machine
NPM	Node Package Manager

OS	Operating System
RAM	Random Access Memory
RAN	Radio access network
P2P	Peer to Peer
RPI	Raspberry Pi
ONS	Object-Naming Services
QoS	Quality of Service
SBC	Single Board Computer
SD	Secure Digital
SQL	Structured Query Language
SSH	Secure Shell
TCP	Transmission Control Protocol
UI	User Interface
VM	Virtual Machine
VNC	Virtual Network Computing
WANET	Wireless Ad-Hoc Network
WAP	Wireless Access Point
WLAN	Wireless Local Area Network
WoT	Web of Things
XMPP	Extensible Messaging and Presence Protocol

1. INTRODUCTION

The Internet has grown extremely in the last decades and evolved to a new form which includes a lot of connected smart objects and environments. As a result of this technological revolution, people have started to demand more and more services in order to ease their daily routines and to make their life more comfortable. Additionally, computational capacity of smart devices is improving and equipments used in these objects are becoming cheaper day by day. Because of these three factors, variety of intelligent objects is increasing in compliance with people's demand and imagination.

In the beginning of the last decade, rise of smart-phones revealed new opportunities in the way of a more comfortable human life. However, this change can be considered as an extension to the connected computers' network since the critical change was the mobility of computing devices which leads to many new communication protocols, technological infrastructures, computational mechanisms, and devices. The following splash in the Internet era will evolve from mobile devices to connected real world objects. Things, as commonly pronounced in the Internet of Things (IoT) domain, will be positioned between the real and digital worlds in order to fill the current gap between them. They can mainly be divided into two groups as sensors and actuators in terms of their functionality. The first group collects information from the environment such as temperature of a room, humidity of soil, acceleration of a patient, weight of an item, then transfers the collected information to more complex systems for further processing. The latter group is generally responsible for activating/deactivating some devices or services in the network. Actuators consist of more complex mechanisms than the sensors which enable analysis of the collected data possible at the network edge. As an example, when temperature of a room exceeds a threshold value, a cooling system can be activated automatically by using both sensors and actuators.

The subject of this thesis is bounded with the home automation domain but there are also other IoT domains like transportation, health care, industrial automation, agriculture, and smart vehicles which enhance our living standards. Since the IoT concept

is applicable for various kind of domains, requirements for the smart systems and networks do not resemble the standard Internet architecture. Low-energy consumption, sleepy equipment, secure communication methods, and reliable data transmission channels can be considered as the common objectives for all IoT domains. These objectives naturally reveal new communication protocols, architectures, software frameworks and processing equipment. For example, HTTP over TCP/IP stack is generally used for reliable data transfer in almost all applications and services on the Web. However, usage of HTTP in the IoT can sometimes introduce us to additional drawbacks and danger reliability because of limited capacity, battery or processing power of IoT equipment.

According to the recent research, there will be 34 billion devices connected to the Internet by 2020 [1] which claims four times increase compared to 2015. As a trending domain, IoT has been revealing lots of opportunities for hardware manufacturers, software companies and application developers. Such an extensive variety and quantity of the IoT devices require proper management/monitoring systems. The appliances may not be always connected to the network because of the device related limitations which differ IoT connectivity from the traditional Internet-connected hosts. Additionally, heterogeneity of objects creates a new problem: implementation of device-specific communication mechanisms. These problems require to be solved by a network orchestrator entity which is positioned near to the appliances. Requirements of such an orchestrator device mostly coincide with the management of environment that is responsible for providing reliable communication channels, monitoring IoT devices, handling faults, reporting errors, and controlling real time data transmission.

In this thesis, we designed and implemented a home automation system considering the mentioned requirements. It is designed as a modular architecture which contains different kinds of layers and services in order to satisfy reliability, security, privacy and scalability. By proposing such an architecture, we believe that our work raised a suitable candidate for other IoT domains or environments. We named it as FOGHA which is a mixture of Fog Computing and Home Automation concepts. It also utilizes Cloud computing mechanisms to perform long term data analysis and time-consuming tasks in an efficient way. However, the network delay between home

appliances and the Cloud forced us to bring computational tasks to the edge of the local network if it is possible. Additionally, volume and variety of the data generated by smart devices are becoming unprecedented which requires time-sensitive analysis and computation at the network edge. Thus, computational tasks are tried to be handled close to where the related data is generated in our implementation. Contrary to Fog computing, Cloud part of our architecture is mainly responsible for receiving and analyzing a portion of the generated data to be capable of making historical analysis and long-term storage. Additionally, it is responsible for the stability of applications and services which run at the network edge.

1.1. History of Home Automation

In order to consider the future of a domain, the history of it must be analyzed and interpreted firstly. In this section, we will show significant steps in the way of modern home automation systems.

Starting from the beginning of 20th century, electricity has changed the model of living and many new inventions has started to emerge. In 1920s, electrical home appliances were created to ease human life but such devices were only available for wealthy people. The Great Depression of 1929 changed the way of thinking and production mechanisms. Through 30's, new designs and models were presented to attendants via The World's Fairs and new ideas, prototypes, devices had begun to raise in order to facilitate daily routines.

Jack Kilby and Robert Noyce invented the first computer chip in 1950, thus the first building block of today's smart home technology entered into our lives. One year later, the world's first commercially available computer UNIVAC-I paved the way for today's smart controllers. In 1964, the first computer with digital display feature was produced and this progress formed another significant development infrastructure for the idea of home automation which was flirted with in the 1930's World Fairs.

In 1966, Jim Sutherland developed the first home automation system “ECHO IV” which was able to make a shopping list, control temperature, and activate/deactivate devices. The first implementation of the smart home idea was never commercially sold because it was so big for the home environment. DARPA introduced ARPANET in 1969 which is the world’s first network and it led to the communication of today’s smart devices. Second important step in the network domain is the entrance of wireless communication technologies which starts with the invention of “IEEE 802.11 Wi-Fi” family of devices in 1981.

In 1984, the term “Smart Home” was first coined by the American Association of Home Builders. A home equipped with the ability of following a schedule in an automated manner is accepted as a “Smart Home”. Pioneers of today’s smart homes were marketed through 90’s but people had a tendency to demand more intelligent services to improve their life qualities. The movie -Smart House- introduced the idea of home automation to the masses in 1999 and starting from the millennium, the attention towards smart homes started to increase exponentially.

Starting from 2000’s, electronic devices have been progressively becoming more affordable thus science fiction ideas of the past are becoming today’s realities. Currently, we are controlling our smart appliances, lights, doors via smart-phones or controllers and every day we are introduced with a new smart device. In the near future, home automation domain is expected to be borrowing ideas from other computer science fields (e.g. Artificial Intelligence, Deep Learning, Big Data Analysis, Virtual Reality).

1.2. Future of Smart Homes

Most people tend to demand a better life quality and more automated services to ease their daily routines. In the modern world, apart from the working environment, people mostly spend their times at homes. Therefore, converting traditional houses to smart homes and providing comfort services are becoming necessities.

Brush *et al.* conducted a study [2] by visiting families who have whether Do-It-Yourself (DIY) and commercial smart home systems and observed two levels of automation in use: user controlled and rule-based. Turning lights on automatically after sunset can be considered as a rule based automation scenario. Another example would be locking a door remotely through the Internet which is included in the user controlled home automation category. Of course, future smart homes should satisfy both of these functionalities by combining them with a manageable user interface. They should be aware of environmental conditions to control devices and make decisions based on the current context. However, the automation logic should not be limited with those kind of scenarios. For example, analyzing psychological conditions of an inhabitant in real-time is currently a more attractive smart home service. Making automated decisions like opening a short movie or music based on the mood of an inhabitant will definitely attract people much.

Additionally, it is claimed that increase in the average age of human population causes a rise in aging related illnesses which require more skilled health-care services in terms of quantity and variety [3]. When we changed the profile of smart home inhabitants from general to elderly or handicapped people, the needs mostly shift from the entertainment or comfort services to the health related ones. This reality is the reason of increasing number of influential health-care applications which mainly target to ease the life of elderly and handicapped people. Most of them analyze behavior patterns of inhabitants via the data coming from different kind of sensors and inform related people or organizations about the abnormal events. Detection of an unusual behavior is a time critical task which requires real-time analysis of big data and a reliable, fast communication channel between sensory devices and processing nodes. In addition, storing sensor data for long term analysis is another important requirement for the health-care applications for tracking illnesses.

Another attractive area related to home automation is the ability to prevent disastrous events which are caused by humans or the nature. Prevention of a fire before it happens or extinguishing it immediately by activating/deactivating some appliances would be the main criterion while selecting a home automation system since the dam-

ages would be irreparable. In cases like an earthquake, prevention or minimization of the damages may not be possible but the system can save a life by immediately transmitting related information to the rescue teams. In such cases, the Internet may be unreachable that also increases the importance of data processing and storage at the network edge.

Contrary to human population growth, natural resources are diminishing day by day. Thus, governments and organizations are preparing new ecological precautions for efficient use of resources. In addition, people are more conscious about the possible pitfalls because they started to face with the ecological problems directly. In the domain of home automation, energy efficiency of smart appliances deserves a great importance for these reasons. However, only producing energy saving appliances are not enough in the modern world. People expect automatic energy saving mechanisms which are required to be integrated into the home environment as a whole. For example, a thermostat can automatically be capable of regulating the heat level but it should also be able to adjust its working hours based on the presence of inhabitants.

1.3. Objective of the Thesis

In this thesis, we designed and prototyped a home automation management architecture named as FOGHA. One of the main objectives of this research is to provide a flexible IoT enabled automation system. Additionally, we believe that it can be considered as a plug&play tool for the further research or projects in the domain of computer networks. While designing the FOGHA, we considered seven significant criteria:

- “Management of Smart Appliances”: Orchestration mechanism of the architecture needs to be designed by considering a large number of nodes and huge amount of data to satisfy scalability. In addition, the management interface is required to monitor data transmissions between nodes and take automatic decisions in faulty cases to provide system reliability.
- “Real Time Communication”: Events could be categorized under two main domains: system related and sensed data events. The formers are used for monitor-

ing of a home automation architecture which consists of more than one processing node. The latter events are obtained from sensors and are utilized for the long term analysis of the environment or triggering some services/applications. Both of them require secure and reliable real-time data transmission which is able to send critical alerts to users or other services.

- “Layered Framework Architecture”: The system should be divided into different layers in order to shorten the development process and provide module-based services. It is especially critical when a change in the architecture logic is required because any minor update could cause huge changes in other parts.
- “Data Priority”: Successful transmission of some requests have more critical than others. System level failure alerts or disaster related events are included in such kind of requests and requires a higher priority over ordinary events.
- “Low System Resource Usage”: Smart devices which are able to collect data from sensors and activate other smart appliances mostly have limited processing capabilities and memory. Therefore, the home automation services and software should be efficient in terms of system resource usage.
- “Flexible Programming Environment”: Home automation architectures should present extendable programming infrastructures. This shortens the development processes and enables easy deployments for various kinds of automation scenarios. Since smart environments consist of many heterogeneous devices, the system should be able to integrate required software packages in order to create communication channels for each device type.
- “Graphical User Interface”: Since most of the users would have lack of programming experience, the system should provide an easy-to-use graphical user interface for end users. Access to the system from Web-browsers or smart phones is another requirement for the graphical user interface.

2. FUNDAMENTALS OF HOME AUTOMATION SYSTEMS

Current technological improvements in mobile devices, machine to machine (M2M) communication and wireless sensor networks could be seen as the initial phase of IoT. In the recent past, we have started to control different kinds of appliances with the help of Cloud enabled technologies. While huge diversity of smart objects, communication protocols, frameworks and infrastructure elements create new opportunities for the automation of different types of tasks, they also reveal new challenges for the IoT systems. As a trending concept, IoT is expected to adopt various technologies by connecting smart objects with supporting intelligent decision making mechanisms. This distinguishes IoT from traditional wireless sensor networks and M2M communications by imposing new design and implementation challenges.

In this work, we mainly focused on the home automation domain by bringing computing elements to the edge of network where the data is created. Since there is supposed to be lots of heterogeneous devices in a smart home environment, management of these appliances is definitely a significant research area. In the following parts of this thesis, available management solutions will be presented by detailing both software and hardware parts. Additionally, we will justify why FOGHA is designed as a distributed architecture by inspecting other similar approaches in the IoT domain.

2.1. Division of Tasks and Services

Fog computing is a popular area which utilizes the capabilities of devices located at the edge of the network to compute, store, and exchange the data [4]. Compared to Fog computing which generally contains multiple physically distributed nodes, centralized remote servers are responsible for storing and processing data in Cloud computing. Although they have the ability of providing similar services, the distribution of the tasks should be handled properly in order to create a robust home automation system.

Cloud computing has been very popular and Cloud technologies have been highly integrated into our lives. Ability to perform resource demanding tasks within a small amount of time has brought lots of opportunities as well as difficulties. Nowadays, Cloud computing is facing with a growing number of challenges especially in the IoT area since the requirements of IoT systems do not match up with today's Web based architectures. Unlike the more mature technology fields which have solid design and implementation structures, further research is needed for new architectural principles for the emerging systems such as IoT, cyber-physical systems, and embedded AI [5].

Using only Fog or Cloud computing for an automation system is not a coherent approach because each of them better suits for different kind of computational tasks. Cloud and Fog computing are complementary, particularly for data management and analytics [6]. Therefore, our home automation architecture is designed in a modular way by defining responsibilities of Fog and Cloud elements. Architecture design is mostly bounded to functionality allocation decisions: who does what, how to link them, and where to do. Such challenges in the IoT domain require a new architecture which differs from today's Cloud/host computing models. In Table 2.1, main characteristics of Cloud and Fog computing are compared. We will detail them in the following sections in order to justify why we created a Cloud and Fog enabled distributed home automation system.

2.2. Home Automation Requirements

In the following subsections, we will present requirements for home automation systems. It is difficult to construct a globally accepted requirement list but the most commonly pronounced ones are selected to be shown after a detailed literature review.

2.2.1. System Level Requirements

Internet of Things and home automation concepts are gaining more attention which triggers new technologies, software, communication protocols and embedded devices. Because of this diversity, home automation systems should properly manage,

Table 2.1. Fog - Cloud Comparison.

Property	Fog Computing	Cloud Computing
Latency	Very Low	High
Location of Services	At the edge of network	Physically far
Distance Between Server/Host	One hop (generally)	Multiple hops
Environmental Information	Available	Not available
Number of Servers	Many	Few
Distribution of Servers	Geo-distributed	Centralized
Mobility Support	Available	Limited
Real Time Interactions	Available	Limited
Computational Power	Limited	Very High

monitor and control smart environments with the help of emerging technologies. In the design phase of these systems, a layered structure which enables high degree of modularity is required. Although each IoT system has different necessities, there are many common concepts which should be taken into consideration like reliability, scalability, security, privacy and orchestration. We believe that home automation systems should provide five basic services:

- “System Management at Edge Network”: Most IoT appliances can send/receive data via Cloud however they should also be manageable locally.
- “System Management via the Internet”: Home automation systems require an orchestration structure which can also be accessible via Internet.
- “Continuous System Monitoring”: Since reliability is a key concern for home automation architectures, real-time monitoring of system nodes and smart appliances is required.
- “Instantaneous Event Delivery”: The system should transmit events to the related services as quickly as possible. This is particularly critical for disaster cases.
- “Long Term Data Storage”: In addition to real-time communication, the data which is obtained from smart devices should be stored for long term usage.

2.2.2. Smart Devices' Requirements

Smart appliances can be divided into two groups: actuators and sensors. The first group can activate/deactivate other devices in the home environment while operating automation rules. On the other hand, sensory devices periodically collect environmental information and transmit data to the actuators. Following requirements are both valid for sensory equipment and actuators except the last one which only suits for the latter.

- “Energy Efficiency”: It is expected that new generation smart homes will contain dozens of smart appliances. Because of that, adopting energy efficient devices and communication protocols have critical importance.
- “Ease of Configuration”: Ideally, all devices should have plug and play functionality however most of them require an initial configuration. At least, the setup phase of devices should be handled with minimal effort. In addition, they should have standardized functions, APIs and communication interfaces which enable them to connect smart environments easily.
- “Reliable Communication Channel”: Reliable data transmission is required for both sensors and actuators to satisfy the reliability of the whole system. In addition, wireless communication with multi-hop support is a key requirement for the devices for most of the home automation scenarios. However, multi-hop support may not be required for specific scenarios or sensory equipment.
- “Device to Device Interaction”: The devices which are included in the control logic of the system enables D2D interaction via standardized functions. Since there are enormous number of smart appliances, controller devices should be programmable for writing device specific interfaces.

2.2.3. Management Software Requirements

Number of smart devices is gradually growing and this trend brings an increasing heterogeneity to the environments which are covered with these appliances. To solve this issue, management of the devices should be properly provided which leads to new

management software. In the smart-home area, they are named as home automation frameworks which ease communication with the appliances by providing a code base to the developers. Whether a framework is used or not for the management of environments, there are some common software requirements based on the characteristics of a smart-home:

- “Flexible Programming Environment”: Home automation frameworks should present extendable programming resources for developers. This shortens the development processes and enables easy deployments for various kinds of automation scenarios. Since smart environments consist of many heterogeneous devices, the framework should present plug-in architectures which provide communication channels for each device type. In addition, usability of already available plug-ins for new plug-in development is another key criterion.
- “Real Time Communication”: Events could be categorized under two main domains: system related and sensed data events. The former are used for monitoring of the home automation architecture and mostly responsible for system reliability. The latter events are obtained from sensor devices and may be used for the long term analysis of the environment. Both require secure and reliable real-time data transmission. System stability is observed via control events in order to detect failures and taking appropriate actions. Additionally, the system should send critical alerts to users or other services according to sensed data in a disaster case.
- “Graphical User Interface”: Since most of the users lack programming experience, the framework should provide an easy-to-use graphical user interface for end users. Access to the system from Web-browsers or smart phones could be another requirement for the GUI part.
- “Layered Framework Architecture”: Architecture should be divided into different layers in order to shorten the development process and provide module-based services. It is especially critical when a change in the architecture logic is required because any minor update could cause huge changes in other parts.

- “Data Priority”: Successful transmission of some requests are more critical than others. System level failure alerts or disaster related events are included in such kind of requests.
- “Low System Resource Usage”: Smart devices like Raspberry Pi, Beaglebone Black have limited processing capabilities and memory. Therefore, the home automation framework which runs on them should be efficient in terms of system resource usage.

2.2.4. Latency Requirements

Recently, the number of IoT devices and sensors have gradually been growing. This trend creates novel computational needs and approaches as Fog computing for latency-critical applications which contains many geographically distributed devices. Cloud computing technologies and services have attracted attention to create various types of applications without considering low level details such as where the computation or storage takes place, how the data is transferred inside the network, and how the security is provided. Hiding these types of details from the developers eases and fastens the implementation process of the provided services in Cloud computing. Contrary to the Cloud, Fog is a more suitable and trending approach for the IoT applications because a large majority of these devices contain time-critical processes, so very low latency is a significant requirement which is not satisfied with Cloud computing. For instance, vehicle-to-vehicle communication which is the bricks of today’s autonomous cars requires a few millisecond latency. In the home automation domain, some services are more delay tolerant than the others. For instance, turning on/off a light switch within a hundred milliseconds delay may be satisfactory for the inhabitants. Even if such a relatively high latency is acceptable for the users, controlling physically near switches via Cloud is definitely an inefficient approach and may danger the reliability of the system because of the high latency or timeouts which can be caused from the propagation delay between the host and the server, instantaneous traffic in the network, link failures, and network congestion. In addition, direct communication between Cloud and sensors/devices may not be possible since some appliances do not support

high level protocols which are required for the Cloud based technologies.

In recent years, augmented reality (AR) applications have become widespread all over the world which is triggered after the rise of smart-phones, tablets, and wearables (e.g. smart glasses, smart-bands). Thus, designing a smart-home without including AR applications and services would be considered as old-fashioned. They mostly require high processing power to process video frames, voice data, and sensory inputs synchronously and they should additionally produce some outputs after a fraction of a second. Since people are highly sensitive towards latency, a processing delay of more than tens of milliseconds can lead to a poor user experience [7]. In time-critical processes, usage of nearby computational nodes decreases both latency in data transmission and processing whilst increasing efficiency. On the other hand, Cloud computing is more beneficial for the long term data analysis and storage because of the resource limitations of Fog computing nodes.

Fog enables data analytics and time-sensitive control functions at the network edge which is essential for not only commercial applications but also for the Tactile Internet [8] vision to enable embedded AI applications with millisecond reaction times [5]. For example, real-time wireless virtual reality gaming which can be an attractive service for a smart home environment requires 10 ms latency. This necessity cannot be achieved via Cloud computing or globally accepted wireless communication technologies like Long-Term Evolution (LTE) which achieves a typical round-trip latency of 25ms. As a result of these challenges, the Fog computing paradigm which brings computation and control processes from remote centralized nodes to the physically near computing elements by decreasing the network latency to the minimum is suitable for home automation scenarios.

2.3. Home Automation System Components

The Internet evolution started with the aim of connecting two computers. Afterwards, it received a new form by connecting a significant number of computation nodes. Starting from this stage, Internet has reached to the masses and became the

most significant tool in the way of a more comfortable human life. Later, people have connected to the Internet via mobile devices and their identities have been carried to the social networks. Nowadays, everyday objects have been gaining computational power and we started to call them as smart objects. The last evolution was globally accepted as the era of IoT.

IoT applications can be grouped under several domains such as agriculture, transportation, health-care, industry, and home automation. In this thesis, we selected home automation as the target domain and tried to apply an innovative approach while utilizing novel appliances and technologies. Smart homes increase the quality of life by utilizing context aware services, automated appliance control mechanisms and predefined rules based on the home conditions [9]. Although home automation is perceived as controlling home appliances, it is not only limited to control mechanisms. A structured management system is required for modern smart homes which is able to bring down the cost, ease daily routines, suggest personalized actions, monitor the environment instantaneously besides automation and control mechanisms [10].

Even if new smart appliances, technologies, protocols and software packages arise almost every day, the skeleton of a modern smart home is constructed on top of some corner stones. In the following sub-sections, main building blocks of a modern smart home is explained in detail.

2.3.1. Single Board Computers

In the previous sections, we discussed why the division of tasks between Fog and Cloud is important and explained why they complement each other. Instead of owning and managing private data centers, Cloud computing which provides “pay as you go” pricing models emerged as a more efficient alternative for creating various kinds of applications and services [6]. Additionally, capabilities of processing equipment and storage devices have rapidly grown which makes Cloud services more affordable.

Nowadays, intention of academia and industry has shifted from mobility of devices and Cloud computing to IoT and Industry 4.0 concepts. In the first half of this decade, home automation architectures [11], [12], [13], [14] which are solely dependent on Cloud computing was proposed by researchers. In order to perform computational tasks at the Fog Layer, single board computers (SBC) are used as computing nodes in every IoT domain. For a distributed home automation architecture, we should consider some significant criteria such as size of the device, price, programmability and processing capabilities. In order to find the optimal SBC, we tried to find the one which provides the most processing power with the minimal cost and the maximum programming flexibility. We did not take under review the boards which costs more than \$50 because they are mainly produced for industrial operations and our distributed home automation architecture should be affordable. Since new SBCs which mainly target the IoT domain are continuously becoming commercial nowadays, we also considered popularity in academic studies, community support, and user reviews of them while limiting the following list.

2.3.1.1. Arduino. Arduino [15] is an open-source electronics platform which provides a set of hardware and software tools to the users. Arduino boards are capable of receiving inputs from various sensors and affecting their surrounding by controlling other actuators or appliances. In order to program the board, Arduino programming language and a specific Integrated Development Environment (IDE) are used by sending a set of instructions to the micro-controller which is placed on it. There are several Arduino Board models and some of them are listed under the IoT category by the Arduino community. However, none of them is capable of running a modern operating system and presenting a graphical user interface. Even the most powerful Arduino models (e.g. Arduino TIAN) do not exceed 560 MHz microprocessor clock speed and 64MB DDR2 RAM. Time-consuming home automation scenarios like detecting someone who enters the home using image processing or monitoring the overall system reliability definitely require more processing capabilities. We believe that Arduino boards are more suitable for reading bit-based or analog sensor data rather than operating as a management node in a home environment.

2.3.1.2. Beaglebone Black. Beagles [16] are small, cost-effective boards and provide almost every capability of modern computers. The most powerful Beagle model is BeagleBoard-X15 which has TI AM5728 1.5 GHz dual core ARM processor and 2 GB DDR3 memory. However, its price currently around \$250 and this would make a home automation system which uses the computation nodes in a distributed manner less affordable. Because of that reason, the most popular model of Beagles is Beaglebone Black which has AM335x 1 GHz ARM Cortex-A8 processor, 512 MB DDR3 memory and 4GB 8-bit eMMC on-board flash storage. It costs around \$50. Therefore, it is widely used in academic and industrial purposes instead of less cost-efficient models as BeagleBoard-X15. Since it has not an SD card slot, you are limited to use the 4GB internal storage which limits the usage of most operating systems. The main difference between it and Arduino is that it supports modern, GUI supported operating systems as Linux or Android, therefore it can be called as a mini-computer which is able to run various programs on these operating systems. Additionally, this support annihilates the dependency of writing IoT appliances with a single programming language.

2.3.1.3. Intel Edison. Intel Edison Compute Module [17] is another mini-computer which has built-in Wi-Fi , Bluetooth Low Energy (BLE) modules and an SD card slot. It has dual-core 500 MHz Intel Atom processor and 1 GB DDR3 RAM and 4GB eMMC on-board flash storage. It is also around \$50 and it provides a less powerful processor but more memory than Beaglebone Black. Therefore, performance evaluation of them changes according to their usage areas and scenarios. Galileo is another Intel board which has less processing power and built-in support than Edison, so we selected the latter as a candidate control node for our system design. Intel firmware is the only available operating system option and supported programming languages are limited to Arduino, Java Script, Node.js, C/C++, Java, and Python. In addition, it does not provide any visual experience, mouse or keyboard support. Therefore, programmers are dependent on another computer for writing code on it. Because of such limitations, we can claim that it is a better choice for embedded systems, and not for distributed IoT architectures.

2.3.1.4. Udoo NEO. It is an Arduino-powered Android / Linux single board computer equipped with motion sensors, Bluetooth 4.0 and a Wi-Fi module [18]. It has a 1GHz dual-core FreescaleTM i.MX 6SoloX applications processor and 512 MB (in Basic Udoo Neo) - 1 GB (in Udoo Neo Extended) RAM. Basic version of it is around \$50 and also lacks of built-in Wi-Fi and BLE 4.0 modules as opposed to extended version which costs \$10 more. It also has a micro-SD card slot on board which enables booting various operating systems from a mini card. In addition, it can reflect the operating system GUI on a monitor via HDMI output by easing programming on it. Modern SBCs commonly use voltage range 3.3-5V, it requires a 12V power supply. This board mainly targets to become a mixture of Arduino and Raspberry Pi boards by taking only desired features of them. Using it in the IoT projects could be a good selection except its relatively high price.

2.3.1.5. Raspberry Pi 3. Raspberry Pi [19] is another small computer with just enough processing power while still preserving inexpensiveness. Since hobbyists, researchers, educators, and inventors are the target audience, it has enough processing power to perform common tasks such as playing 1080p videos, Web browsing, and playing games. The latest Raspberry Pi 3 has 1.2Ghz 64-bit quad-core ARMv8 Cortex A53 CPU and 1GB RAM of 900Mhz, 4 USB ports, 1 HDMI port, 1 audio I/O port and 1 Ethernet port. It does not have any built-in storage equipment but micro SD card slot is available for this purpose. It supports various operating systems such as Windows 10 IoT core, Ubuntu Mate, Risc OS, Raspbian. It means that there is no dependency to any programming language, software IDE, etc. so you can run any program which is supported by the operating system. It also has built-in Wi-Fi and BLE modules like Udoo NEO but Raspberry Pi 3 provides these features with a low-cost price, \$35 which can be considered as a critical decision factor for distributed systems. Raspberry Pi 3 is a globally popular device which has a very active community and the Internet is full of Do It Yourself (DIY) projects developed using it. Lastly, it is very popular among researchers therefore we are able to compare our architecture with other Raspberry Pi enabled systems. Mainly for these reasons, in this thesis, we selected Raspberry Pi 3's as our computing nodes which is located at the network edge.

2.3.2. Home Automation Frameworks

Smart environments contain many heterogeneous smart appliances and the communication between them requires various technologies, and protocols. Therefore, proper management of the environment is a significant necessity but writing an IoT management software from scratch is an inefficient and time consuming solution since it requires the support of different kinds of smart appliances, technologies, protocols. Instead, there are some open-source IoT management frameworks which can be used as a code skeleton and customized according to the specific needs.

In this thesis, we tried to find out the most suitable home automation frameworks by comparing different features of them. First of all, it should support popular single board computers such as Raspberry Pi (RPI), Beaglebone Black and use processing equipment efficiently because SBCs have very limited computational power compared to modern Cloud servers. Scalability, ease of programming, and the number of available software packages to interact with smart appliances are the other important criteria. We selected and compared nine popular, updated IoT frameworks which have support and special software for the popular SBCs in order to perform management processes among the most popular 21 tools [20]. The general properties of them are presented in Table 2.2.

Plug&Play level changes from 1 to 5 such that the higher value represents the easier usage and development. Please note that it reflects our personal opinions based on the difficulties which were experienced in setup and development processes.

2.3.3. Infrastructure Protocols

In this section, we will briefly describe the most commonly used infrastructure protocols in the IoT domain. Since the home environment is highly heterogeneous due to the variety of devices, we cannot create a network topology which only supports one of them. Instead, our system is designed considering flexibility and ability to support different technologies.

Table 2.2. Home Automation Frameworks.

Framework	Written In	Application Languages	User Interface	Plug &Play Level
Machine.io	C++	Javascript C++	Web	1
Kura	Java	Java	Web	1
Kaa	Java	Java, C, C++, Objective C	Web, iOS, Android	2
Home Assistant	Python	YAML Syntax	Web, iOS, Android	3
Node-Red	Javascript	Javascript	Web	5
OpenHAB	Java	Framework Specific	Web, iOS, Android	3
IoTivity	C	C, C++ Java	-	1
PlatformIO	Python	C, C++	-	3
Pimatic	Javascript	Framework Specific	Web	3

2.3.3.1. Z-Wave. Z-Wave is a low-power wireless communication protocol which is widely used for the automation scenarios in residential surroundings [21]. It cannot be expected to transfer large amount of information or streaming data because the main aim of the protocol is to transport small data packets from the source to one or more nodes in the network reliably [22]. It generally operates in the 900 MHz ISM bands (e.g., 868.42 MHz in Europe, at 908.42 MHz in the US and Canada) but this can change based on the regulations of countries. Therefore, signal interference with Wi-Fi, Bluetooth or other systems that operate on the crowded 2.4 GHz band is avoided. The throughput is around 40 Kb/s and communication range between two nodes is about 30 meters. However, the recent 500 Z-Wave 500 series chips are capable of providing a few hundreds Kb/s data rate and nearly 40 meter communication distance.

Both ZigBee [23] and Z-Wave provide short range wireless communication with low cost and energy consumption. ZigBee operates in the frequency band of 2.4 GHz, while it is less than 1 GHz in Z-wave. Although the Z-Wave network can support 232 slave nodes, the limit for ZigBee is 65000 slave nodes. Lastly, Z-wave is simpler in implementation than ZigBee [24].

2.3.3.2. Bluetooth Low Energy. Bluetooth Low Energy (BLE) is another short range wireless communication protocol and has a coverage about 100 meters which is ten times bigger than the classic Bluetooth. The main advantage over classic Bluetooth is power efficiency that enables BLE devices to operate for a long time (even for years) with a small battery. These advantages makes it a very suitable choice for IoT applications. Additionally, BLE is more efficient than ZigBee in terms of battery consumption and the energy usage for per transmitted bit [25].

Smart phone vendors integrate BLE equipment to their devices because BLE provides energy efficient communication channels. Other than smart phones, lots of new IoT appliances and sensors support BLE. Therefore, home automation systems should be capable of supporting BLE devices.

2.3.3.3. LTE-A (Long Term Evolution - Advanced). Architecturally, LTE-A network consist of two main layers. Core Network (CN) controls mobile devices and handles IP packet flows while Radio Access Network (RAN) is responsible for wireless communication and establishing user and control plane protocols [26]. RAN contains base stations (evolved NodeBs) and the X2 interface between them. Mobile devices can connect directly to them or through the MTC gateway.

IoT friendly LTE-A chipsets lead to new device deployments by providing flexible, efficient, affordable solutions in IoT applications. It enables a communication bandwidth of 1 Gbps up-link and 500 Mbps down-link which differs it from other IoT infrastructure protocols. In addition, achieved communication distance between two nodes is greater than popular protocols like ZigBee or BLE.

2.3.3.4. RFID. EPCglobal is the organization which is responsible for managing and developing RFID and Electronic Product Code (EPC) standards. EPC is a unique identification number which is stored on an RFID tag. The RFID system consists of two basic components: the tag and tag reader. The first contains a chip to keep the unique identity of the item and an antenna to perform wireless communication between the tag and tag reader. The latter receives the tag's identity number using radio waves and passes it to an application which is called as Object-Naming Services (ONS). ONS retrieves tag details from a database and enables further specific procedures. RFID technologies are widely used in many IoT scenarios but it has a very limited usage area in the home automation domain.

2.3.3.5. IEEE 802.15.4. IEEE 802.15.4 protocol was created to satisfy low power consumption, data rate, cost, and high message throughput, therefore it is a suitable candidate for IoT scenarios [26]. It is the base protocol for the ZigBee and both of them target providing low data rate services on power constrained devices.

Based on the selected frequency channels, it can provide 250 Kbps at 2.4 GHz, 40 Kbps at 915 MHz, and 20Kbps at 868 MHz. Higher frequency channels definitely present more bandwidth and lower latency whereas smaller distance coverage and sensitivity. It resembles the ZigBee protocol in many aspects, but it provides a simpler implementation option compared to ZigBee. Since it is not common as the other short range wireless communication protocols like BLE or Z-Wave, it could be cumbersome for home automation scenarios to find smart equipment which supports it.

2.3.3.6. 6LoWPAN. Low-power wireless personal area networks (LoWPAN) are described as a large number of wireless connected low-cost devices [27]. The main advantages are small packet sizes, low throughput, and energy efficiency which well suits for IoT applications. It combines IPv6 and LoWPAN and supports all capabilities of IPv6 [28]. Since IPv4 was not designed for the billions of connected smart things, IPv6 was offered to overcome the critical addressing issue.

Using IPv6, assigning IP addresses and connecting enormous number of smart devices to the Internet has become easier. Because of these reasons, 6LoWPAN is widely used in IoT scenarios while increasing its popularity. Data rates are between 20 to 250 Kbps and the communication distance is between 10 and 30 meters which would be considered as critical limitations for dense IoT scenarios.

The general features of the infrastructure protocols are presented in Table 2.3. Their use cases vary according to the requirements of an IoT based system.

Table 2.3. Overview of Infrastructure Communication Protocols.

Protocols	Freq. Band (MHz)	Data Rate (Kbps)	Range (meter)	Spreading	MAC Access
Z-Wave	898 908 2400	9.6 40 200	30(in) 100(out)	-	CSMA/CA
ZigBee	868 915 2400	20 40 250	10-100	DSSS	CSMA/CA
IEEE 802.15.4	868 915 2400	20 40 250	10	DSSS	TDMA CSMA/CA
6LowPAN	868 915 2400	20 40 250	10-100	DSSS	CSMA/CA
BLE	2400	1024	100	FHSS	TDMA
LTE-A	varies	1 Gbps	varies	Multiple CC	OFDMA
RFID	from 860 to 960	from 5 to 640	5(passive) 100(active)	DS-CDMA	ALOHA

2.3.4. Application Protocols

In this section, we will briefly describe the most commonly used application protocols in the IoT domain. Today, many smart appliances have the ability to support multiple of these protocols which facilitates the integration of new devices to the IoT systems.

2.3.4.1. Message Queue Telemetry Transport (MQTT). MQTT was proposed to connect embedded devices and networks with applications and middleware by supporting one-to-one, one-to-many, and many-to-many relationships [26]. MQTT is a lightweight communication protocol and uses a publish/subscribe pattern in order to satisfy flexibility and simplicity. It is created on top of the TCP protocol by targeting resource constrained devices and unreliable communication links.

It is comprised of three components: broker, subscriber, and publisher. The broker is responsible for transmitting messages which are received from publishers to the subscribers which were previously subscribed to specific topics. Nowadays, MQTT is widely used in IoT applications because it provides a lightweight, simple, QoS supported, and energy efficient communication solution for the resource constrained devices and bandwidth limited networks.

2.3.4.2. Extensible Messaging and Presence Protocol (XMPP). The XMPP is an instant messaging protocol based on XML streaming protocols which provides great scalability, addressing, and security capabilities. Common use cases of it are multi-party chatting, voice and video calling. XMPP is a flexible protocol which allows applications to achieve encryption, authentication, access control and enables bidirectional messaging with other networks [26].

XMPP architecture contains three different components: gateway, client and server. The gateway is responsible for supporting reliable communication between heterogeneous systems while the server manages communication links and message

routing mechanisms. The client connects to the server and sends messages based on the XML streaming protocol. XMPP is a good candidate for peer-to-peer communication purposes in the home automation domain.

2.3.4.3. Constrained Application Protocol (CoAP). CoAP is an application layer protocol based on the REST (Representational State Transfer) architecture and provides modified HTTP functions to enable resource limited devices to perform RESTful interactions. REST is used to transfer data between clients and servers by providing cache mechanisms and very popular among mobile and Web applications. Unlike REST, CoAP is built on top of UDP (not TCP) which makes it more appropriate for IoT applications and resource limited appliances [24].

CoAP uses a request/response messaging model which supports group communication and push notification but not broadcasting. It can be used with any message format such as XML, JSON, CBOR. In addition, it is able to work on micro-controllers with as low as 10 KB of RAM and 100 KB of code space [29] which validates its popularity in IoT applications.

2.3.4.4. Advanced Message Queuing Protocol (AMQP). AMQP is an application layer protocol which targets message oriented environments by providing message transmission services like store and forward, publish and subscribe, message distribution, message queuing, context-based routing. AMQP relies on tracking all messages and ensuring each is delivered properly, regardless of failures or reboots [30]. Therefore, a reliable transport protocol like TCP is used to exchange messages. In the IoT domain, it is a very suitable choice for the control plane and system monitoring.

2.3.4.5. Data Distribution Service (DDS). DDS is another publish/subscribe protocol which is used for real time M2M communications. Compared to other publish/subscribe application protocols like AMQP and MQTT, it does not require a broker for data transmission purposes. DDS supports 23 QoS policies by enabling the selection of many communication criteria such as reliability, security, urgency, priority.

In DDS, publishers can create topics (e.g. temperature and humidity) and publish messages to those topics. Subscribers of a specific topic receive messages of that topic. DDS nodes use peer-to-peer communication architecture by enabling UDP multicast without the need for a central router node. This also decreases the communication delay below milliseconds which perfectly fits for the M2M communication.

2.3.4.6. Websocket. The Websocket protocol targets to ease communication channels over the TCP and it is neither a request/response nor a publish/subscribe protocol [31]. Websocket was built over TCP protocol to decrease the communication overhead while enabling real-time full-duplex communications. It does not have additional reliability mechanisms other than the ones provided by the reliable TCP protocol.

Websocket is less suitable for resource limited devices because of its server/client based architecture. However, it enables reliable and secure real-time communication by minimizing the packet overhead. Therefore, utilizing it for providing reliable communication between more powerful nodes in an IoT architecture would be beneficial.

The general features of application layer protocols are presented in Table 2.4. There are also other protocols for the application layer but we listed the most common ones in the IoT domain.

Table 2.4. Application Layer Protocols in IoT Domain.

Protocols	Transport Layer	Architecture	Security	QoS Support
MQTT	TCP	Publish/Subscribe	SSL	Available
XMPP	TCP	Publish/Subscribe Request/Response	SSL	-
CoAP	UDP	Request/Response	DTLS	Available
AMQP	TCP	Publish/Subscribe	SSL	Available
DDS	TCP	Publish/Subscribe	SSL DTLS	-
Websocket	TCP	Publish/Subscribe Client/Server	SSL	-

3. LITERATURE SURVEY ON HOME AUTOMATION ARCHITECTURES

In this thesis, we applied Fog computing strategies, technologies, and functionalities to the home automation domain since related scenarios are becoming more complex and resource demanding day by day. Thus, a modular and well defined architecture appears to be a critical requirement in order to be able to perform these kind of tasks at smart environments. Such kind of architectures are mostly pronounced with the Fog computing concept of the IoT domain which also contains a wide range of research areas. This chapter is divided into two sections as home automation and Fog computing and the prominent implementations or proposed architectures will be presented.

3.1. Home Automation Domain

Functional requirements of an ideal smart home could be categorized into four: automation, intelligence, dynamicity, and ease of configuration [32]. The system is required to be able to interact with smart objects and perform required operations according to contextual data automatically. In addition, the devices which have the capability of performing computational tasks should be able to make intelligent decisions based on the context or historical data. Thirdly, most of the smart objects have mobility therefore the system should adapt itself to the new context. Lastly, the configuration or maintenance of the home automation system should be easy because most of the users will not be able to understand technical details.

In order to categorize a smart home under the IoT domain, the objects are needed to be controllable through the Web. For this purpose, there are two commonly accepted solutions: direct Web connectivity on smart devices or through a bridge node (commonly called as gateway or proxy). As with the development of Cloud technologies and Wi-Fi enabled appliances, Internet based remote control implementations [33, 34] and architectures [35, 36] for home automation have been proposed. In 2010, it was

claimed [37] that most embedded platforms will have native support for the TCP/IP connectivity in the near future. However, it has not become a reality until now and does not seem so in the near future because of the resource limitations and priorities of the devices. Instead, new architectures like Fog computing and rise of single board computers as forwarding elements have started to annihilate this need.

For the purpose of controlling the home appliances remotely, Jamil *et al.* [38] proposed a pilot study which uses a RPI board and Wi-Fi protocol for control purposes but it is far from satisfying real home environment requirements. In another study [39], Bluetooth and Wi-Fi protocols are adapted to perform simple scenarios in the home environment by targeting elderly and handicapped people with the lack of a comprehensive architecture. By targeting solely aged people, another research [40] was conducted in order to track daily activities of the people via an Android application and it also does not consider the necessities of a smart home. Home automation systems which aim to automate homogeneous ZigBee-based networks are proposed in [41, 42].

A further study [13] aimed to utilize Arduino hardware as the control unit and ZigBee protocol as a solution for networking smart objects by easing communication between them and enabling Cloud services. A similar architecture was proposed by Piyare *et al.* [43] using Bluetooth technology for home automation using an Arduino board as a wireless control system. In this work, a cellphone application was used to send messages over Bluetooth to the Arduino board which runs a Python script to control home appliances.

In the beginning of this decade, improvements on Cloud technologies attracted people to produce remote solutions. Several architectures [37, 44] were proposed to provide a solution for the highly heterogeneous IoT environments by utilizing the WoT (Web of Things) concept. They claimed that the network delay will not be an issue for most of the scenarios by advocating the simplicity and scalability of RESTful architectures. However, it is started to become a major problem for most of the IoT scenarios after recent interactions between IoT and other computer science domains.

By the nature of IoT applications, a home automation network consist of many heterogeneous devices which support various infrastructure protocols like Bluetooth, ZigBee or Wi-Fi. Therefore, constructing a system which depends only one of them is an infeasible way in today's world. For this reason, single board computers are commonly used as home gateways which mostly have a running server and operating system [45]. ZigBee, Bluetooth, and Wi-Fi are used heterogeneously in the local network while the home gateway is handling connectivity between local network and the Cloud.

RPI is a popular single board computer and it has the highest performance to cost ratio [46] which makes it an excellent candidate for the home gateway node. In [47], a basic home automation application which controls the environment via reading the topics of e-mails is developed using RPI and its supported programming environment is Python. There are also other studies [48–50] which use a RPI board as the home gateway, however they lack of proper management and monitoring mechanisms.

A distributed, inter-operable architecture [51] for the IoT is proposed to handle common problems in this domain. It is a layered architecture which enables the addition of new appliances seamlessly and targets to overcome the issues like scalability, heterogeneity and interoperability.

In another research [52], conducted by Gonnot and Saniie, the interaction between home appliances is the main focus and a 6LoWPAN network architecture is proposed for this purpose. Their first aim is to provide a standardized mechanism for connecting devices based on their categories. The second is to provide a communication channel via a central node. Additionally, Home Automation Devices Protocol (HADP) is proposed as an application layer protocol to decrease the consequences which result from the heterogeneity of the environment.

Korkmaz *et al.* [53] proposed and implemented a Cloud based home automation system which consist of three main components: i.MX53 embedded board to transfer data between the Cloud and Fog environment, a Web server which is responsible for

storage and data processing, and a smart-phone application which is capable of controlling home appliances. Their testbed is capable of performing some kind of tasks which only require small amount of data exchange such as turning on/off the devices located in a house remotely.

Nowadays, single board computers such as RPIs are very popular in the home automation domain. Pavithra and Balakrishnan propose a home automation implementation [54] which is used for controlling and monitoring a house via a RPI. Their system enables remote control of physical devices such as lights, door through a Web browser and data collection from the sensors which are connected to the RPI board. Their system is not capable of handling resource demanding tasks. Because, they preferred to put the entire application logic inside a single RPI board which has an Internet connection instead of adding a Cloud node. In addition, similar systems [55], [56], [57] that have the same limited capabilities are proposed with the aim of controlling the physical devices.

Besides those limited systems, there are also other systems which can be considered as seminal for modern smart homes. One of them that is conducted by Lohokare *et al.* presents an implementation of a scalable wireless home system at city level by using several prominent software tools in embedded boards [58]. Although the tools used in the Cloud servers (mainly consist of MongoDB, Apache Spark, Apache Kafka and MQTT Broker) seem to be impressive the performance evaluation part is missing in this research. Additionally, data processing is handled in remote servers instead of local boards which makes it a Cloud computing solution only.

In the home automation domain, most of the emergent systems are far behind the current technological improvements and user demands. They generally utilize an Internet connected SBC and user interface to turn on/off home appliances. Therefore, they cannot be associated with the Fog computing paradigm since we cannot mention any kinds of computational tasks, excluding switching on/off appliances. Another common approach is publishing data to Cloud for processing purposes via gateways which has critical disadvantages in terms of communication delay and high bandwidth

usage. For these reasons, we will continue with Fog computing architectures which can be beneficial for home automation purposes in the following section.

3.2. Fog Computing Domain

In addition to home automation architectures and implementations, our system is also strictly bounded to the Fog computing paradigm. Although Cloud computing provides an effective way of storing, processing, and analyzing huge amount of data, it cannot be a standalone solution for delay sensitive applications. Additionally, it is mostly pronounced with an ineffective usage of network bandwidth because of the need for continuous data transfer between a physical environment and Cloud servers. The main purpose of Fog computing is to present a novel solution for these problems. In this section, Fog computing architectures and implementations which would be beneficial for home automation domain will be presented.

EABS [59] was proposed to minimize end-to-end delay in the network and prioritize some packets from the others by tagging them as emergent. It mixes shortest path and backpressure-based queue model for the selection of the next hop in order to eliminate network congestion and minimize forwarding of emergency packets. Their work coincide with geographically distributed Fog computing architectures and latency sensitive applications.

Yi *et al.* [60] implemented a proof-of-concept Fog computing platform that consists of local servers and a Cloud server. They installed OpenStack [61] software management platform on each of them with four additional modules. The functionalities of them are as follows: authentication & authorization, VM image management, computation, and block-level storage. In their work, a VM offloading scheme is also presented which enables migration of one VM to another Fog cluster. As a use case, a face recognition application which takes photos via a mobile-phone is developed and Fog-Cloud performance comparison is presented based on the response times.

In another research, Fog and Cloud computing layers are connected via smart gateways which are responsible for collecting sensor data from an environment and communicate with remote servers [62]. These smart gateways manage the Fog environment and process the data before sending it to the Cloud that results with better utilization of Cloud and network resources. They mentioned about the multi-hop communication with these smart gateways which may be considered as a critical requirement in terms of Fog computing scalability. However, there is not any implementation and performance evaluation related to these type of communication.

Li *et al.* [63] proposed a data centered Fog platform namely EHOPES for smart living by providing real-time interaction with the aim of bringing computing power from Cloud servers to the near of users. They claim that the processing capacity of Fog computing minimize the interaction and data transmission with the Cloud which is very beneficial for the entire Internet. Foglets are software which can be easily deployed to the hardware devices in the Fog environment in order to control resources such as VMs and services. Their architecture includes two types of hardware devices: Fog Edge Nodes, Fog Servers. While the former group which has enough processing, communication, and storage capabilities to run Foglets enables communication with smart devices, the latter group has the same capabilities but also it responds to the requests generated from Fog Edge nodes and connects them to the Cloud data centers.

Hong *et al.* [64] proposed a high level programming model for Internet applications which are geospatially distributed, large-scale, and latency sensitive. Their first goal is to create a programming model which eases software development on a great number of heterogeneous and geographically distributed devices. Secondly, they aimed to support dynamically scalable applications based on their current workload while enabling Fog and Cloud interaction. Their proposed framework has a set of default functions which are invoked when a certain type of event occurred. Additionally, it is claimed that their architecture runs the same code on various devices such as smart phones, smart-cameras and server nodes in the Fog and Cloud. Thus, the main contribution is done for the decreased heterogeneity of smart appliances, especially for Fog and Cloud servers.

The Fog computing paradigm is also commonly used in the health care IoT domain because real time responses are critical and huge amount of data is generated in this domain. Health care services are also commonly pronounced with smart homes currently. Zao *et al.* [65] proposed a Fog computing system which can realize users' brain states using wireless EEG headsets by utilizing MQTT publish/subscribe protocol and a multi-tier Fog/Cloud infrastructure. They use Fog servers and mobile phones close to the sensors in order to process data in real time. Resource demanding tasks like machine learning and big data mining is performed in Cloud data centers for a deeper analysis.

Face recognition is a popular research scenario which is very suitable for Fog computing and home automation concepts because it is delay-sensitive and requires real-time processing of frames. Hu *et al.* [66] suggest a Fog computing based face identification framework in order to decrease bandwidth usage and network delay. Their implementation uses local binary patterns (LBP) for identifier generation and Euclidean distance method for matching by offloading some computations from Cloud to the edge of network. In their work, Fog Nodes (FNs) are located at the network edge and have enough processing capabilities for certain tasks like face detection, image preprocessing, and feature extraction. Later processes run on the Cloud nodes. Thus, data size transmitted to the Cloud nodes and response time for the face identification task are remarkably decreased.

4. FOGHA HOME AUTOMATION ARCHITECTURE

In today's world, expectations from a smart home is far beyond only activating or deactivating a switch. Additionally, turning on a coffee machine at a specific time is considered as a trivial home automation scenario nowadays. For such scenarios, we cannot claim considerable amount of data and processing. However, such scenarios are pronounced frequently as smart home ideas by claiming that Cloud technologies can handle such jobs. Communication via Cloud can be beneficial to control a physical environment remotely but it is an error-prone communication type. Also, it is not effective compared to communicating at the edge network when the source which sends an instruction and the receiver are in the same physical range. Latency is another issue for the Cloud computing and deserves higher importance for latency critical applications. Therefore, the tasks should be handled at the network edge as much as possible and Cloud nodes should be implemented as subsidiary elements in order to perform CPU intensive jobs or access the environment remotely. In this chapter, the architecture of our system will be explained. The implementation details will be provided in the next chapter.

4.1. General Overview

In this thesis, we tried to design our system as modular as possible since heterogeneity and scalability concepts have substantial impacts on smart environments. In addition, we aimed our architecture to be applicable for not only small environments but also for larger areas like a skyscraper or a campus. Nowadays, IoT appliances are commonly utilized for single use cases or tasks in industry which makes information exchange between devices nearly impossible. As a result, environments which are claimed to be smart come into existence as disconnected black boxes. Therefore, we aimed to create an architecture for smart areas by supporting as many IoT devices and services as possible. Theoretically, supporting all of them is not possible but creating a system which allows developing communication middleware between smart appliances via widely used programming languages is an important objective of this thesis.

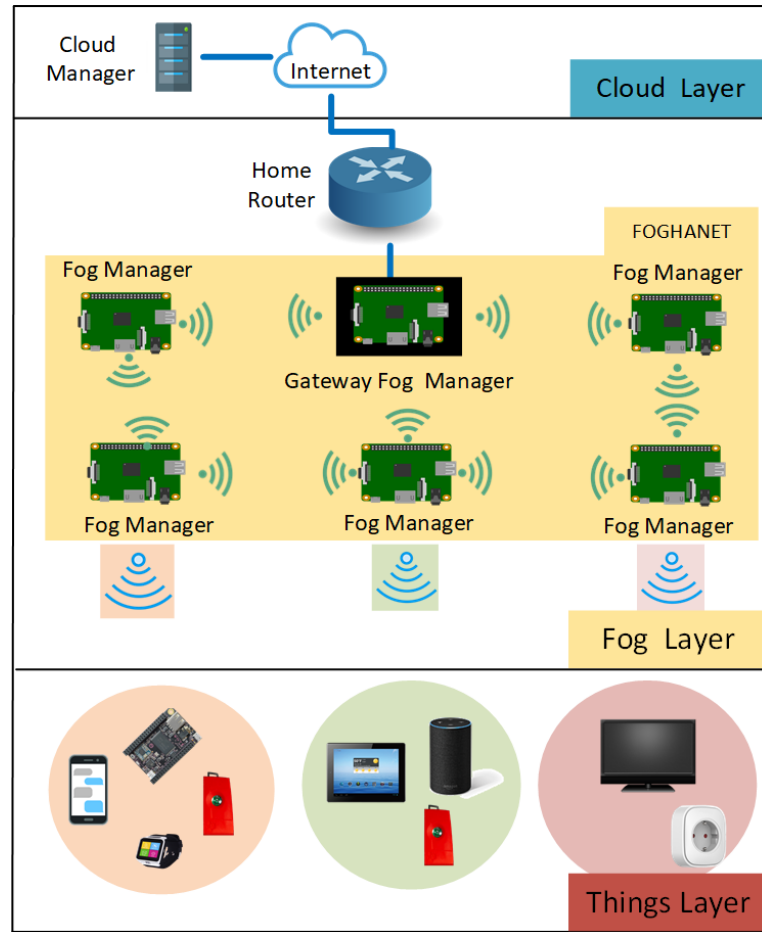


Figure 4.1. Layered System Architecture.

FOGHA architecture mainly consist of three layers as seen in Figure 4.1. Cloud Layer could be located anywhere in the world but it is better to select a place which is close to the smart environment. In that way, communication delay would be shortened. The same selection option is not valid for the Fog and Things Layer since they contain all smart appliances in the physical environment. While the Fog Layer consists of single board computers as processing equipments, the Things Layer includes sensors and other smart devices. Data collection, processing, decision mechanisms and automation rules are all parts of the Fog Layer which makes it the brain of our architecture. Contrary to Fog, the Things Layer is responsible for sensing and applying commands taken from upper layers.

In our architecture, we used multiple Raspberry Pi (RPI) computers and distributed them geographically in the home environment. The main reason behind this choice is to cover and control larger areas by making the system scalable. In this way, we believe that our architecture is applicable not only for smart homes, but also huge physical areas like factories or campuses. For a healthy system, number of RPI computers should be determined based on the physical features like the number of walls, thickness of walls, and the size of the environment. Additionally, they should have reliable communication channels between themselves in order to forward sensed data to the related node in the network. The most reliable communication channel would be using Ethernet cables to connect RPI boards. However, this is an infeasible solution in terms of scalability considerations. Another possibility is connecting them to a router as wireless clients which could also create two major problems. The first one is the need for additional access points because of the limited wireless coverage area of them. The second one is the single point of failure risks caused from the lack of redundant routers.

Because of these drawbacks, we selected to create a Wireless Ad-Hoc Network which is named as FOGHANET and connect all the Fog managers (RPI boards) to this network. Therefore, we can claim that our architecture is a portable automation solution which is not affected from the physical form of an environment. This is the main reason of the In our research, we named RPI boards as Fog managers because they are the orchestrator nodes of the Fog Layer. However, these equipments could also be called as Fog servers. When at least one Fog manager powers up, the network is automatically created and each of the managers is connected to it by using a static IPv4 address which is unique across the network. It additionally decreases the complexity of routing procedures and the need for a DHCP server which assigns IP addresses dynamically. Inter Fog manager communication is provided via the wireless FOGHANET in the Fog Layer whilst a gateway manager has an Ethernet connection to the home router in order to connect FOGHANET to the Internet. At the bottom side, data transmission between the Fog managers and devices located at the Things Layer is supplied via different kinds of technologies and protocols. The communication type inside FOGHANET is named as the control plane and Fog-things channel is referred

as the data plane. The data plane is more complex than the control plane because the Things Layer could include lots of heterogeneous smart devices. Thus, communication middleware between Fog and Things Layers requires the support of various protocols and technologies. In order to illustrate the usage of RPI boards in a house, Figure 4.2 is presented.

Consequently, our architecture is designed as a modular system which can be adaptable for both smart homes and larger environments. The modularity is provided by separating hardware components into different layers based on their physical locations or job definitions. Additionally, we utilized and created adaptable software modules/infrastructure in order to satisfy scalability, reliability, security and adaptability. If we aimed to just control, activate or deactivate several smart appliances in the home environment, our architecture would be considered as overqualified. However, we desired to compose a system which is able to manage an environment via smart appliances and also perform any kinds of processing power/time critical tasks such as face recognition, disaster monitoring, video streaming. In the following sections, FOGHA components and their interaction will be presented in detail. Since the FOGHA architecture and implementation are highly bounded to processing mechanisms at the edge of network, the order of subsections is organized as follows: Fog Layer, Things Layer, Cloud Layer, and the user interaction part. In each section, implementation details will also be presented in order to make the thesis structure more compact and to prevent repetitions.

4.2. Fog Layer

Fog computing term is firstly pronounced by Cisco claiming that the data produced by devices, people, and things will reach 500 zettabytes, and 45% of IoT generated data will be processed at the network edge by 2019 [67]. This prediction shows that physical environments need to be covered by local servers which are even capable of resource demanding tasks. This movement towards Fog computing (also pronounced as Edge computing, Fog networking) also benefits from the developments in hardware equipment. Since edge of the network is clearly different than the centralized data

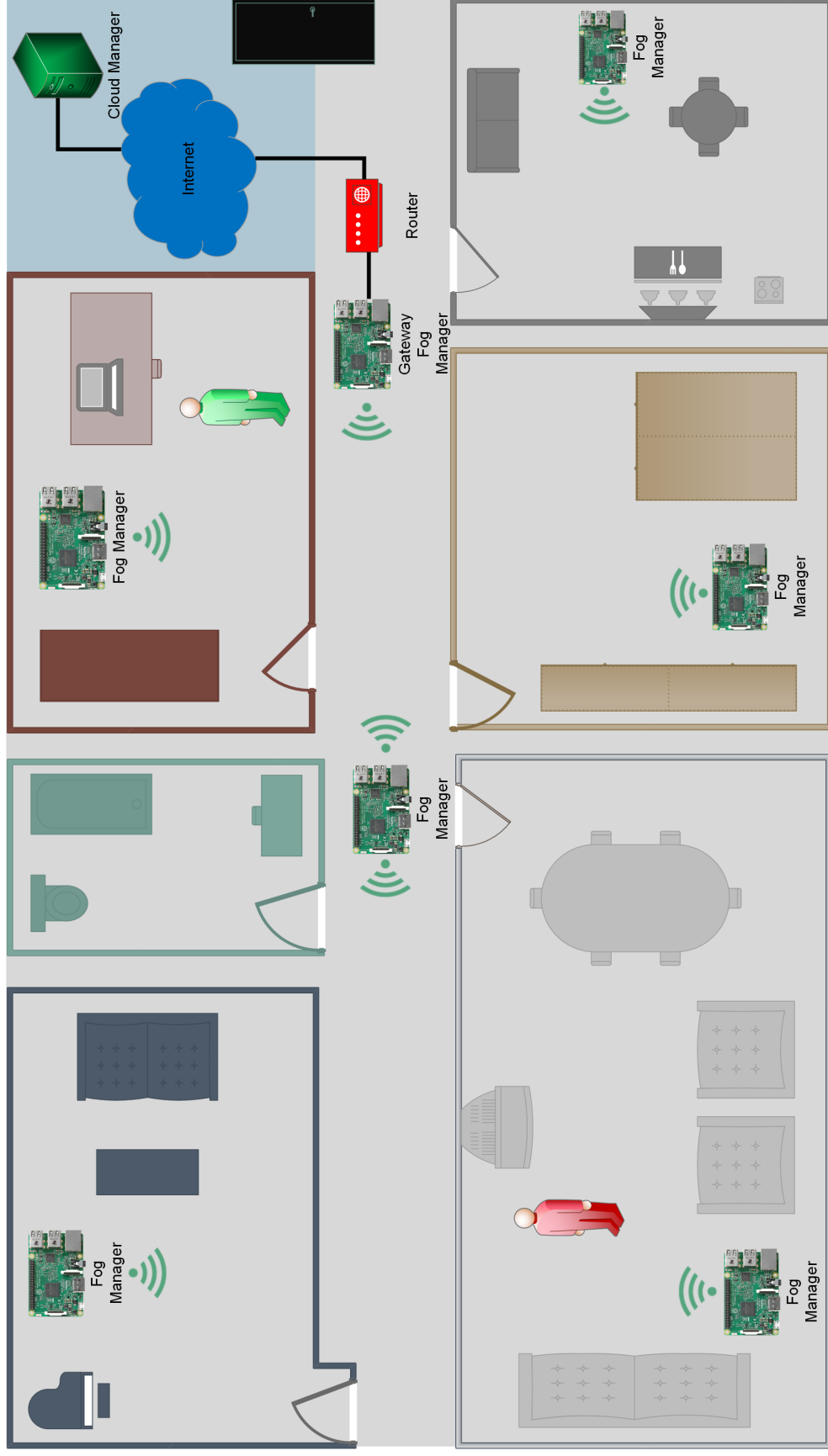


Figure 4.2. Distribution of RPI Boards in a Smart Home Scenario

centers, Fog computing needs geo-distributed low-power local servers. This shift from Cloud to Fog leads to the rise of single board computers which are ideal for the edge of network in terms of their physical size, energy efficiency, and satisfactory computational power. However, this shift also brings additional complexity to the network edge which should be orchestrated efficiently. In our architecture, the Fog Layer term is used for describing our proposed solution for the edge of network. The main physical components of this layer are illustrated in Figure 4.3.

4.2.1. Data Processor Nodes: Fog Managers

Functionally, physical components in any IoT system could be divided into two as actuators and sensors. In our system, we selected RPI (version 3) single board computers as the main actuators because of their popularity, programmability, performance, community support, and cost. It contains a Quad Core 1.2 GHz Broadcom CPU, 1 GB LPDDR2 (900 MHz) RAM, Broadcom Video Core IV GPU, built in Wi-Fi and BLE

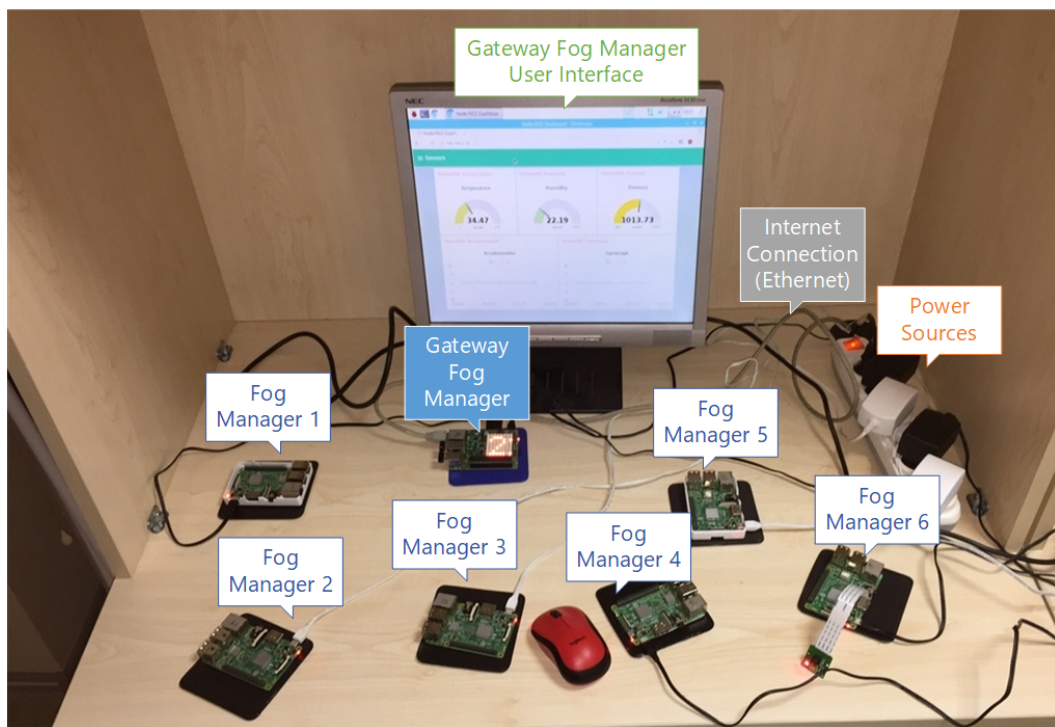


Figure 4.3. The Physical Components of the Fog Layer.

modules, 4 USB-2 ports, HDMI support, CSI camera port, micro USB power source, and micro SD port for storage. These technical specifications makes it an available candidate for the purpose of controlling other smart appliances and processing data locally. Compared to a general purpose computer, its capabilities are relatively limited however this does not prevent it from running CPU intensive tasks such as image recognition or big data analysis.

Executing such tasks only on one node could not be feasible when we consider the processing power of single board computers. However, distributing time-consuming tasks over a set of nodes at the Fog Layer would provide us significant processing capabilities. Compared to Cloud computing, creating a distributed system at the Fog network enables minimum network latency and communication overhead. This does not mean that Cloud and Fog computing are opponents. The maximum benefit would be acquired if we used them as complementary domains.

In addition to processing power achievements, composing a distributed system using RPI boards allows us to cover larger environments geographically. Because of the physical range limitations of widely used communication technologies like Wi-Fi, BLE and ZigBee, we are required to construct an architecture which contains geographically distributed RPI boards for the purpose of controlling the environment and other smart devices. Most contemporary electronic equipment produce data and publish it for further processing in order to extract meaningful results. In our system, the produced data are collected, processed and transferred to the other Fog control nodes or remote nodes based on the task requirements.

4.2.1.1. Operating System. Raspbian [68] is a Debian based Linux operating system which is officially supported by the Raspberry Pi Foundation. There are also several third party alternatives for Raspbian, including, but not limited to, Ubuntu Mate, Snappy Ubuntu Core, Windows 10 IoT Core. Raspbian is selected as our Fog/Cloud managers' OS because of the following reasons: Firstly, it is an open-source software and the community behind it is so active. Secondly, it is specialized for RPI hardware

by providing over 35,000 software packages and pre-compiled applications. Thirdly, it is a lightweight OS compared to the alternatives which is a fundamental point for tiny single board computers.

In the FOGHA architecture, the latest distribution of Raspbian OS, PIXEL is used in all data processor nodes including Cloud managers. Using a Linux based OS by enabling GUI leads to easier software development. Additionally, it presents a graphical interface to the users for the purpose of controlling and monitoring the environment. Lastly, it allows us to create a processor node that is able to run any kind of software such as a server, middleware, database, REST API. Furthermore, these software can be integrated with other libraries, plug-ins, and frameworks. This elasticity paves the way for attractive automation scenarios which are required in the FOGHA architecture.

4.2.1.2. Software Development on Fog Managers. RPI (version 3) boards contain an HDMI, Ethernet port, and four USB interfaces. In this way, we are able to use it as a standard desktop computer with the help of a monitor, keyboard and mouse. Moreover, additional driver installations for these devices are not required due to the usage of Raspbian OS. Another option would be making an SSH connection [69] between a board and the desktop computer that prevents you using the desktop environment of Raspbian OS. Both of these solutions are not efficient if the system includes multiple boards, since the connection elements need to be removed from a board and plugged to another board even for a simple configuration during the development phase.

Instead of depending on hardware equipment, a better solution is to use additional software in order to connect RPI's command line and GUI interfaces. For this purpose, Real VNC [70] is used as a graphical desktop sharing system while making developments on RPI boards. It enables you to remotely control an RPI board which is running a VNC server application from another computer via a VNC Viewer software. The VNC server starts when the RPI is powered on and listens for the connections from VNC clients. It is a secure connection type like SSH because credentials of an existent Raspbian OS user must be entered before the connection is established. Unlike an

SSH connection, the desktop interface of the OS is available for the users that is also important for visualization requiring tasks like video streaming or face recognition.

4.2.2. Wireless Ad-Hoc Network: FOGHANET

FOGHA architecture is designed by considering scalability of the Fog managers at the network edge. It is a fact that coverage of a larger area requires more processing nodes. Therefore, the Fog managers should be able to communicate with the other nodes in the environment in a secure, fast and reliable way. There are several ways to connect managers of Fog Layer which can be seen in Figure 4.4.

The first solution is to connect them to a wireless access point (WAP) by using its built-in wireless module interface. In this type of connection, each Fog manager communicate with the others over a modem which is connected to the Internet. The only requirement is to store credentials of the wireless network at the device. In that way, they are able to connect to the other Fog managers and the Internet. The first drawback in this structure originates from the limited signal coverage of the WAPs. Thus, additional WAPs needs to be installed if the covered physical area is large or contains too many obstacles/walls. Secondly, the Fog managers will be vulnerable

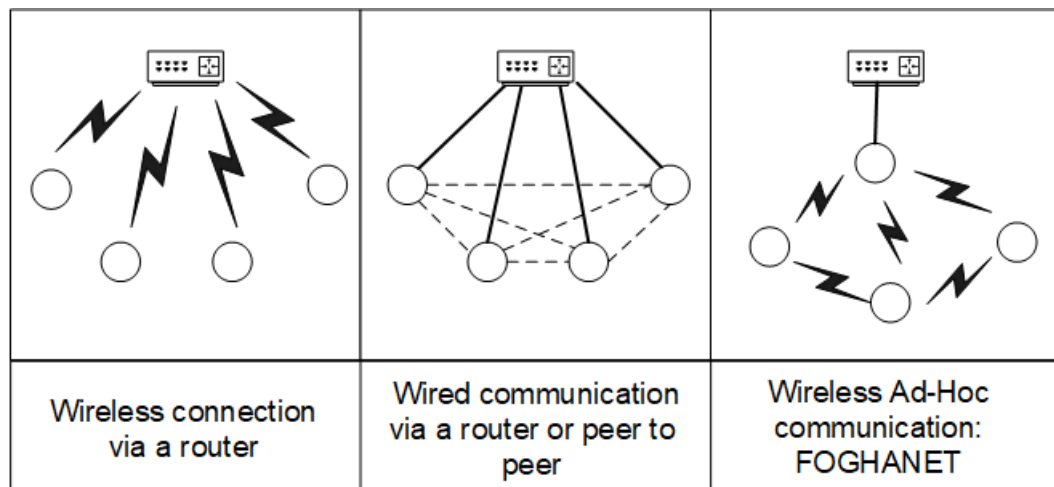


Figure 4.4. Data Transmission Methods at Fog Computing Layer.

for any type of attacks since they are connected to the Internet via a home router. In addition to security issues, sensitive personal data collected from the environment could be deciphered by attackers that arises a catastrophic privacy problem.

The second solution is similar to the first one and it does not contain any wireless connection interface. Each of the Fog managers is connected to a home router with Ethernet cables. This is very similar to the first solution but the only difference is the communication medium which presents a much secure data transmission at the network edge. In addition, it has lower communication delay and more bandwidth compared to any type of wireless data transmission. However, it requires a lot of Ethernet cables to cover even small areas like houses. Such kind of an infrastructure which includes a considerable amount of cables are definitely not desirable for the users. Additionally, it is a big disadvantage for portability and also increases system installation time and effort. Since our aim is to create a scalable, reliable and portable home automation architecture which is independent of the environmental conditions, data transmission via Ethernet cables could not be an ideal solution for the inter-connection of processor nodes.

The third solution is to create an additional network of the Fog managers and select at least one of the Fog managers as a gateway for the Internet connectivity. Peer to peer (P2P) connections between managers would be formed in two ways as cabled and wireless. Using Ethernet cables which are indicated with dashed lines in the second column of Figure 4.4 would be an option but this has the same drawbacks with the previous solution. Thus, we believe that the best way to form a connectivity between Fog managers is the creation of a Wireless Ad-Hoc Network (WANET) that is named as FOGHANET in the FOGHA architecture. A static IPv4 address is defined for each one and they are connected to this WANET via WLAN interface of the RPI on boot. In that way, IP address conflicts are prevented. Additionally, separation of RPI's ad-hoc network from other networks provides additional security because the Internet connectivity is performed by the gateway Fog controllers. Thus, incoming and outgoing data traffic to FOGHANET is allowed or rejected by these specific nodes via predetermined rules and procedures. Reliability of the system could also be increased

by adding additional gateway nodes. An extra 4G/3G/GPRS module could be integrated to the RPI devices in order to have an alternative Internet connection interface by considering disaster cases and infrastructural problems.

4.2.3. Software Containerization

Software containerization is a virtualization technology used to run and deploy distributed applications by isolating each application from the others without usage of any virtual machine (VM). Multiple applications/services can be launched on the same host with the usage of the same OS kernel which is the main difference between containerization and virtual machines (Figure 4.5). Docker [71] is selected for software containerization in this thesis, however there are also other alternatives in the market which are compared in Table 4.1. As indicated in the table, application level containerization is only supported by Rkt and Docker. On the contrary, other tools only support system level containerization that is very similar to virtual machines in terms of functionality. For example, a database, a Web server and an in-memory cache software can be separated into different isolated containers in the host machine. Software modifications in a container does not affect the other containers which absolutely eases software development and improves system stability and security. Thus, using Rkt or Docker is more suitable for creating a modular software architecture at the expense of increased complexity.

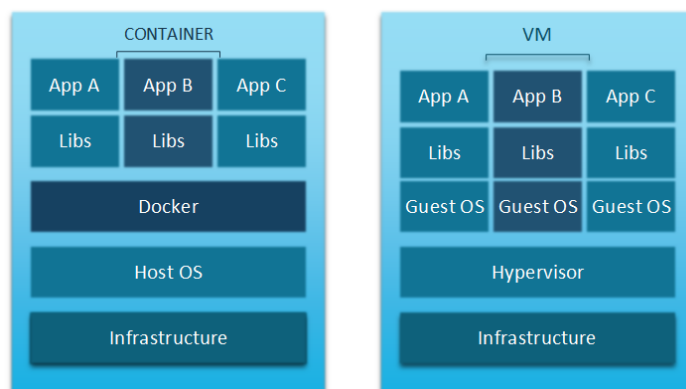


Figure 4.5. Docker Containers and Virtual Machines.

Another important criterion for our system is the support of ARMv7 processors because they are the main processing units of RPI boards. Although, Rkt supports Docker containers, it currently does not support the ARMv7 processor architecture. In addition to this, Docker is the most popular containerization technology nowadays and lots of publicly available containers could easily be adopted to the system without building that software on the device. This is quite important because some complex software requires hours of building processes on the device. Lastly, a Docker container could be pushed to the Cloud for further usage or backup purposes from a host and even it can be pulled from another device automatically. When our architecture is taken into account which has many Fog managers, this elasticity brings a huge advantage for the software updates. For all these reasons, we believe that Docker is the most suitable containerization tool in the market. Technical details of Docker containers is presented in the following subsection.

Table 4.1. Comparison of Containerization Software.

Feature	Docker	LXC	LXD	OpenVZ	Rkt	Linux-VServer
Virtualization	OS level	OS level	OS level	OS level, hypervisor	OS level, hypervisor	OS level
Container type	Docker	Linux (LXC)	Linux (LXC)	Virtuozzo	appc, Docker	Security Context
Platforms	Linux, MacOS, Windows	Linux	Linux	Virtuozzo Linux (only)	Linux, MacOS, Windows	Linux
Application Container	Yes	No	No	No	Yes	No
System Level Container	No	Yes	Yes	Yes	No	Yes
Programming Language	Go	C, Python Shell	Go	Go	C	C

4.2.3.1. Docker Containers. A Docker container image is an executable package of a set of software which also include additional resources such as required packages, installations, binaries, libraries and configurations for each of them. Containers are living instances of those images and each one is isolated from the others. In that way, any failure in one of the containers only influences this container and can be fixed without a catastrophic system level failure. A container can be constructed from scratch by creating a Docker file which can include any valid Linux shell commands. Alternatively, publicly available Docker containers could easily be used as a base with just one Docker pull command.

A stable version of a container can be exported to an image that can be stored in Cloud or locally. Usage of Docker containers are illustrated in Figure 4.6. Additionally, they can be shared with the other people publicly via some Cloud services like Docker Hub. For the development and update processes of an IoT system, this feature is very critical because it enables the usage of a set of software by hiding building and configuration procedures. In this thesis, we tried to separate applications/services to the different containers as much as possible because of the mentioned reasons.

Fog Layer of FOGHA consists of geographically distributed processor nodes which run Raspbian OS and Docker container stack on top of it. There are two types of

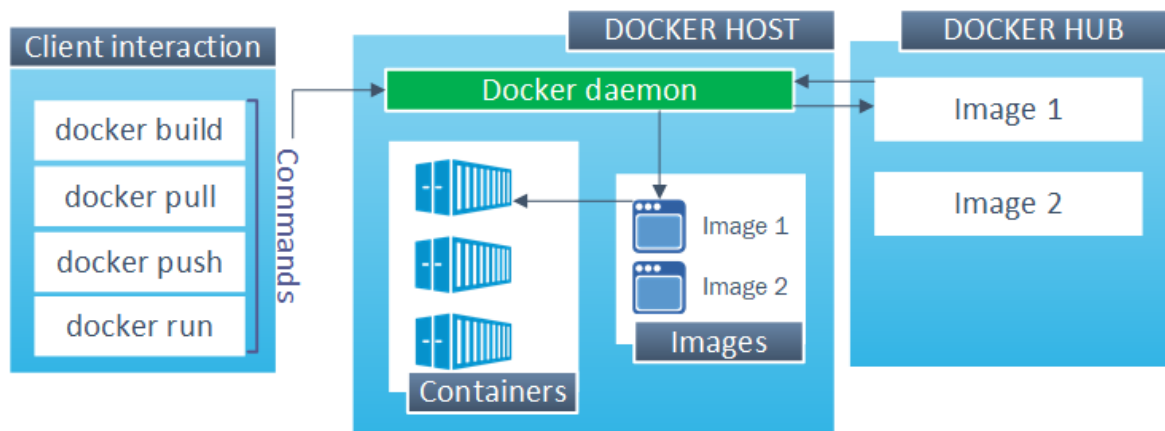


Figure 4.6. Software Development and Deployment via Docker Containers.

containers in the FOGHA architecture: base and auxiliary containers. Each base container is configured to be launched on system boot and runs forever unless it is externally stopped. We named them as base containers because they form the software infrastructure of FOGHA managers. If any of them does not perform in a stable manner, the FOGHA system also will not work properly. On the other hand, auxiliary containers are optional for the architecture and can be deployed, started or stopped based on the requirements.

4.2.3.2. Docker Swarm Mode. Management of containers is another important domain for containerization technologies. Since RPI boards are not so much powerful in terms of computational resources, the management of software running on them should be properly managed. If orchestration of them is not considered as a critical point, RPI boards may become non-responsive nodes or even break down because of the high load. In addition, some nodes may require a specific type of services while others do not.

For example, only two RPI boards in the home environment may include a camera module which requires a specific container for the face recognition task. In this scenario, running that container/service in all the nodes would be useless. Thus, a manager node should be able to dictate which service is belong to which node. On the contrary, a specific task which is packaged into a container may require multiple nodes for processing purposes. In this case, a master node should be able to scale that service to the multiple nodes as running containers. In order to achieve high availability, a second master should also be able to keep the system running by performing the master node tasks if it is down. For container orchestration, there are a group of tools as shown with their features in Table 4.2. In our architecture, we used Docker Swarm [72] for the orchestration and clustering of containers. Since it natively supports the ARMv7 processor architecture, it fulfill our requirements, and works with Docker containers seamlessly.

Nodes in a swarm cluster can be separated into two groups in terms of their roles as swarm managers and workers. The first node joins the swarm cluster as a swarm

Table 4.2. Comparison of Container Orchestration Tools.

Feature	Docker Swarm	Kubernetes	Mesos	Nomad
Supported Containers	Only Docker	Docker, Rkt	Docker, Rkt	Docker, Rkt, LXC
Service Discovery	Yes	Yes	Yes	Third Party (Consul)
Master Node Replication	Yes	Yes	Third Party (Zookeeper)	Yes
Load Balancing	Supports External Balancer	Supports External Balancer	Third Party (Marathon)	Third Party (Consul)
Auto Scaling	No	Yes	Yes	No
Container Monitoring	Third Party	Third Party	Third Party	Yes
Container Logging	Third Party (ELK)	Third Party (ELK)	Yes	Yes
Secrets Management	Yes	Yes	No	Third Party (Vault)

manager node by default and receives secret tokens for swarm workers and managers. Later, each joining node is required to specify its own role and related secret token. A swarm cluster can include multiple swarm worker and manager nodes however only one of the managers is selected as the leader. If it is in an inactive state, one of the swarm managers is automatically assigned as the swarm leader and orchestration of the containers and services is handled by this node. Management of them is illustrated in Figure 4.7 by providing distribution of example swarm services. In our architecture, a node in the Cloud Layer dictates the roles for swarm cluster nodes based on their current resource usages and properties which will be explained in the following sections.

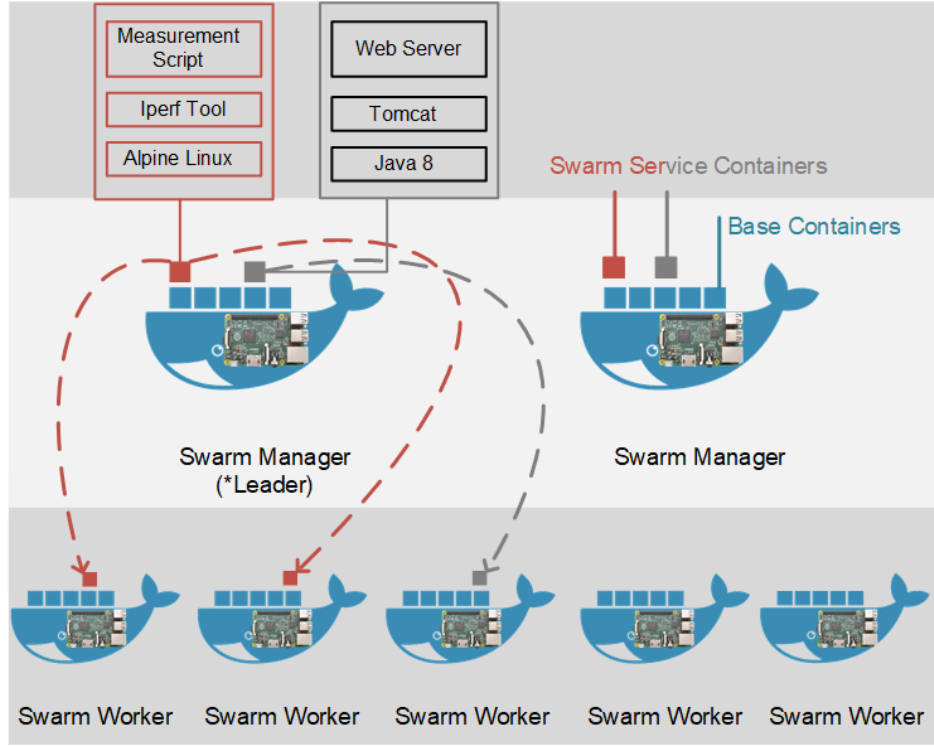


Figure 4.7. Orchestration of Containers and Services.

4.2.4. Software Stack of Fog Managers

In the previous sections, modular software packaging via Docker containers and their orchestration with Swarm Mode are explained architecturally. In this section, our implementation will be detailed starting from the base containers that construct processing, communication, and storage infrastructure of the RPI boards. In pursuit of them, optional service containers which managed by Docker Swarm will be explained. For processing purposes, we need to install a lightweight home automation server on boards which is able to interact with sensors and actuators in the environment. For long-term storage purposes, we are required to configure a database. An in-memory cache mechanism is also required for short-term data storage and instant replies. Lastly, there should be a mechanism which is able to communicate with the other nodes in the Fog Layer. Each of these infrastructural mechanisms correspond to different Docker container in the FOGHA system. In the following subsections, those will be explained in detail.

4.2.4.1. Node-Red Server. In Section 2, we compared software frameworks that are mainly created for home automation scenarios. While doing this comparison, we selected the most popular ones and measured their performances. Based on these measurements which will be presented in the following sections, Node-Red framework is selected in order to program RPI boards. In the FOGHA architecture, official dock-

Table 4.3. Node-Red Features.

Feature	Detail
Development Language	Javascript
Underlying Software	Node.js
Open-source packages	Available in Node Package Manager(npm)
Supported Devices	Local computers, SBCs, Cloud servers
Docker Support	There is an available official Docker image
Development Tool	A Web browser based text editor is available
User Interaction	UI nodes are available in the palette which enables data presentation in graphs or user interaction
UI development	It accepts any valid Html or Angular.js commands for UI development
Categorized Nodes	There are many nodes in the palette which are developed for various kind of tasks.
Flow Logic	All data processing mechanisms/routines are defined in the flows which wire a wide range of nodes
Routines & Triggers	It enables auto-running a flow/job in every x second, or triggering specified functions after a related event
Xbee Communication Support	Available via third-party nodes
BLE Communication Support	Available via third-party nodes
Available Protocols as Nodes	There are many nodes which either officially deployed or developed by third-parties such as MQTT, XMPP, AMQP, CoAP, UDP, TCP, Websockets
File Operations (Read & Write)	Ability to create/change files in any format

erized version [73] of Node-Red is used as the base Docker image and our customized version [74] is pushed to the Cloud as a public repository. It runs on top of a Node.js server and provides a browser-based programming environment for the developers. A detailed analysis of its features is presented in Table 4.3.

Its main power comes from the available software packages (around 1300) which is specifically written for the Node-Red framework. These packages mostly abstract lower level implementation details and eases integration with other hardware devices. Additionally, any Node.js package (around 500 thousand) can be installed via the node package manager (NPM) [75] and integrated to the system. For software development part, it presents a browser-based GUI which is shown in Figure 4.8.

To sum up, Node-Red framework eases the implementation of home automation scenarios by presenting many packages, nodes and flexibility. Turning on/off home appliances could be implemented by utilizing various communication technologies and protocols. In addition, it allows you to launch any kind of executable scripts/programs. For automation scenarios, a predefined job could be started with a delay or immediately after a specific event is occurred. For continuous data streaming/reading purposes, a very large range of protocols are available to be integrated to the system. There is not any limit for home automation scenarios in this era. However, we believe that FOGHA architecture is raised as a flexible, improvable, scalable and integrable system with the help of benefits provided from the Node-Red framework. In Chapter 5, our implementation will be explained in detail through the use cases of FOGHA.

Reading sensor data from BLE enabled devices is a common scenario for smart homes. In Figure 4.9, efficient use of different software packages is shown for BLE sensor data reading in RPI boards using the Node-Red framework. As shown in the figure, OS level software are utilized by Node.js packages which form the base for Node-Red libraries. This collective structure abstracts low level software implementations and increases code reuse. For this scenario, after those packages are installed, the developer only needs to add a new Node-Red node to the logic and enter some configuration parameters via its Web-browser interface. This is a complex example which is selected

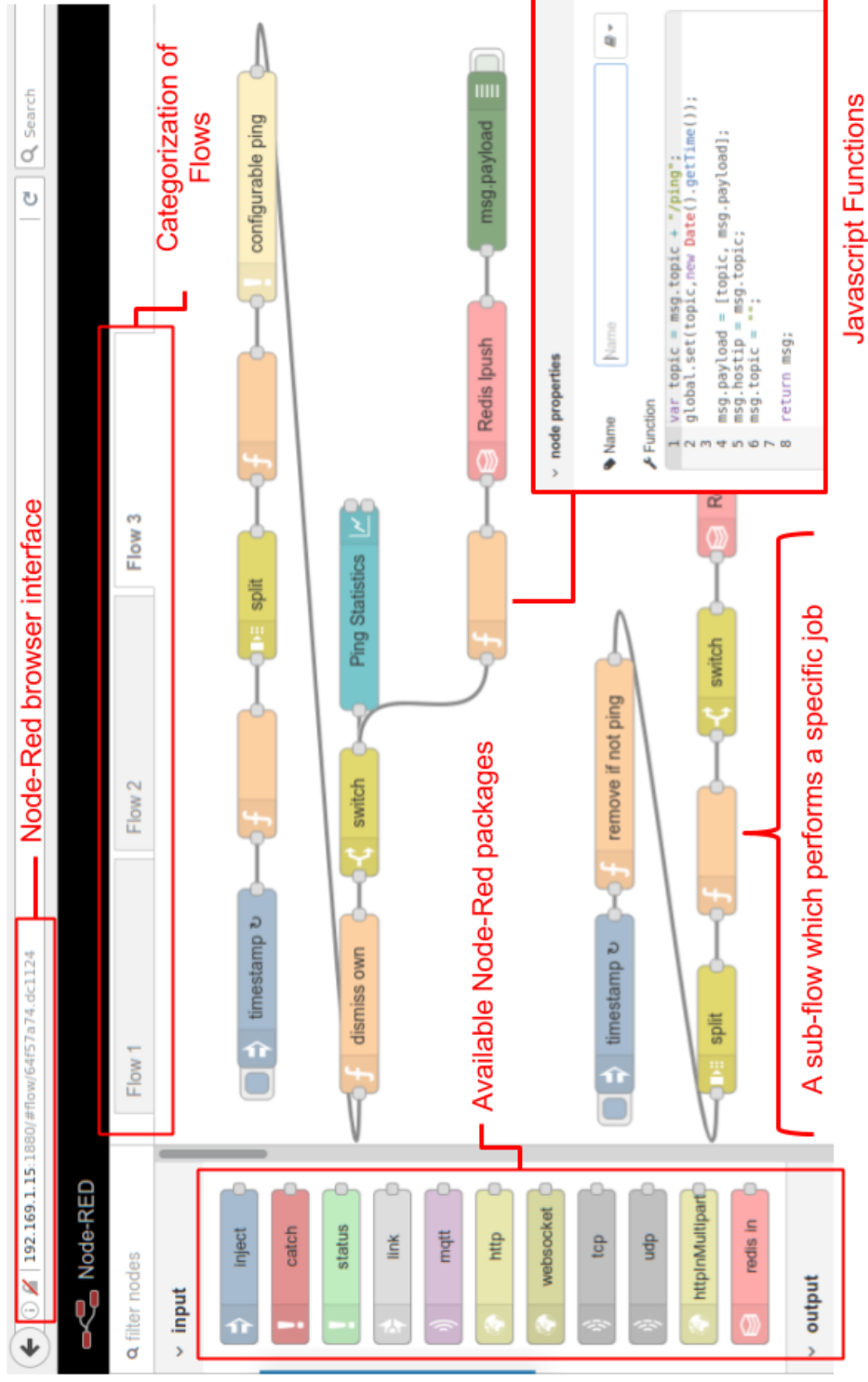


Figure 4.8. Node-Red Development Environment

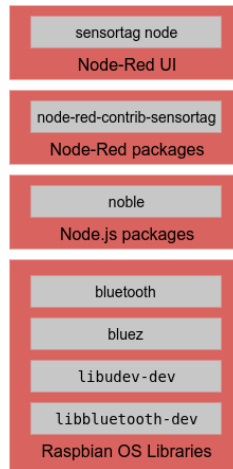


Figure 4.9. Example Scenario: Communication with BLE Sensors.

to illustrate the interaction between different levels of the software stack. In most cases, installing only one package which is included in the Node-Red packages is adequate.

4.2.4.2. InfluxDB Database. InfluxDB is an open-source database which mainly targets to store the time series data. It is optimized for the retrieval of time series data by enabling various use cases such as application metrics, Internet of Things sensor data, and real-time analytics. It provides a query language which is very similar to the SQL (Structured Query Language). Additionally, any kind of data can be written in the database using the HTTP, TCP, and UDP protocols which is a significant advantage for IoT applications. Multiple databases can easily be created with InfluxDB which brings an important benefit which is the usage of retention policies.

When you create a database, the default policy is applied which has an infinite retention time. This is not suitable for the IoT scenarios because the sensors produce significant amount of data at high frequencies which need to be processed immediately. Secondly, the local servers like RPI boards have limited storage capacity. Because of these reasons, we applied several retention policies to the databases based on the data type. For example, the CPU and memory usage data of the RPI boards are continuously written in the database and automatically removed after 7 days by InfluxDB. However, the Cloud managers can store that data throughout one year for the long

term analysis with the help of retention policies. Additionally, the Node-Red framework provides required plug-ins to write and read data from the InfluxDB. To sum up, we believe that the InfluxDB is very suitable for the FOGHA architecture and the IoT scenarios.

4.2.4.3. Chronograf. Chronograf [76] is the multi-purpose data visualization and management software which is supported by the InfluxDB. The key features of it are the data visualization, infrastructure monitoring, alert management, database management, and multi-user support. Since InfluxDB and Chronograf is the tools of the same company, the integration of them is very easy. In the FOGHA architecture, we selected it to continuously visualize the sensor and infrastructure data in addition to the database management. For example, we are able to analyze and monitor the sensor data which are published to the Fog managers in real time and take an action based on the average values of the data. Additionally, we can set a certain threshold value and receive an alert when the data exceeds it in real-time. To sum up, it provides a beautiful UI for data monitoring and analysis purposes which is accessible via a Web browser. The resource usage of the Fog managers is visualized via configuring Chronograf dashboard according to the data type. A sample resource usage of a Fog manager is shown in Figure 4.10.

4.2.4.4. Redis In-Memory Cache. Redis [77] is an in-memory key-value storage tool which can be utilized as a database, message broker, and cache mechanism. Since the main advantage of it lies behind the fast access to any type of data, we placed it as the in-memory cache infrastructure in the FOGHA architecture. It supports a wide range of data structures such as strings, lists, sets, bitmaps and hashes. In our architecture, it is used in the form of a Docker container [78] for the tasks which require real-time processing or instant data access. For example, sensors produce high amount of data and this data is stored for long-term access. However, this raw data needs to be analyzed for triggering a home automation scenario. For this analysis, retrieving the whole related sensor data from InfluxDB is very inefficient and slow. Because, reading a huge amount of data from the disk (in our case RPI's micro SD card) and extracting

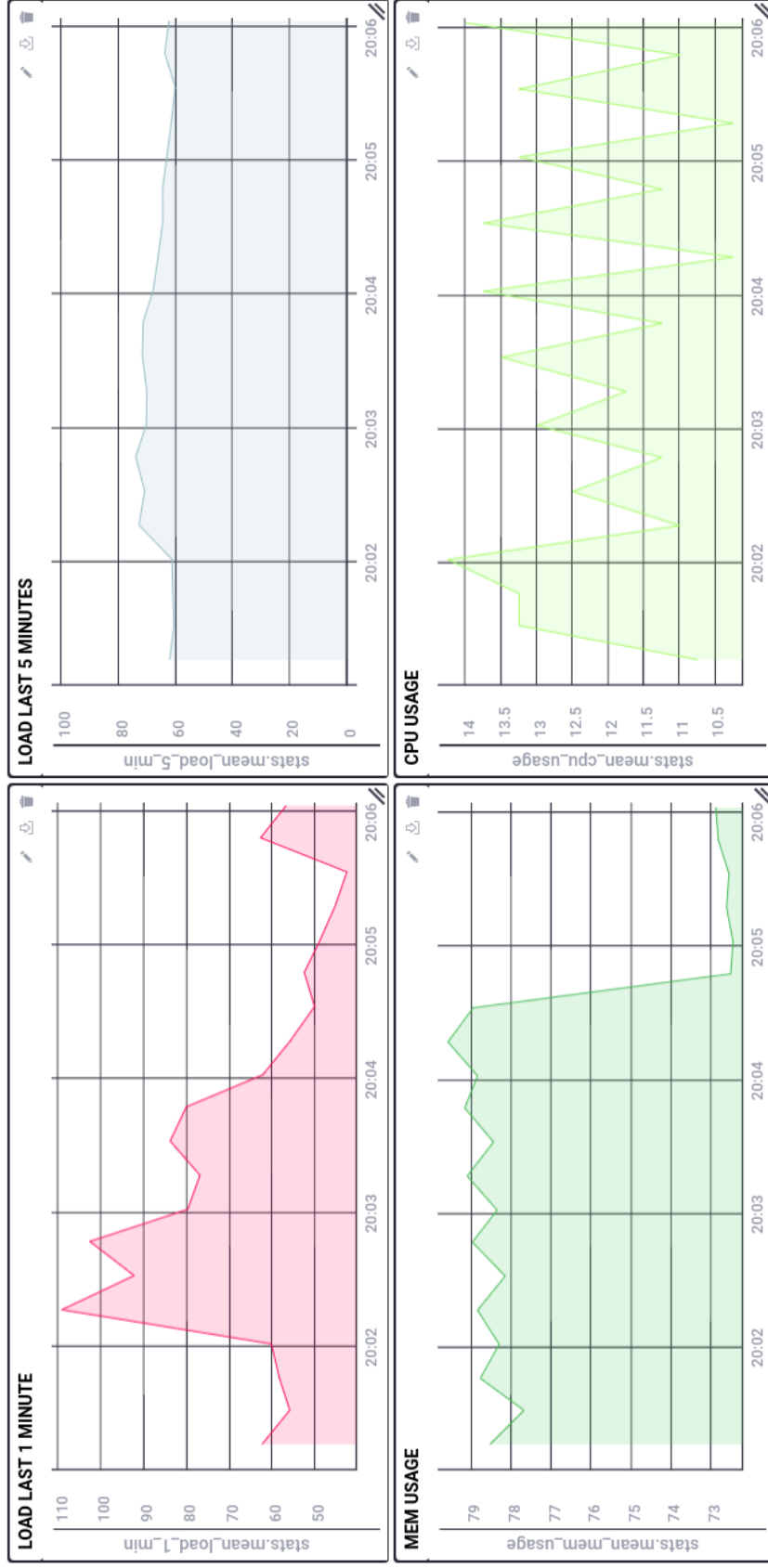


Figure 4.10. Visualization of the Resource Usage of the Fog Managers.

a meaningful result will take a considerable time. Thus, the collected data is processed and converted to a more structured form in RPI boards as soon as possible. After then, it is stored in memory via the Redis tool by considering the algorithms which require that processed data.

The second usage of Redis is related to the retrieving a single object from the storage instantly. In this case, direct data retrieval from InfluxDB can be a bottleneck for our architecture if the InfluxDB tables contain huge amount of records. Thus, we believe that storing frequently accessed data in memory using Redis will eliminate this problem. It does not mean that InfluxDB and Redis are competitors for data storage purposes. They serve as complementary elements for each other based on the usage of collected data which is shown in Figure 4.11.

4.2.4.5. MQTT Broker. As explained in Chapter 2, MQTT is a lightweight publish/subscribe based messaging protocol. It is claimed [79] to be an ideal protocol

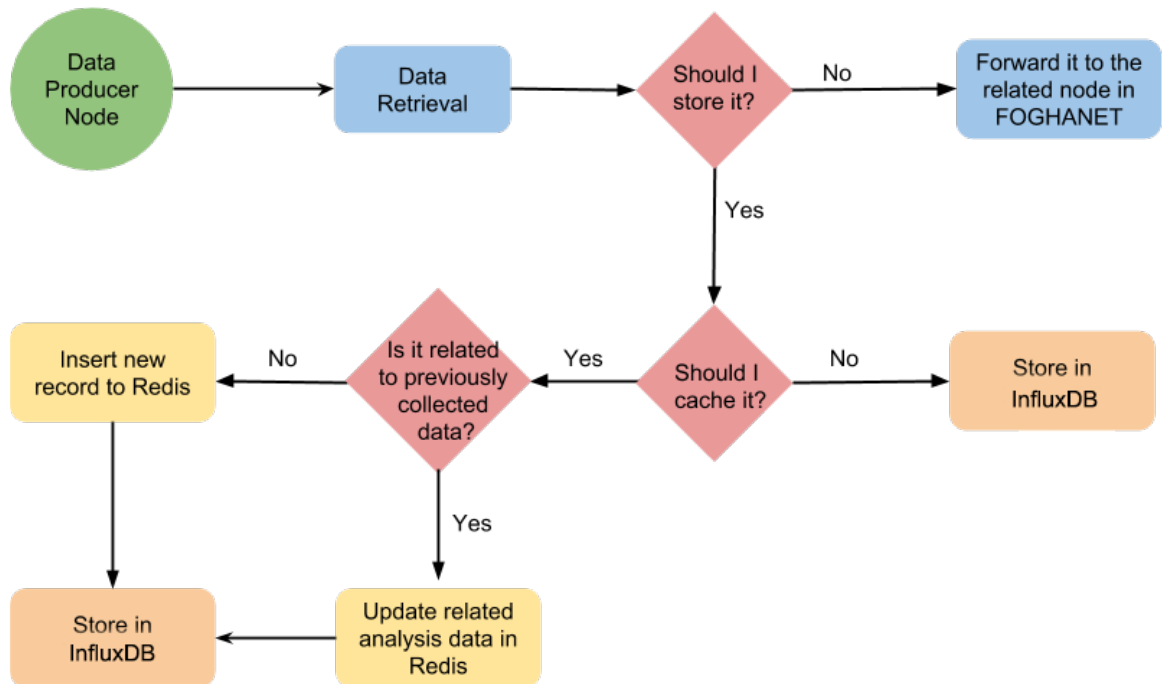


Figure 4.11. Data Storage Mechanism.

for IoT and machine-to-machine (M2M) connectivity because of its low power usage, minimized data packet size, and efficient sharing of data between multiple hosts. In the MQTT protocol, messages are sent to a broker which distributes that message to the subscribers. Each message should be published to a topic and later it is received by all hosts which subscribe to that topic. The basic communication mechanism is shown in Figure 4.12. As shown in the figure, clients must communicate with each other via an MQTT broker. However, our architecture does not suit for a single central broker because geographically distributed Fog managers may not be in the signal range of the broker node. In our architecture, besides producing data as an MQTT client, each RPI board is also an MQTT broker which is able to route traffic to the related nodes in FOGHANET.

In FOGHA, RPI boards (Fog managers) are physically distributed in the environment and connected to the FOGHANET as wireless nodes. We separated communication mechanisms in FOGHANET into two as monitoring and data planes. The former is only responsible for the transmission of the system monitoring related data such as temperature of boards, resource usage of boards, available bandwidth which will be explained in the next section. Data transmissions in this plane do not affect any home automation scenario directly. Instead, they provide information about system stability and infrastructure. These data are routed to a gateway node in the Fog Layer and later consumed by the Cloud Layer. In that way, availability of each node and link is known in FOGHANET that determines routing decisions and distribution of Docker Swarm services.

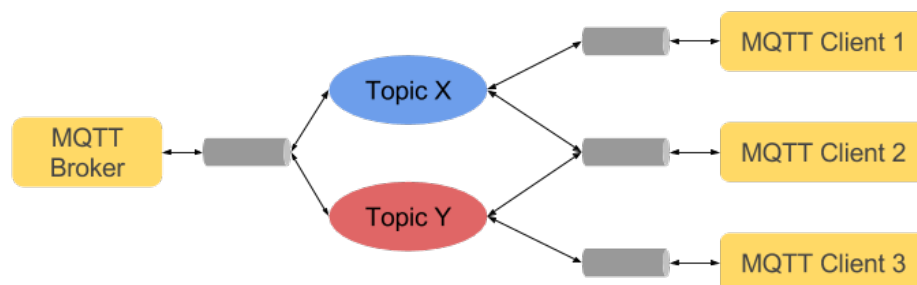


Figure 4.12. MQTT Communication Mechanism.

4.2.5. Management of Fog Layer

Fog Managers are geographically distributed nodes which are able to communicate inside FOGHANET with MQTT pub/sub protocol. FOGHANET is the name of a wireless ad-hoc network that each Fog manager is joined by default. Since each Fog Manager only has one wireless module, the connection between the Cloud Layer and FOGHANET is provided by a Gateway Fog Manager which is connected to the Internet via an Ethernet cable. Each node in FOGHANET sends ping requests to the other hosts every second. In this way, they are aware of the neighbor nodes.

FOGHANET contains geographically distributed nodes and the main advantage of this distribution is the increased signal coverage of the home environment. Because there can be lots of IoT devices which support wireless communication with Fog Managers. Absolutely, using only one RPI board is not enough for even a small house in terms of signal coverage if we consider such a scenario that each room has at least one BLE temperature sensor. Thus, we propose to deploy multiple Fog Managers based on the size of the environment.

Each of them is able to collect data from the Things Layer. However, they have limited processing and storage capabilities. Thus, transmission of the data to the Cloud nodes are required. We propose to arrange the distance between Fog Managers as approximately 30 meters based on our experiments which will be provided in Chapter 6. For such a setup, it will be meaningless to position the other Fog Managers in the coverage area of the Gateway Fog Manager. Thus, the data should be transmitted to a Gateway Fog Manager using multi-hopping techniques. This is achieved via the MQTT broker of a Fog Manager which receives the data from a neighbor node and forwards it to the next node. This process continues until the message is received by a Gateway Fog Manager.

We used one Gateway Fog Manager in our implementation but the number of them can be increased by connecting them to a router via an Ethernet cable in order to increase system reliability. Even the system can be converted to a more robust

architecture. For doing this, one of the Gateway Fog Managers could be connected to the Internet via 3G/4G modules. In this way, connection to the Internet is backed up with an alternative connection channel.

The availability of communication links between hosts has a crucial impact on system reliability. In order to monitor these links, default ping node of Node-Red is used by sending a ping request to the other RPI boards in FOGHANET. The interval time is selected as 500 milliseconds in order to understand if a link or a node is broken. Ping statistics are also sent to the gateway Fog manager which broadcasts this information to the Fog Layer via the MQTT protocol. In this way, each node not only knows its neighbors but also the overall topology. If the gateway Fog manager does not get any ping information about a specific node in 3 seconds, it is accepted by all hosts that the node is not alive anymore.

4.2.6. Auxiliary Scripts

Separating jobs/tasks into the Docker containers has many advantages which is discussed previously. Thus, we believe that applications and services should be packaged into the containers as far as possible. However, this does not mean that we should only use containers for software development. For certain tasks, running shell scripts directly on the operating system is a better solution because of the need for simplicity. In addition, some of them are required to be launched on system boot and kept running always.

Crontab [80] is a tool used in Linux distributions which contains the schedule of shell commands to be run at specified times. A Cronjob is the set of instructions that consist of the command to be executed and its launch time. Cronjobs are defined in a Crontab file whose location differs from OS to OS. We believe that using Crontab is a good solution for integrating any kind of executable applications to our system and we defined auxiliary scripts as Cronjobs in our implementation which is listed below.

4.2.6.1. Monitoring Raspberry Pi Boards. As indicated previously, monitoring of Fog managers is very critical for system stability and prevention of data losses. For this purpose, we created an always working shell script and integrated it to Crontab as a start-up script. It collects information related to the current status of RPI boards every 5 seconds. Later, the data sent to the local MQTT Broker which forwards it to the Node-Red container in the same host machine. In the Node-Red container, the data is parsed, processed, stored and also sent to a Swarm master node as a part of the whole monitoring process. In Figure 4.13, this mechanism is illustrated in detail by hiding the mechanisms outside of the host machine.

In our implementation, the overall resource usage of RPI boards are measured by using a set of Raspbian packages which are indicated as gray boxes in the figure. In addition, the resource usage of each container is collected through the Docker's default statistics tool which is critical for the container orchestration. In this way, the performance of a service which is packaged into a container could be analyzed precisely and containers (excluding base containers) can be transferred from a busy host to a more appropriate host in FOGHANET.

4.2.6.2. Local Communication Mechanisms. In the Docker architecture, containers are isolated from each other and host OS in order to prevent security attacks and minimize the effects of errors. However, communication between the containers that are placed in the same host should be handled properly. For this requirement, containers' ports should be bound to host OS's ports while launching a container. Port expose feature of Docker enables this type of bridging which is also included in our system (Figure 4.14). Docker containers are able to consume the data which is sent to a specific port of the host OS, externally or internally, through port binding.

MQTT Broker, Redis, Chronograf and InfluxDB containers use their default ports for communication among base containers. However, Node-Red is the brain of our software stack and it communicates with the other base containers to cache, store, and distribute the data. InfluxDB, Redis and MQTT packages of the Node-Red framework

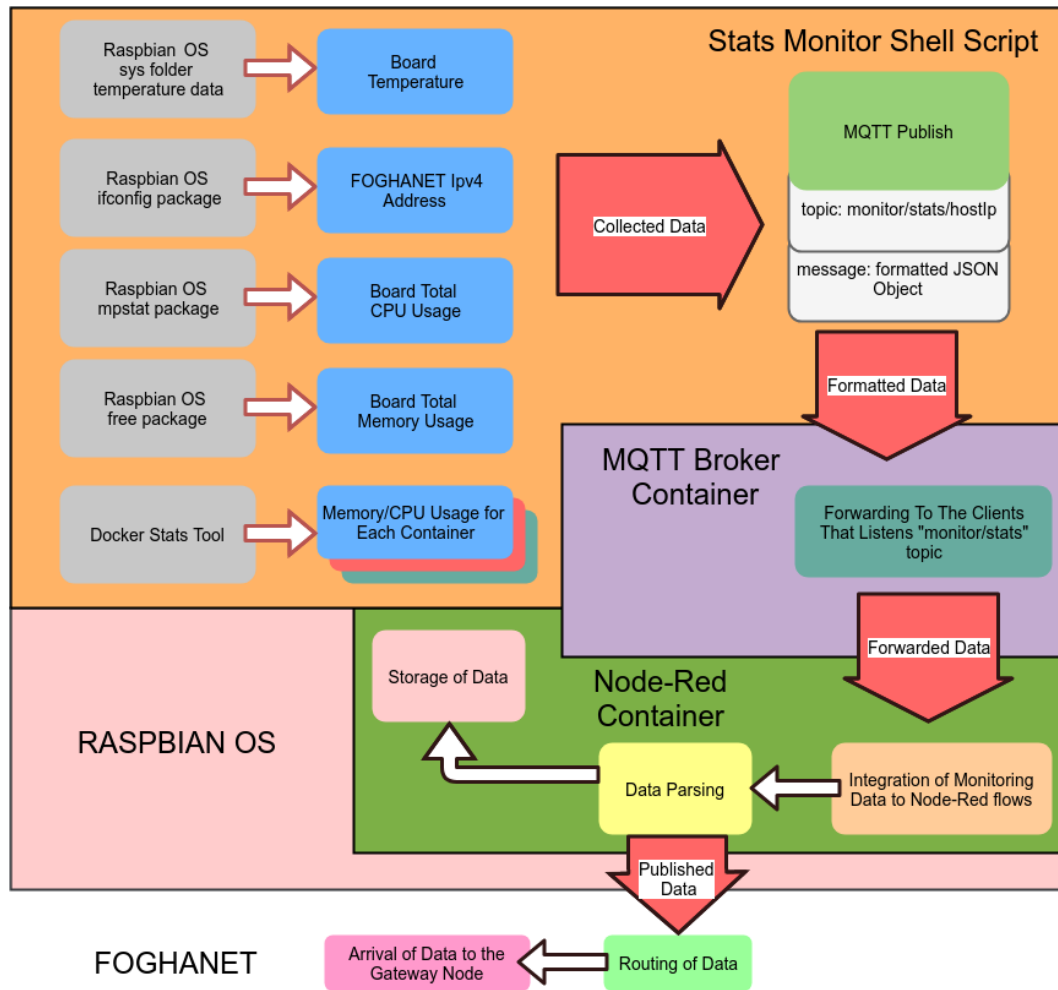


Figure 4.13. Monitoring of RPI Boards.

are installed for this purpose and these components are integrated to the software logic through required parameters.

In addition to container-container interaction, the applications which run directly on host OS should be manageable by a central software in order to prevent software faults. For example, usage of RPI camera is required by three different services in our implementation which are training face images, recognition of faces in video, and video streaming. Running these three tasks at the same time both end up with software crashes and extreme usage of system resources according to our experiences. Thus, additional commands may need to be executed based on the instructions. Independent of the use case, ability to start/stop processes and applications from a central point

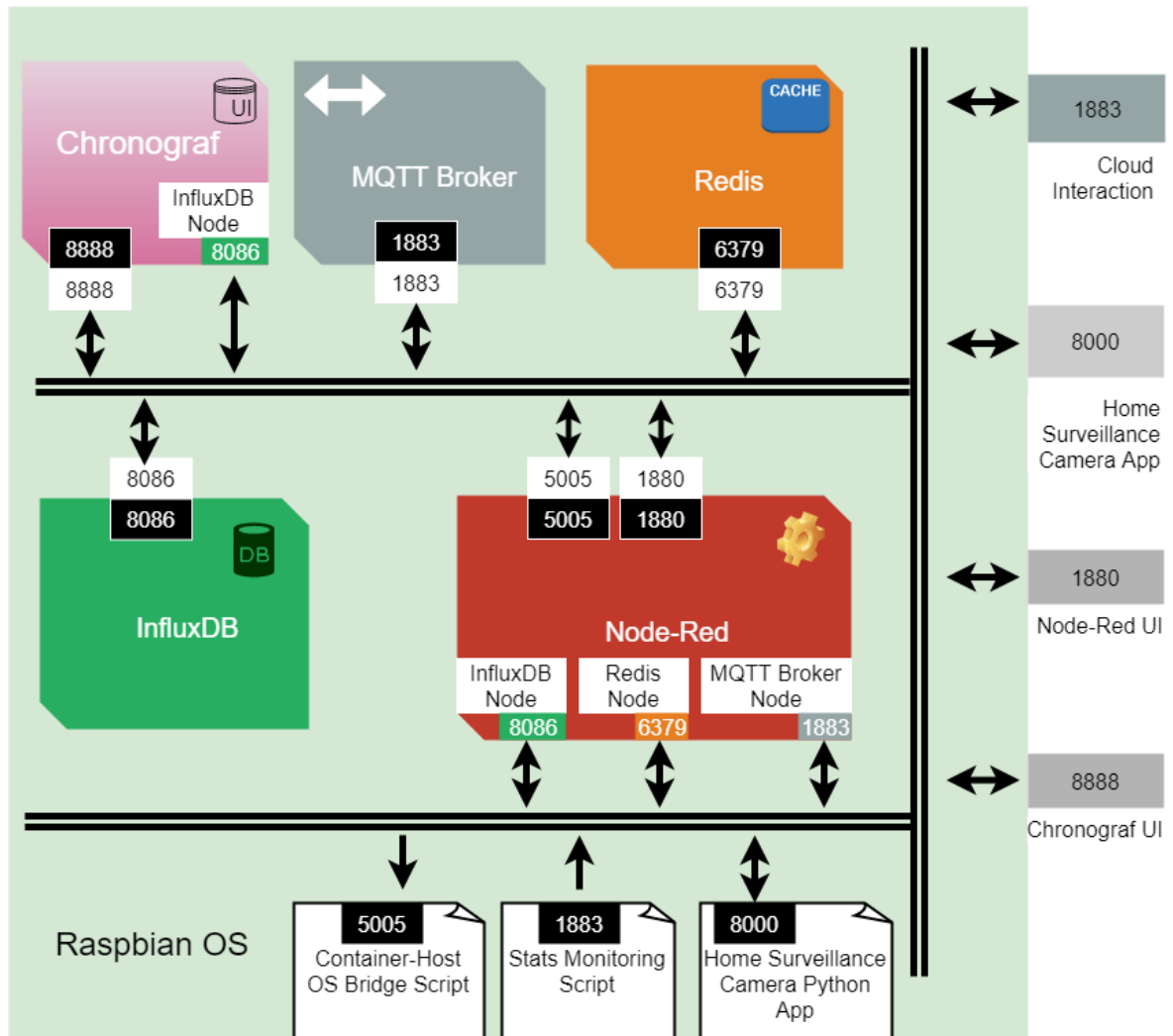


Figure 4.14. Host OS-Container Bridge.

is also required for generic purposes. For this need, we created a python script which listens TCP 5005 port continuously and executes commands based on the received instruction. Since our system logic mostly resides in the Node-Red container, these instructions are sent from Node-Red to the related OS port and immediately received by the python script which starts listening after the system boot through Crontab.

4.3. Things Layer

In the previous section, we explained that how the Fog Layer brings the Cloud technologies and processes to the network edge. In this section, we will explain the

other hardware devices which are placed in the same physical area with the Fog managers. We grouped those appliances under the Things Layer because it is the most dynamic layer which contains a broad range of hardware equipment. In addition, each device has its own configurations, requirements, and software infrastructure. Thus, integrating a new smart device or sensor to the system mostly requires additional processes. These integrations should be eased as far as possible in order to increase the system adaptability.

Popularity of the IoT domain leads to the rise of new hardware equipment, technologies and protocols which bring additional complexity for the collection and analysis of data. In order to decrease this complexity, we added the Fog Layer on top of the Things Layer. The interaction between them has a fundamental importance since the Fog Layer both manages the smart appliances in the Things Layer and enables new integrations. When a new device is needed to be added to the FOGHA system, requirements of that device may even require installing some OS packages. However, we facilitated this procedure by using the Node-Red framework in our Fog managers (RPI boards). It enables the use of readily available software packages that is specially written for a device and subsidiary packages which can be utilized to create a new integration structure.

Things Layer cannot be restricted with a set of technologies, equipment and protocols. It will be expanding continuously in parallel with the technological progress. For this reason, the Fog Layer is positioned as the brain of FOGHA so it is implemented as an integrable structure instead of putting any logic inside the Things Layer. In this thesis, we implemented the Things Layer through integrating different kinds of hardware devices to the system. Not only the implemented use cases but also the possible ones will be explained in the following sections.

4.3.1. Communication Protocols and Technologies

In the Fog Layer, Fog managers are connected to each other with the MQTT protocol because it is lightweight and supports publish/subscribe type of message

Table 4.4. FOGHA Supported Protocols.

	Status	Implementation Detail
HTTP Protocol	Supported Integrated	Support for interaction with an HTTP Server
UDP Protocol	Supported Integrated	Supported by default
TCP Protocol	Supported Integrated	Supported by default
MQTT Protocol	Supported Integrated	MQTT Broker as Container Support in Node-Red
Bluetooth Low Energy	Supported Integrated	Support for generic BLE Devices Support for Sensortag devices
Wi-Fi Infrastructure Mode Wi-Fi Ad-hoc Mode	Supported Integrated	Support for Wi-Fi enabled devices
XBee	Supported	Support for XBee commucation
XMPP Protocol	Supported	Support for interaction with an XMPP Server
AMQP Protocol	Supported	Support for interaction with an AMQP Server
Websocket	Supported	Support for Websocket communication

distribution. On the contrary, we cannot select only one protocol or technology for communication purposes in the Things Layer. Because it contains a wide range of heterogeneous devices which requires the implementation of corresponding software. In FOGHA architecture, we integrated the most popular communication technologies and protocols which can be seen in Table 4.4. These are the currently supported ones and they can be increased in the future based on the needs.

4.3.2. Actuators

There are mainly two types of devices in the Things Layer: actuators and sensors. Actuators may also be able to sense the environment and take actions based on the

given instructions. Actuators differ from the ones in the Fog Layer even they have computational power. Their computational resources are fundamentally responsible for activating or deactivating relays, lights and similar smart appliances based on the received commands from the Fog Computing Layer nodes.

Through amazing technology improvements in the last decade, most people are expected to control appliances via a smart-phone or Web-browser. Thus, many vendors have been changing their products and their functionalities in parallel with the IoT concept. For instance, modern household appliances provide several types of communication technologies and protocols by providing well-defined APIs. Since they are mostly able to sense and control the environment, sensing and actuating parts can be handled in the same device. On the other hand, this trend leads to a more heterogeneous environment because of the lack of standardizations in the IoT domain. Each device still has its own configurations and settings. This problem is tried to be solved by home-automation frameworks which can be integrable with the thousands of them. We previously explained our selected solution to this problem which is the Node-Red framework.

Additional to those type of smart appliances, all of us have traditional equipment in our homes. In the near future, we cannot expect that every device will be converted to a smart appliance. Therefore, additional hardware are needed to control such equipment. They can operate as a bridge between the smart and non-smart parts of an automation system. In order to solve this problem, we implemented a smart socket using a micro-controller, breadboard, relays, and cables which will be detailed in Section 5. The working principle of this socket is shown in Figure 4.15 by hiding the implementation details.

4.3.3. Sensors

There are a wide range of sensors available in the market currently. Some of them provide plug&play functionality whilst the others need to be integrated with additional boards like Arduino, Raspberry Pi. Both of them are integrated to our

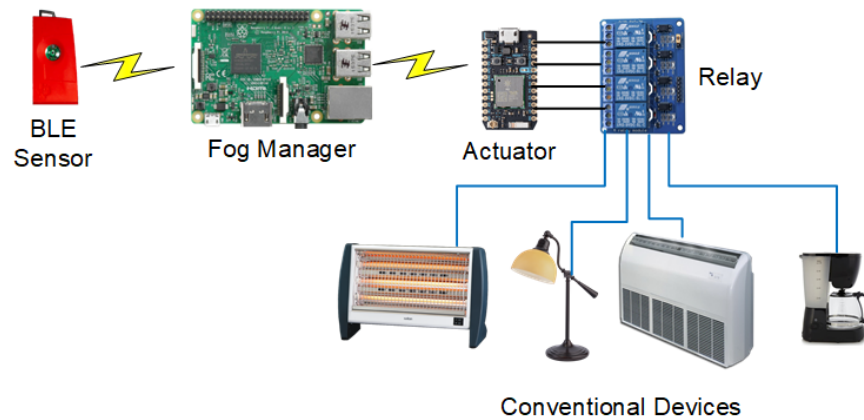


Figure 4.15. Functionality of Actuators.

implementation for demonstration purposes. Based on the needs, various types of sensors could be added to our system but this is not the main purpose of this research. Thus, we provide an infrastructure which is capable of integrating a wide range of sensors.

We integrated SensorTag CC2541 devices to the system by installing a set of Linux OS packages, NPM, and Node-Red libraries in order to connect a BLE communication. In that way, RPI boards are able to sense the environment using SensorTag's IR temperature, humidity, pressure, and accelerometer, gyroscope, and magnetometer sensors. The sensed data is stored, analyzed and used for the home automation scenarios based on the users' needs.

Since SensorTag devices provide data in a structured way, we can categorize these devices as advanced sensors. Additionally, we used Arduino boards by manually attaching additional sensor modules. In that way, we showed our architecture is both compatible with end-user products and hand-made equipment.

4.3.4. Fog Layer and Things Layer Interaction

The communication between Fog and Things Layers requires additional implementations and mechanisms since the Things Layer contains a large heterogeneity

compared to other layers. Different hardware, software, protocols, and configurations are all the components of this layer which make the communication with the Fog Layer difficult. There is not a unique way to form a global communication channel between these layers because of the large heterogeneity. Instead, we believe that supporting common technologies and protocols is a better solution. This does not mean that device specific implementations and configurations are not required. In the FOGHA architecture, Node-Red containers running on RPI boards are mainly responsible from the two-way interaction between Fog and Things Layers. They provide the integrity with the Things Layer and act as a brain for the FOGHA by running all rules, triggers, flows and decision mechanisms at the edge of the network. In Figure 4.16, signal coverage of RPI boards and currently available communication interfaces are illustrated.

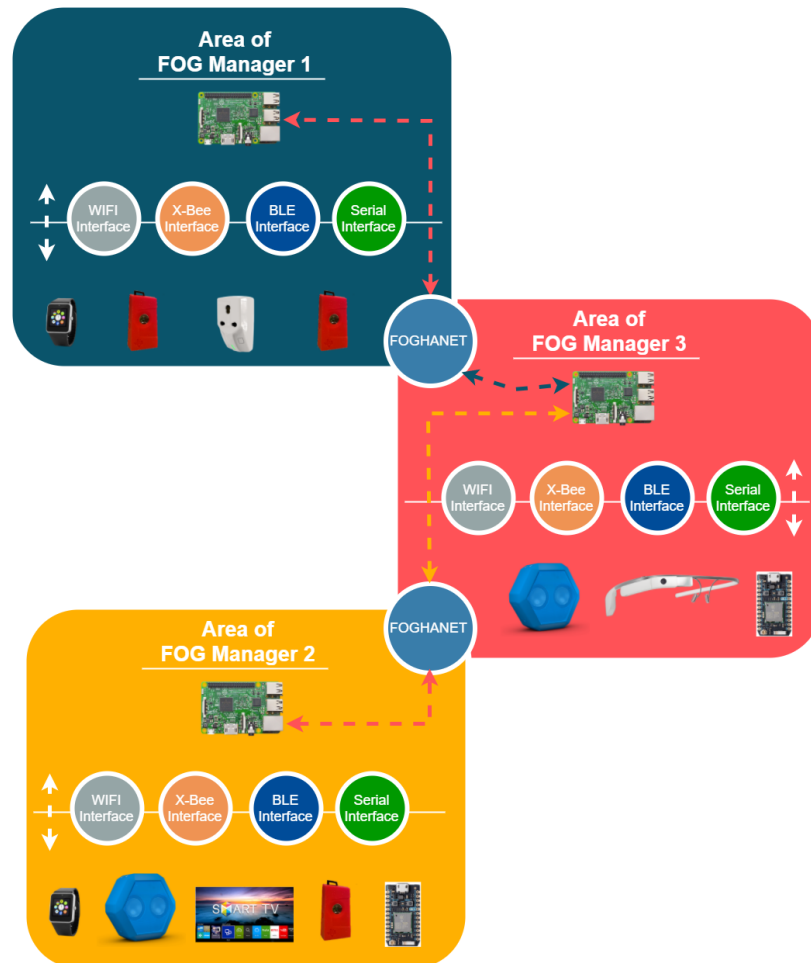


Figure 4.16. Interaction Between Fog and Things Layers.

4.4. Cloud Layer

Cloud computing is not a rival for the Fog computing paradigm and it is able to perform resource demanding tasks which may not be suitable for Fog computing. While Fog computing brings computational tasks to the edge of network by shortening communication delays, Cloud computing executes resource intensive tasks in a short time with low cost. Thus, we believe that combining these concepts is beneficial for our FOGHA architecture. As explained before, the Fog Computing Layer contains geographically distributed servers and collects data from the Things Layer by minimizing the communication delay. However, these servers do not have enough storage and computational power in order to handle huge amount of data that is received from the sensors. Thus, we added the Cloud Layer on top of the Fog Layer and connected them over the Internet.

4.4.1. Cloud Managers

In addition to Fog Managers, the FOGHA architecture also contains more powerful Cloud managers which are located in remote positions. Since they have more computational power compared to RPI boards, time-consuming tasks are designed to be handled in these nodes. In addition, they continuously monitor the status of the nodes located in the Fog Layer via the data published from Gateway Fog Managers. Therefore, the overall view of the architecture is available in Cloud managers and they are able to establish a two-way communication with any node in the Fog Layer.

In our implementation, we used the Scaleway Cloud computing platform [81] because of its low cost, ARM processor support and stability. Any other alternative platform is suitable for the FOGHA architecture because our applications and services are available as Docker containers. The only requirement is the availability of ARM processor support and any distribution of Linux for the Cloud servers, since we wanted to use the same software stack in both Fog and Cloud Layers for simplicity and manageability.

4.4.2. Software Stack of Cloud Managers

In the Cloud Layer, we also used Docker containers in order to benefit from a modular software design and isolation. The base containers are exactly the same with the Fog managers: Node-Red, MQTT Broker, Redis, InfluxDB and Chronograf. Architecturally, we can make the Cloud Layer as distributed but we used only one server for simplicity in the implementation. Since we use a Cloud server which contains ARM processors and Debian Jessie 8.2 OS, all of our containers in the FOGHA system can be deployed to any node which is working as a server.

All the calculations, decision mechanisms, data collection and analysis are performed inside the Node-Red container. This container is not the same as the Fog managers' Node-Red container, thus we published it to the same Docker Hub repository with a different tag namely the Cloud. For example, communication via BLE is not required for the Cloud nodes so we did not configure this node as compatible with BLE devices. Additionally, all of the software logic is completely different from the Fog nodes.

Redis is mainly used for the fast data access and caching purposes. On the contrary, InfluxDB is responsible for the long term storage of any kind of data. The collected data in the Fog Layer firstly reaches to a gateway Fog manager and is immediately published to the Cloud Layer via the MQTT protocol. Thus, an MQTT broker which works in a Docker container is a must for Cloud managers.

4.4.3. Fog - Cloud Layer Interaction

As explained before, the MQTT protocol is used for data transmission purposes between Fog managers. We also believe that MQTT that is a lightweight publish/subscribe protocol is very suitable for connecting Fog and Cloud Layers. Since there is a continuous data transmission between these layers, request/response based protocols would not be satisfactory for our system. In addition, the collected data in the Fog Layer is very heterogeneous that can cause compatibility problems. However,

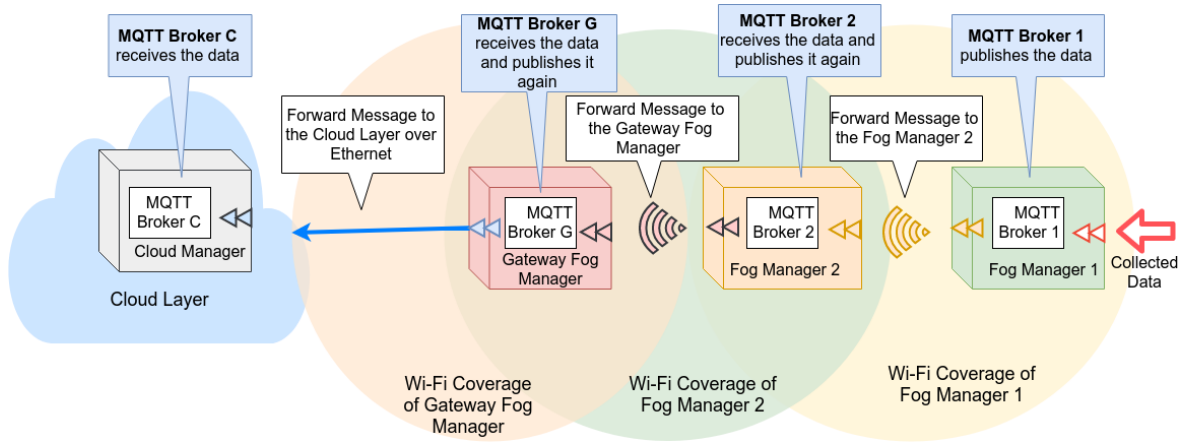


Figure 4.17. The Data Forwarding Mechanism.

only plain text could be sent using the MQTT protocol.

In order to minimize the problems above, Fog managers enforcedly collect data in any form and later convert it to a valid JSON object. If the node is not a gateway node, it tries to convey the data to a Fog gateway node via the multi-hopping mechanism. After a gateway node receives the data, it is turned into a string and published to an MQTT broker which is located in the Cloud Layer by specifying the topic. In this way, the subscriber Cloud node receives the data and converts it to a JSON object. MQTT Broker container of the Cloud node receives the data and forwards it to its Node-Red container. Therefore, the data produced in the Things Layer reaches to the Cloud Layer in a formatted way. A simplified version of this process is shown in Figure 4.17.

4.5. User Interaction

In the previous sections, we mostly explained our architecture and implementation details by dividing it into the layers and modules. Our architecture is implemented by considering security, reliability, availability, and scalability aspects. In addition to those system quality attributes, the interaction between the users and FOGHA system has a crucial importance. Because, a modern smart home should be able to automate

tasks as much as possible in order to decrease user effort. On the other hand, users of the system should be able to control home appliances and provided services via a graphical user interface even they are in a remote location. To achieve these, we implemented two user interfaces as a Web Browser interface and an Android application. To be able to access these interfaces, users should be authenticated via the Cloud manager node which requests a user-name and password. We defined three user roles in the system as parent, child, and assistant because there may be some critical actions which should be performed by a subset of the users.

4.5.1. Web Browser Interface

As we explained before, Node-Red framework presents a UI which is accessible via a Web browser for development purposes. Additionally, Real VNC that is a screen sharing software is used to connect RPI boards. These two options are mostly suitable for development purposes. The system should also be able to present a UI for the end users by easing home automation scenarios. Thus, we integrated the dashboard module of Node-Red framework which enables front-end development via Angular.js and HTML. In that way, we connected the application logic and user interface by providing a Angular.js [82] based Web application which is accessible via port 1880 of RPI boards. For example, a device in a FOGHA included house can be easily switched on/off using a browser even if the users are away from the environment. As an additional example, they can watch their houses through a camera included in the RPI board which is located in the Fog Layer in real time. In the next chapter, the implementation details are presented.

4.5.2. Android Application

Smart phones and applications are gradually becoming a central part of our daily routines. In addition to their processing power capabilities, mobility is the key feature of these devices. Although a smart home resident is able to access FOGHA UI via the browser of a mobile phone, the rich feature set of those devices cannot be utilized in that way. For example, tracking location of a home resident requires a mobile application

which is able to collect location information in the background. In addition, users want to be immediately informed about a critical event related to the home which requires push notifications. After considering such kind of scenarios, we decided to implement a mobile application in order to manage the home environment and continuously collect information related to the user. Implementation details and features of the mobile application will be explained in the following chapter.

4.6. Integrating New Smart Devices and Applications to the FOGHA

Multi-layered, containerized system architecture of the FOGHA enables the integration of a new hardware equipment or software component to the system even if the user is not an experienced software developer. Node-Red framework presents many implemented plug-ins (around 1335) which can be integrated to any home automation scenario with a drag and drop action on the Web browser interface.

For instance, Amazon Echo [83] is a popular home automation equipment which presents a smart and voice-controlled assistant, namely Alexa, to the users. It can be integrated to the FOGHA system by installing “alexa-home-skill” [84] plug-in from the Web interface of the Node-Red framework. In this way, the Fog managers will be able to communicate with the Amazon Echo device and turn on/off a bulb which is connected to another sub-system after the user says “Alexa, turn on the Kitchen Lights”. The only application logic which should be written in here is checking the received command and triggering the other system based on it. A small piece of JavaScript code is the only requirement for that part.

It is a significant advantage that most of the popular IoT devices are currently supported by the FOGHA architecture. However, some IoT devices require additional installations for a proper integration. In order to satisfy those requirements, you may need to connect your Raspberry Pi desktop using Real VNC tool and install some OS packages from the terminal. After the prerequisites are satisfied, the procedure is turned to a simple form like Amazon Echo integration. Most of the Node-Red plug-in developers provide the requirements in the Node-Red website of the plug-in which

decreases the integration complexity of the Things Layer.

In the application part, we also tried to simplify the procedure by dividing the applications to Docker containers. For example, a Docker container which performs the face recognition tasks could be transferred from a remote repository (generally hosted by the Docker Cloud Hub) to the host machine with just one command (i.e `docker pull "arm32v7/chronograf"`). Later, this container could be start with a single command (i.e `docker run "arm32v7/chronograf"`). Since all the dependencies, software packages are included in the container, the user is not required to handle low level implementation details. Our plan is to provide attractive services using this mechanism to the FOGHA enabled houses.

4.7. System Level Evaluation of the FOGHA Architecture

Business level requirements are shaped based on user demands and we tried to implement those by providing some use cases. Since they are subjective elements, we will present service level requirements which are generic for all kinds of software architectures in this section. According to Cade *et al.* [85], there are seven service level requirements which should be addressed while evaluating a system architecture: performance, reliability, security, scalability, availability, manageability, maintainability and extensibility. Additionally, we believe that privacy is another important criterion for this domain. We evaluated the overall FOGHA architecture considering these requirements in the following sections.

4.7.1. Performance

In order to create a robust automation system, performance of the system should be analyzed using proper metrics. When we mention about the performance of the FOGHA system, we should consider Fog Layer managers and FOGHANET as possible bottlenecks. Because, the Cloud Layer nodes could be changed with more powerful servers with a little effort. Our system enables such a change because FOGHA applications are in the form of Docker containers which can be deployed easily to another

Cloud server. In the Performance Evaluation chapter, the overall system performance is presented for different scenarios.

Similarly, the Things Layer may consist of lots of heterogeneous devices but the data generated by them is sent to the Fog Layer for processing or an activation command is again received by Fog Layer nodes. Since FOGHA is a modular system, resource usage and current situation of each RPI board are continuously obtained by the gateway Fog manager and then sent to the Cloud Layer. In that way, some precautions could be taken automatically by inspecting the overall node performance. Additionally, we prepared a GUI for manual investigation which shows this information instantaneously per node. Other than the overall node status, current resource usage of each Docker container which runs inside an RPI board can be monitored via the FOGHA GUI.

4.7.2. Reliability

Reliability is another important requirement for the FOGHA architecture since it contains multiple layers and the data transaction between nodes should rely on reliable communication channels. In order to satisfy reliability, MQTT pub/sub protocol is used for inter-manager communication including FOGHANET and Fog-Cloud connection. It is built on top of the TCP protocol and provides three levels of QoS. We selected QoS level 2 which ensures that each message reach the receiver exactly once. In addition, it is a lightweight protocol and commonly used in IoT scenarios.

On the other hand, there could be other nodes between a sender and receiver manager in FOGHANET which perform data routing similar to multi-hopping techniques. A faulty node or link in this wireless ad-hoc network may affect the system reliability adversely. Thus, we propose that every node in FOGHANET should at least two connections to the other nodes excluding terminal nodes. In this way, there will be always an alternative route between a source and destination. Additionally, we are aware of that using only one Fog gateway manager is not appropriate because it creates a single point of failure risk. To increase the reliability, we suggest an additional

gateway manager which will forward data to Cloud Layer over the Internet. A home router which connects the gateway Fog manager to the Internet also have the same potential risk. Thus, instead of connecting the second gateway manager to the router, using 3G/GPRS modules in RPI boards could be an excellent alternative for increasing the system reliability.

4.7.3. Scalability

In the design phase of the FOGHA architecture, we tried to create a scalable system in both hardware and software level. We selected RPI boards in order to perform automation scenarios in the Fog Layer which contributes to create a scalable architecture because of their cost efficiency and small size. In order to cover a large physical area, increasing the number of RPI boards will be a possible solution. On the other hand, FOGHANET is an ad-hoc network and the nodes do not need to connect to a central router which even enables new node deployments in real time.

In the software part, scalability is also important and our system satisfies this requirement via Docker containers. Docker Swarm works as a container management tool which enables the scaling of containers over FOGHANET. For example, an application which is packaged as a Docker container could be deployed to multiple nodes as a service. Docker Swarm also enables differentiation of hosts by giving flags for each one. In that way, a subset of nodes could be responsible for running certain kinds of services.

4.7.4. Security

In this section, we will describe security mechanisms of the FOGHA architecture. Popularity of IoT devices also leads to potential security vulnerabilities. Because those devices are mostly lack of enough resources to implement security services. However, RPI boards have enough power to operate these mechanisms on top of modern operating systems like Raspbian. Since the Things Layer has a wide range of heterogeneity, the security mechanisms should be constructed for each device type which is not in the

scope of this thesis.

FOGHANET is a wireless ad-hoc network which drops any IP based data packet coming from an unknown node. In order to join FOGHANET, each host should have SSID (Service Set Identifier) and password of this network and FOGHA nodes have this information by default. In order to make it a more secure network, the password could automatically be changed by the Cloud Layer after sending the credentials to all FOGHANET nodes over the gateway Fog manager. Additionally, we restricted the Cloud-Fog interaction in a way that any packet which is coming from a non-FOGHA node is dropped.

In addition to FOGHANET security, UI applications (accessible via a browser and FOGHA Android application) can only be accessed by entering a username and password. The same automatic password change logic could also be implemented for the UI part. Additionally, a similar username-password protection which is provided by the MQTT protocol is utilized to make inter-manager communication more secure. Lastly, the OpenSSL [86] tool was used to create an SSL certificate to provide an encrypted communication channel for both HTTP and MQTT protocol in any part of the FOGHA architecture.

4.7.5. Availability

Availability guarantees that a service is always accessible even if some of the components fail. It is close to the reliability, but a service could be active in a failure case with the help of redundant components. As we explained in the reliability part, we suggest the usage of additional gateway Fog managers and adjusting positions of each Fog manager in a way that each node is able to communicate with at least two Fog managers. In this way, we tried to maximize the availability by maintaining services and automatically reconnecting Things Layer devices to another Fog manager if possible.

4.7.6. Manageability

The status of Fog managers is continuously monitored by the Cloud Layer which is able to rearrange Fog Layer nodes. This arrangement should not affect the functionality of the system. In order to satisfy manageability, changing some configuration parameters or job definitions should be enough to sustain a service. For example, the Cloud Layer is able to determine the gateway Fog manager in the FOGHA architecture because the gateway node always sends small messages to the Cloud to inform its status. If Cloud managers does not receive these status messages in a predetermined interval, alternative gateway manager which is connected to the Internet is selected as the new gateway Fog manager and the system resumes its tasks.

4.7.7. Privacy

Privacy is a critical requirement for any kind of software service because people are so sensitive for the cases that cause the violation of privacy. Therefore, people may tend to avoid the storage of personal data in Cloud providers. In the FOGHA system, this tendency could be eliminated by disabling data storage in the Cloud Layer. In this case, mechanisms which are only required to sustain system functions would work and the collected data is analyzed in the Fog Layer. Additionally, sensitive data such as camera frames could only be stored in this layer which requires usage of additional storage devices for RPI boards.

5. FOGHA USE CASES

In the previous sections, we explained the FOGHA architecture and its' components by hiding implementation details. In this part, we will express the features and capabilities of our system and possible use cases. Every modern smart home are required to enable controlling home appliances. This is the fundamental and commonly only skill of traditional home automation implementations. However, we believe that the usage of Fog computing and home automation concepts together would pave the new ways for more complex scenarios. The feature set of a home automation implementation is boundless and depends on the user demands. For this reason, we firstly implemented basic functionalities like turning on/off the devices and later added attractive use cases which will be presented in the following section. In the second section, supported use cases and the other attractive home automation scenarios will be explained.

5.1. Implemented Use Cases

In this section, we will provide our implemented home automation scenarios using the FOGHA system. We tried to diversify the use cases in order to present the capabilities of the architecture. Since the Things Layer can contain thousands of different scenarios and smart devices, we selected a set of scenarios which contain resource demanding/attractive use cases like face recognition using RPI boards additional to the common use cases such as turning on/off a device.

5.1.1. Controlling Home Appliances via GUI

A wide range of smart devices are available for different purposes. Most of them support common technologies and communication interfaces. However, only few of them provide well-defined APIs and SDKs for a proper integration with other smart appliances. Instead of adding such commercially available controllable devices into our system, we preferred to create a smart plug which is able to switch on/off any

kind of electronic devices through a Photon micro-controller [87], four relays and the cables. The photon micro-controller supports communication via Wi-Fi which makes it a suitable choice for our system. It receives the command from a RPI board using the MQTT protocol and outputs either 3.3 V (high) or 0 V (low) to its GPIO PINs. Based on that value and PIN information, it activates or deactivates one of four relays. In that way, any electrical equipment can be activated or deactivated even remotely.

Any kind of trigger logic and data analysis results which operate inside the RPI board can be used for switching on/off the devices. Since our system presents a nice GUI to the users via the Node-Red Framework, complex scenarios can be easily adopted. For example, users may enable to automatically turn on an air-conditioner if the temperature is above/below a user defined threshold value. Another example could be turning off a set of electronic equipments if all of home residents are away from the house that is determined via user location information received from the FOGHA mobile application. The developed smart plug is shown in Figure 5.1.



Figure 5.1. The Developed Smart Plug.

5.1.2. Disaster Monitoring and Reporting

It is clear that a smart home should be able to take the required actions in disaster cases to prevent adverse consequences. For this purpose, we integrated the SenseHAT [88] add-on board to the Fog managers over the GPIO pins of RPI boards. The required software packages were installed in the Node-Red container to communicate with the SenseHAT. The Node-Red container continuously receives the data from the SenseHAT which publishes the temperature, pressure, humidity, gyroscope, and accelerometer sensor data. The data is presented to the users via a dashboard UI which enables the monitoring of disaster cases.

The sensed temperature and humidity data are variable whilst the gyroscope and accelerometer values are nearly static unless we change the positions of the RPI boards. A sudden increase in the sensed temperature data could be the reason for a fire case. Since our implementation enables the users to add various trigger rules, exceeding a predetermined temperature threshold can be sent as a message to the users' mobile phones which always listen the MQTT topics of the Cloud and gateway Fog manager. For an earthquake case which is very common in Turkey, gyroscope and accelerometer sensors of SenseHAT would be beneficial to detect the seismic anomalies up to some extend. In order to measure the accuracy of the SenseHAT sensors, we analyzed the sensor data when the RPI board is in a fixed state through 3 minutes (Table 5.1).

Table 5.1. The Analysis of SenseHAT Sensors.

Sensor Property	Standard Deviation	Average	Maximum	Minimum
Gyroscope X	0.00070	0.00779	0.0107	0.0051
Gyroscope Y	0.00064	-0.01566	-0.0139	-0.0198
Gyroscope Z	0.00034	1.00088	1.0028	1.0002
Accelerometer X	0.00126	0.00026	0.0043	-0.0043
Accelerometer Y	0.00110	0.00024	0.0042	-0.0033
Accelerometer Z	0.00084	0.00009	0.0028	-0.0028

According to the observations, we can claim that the error rate of the sensors is in an acceptable range for an earthquake case. Therefore, small changes in the accelerometer and gyroscope data of the SenseHAT which are outside of the error range indicates an anomaly which is shown in Figure 5.2. Such kind of anomalies can continuously be monitored, detected, and reported in the implemented FOGHA system. In addition to the real-time recognition of disasters, detecting the amount of damage in a building is a possible use case. For such a scenario, the distance sensors which continuously monitor the distance between intersecting walls could be deployed to the corners of the ceilings per room. Throughout an earthquake case, the current situation of each room could instantly be observed by the FOGHA and this information could be shared with the remote emergency services automatically. Additionally, the number of affected residents and their locations could also be provided to those services in order to take required actions immediately.

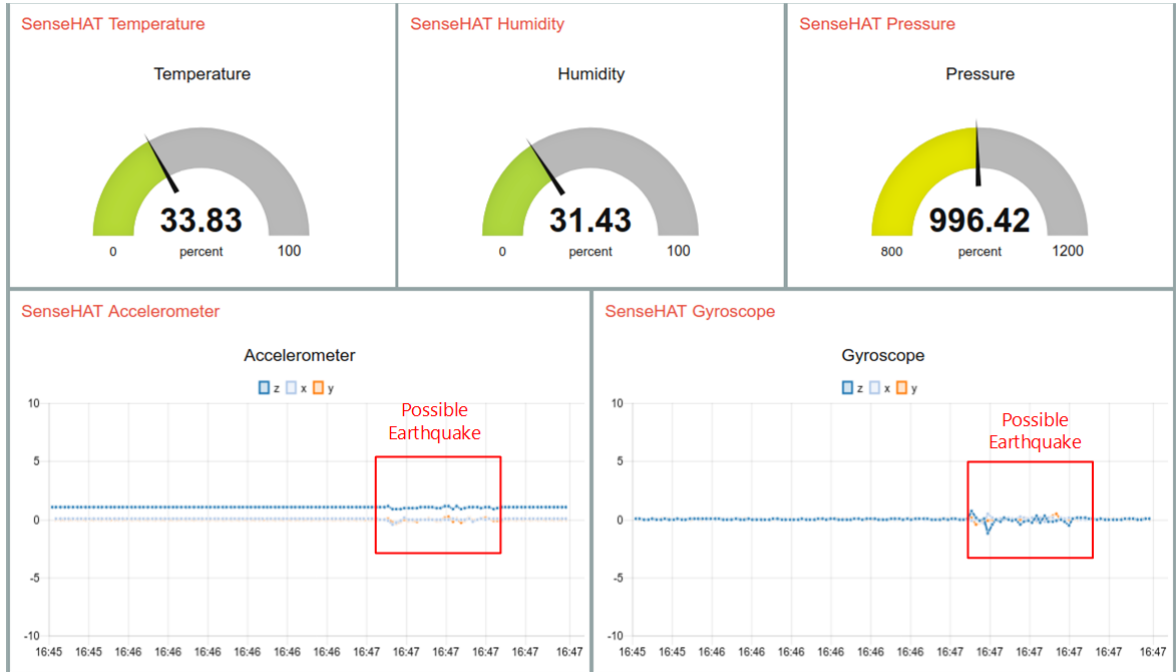


Figure 5.2. Disaster Monitoring with the FOGHA.

5.1.3. Location Based User Tracking

Smart phones provide many services which are beneficial for home automation scenarios. Since each person has at least one smart phone nowadays, various kinds of data related to the user can be collected and analyzed. Recent popularity of Wearable devices like smart watches and wristbands also contributes to this trend. For example, elderly people who suffer from a disease can be tracked without any user interaction via such devices. Since each illness has separate features and requires different data analysis mechanisms, we implemented location tracking as an alternative scenario.

For this purpose, we added a location service in our application which starts as a startup process and receives location updates from the GPS (Global Positioning System) provider at predefined intervals. The location collector service receives position updates when a user location changes more than 100 meters or the interval time (configured as 3 minutes) is exceeded after the last location update. Later, the data is sent to the Cloud Layer in a background service to visualize home residents on a map and trigger other functionalities. The location visualization part is shown in Figure 5.3.

This service could be beneficial for various location based scenarios. For example, an air-conditioner (or any other service/device) could be automatically activated if a resident approaches to the house. Additionally, users can define the frequently visited places inside the map. In this way, a notification message could automatically be sent to the smart-phones of all residents if a resident enters to or exists from a specific region. We believe that those kinds of automation logics increase the quality of life and decrease the amount of time which is spent for the daily routines.

5.1.4. Face Recognition in Fog Layer

In order to measure the system performance, usage of resource demanding tasks are required. Thus, we selected to implement a face recognition application at the Fog Layer using RPI boards, RPI Camera v1, Python 2.7 and OpenCV 3.1 library [89]. OpenCV is a popular and powerful tool for image processing purposes and it has Python

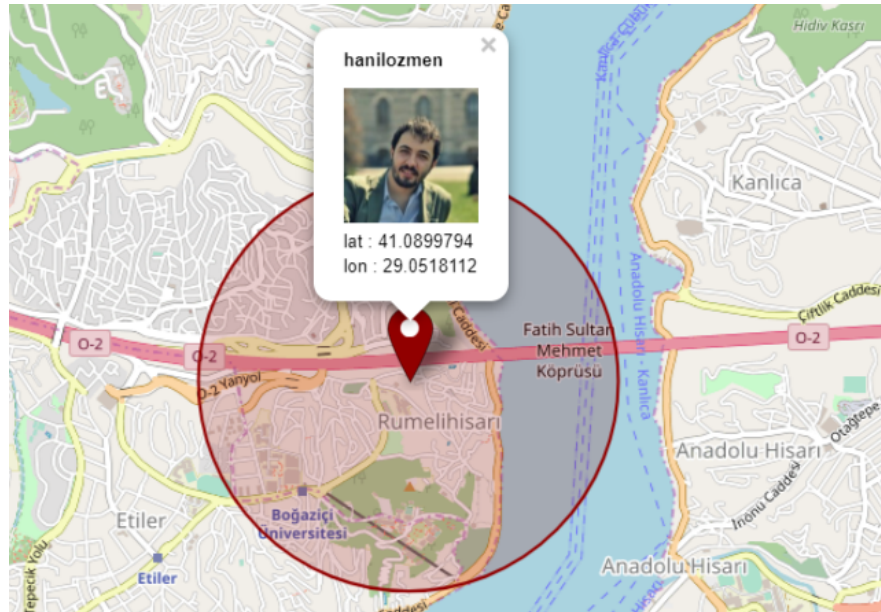


Figure 5.3. User Location Tracking Feature.

bindings for the easy application development. It also has a specific module for face recognition purposes. The face recognition application which is written in Python language captures image frames and processes these frames in order to recognize a person who is in the range of the RPI camera. An example screen-shot is shown in Figure 5.4.

Frames are captured and processed continuously using OpenCV and shown to the user using GUI skills of OpenCV. Training of the face images is the first step that converts the images of home residents to a dataset file by separating users based on their IDs. As a face recognizer, Local Binary Patterns Histograms (LBPH) provided by OpenCV is used for training images. This procedure is done only once and produces a “yaml” file for the recognition step if the training image set does not change. Detection of a face is the second step and the recognition procedure is performed on the detected face area.

To summarize, our system enables real time face recognition using a video stream generated by the RPI camera. There are a wide range of algorithms and tools for

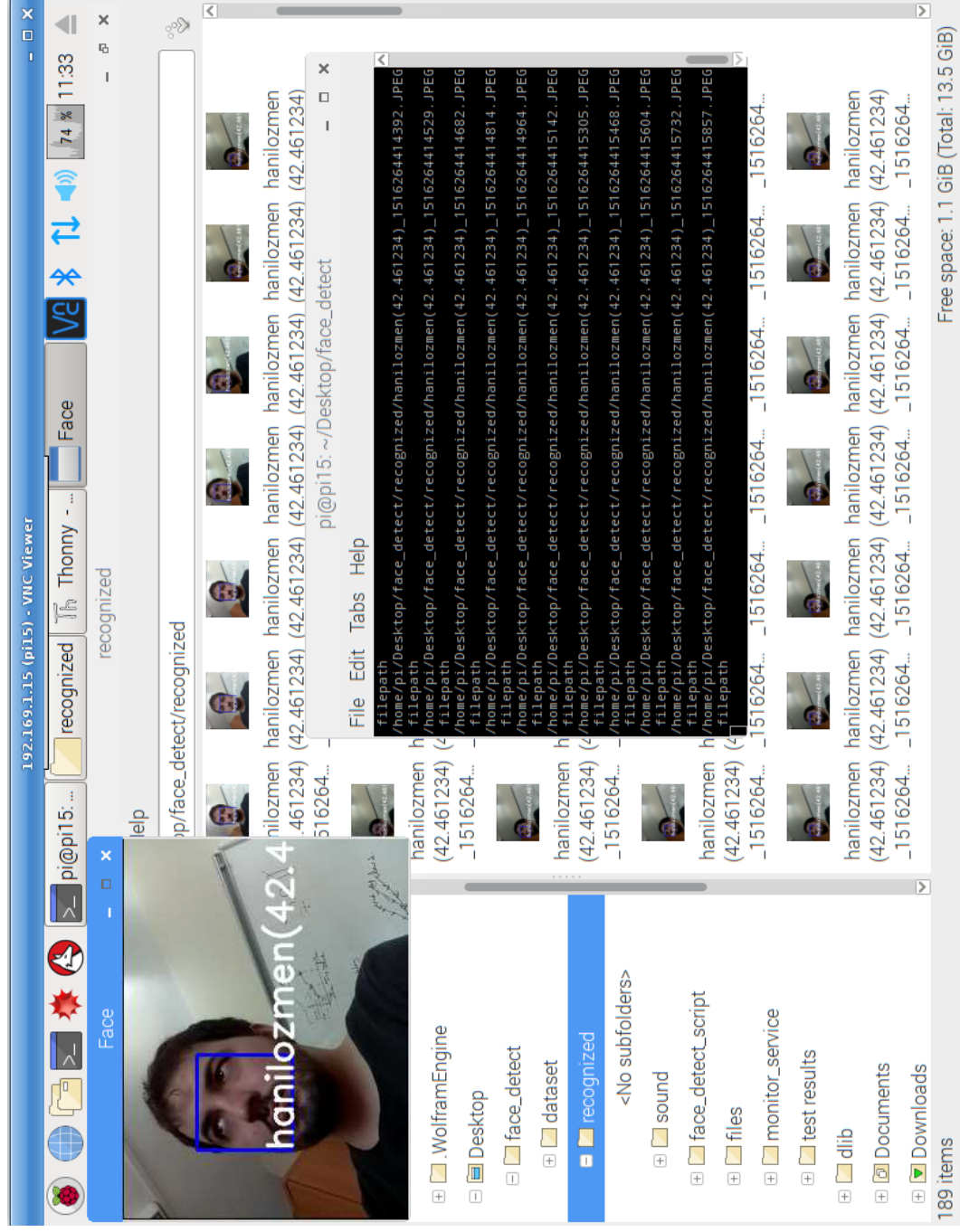


Figure 5.4. Face Recognition with OpenCV.

this kind of tasks in the literature with better recognition results. However, the face recognition task is out of the scope of this thesis and we implemented the algorithm in Figure 5.5 using Python and OpenCV in order to prove that resource demanding and latency critical tasks could be handled in the Fog Layer. Since installation and building time of these tools take a considerable amount of time, a Docker image for the face recognition task is created with required installation commands which makes it easily deployable to any node in the system. In the next chapter, performance evaluation of the face recognition procedure will be presented to show the capabilities of the FOGHA architecture.

5.1.5. Video Streaming in Fog Layer

Nearly every smart home system provides visual services to the customers such as remotely monitoring the home environment via cameras. It is a real requirement for a modern smart home that attracted us to implement a camera enabled home surveillance application. In addition, it provides valuable information about the Fog Layer in terms of bandwidth and computational resource usages.

We have created a Python server for this purpose that runs automatically after a camera integrated RPI boot-up. In our implementation, we adjusted the resolution of the camera as 640x480 pixels and frame-rate as 24 per second. The application reads the video stream provided by the RPI camera and starts publishing the data over TCP port 8080 until it is stopped externally. The UI part is completely written with HTML which shows the stream to clients utilizing sockets. The data is published in Motion JPEG (MJPEG) format using “Socket Server”, a Python package, because of its lightness and bandwidth efficiency. In that way, a resident can continuously observe the house using any kind of Web browser in real time. In addition to the browser interface, we added this functionality to the FOGHA mobile application.

```

face_cascade ← read openCV cascade(lbpcascade_frontalface_improved.xml)
video_stream ← start video stream
face_recognizer ← create face recognizer(LBPH) and load training images
while true do
    frame ← loadFrame(video_stream)
    gray_frame ← convertToGray(frame)
    equ_frame ← histogramEqualization(gray_frame)
    faces ← openCVDetectMultiscale(equ_frame)
    if length(faces) > 0 then
        faceRecognition(faces, equ_frame, frame)
    end if
end while

function faceRecognition(faces, equ_frame, img)
for x_pos, y_pos, width, height in faces do
    printRectangleToGUI(img, x_pos, y_pos, width, height)
    predictor ← initialize openCVMinDistancePredictor
    face_area ← subFrame(x_pos, y_pos, width, height)
    result ← openCVPrediction(face_recognizer, face_area)
    ID, distance ← getLabelAndDistance(result)
    if distance < THRESHOLD then
        printIDAndDistanceToGUI(ID, distance)
        return name, ID
    else
        return UNKNOWN
    end if
end for
end function

```

Figure 5.5. Face Recognition Algorithm.

5.1.6. Chat-bot and Natural Language Understanding

Actuating home appliances are generally performed via some predefined buttons, switches, drop-downs using a Web application. This is the traditional way to control them which is included also in our implementation via the Node-Red Dashboard. Nowadays, people are demanding more attractive services which eases any kind of daily routine. Thus, we believe that users should be able to manage their environments in a more innovative way. In Figure 5.6, our FOGHA chat-bot is shown with using example inputs.

Chat-bots are gaining popularity day by day and they are very suitable for the management and triggering of automation tasks. Thus, we implemented a chat-bot in our FOGHA Android application to control home appliances. Instead of turning a

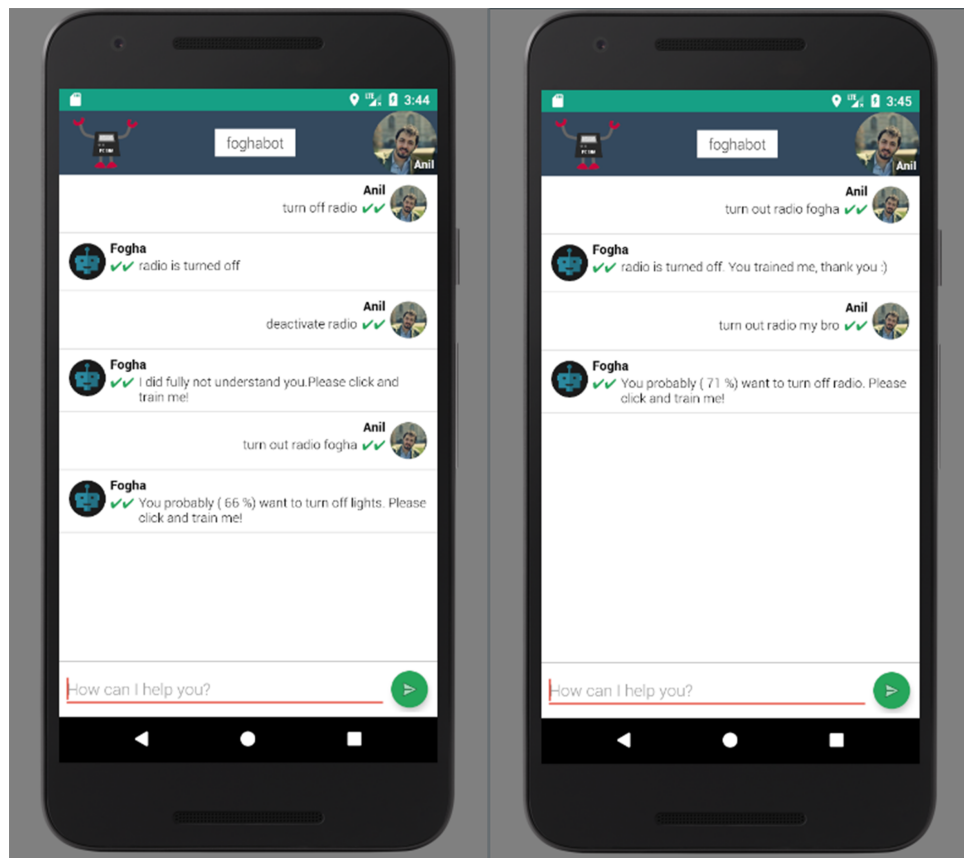


Figure 5.6. Chatbot Implementation and Training of the Foghabot.

switch from off to on state, users are able to write a sentence to the chat-bot such as “turn on lights in kitchen”. As an alternative to writing such a sentence, our application also provides a speech input inside chat-bot which is then automatically converted to the text. Naturally, the same request could be written or spoken in different ways which requires an intelligent mechanism to analyze sentences. Dividing sentences into the words and finding synonyms of them have a crucial importance for this analysis.

For example, a user may say “turn lights on” while another person request the same action with “activate lights”. At this point, the value of synonyms comes into existence. In our implementation, a similarity score is calculated by comparing the similarity of words and their synonyms in a sentence that is provided by the user to the chat-bot. The sentence which is written by a chat-bot user is compared with the available sentences which are saved into the database. This comparison is made based on the similarity of tokens of sentences and their synonyms. In order to find the synonyms of a word, WordNet database [90] is used. Since we need this tool in our main logic which is Node-Red and there is not a wrapper for it, we developed and published our Node-Red package publicly [91]. The overall logic of this analysis is illustrated in Figure 5.7.

5.1.7. Real Time Information Exchange

Besides from the pre-configured automation scenarios that takes actions automatically, users of a smart home should be informed about the critical events in order to take a fast action. Most of the similar systems handle this requirement by sending an e-mail or SMS to the users. These types of information services are considered as obsolete solutions since the user cannot take any action directly.

In our proposed implementation, we implemented an MQTT background service in the FOGHA Android application which runs continuously even the phone is rebooted. The MQTT service subscribes to the MQTT brokers that runs as a Docker container at Fog or Cloud Layers based on the location of the smart phone. In that manner, two-way communication is always in an active state if the phone has an Inter-

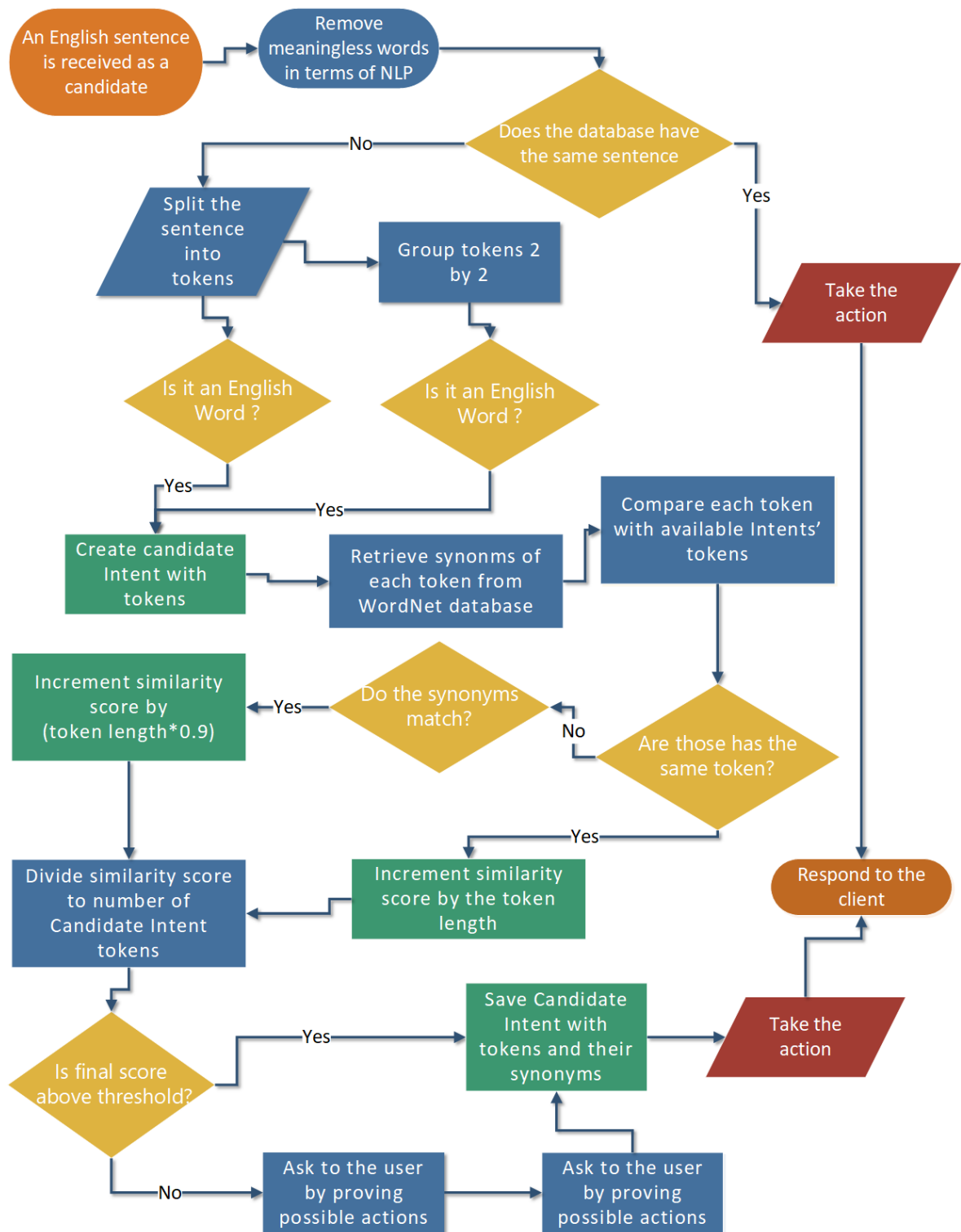


Figure 5.7. Natural Language Understanding Mechanism of FOGHA.

net connection. In our implementation, any kind of event can be sent to the residents' smart phones even if the FOGHA application is completely turned off. After the data is received in the background, it creates a push notification on the smart phone to immediately inform the user. By clicking this notification, users will be directed to the FOGHA application which shortens the time of taking an action.

5.2. Supported Use Cases

In the previous section, we detailed the currently available applications which are provided by the FOGHA architecture. However, a wide range of applications could be developed and easily deployed to the system because of the flexibility and modularity of the FOGHA. Apart from home automation scenarios, many attractive applications and services could be provided to the users in various IoT domains such as health-care, agriculture, smart-vehicles, and industrial monitoring. In the following subsections, we will detail the use cases that are performable by the FOGHA architecture through providing some attractive ideas. It is impossible to cover every use case in the IoT domain and integrate them to the FOGHA, therefore we decided to provide the most prominent IoT scenarios and start-ups presented by the community of IEEE IoT Initiative [92]. We will present the scenarios which are suitable for the FOGHA architecture and discuss whether the FOGHA architecture is able to provide similar services that are provided by these start-ups.

The aging population and the technological improvements to cure illnesses lead to attractive scenarios for the IoT domain. In a scenario [93], an elderly home resident opens the bedroom windows and forgot to close them. The home automation system detects a decrease in the temperature and informs all of the home residents about this situation. Optionally, a distance sensor can be attached to the window to better indicate the reason for the temperature decrease. Such a scenario could easily be implemented by the FOGHA by deploying several temperature sensors (e.g. SensorTag sensors) and connecting them to the Node-Red server of Fog managers via the supported communication channels (e.g. BLE). Therefore, the anomaly is detected in the Fog Computing Layer and this information is transferred to the Cloud Layer which

immediately sends a push notification to the mobile phones of all residents.

Home automation domain is frequently pronounced with the health-care scenarios nowadays because of the aging population. One of them [94] offers “continuous care” concept which helps patients and elderly people with serious diseases by providing health monitoring services. In this way, the frequency of doctor visits and the number of relocations to the nursing homes could be minimized. This can create very positive effects on them along with the economical impacts. In the scenario, a smart device is proposed which enables to monitor the blood pressure, blood glucose level, etc., and send an immediate alert to the related people or health-care organization in a fall case. These types of services are required to be provided in a continuous manner because the consequences could be dramatic. The FOGHA architecture is very suitable to collect the data from such devices with the help of the provided communication technologies. The continuous data stream could be analyzed in the Fog managers without the excessive use of the network bandwidth.

In another scenario [95], a small village in the countryside invests in a centralized server which enables the citizens to store their data and receive some attractive services. In this case, each house only owns a part of the overall data and has access to that data. This scenario can be considered as a smart city use case which is suitable for the FOGHA architecture. In order to implement it, the Cloud Layer of the FOGHA should have more powerful computers and an authentication mechanism which gives the proper access rights to the each home automation subsystem. In that way, citizens can select the provided services and restrict transfer of the certain types of data in order to protect their privacy. In addition, a FOGHA enabled smart home can automatically communicate with the neighbor smart homes in a disaster case to transfer the critical information to another smart home unit which has the Internet connectivity. To sum up, improvements in the Cloud Layer of the FOGHA can lead to the implementation of connected homes and smart cities.

GreenIQ [96] is an IoT service which targets to decrease the outdoor water consumption in the gardens via a hub. It is able to connect to the Internet via Wi-Fi or 3G.

The system also provides a mobile application in order to manage irrigation scheduling remotely. In addition, the hub receives forecasted weather information from the Internet and make automated decisions accordingly. Such a system could be very beneficial for the houses which have large gardens. However, it is unable to analyze the environmental conditions such as the humidity of soil. This kind of use case would be integrated easily to the FOGHA along with the analysis of environmental conditions. The mechanical equipments which is responsible to turn on/off the irrigation system could be connected to the GPIO pins of the Fog managers. Thus, the same service could be provided with the FOGHA as a subsystem rather than deploying a task-specific equipments.

The number of wearable IoT devices are gaining popularity and needs a closer attention for the smart environments. There are many prototypes and commercially available hardware equipments. FabriXense [97], a unified biometric sensor embedded in fabrics, is one of these wearable ideas which was proposed to enable the measurement of biological parameters like heart beat rhythm, breathing, and blood pressure using a single sensor. However, the data produced by this sensor needs to be collected by a mobile device, ideally a smart phone. FOGHA Android application could be improved for this purpose. However, those kinds of sensors produce data very frequently which requires the transferring them to a server for the storage and analysis. The best candidate for this purpose is the Fog or Cloud managers according to the user location.

Most of the use cases which are applicable for the home automation domain mainly consist of the health-care services. In this section, we provided some of them to illustrate that the FOGHA system is able to provide similar services in both the Cloud and Fog Layers. In addition, container logic used in the FOGHA enables deploying any kind of application to the Fog managers in real time without the need for additional configurations. For example, a new Docker container could be deployed to the Fog managers over the Internet to collect the data produced by a body sensor. The containers can also include some complex algorithms, models, and tools in order to produce beneficial results based on the data.

In addition, the FOGHA could provide many entertainment services like personalized audio-visual content streaming and interactive games. The mood of a resident could be analyzed using cameras or sensors which leads to mood-based services and suggestions. Such services could be deployed as the Docker containers to the Fog managers. Moreover, a FOGHA software market which enables users to download various kinds of applications to their RPI boards could be provided with a GUI similar to the mobile application markets. Apart from the entertainment services, the energy usage of the household goods could be continuously monitored and analyzed for the energy saving. Based on the outcomes, the user may be warned about the situation. Furthermore, some household goods could automatically be activated according to the period of time which has the lowest electricity fee. To sum up, the number of possible scenarios is infinite but the FOGHA architecture has the required flexibility for their implementation and integration.

6. PERFORMANCE EVALUATION

In this chapter, we will analyze our FOGHA implementation by presenting different kinds of performance evaluation results. Since a home automation system consists of various kinds of devices, there is not an easy way to compare it with similar implementations. Instead, we mostly presented the capabilities of our implementation by performing real experiments on the devices, contrary to using simulators. In addition to using benchmark tools, some of the applications which are explained in the previous chapter are used for the evaluation. In the first section, we compared the most popular four home automation frameworks by analyzing their CPU and memory usages. In the second section, we measured the system scalability by increasing the number of MQTT clients which are connected to the RPI boards. In the third section, utilization of the Docker containers which are used in the Fog managers are analyzed. In the third section, we analyzed the effect of using Docker containers instead of running applications directly on the host OS. Network performance of the FOGHA architecture is analyzed in the fifth section. In the sixth section, we selected the face recognition as a resource demanding task and compared the task completion times of the different layers of the FOGHA architecture.

In Figure 6.1, the environment setup which is used for the following experiments is presented. Our main objective was to perform real experiments rather than using simulators and emulators. Thus, we tried to use our FOGHA implementation as much as possible. Normally, a Cloud server (the Scaleway Server 1) which has only 2GB RAM and ARMv7 processor is adequate for the Cloud Layer. However, this server has nearly the same processing power with a RPI board which revealed the need for a more powerful Cloud server (the Scaleway Server 2). Since our system is implemented with the support of Docker containers, we utilized it to deploy the test applications without any additional configuration. In addition, a standard laptop computer is used to perform load tests and generate traffic since it is not meaningful and feasible to buy 100 sensor devices and position them in the Things Layer.

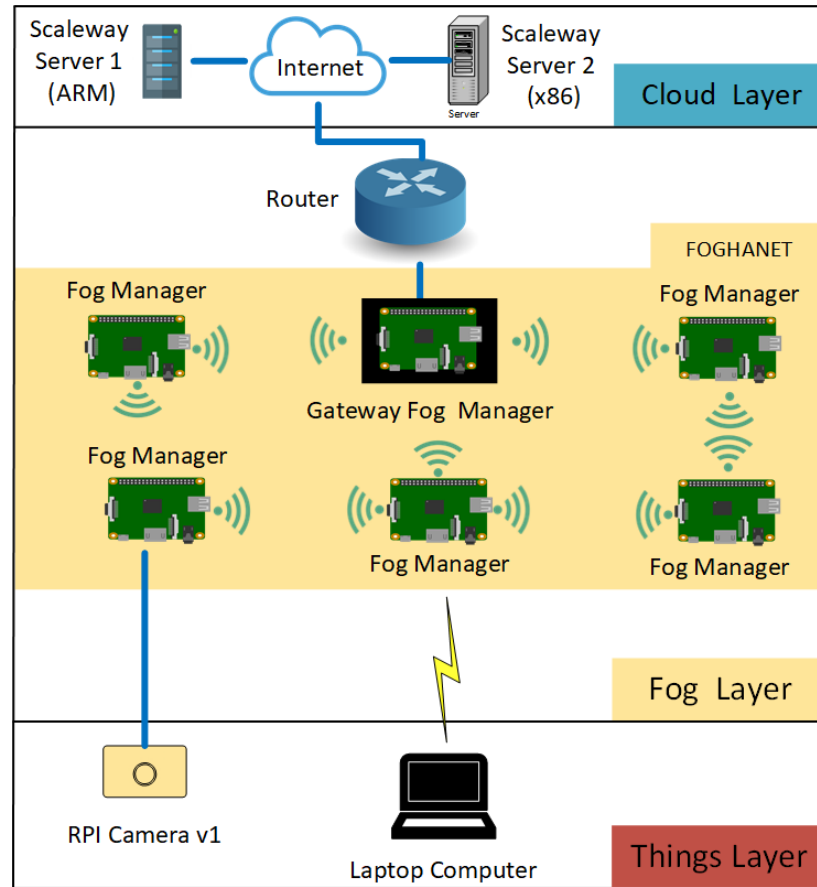


Figure 6.1. The Environment Used in the Performance Evaluation Tests.

6.1. Comparison of Home Automation Frameworks

In the previous sections, we explained why home automation frameworks are required for implementing a modern smart home system. Their functionalities and capabilities are very important and the main reason for our Node-Red selection depends on its' skills. However, their performance and resource usage are the other significant factors. In the beginning of this research, we installed the most popular and dominant ones to RPI boards in order to measure their performance since RPI boards have very limited computational power compared to a standard personal computer. Thus, we compared these tools without installing additional software on them to eliminate subsidiary effects. In Figures 6.2 and 6.3 , their CPU and memory usages are compared. This analysis is very important because they are the main software units of RPI boards

in home automation domain. In this experiment, resource usages of the frameworks were analyzed from the beginning of their startup throughout 150 seconds without running additional computational tasks. We can claim that there is not a significant difference between Home Assistant and Node-Red. However, we believe that the flexibility and adaptability of the Node-Red framework can compensate this disadvantage.

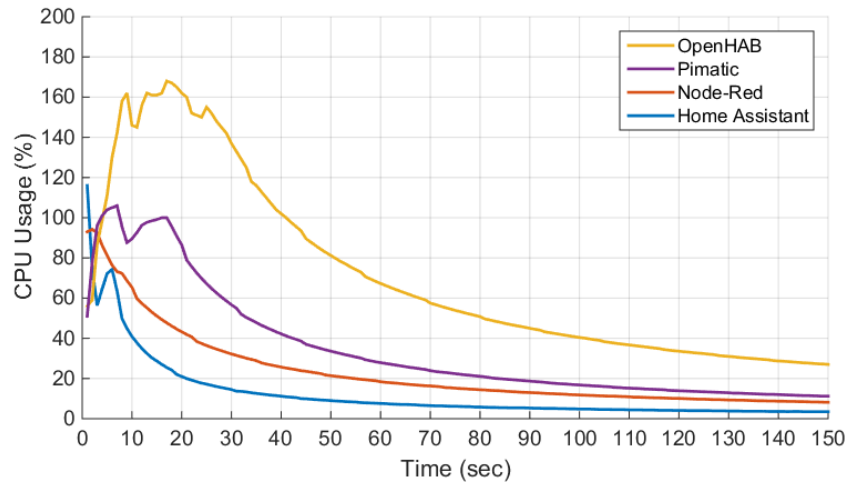


Figure 6.2. CPU Usage of Frameworks

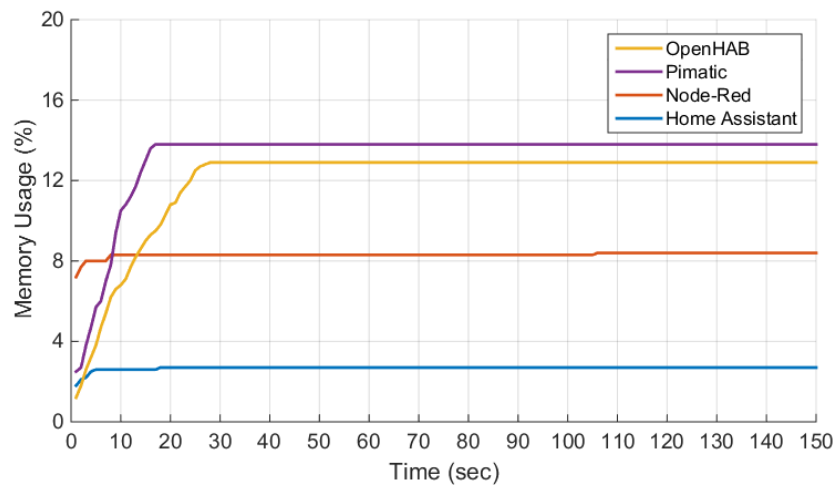


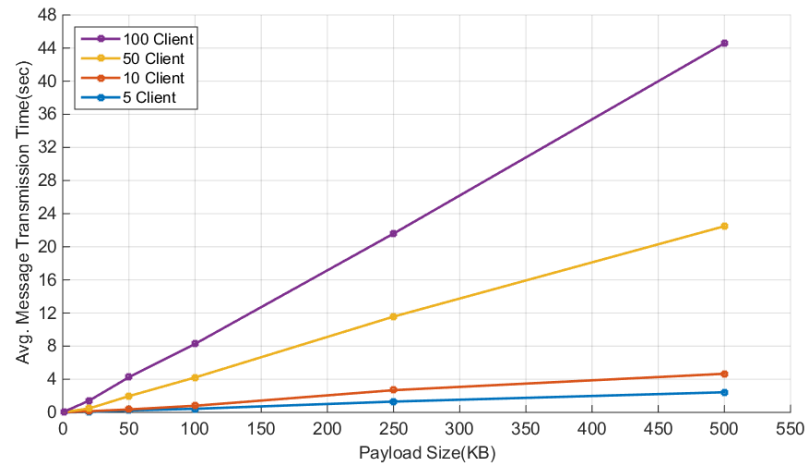
Figure 6.3. Memory Usage of Frameworks.

6.2. Scalability Tests in Fog Layer

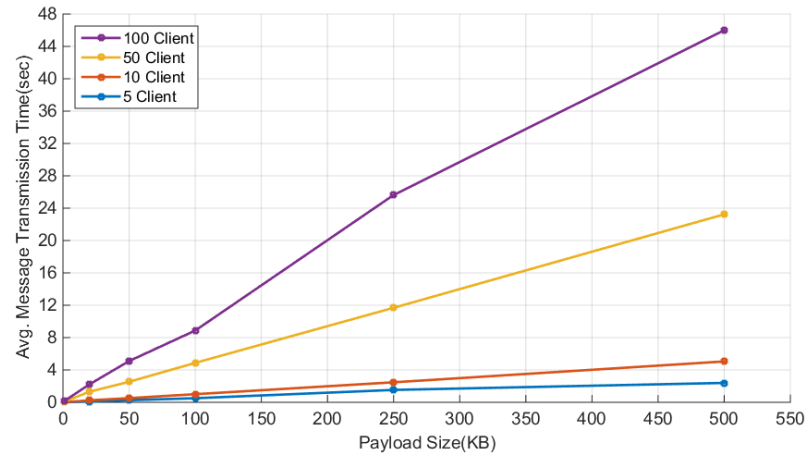
We believe that Fog and Things Layer interaction should be analyzed in terms of scalability, since RPI boards or similar SBCs are not powerful enough to handle a very large number of clients which produce data continuously. For this analysis, we selected the MQTT protocol which is supported commonly in IoT devices and used the MQTT benchmark [98]. Since the MQTT protocol provides three levels of QoS (at most once (0), at least once (1), exactly once (2)), we performed our tests for each of them.

In this analysis, MQTT clients connect an MQTT Broker which runs inside an RPI board and start publishing data. Since using real MQTT supported IoT devices is very expensive for such kind of scalability tests, the clients are created inside a desktop computer using the MQTT benchmark tool. We analyzed the effect of MQTT message body size and number of clients to the message transmission time. The results are shown in Figure 6.4. Based on the results, we can claim that there is not a significant difference between using different QoS levels of the MQTT protocol. Thus, usage of QoS level 2 could be more beneficial in terms of the reliability of data transmission since it guarantees that each message will be delivered to the subscriber host exactly once. In addition, the number of MQTT clients connected to an MQTT broker should not exceed a certain level based on the message payload size. We can also observe that the system shows a robust characteristic for the small sized messages. Based on this observation, we can claim that significant number of smart devices could be managed. However, this robustness probably will not be valid for the scenarios that requires a considerable amount of bandwidth.

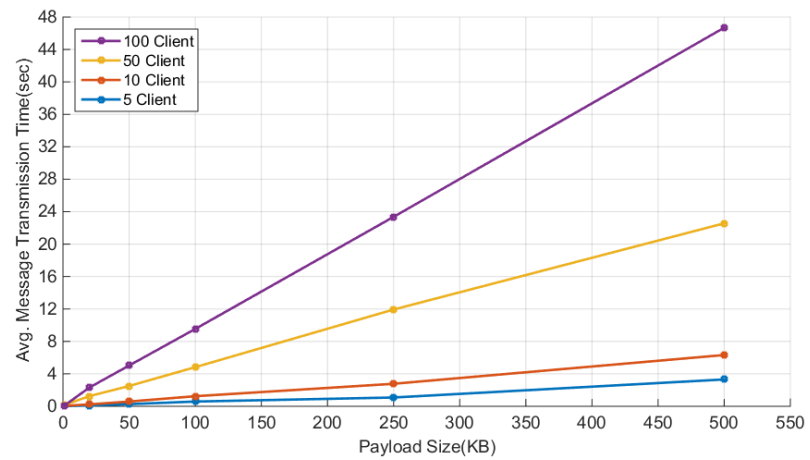
After we analyzed the results which were produced by the MQTT benchmark tool, we decided to perform additional experiments using our FOGHA implementation. Since, it is clear that real experiments which utilize the sources of a system produce more accurate results than benchmark tools. For this purpose, we added various number of MQTT clients to the Node-Red container which runs on the RPI board and analyzed the effect of the number of MQTT clients. Main objective of this experiment is to analyze the interaction between the Things Layer and Fog Layer by mimicking the



(a) Qos 0



(b) Qos 1



(c) Qos 2

Figure 6.4. MQTT Benchmark Test.

smart devices. In this test (Figure 6.5), one of the RPI boards is selected as the data producer node whilst the other one is nominated as the receiver node which sends one byte response after it receives the MQTT message. In order to decrease the subsidiary effects, the other containers and applications were turned off except the Node-Red and MQTT broker containers which are required for this experiment.

QoS level of MQTT protocol is selected as 2 and the payload size of the messages changes in the range of 20 bytes and 312 kilobytes. While we were selecting the range of message size, we considered 20 bytes of data could be a possible payload for a sensor device and 312 kilobytes of data may be a candidate payload for a smart camera. For each payload size, different number of MQTT clients which are shown in the X axis of the figure simultaneously send the data to the receiver node and wait for the response. After the data producer RPI board receives the response, it sends another message to the receiver node with the same payload size and the same process repeats until the end of the experiment. The average values of the response times are grouped based on the number of clients and shown in Figure 6.5. Based on the results, we can claim that the system keeps working in a stable state with smaller MQTT messages (up to

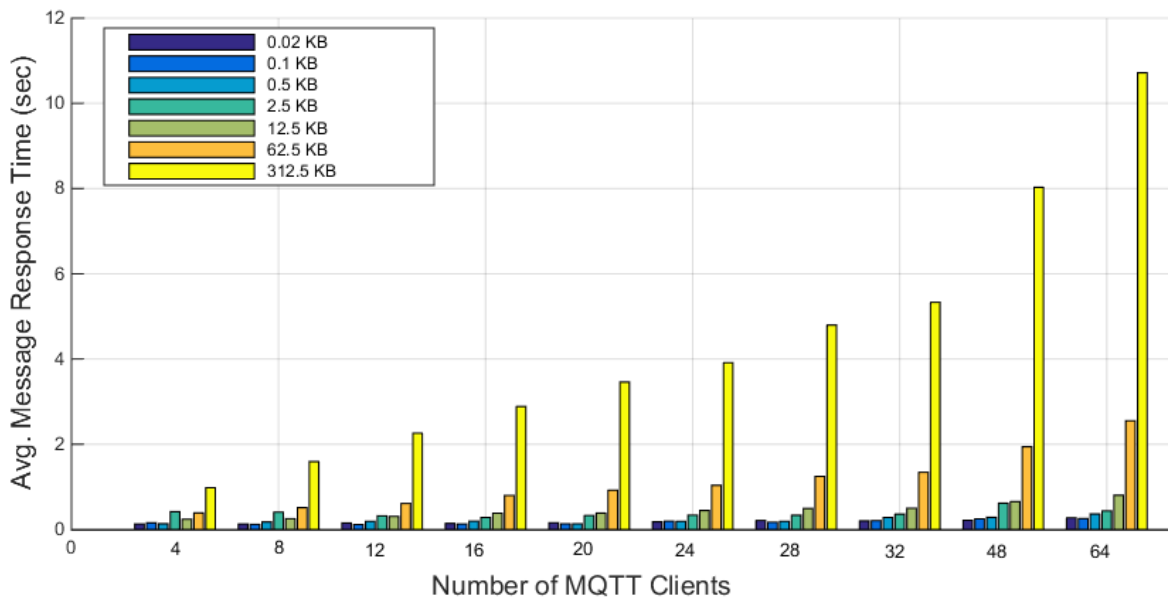


Figure 6.5. The Effect of Number of MQTT Clients.

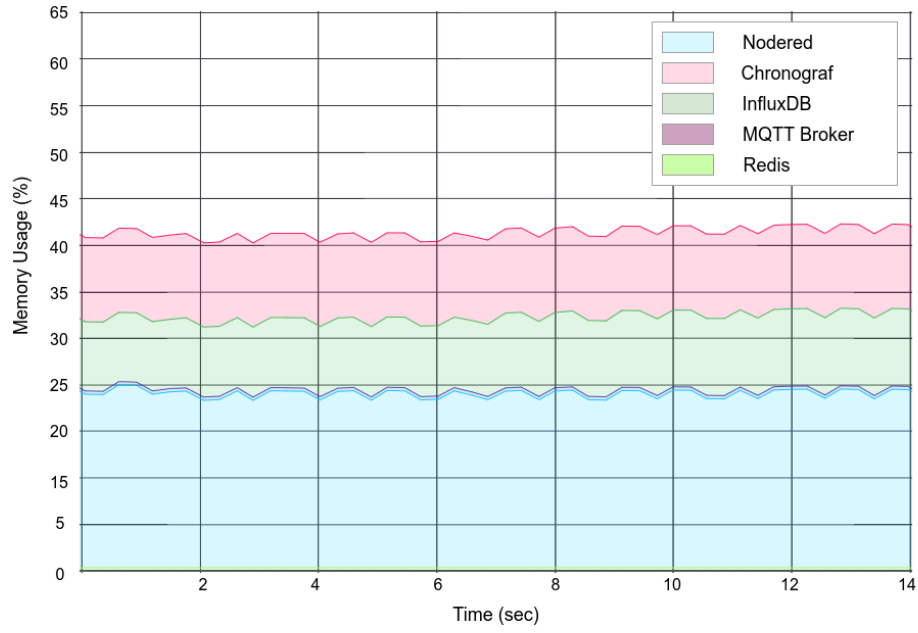
12.5 KB). On the other hand, the increase in the number of clients decrease the system robustness for the larger messages.

6.3. Resource Utilization of Fog Managers

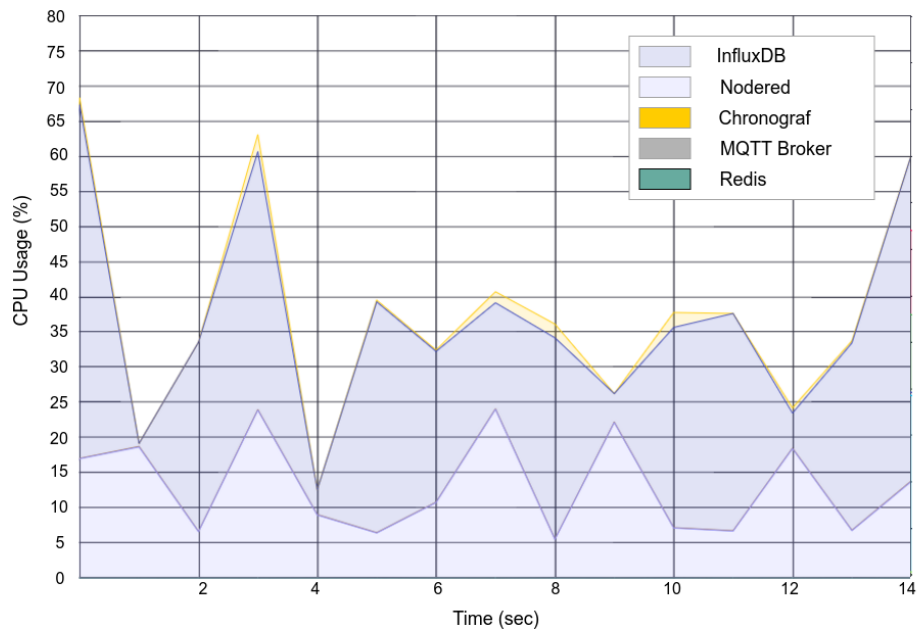
The RPI boards provide us many advantages in order to convert an environment to a “smart environment”. However, their computational power is scarce which requires a continuous monitoring mechanism and properly configured software elements. Therefore, the applications and services which run on them should be lightweight in terms of resource utilization. At this point, the advantages of Docker come into existence since the FOGHA architecture uses containerized versions of the applications like Node-Red, InfluxDB, Chronograf, MQTT Broker, and Redis. The Docker software enables us to stop a container (application) with just a terminal command. Additionally, it provides us the CPU and memory usage statistics of the containers separately via “docker stats” terminal command.

We believe that analysis of each container is required to construct a robust and reliable automation system. Therefore, we wrote a shell script which periodically (every 10 second) collects the CPU and memory usage of each container and directly runs on the host OS. Then, the collected data is sent to the Node-Red container for the analysis. The Node-Red container writes the data to the InfluxDB container with the help of the InfluxDB plug-in of Node-Red. For the data visualization and monitoring part, Chronograf is the target container which continuously consumes the data from the InfluxDB.

In the FOGHA system, the container monitoring can be done continuously via a Web browser which is provided by the Chronograf tool. In order to analyze utilization of each container, we observed one of the Fog managers while the system keeps running as expected (every base Docker container is running) through 15 seconds. We limited the duration of the experiment for simplicity and better data visualization. The experiment duration could be increased but we can claim that the process continues very similarly after a 24 hours observation. The results are shown in Figure 6.6. The distance



(a) Memory Usage of the RPI boards per Container.



(b) CPU Usage of the RPI Boards per Container.

Figure 6.6. CPU Usage of the RPI boards per Container.

between the two consecutive lines expresses the resource usage of that container. In terms of memory usage, we can claim that Node-Red and Chronograf containers are the primary consumers. The resource usage of Redis and MQTT Broker is under 1 percent which is negligible. On the other hand, the InfluxDB container is the most resource demanding element of the FOGHA in terms of CPU usage. The Node-Red container is another CPU demanding unit as expected. The Chronograf, Redis and MQTT Broker containers demand the CPU resources around 1 percent. To sum up, we can propose that the Node-Red and InfluxDB containers are the critical elements of the FOGHA and need continuous monitoring mechanisms.

6.4. Comparison of Docker and Native Implementation

We believe that the applications and services should be deployed as Docker containers as much as possible, especially in the Fog Layer. In that way, a service can be deployed to any node in an easy way without additional software installations. However, it is a common belief that usage of Docker causes a performance decrease compared to native applications which run directly on the host OS. In order to test whether this is a valid hypothesis, we compared these approaches using a popular benchmarking tool, Sysbench [99] and the results are shown in Table 6.1. Based on these results, we can claim that usage of Docker is not considered as a disadvantage in terms of system performance. To summarize creating an application stack which consist of Docker containers is beneficial in terms of both performance and flexibility.

6.5. Network Performance Evaluation

In our implementation, we used seven RPI boards in the Fog Layer by connecting them to the same wireless ad-hoc network. Since they continuously exchange data, measuring the network performance of such a system has a critical importance. Thus, we used a popular network tool named Iperf [100] by selecting one of the nodes as sink and the others as data producers. Iperf tool supports the UDP and TCP protocols for the communication between hosts and we used UDP in the experiments since it produces more detailed reports. By changing the maximum bandwidth parameter per

Table 6.1. Docker Performance Evaluation with Sysbench.

Test Type	Performance Criteria	Native	Docker
Internet Test	Ping (msec)	28,86	29,64
	Download (Mbit/s)	19,85	19,88
	Upload (Mbit/s)	20,48	21,33
CPU Test	Total time (sec), (max prime=5000)	17,53	17,55
	Total time (sec), (max prime=20000)	136,1	142,38
Thread Test	Total time (sec)	11,97	14,26
Memory Test	Operations/sec	1248738	1259454
	MB/sec	1219	1230

client, we are able to see the limits of wireless data transmission. The overall results are presented in Table 6.2. In order to implement the results better, some of the columns need to be explained. The first column indicates that per data producer host tries to send the data with that amount of bandwidth. The second column indicates the total amount of data which is received by the sink node. The total consumed bandwidth for the sink node is shown in the third column while the average used bandwidth per data producer is presented in the forth column. Based on the results, we can claim that the network is in a stable state when the assigned bandwidth per node does not exceed 5 MB. On the other hand, the situation negatively changes when it approaches to 20 MB. In that case, the percentage of lost datagrams and the jitter which are the good indicators for the network stability shows a clear increase. Therefore, the tasks which require high bandwidth need to be performed on the owner host rather than distributing them to the whole network in the FOGHA architecture.

Since the FOGHA architecture proposes geographically distributed servers in the Fog Layer which are connected as wireless nodes, the distance between them is another critical factor for measuring the network stability. The effect of distance between two RPI boards is measured using the Iperf tool and shown in Table 6.3. In this test, two

Table 6.2. Network Tests Using Iperf.

Max BW (MB)	Transferred Data (MB)	BW (Mbps)	BW per Node (Mbps)	Jitter (ms)	Total # of Datagrams	# of Lost Datagrams
100	187.1	51.96	8.66	20.81	1345926	164 (0.012%)
50	199.7	55.05	9.17	19.61	1425343	222 (0.015%)
20	179.8	49.53	8.25	21.1	1282906	217 (0.016%)
10	175.7	48.7	8.11	13.38	1253952	51 (0.004%)
5	107.4	29.8	4.96	5.65	765111	15 (0.0019%)
4	85.8	23.9	3.98	3.68	612312	10 (0.0016%)
2	42.96	12	2	1.15	306214	0 (0%)

RPI boards are used to observe the distance effect by forcing the bandwidth of the data producer node to 100 Mbps. If they are next to each other, the percentage of lost datagrams and jitter are at minimum. However, this kind of Fog manager distribution is not suitable for the FOGHA since it needs to manage all equipments in the home environment. To sum up, the physical distance between two Fog managers is suggested to be between 20 and 30 meters. Therefore, deploying one Fog manager for each room is suggested for a standard implementation.

Table 6.3. Effects of Distance Between Fog Managers.

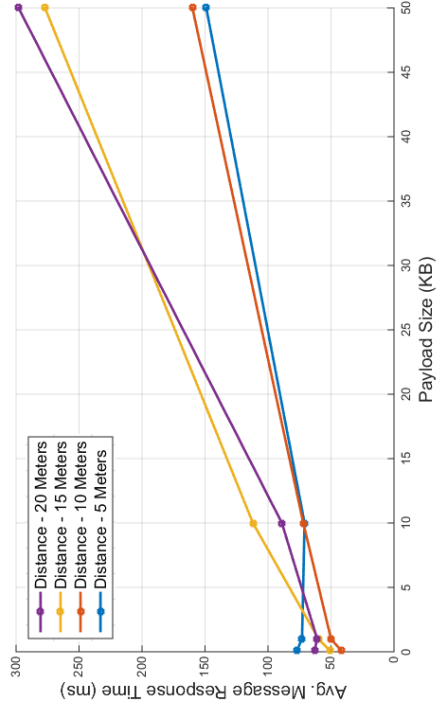
Distance (m)	Transferred Data (MB)	BW (Mbps)	Jitter (ms)	Total # of Datagrams	# of Lost Datagrams
0.1	157.4	43.37	18.84	1122628	29 (0.002)
5	150.3	41.43	18.82	1072223	5158 (0.481)
10	96.6	31.59	19.95	818287	4861 (0.594)
20	13.33	3.66	23.79	96592	1522 (1.575)
30	5.44	1.47	41.23	39116	724 (1.850)

Additional to the experiment which was performed via Iperf tool, we believe that a real test which utilizes the software stack of the FOGHA could produce more accurate results. Therefore, we conducted an additional experiment to measure the effect of distance between two RPI boards. In this measurement, only the Node-Red and MQTT broker containers of the FOGHA are used for two RPI boards similar to the previously provided scalability test (Figure 6.5). In this experiment, we used one of the RPI boards as the MQTT publisher node and another RPI was configured as the MQTT subscriber node which sends one byte success message after it receives a message successfully. Unlike the scalability test, the MQTT messages were sent at certain intervals with various payload sizes. By changing the distance between the RPI boards, we observed the effect of physical distance between them on wireless data transmission and presented it in Figure 6.7. Based on the results, it can be claimed that the effect of distance is not prominent for the small MQTT messages. However, it is so clear that the messages which contain significant amount of data such as 50 kilobytes should not be sent at frequent intervals. This would eventually finish with the non-responsive RPI boards.

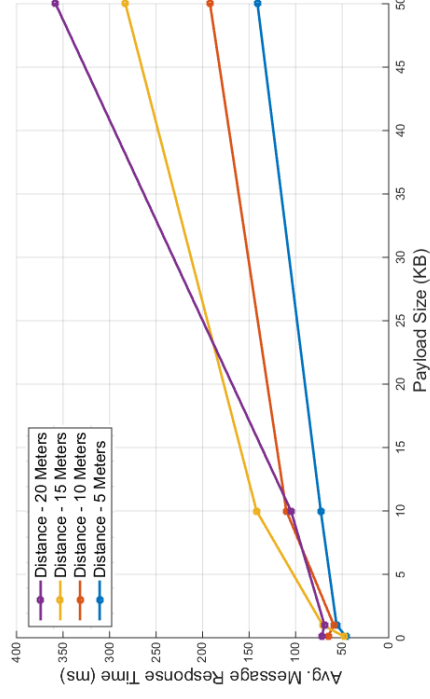
6.6. Face Recognition Tests

Our face recognition implementation is explained in the last chapter and we believe that it is a good indicator to measure system capabilities. Since the face recognition application is in the form of a Docker container, we are able to deploy the same application to the Cloud nodes with minimum effort. This also shows that our architecture is very suitable for any kind of performance evaluation related to Fog and Cloud computing.

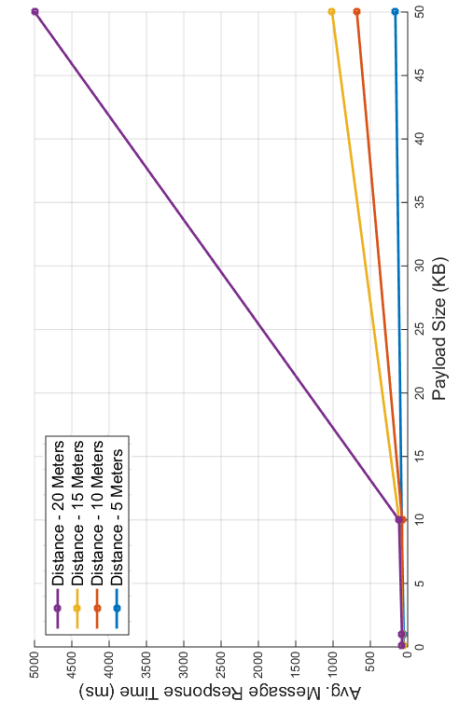
For these tests, we divided the scenarios into four based on the node which performs the task: local, Fog, Cloud arm, and Cloud x86. Local node means that the face recognition application works on the RPI board which has the camera module. The Fog node is another client in FOGHANET which receives the image frames from the other camera integrated RPI. Local and Fog nodes are RPI 3 boards whilst the Cloud nodes are the devices provided by Scaleway Cloud Provider [81] in France. Cloud



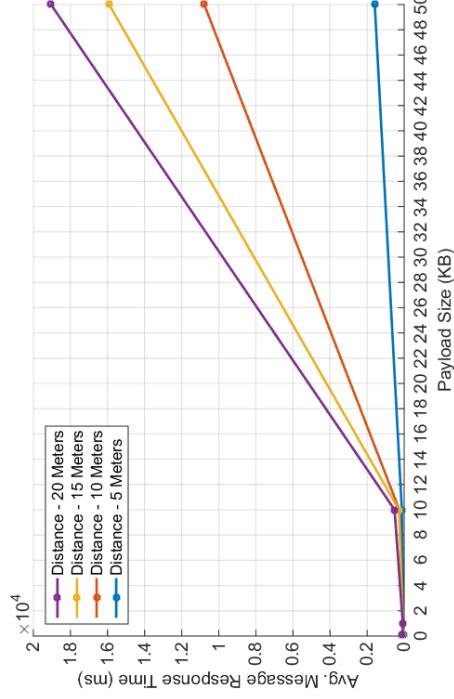
(a) Interarrival Time 1 sec.



(b) Interarrival Time 0.5 sec.



(c) Interarrival Time 0.1 sec.



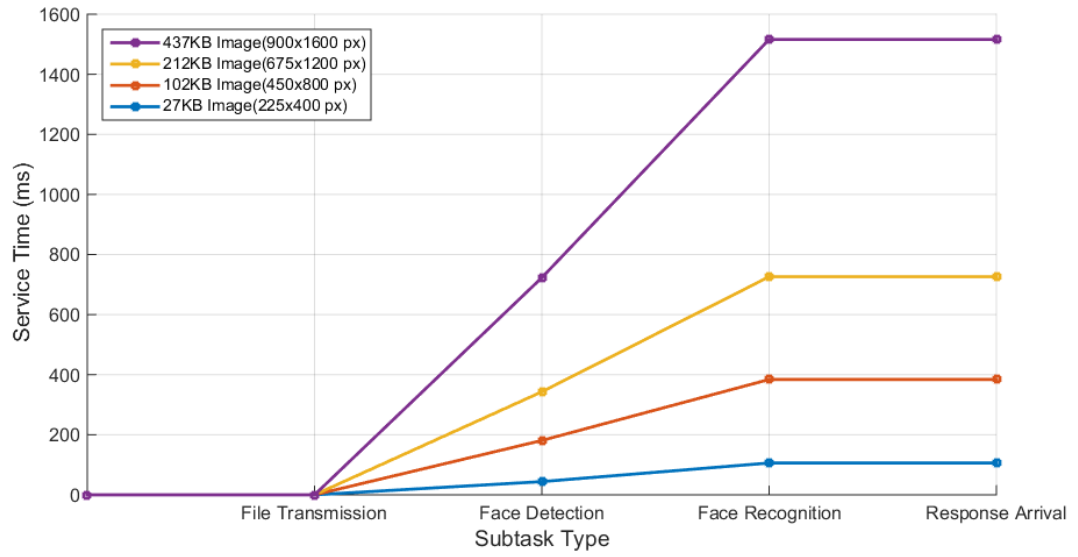
(d) Interarrival Time 0.05 sec.

Figure 6.7. The Effect of Distance for the MQTT messages.

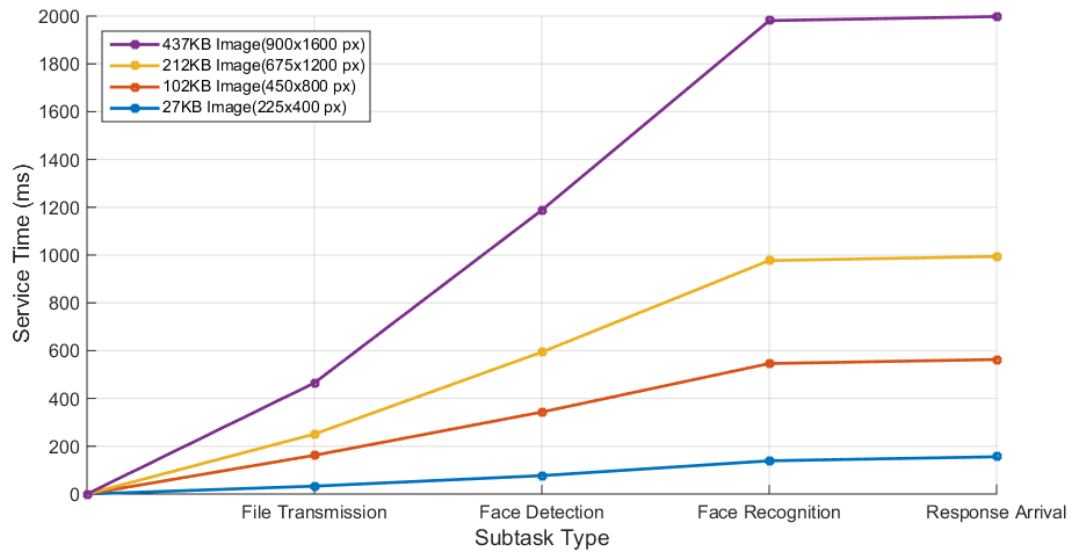
arm node has four 32 bit ARM cores and 2 GB memory which is similar to the RPI hardware. On the other hand, Cloud x86 node has four 64 bit x86 cores and 8 GB memory space. We added the Cloud x86 node into our analysis after we conducted the initial face recognition tests because the Cloud arm machine did not show a significant difference in terms of computation speed compared to the RPI boards.

For the face recognition tests, we captured 30 images for the analysis, and 15 images for the training face images. Those analysis images are compressed to smaller sizes in terms of the number of pixels as 75%, 50%, and 25%. We selected an average image in terms of its size in KB from each category and used four images in total to eliminate side effects. We have written a Python application to send images to the other nodes using TCP sockets. Training of the recognizer is done in the beginning of these experiments so it did not affect the results. The effect of image size is shown in Figures 6.8 and 6.9. We mainly separated the experiment results into two based on the location of the host that performs the face recognition task as the Cloud and the Fog Layer. In this experiment, performing all of the face recognition processes in another device which is different than the camera integrated device corresponds to the task offloading. The camera enabled RPI board sends only the image file to another host and gets the response.

Based on the results, we can claim that offloading a task to another Fog manager provides nearly the same results with the local host because the propagation delay is around a few milliseconds. However, sending the test image to the Cloud Layer and receiving response from it increases the propagation delay based on the distance between the Cloud and Fog Layers. However, the time to perform face detection and recognition tasks in the Cloud Layer is lower than the Fog Layer. We also added a more powerful Cloud manager node since we did not observe a significant difference between the ARM Cloud manager and RPI boards in terms of face detection and recognition time. To sum up, we can claim that offloading the tasks to the Cloud Layer could be beneficial if the task completion time is bigger than the propagation delay.

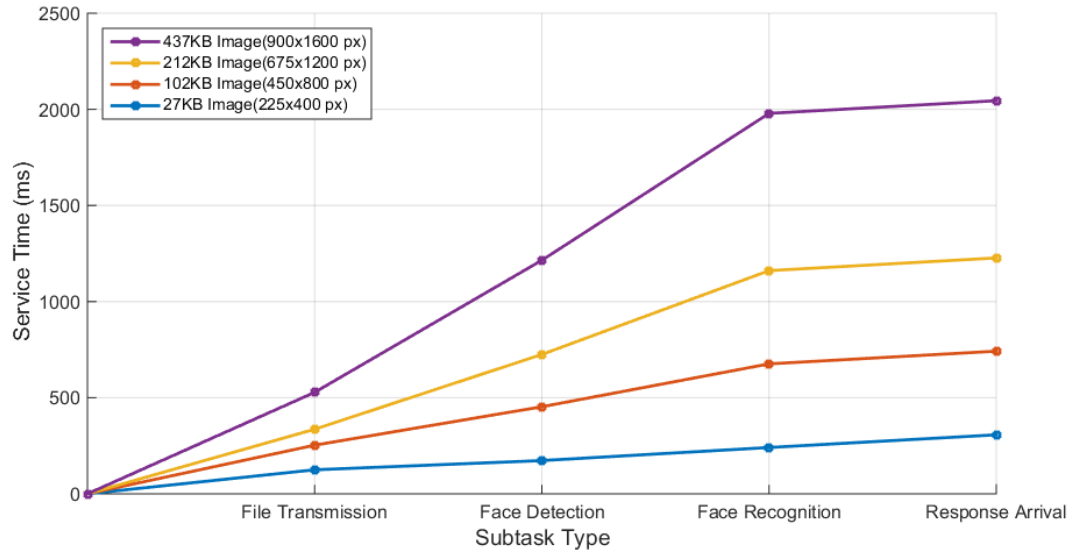


(a) Local Node.

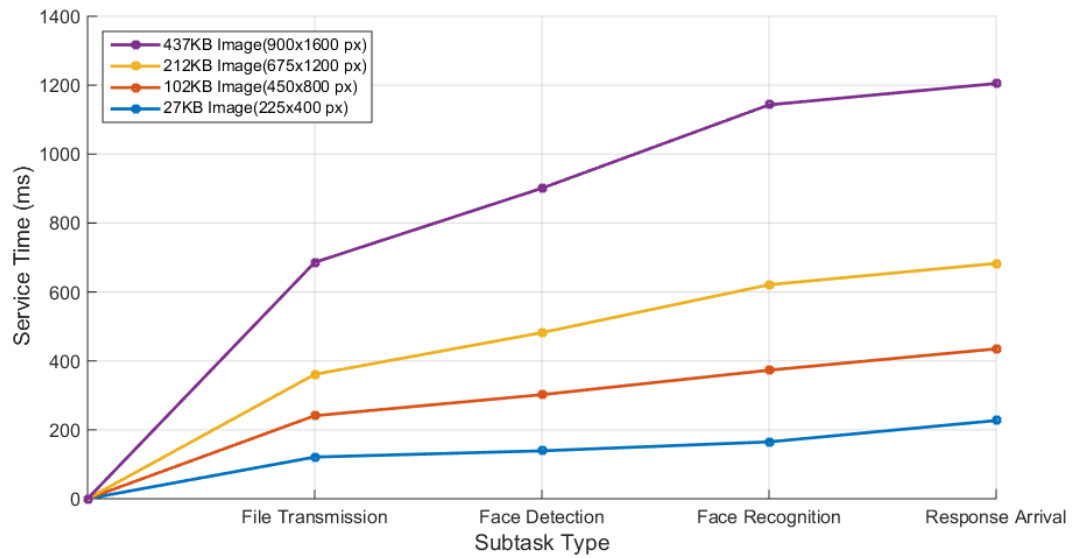


(b) Another Fog Node in FOGHANET.

Figure 6.8. Face Recognition Task Analysis using RPI boards.



(a) Cloud Arm Node.



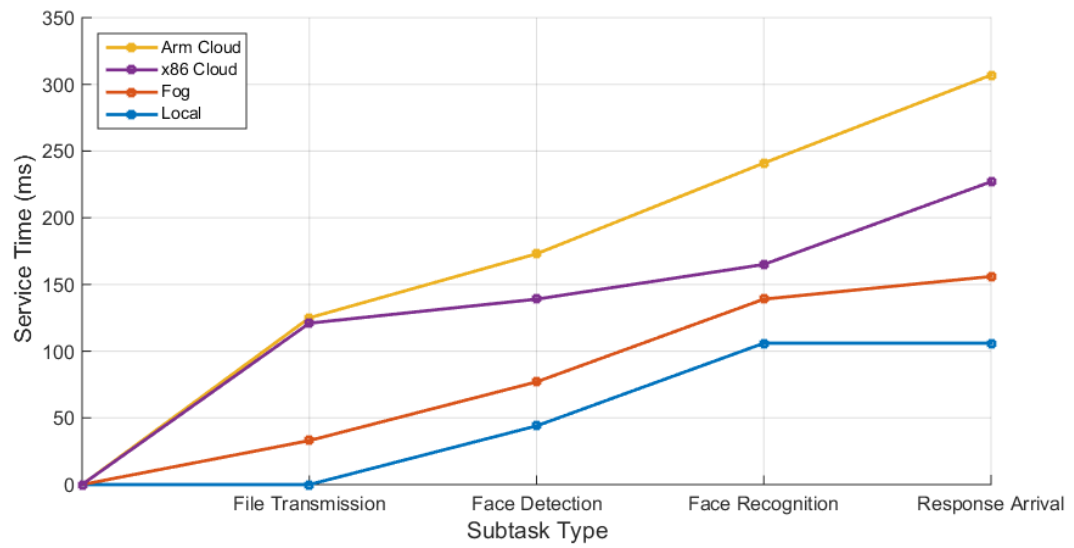
(b) Cloud x86 Node.

Figure 6.9. Face Recognition Task Analysis using Cloud Nodes.

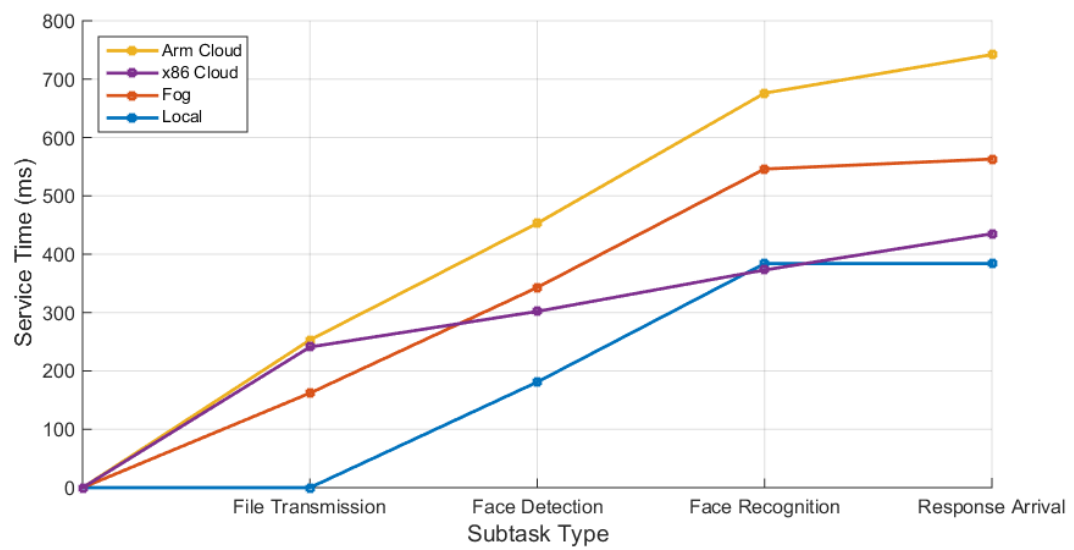
It is clear that performing any kind of task in the local host is advantageous if the host has enough power to perform it. In addition, the distribution of tasks to the different hosts brings additional complexity and propagation delay. However, the geographical distribution of the Fog managers is a critical requirement for the FOGHA architecture which needs the proper utilization of the Fog managers. In addition, the parallelization/offloading of resource demanding tasks like the face recognition is beneficial for the resource limited SBCs such as RPI boards. For example, recognition of face images from a video stream is a resource demanding task which requires the analysis of the video frames. Each frame can be offloaded to a different Fog manager and collected by the camera enabled host. We believe that the effect of node which performs the task is a critical factor for evaluating our proposed architecture that is shown in Figure 6.10 and 6.11. This experiment is divided into four scenarios according to the size of the image file which is used for the face recognition task.

According to the results, we observed that performing the face recognition task in the Fog Layer clearly outperforms the Cloud Layer for the small image files (27 KB). The usage of more powerful Cloud manager (x86) decreases the face detection and recognition times but the propagation delay is the bottleneck for this scenario. It is nearly the same as the overall task completion time of the Fog manager case which indicates the benefits of the Fog computing. To sum up, any kind of computational task whose completion time at local is smaller than the propagation delay is suggested to be performed at the Fog Layer. This situation starts to change for a bigger version of the same image (102 KB). Based on the results presented in Table 6.10b, the overall service time is nearly the same with the local host for the powerful x86 Cloud manager. However, processing the image in the local host is a better choice for this scenario. In case of the real-time face recognition using a video stream, the bandwidth usage could be another disadvantage for the Cloud managers.

For the bigger versions of the test image (Figure 6.11), the x86 Cloud manager outperforms the local and Fog nodes in terms of the overall service time. Because the face detection and recognition subtasks are highly dependent on the size and dimension of the image file. For the bigger image files, the file transmission time is a critical factor

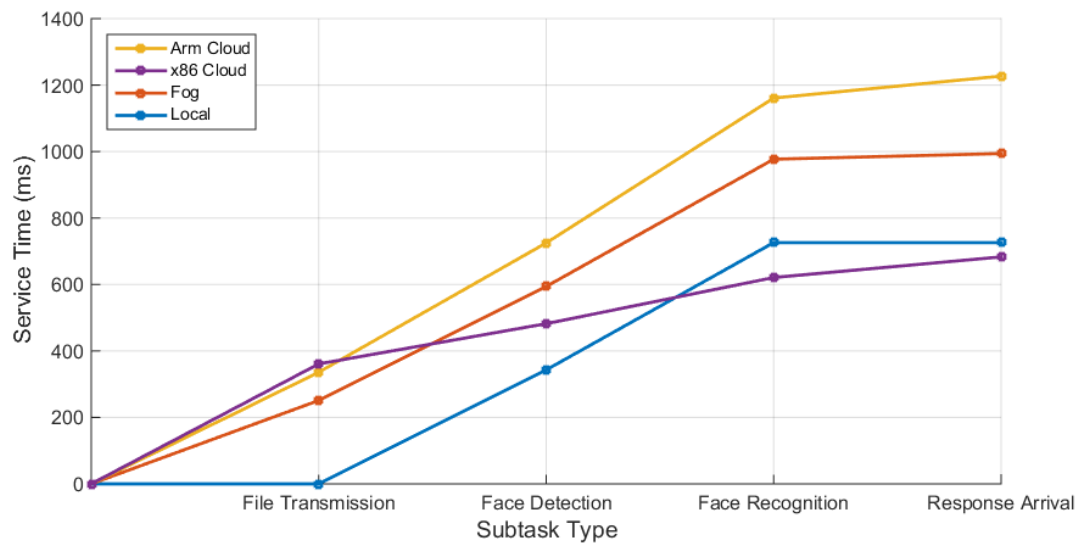


(a) 27 KB (225x400 px) Image.

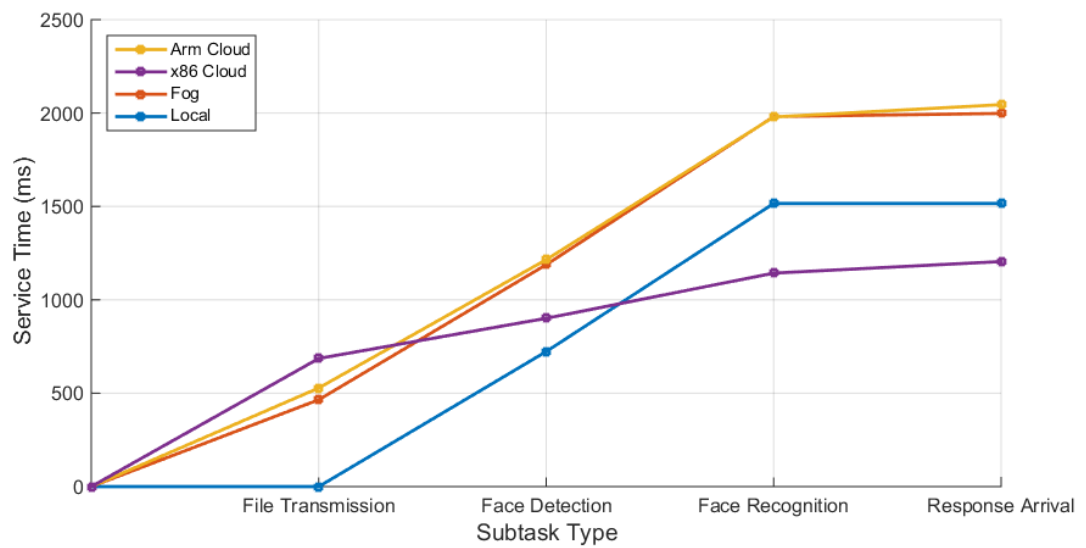


(b) 102 KB (450x800 px) Image.

Figure 6.10. Face Recognition Task Analysis Using Small Sized Image Files.



(a) 212 KB (675x1200 px) Image.



(b) 437 KB (900x1600 px) Image.

Figure 6.11. Face Recognition Task Analysis Using Large Sized Image Files.

and covers a huge percentage of the overall service time. Thus, decreasing the image file size without damaging the quality of it would be a beneficial preprocessing step. In this way, usage of the Fog Layer gains a critical importance for the complex tasks such as image processing. Additionally, the distributed architecture of the FOGHA enables us to perform a task in parallel over multiple hosts. For instance, the face recognition scenario used in this experiment could be extended to a more complex use case such as real-time video analysis. In this case, each frame of the video could consecutively be sent to a different Fog manager for the analysis in FOGHANET that clearly outperforms any other offloading case if the environment has enough Fog managers.

7. CONCLUSION AND FUTURE WORK

In this thesis, we proposed a home automation architecture named FOGHA and detailed the implementation phase of it. In the design phase of it, we mostly considered the system scalability and software modularity. Even if our system targets the home automation domain, we believe that it should be applicable for other IoT domains or physical environments. Additionally, the Fog computing paradigm is very suitable for home automation tasks which are delay sensitive and bandwidth consuming. However, this does not mean that our system solely depends on the Fog computing paradigm. Usage of Cloud computing services or technologies has a critical importance in terms of long term data storage/analysis and resource demanding tasks. Therefore, we adopted these two paradigms by clearly defining their responsibilities.

Most of the research and products related to the home automation lack of proper orchestration mechanisms and constructed for a few specific scenarios. Those kind of systems are not suitable for new integrations and scalability is a big problem for them. To avoid them, we proposed a multi-layer scalable home automation architecture which can also be installed in similar areas, additional to the houses. We used geo-distributed RPI boards for the data processing at the network edge providing minimum network delay and bandwidth usage. The Raspbian OS is used in these boards in order to benefit from its' low resource usage and available Linux software packages. In order to provide a modular software architecture, the Docker technology which actually targets Cloud servers is used in the Fog Layer. In this way, each application that runs on top of Raspbian OS is converted to a Docker container which can be easily deployed to any node in the network. Since they are isolated from each other, a fault in one container does not affect the whole system. We believe that management of these containers needs to be handled by considering the scalability aspect. For this purpose, we utilized the Docker Swarm tool which is able to deploy an application to the multiple nodes in the network, additional to starting/stopping them.

It is clear that performing all tasks at the network edge is not feasible because of limited capacities of RPI boards. Additionally, a smart home should be manageable from a remote location which requires Cloud nodes in order to transfer user requests to the Fog Layer. For this purpose, we proposed the Cloud Layer which is also able to manage nodes of the Fog Layer. In terms of software infrastructure, Fog and Cloud Layers are nearly the same, except the logic of applications and services. At the very bottom which is named as the Things Layer, lots of heterogeneous IoT devices are available conceptually. The interaction between Fog and Cloud Layers is provided by RPI boards which are able to communicate with different kind of devices with the help of Raspbian OS and Node-Red (home automation framework) packages. For our use cases, we used several IoT sensors and actuators but this does not mean that our system only supports a few types of devices. In order to analyze the limits, we also performed experiments and presented results in the previous chapter.

To conclude, we proposed a multi-layer home automation architecture named FOGHA and implemented both basic scenarios like turning on/off some devices and complex scenarios like real time face recognition analysis from a video stream. Additionally, we implemented an Android application to vary the scenarios and ease interaction with the devices. Additional to Web and Android interface, a chat-bot was implemented for this purpose which is able to convert user-typed sentences to system intents. If it does not understand the sentence, it provides available system actions to the user and train itself based on the user response. Other than these functionalities, resource usage and current situation of RPI boards are continuously collected and analyzed to avoid errors. Besides hardware level information, our system also provides application level resource usages with the help of the Docker container architecture. In that way, we are able to observe a specific application and find possible bottlenecks in the system.

In order to improve our system and vary scenarios, new services/applications could be implemented and easily deployed as Docker containers. Because of this functionality, we also believe that it is a suitable candidate for a home automation testbed. Use cases could be increased and additional performance evaluation sections could be

added based on those use cases. In our implementation, we used seven RPI boards which are enough for a standard house however this number could be increased to cover a larger area such as a factory. This would present more detailed results, especially in terms of the scalability aspect. Additionally, the number and variety of devices in the Things Layer could be increased to analyze the Fog-Things Layer interaction.

In addition, integration of new devices and software to the FOGHA could be eased by abstracting low level details from the user. For this purpose, a predetermined set of hardware equipments could be supported by the FOGHA and the integration steps could be decreased to one user click through an integration UI. Secondly, we would implement interesting entertainment services such as games and Virtual Reality applications, additional to health-care services which are critical for the elderly people if we had extra time for the thesis. Thirdly, the number and computational power of the Cloud managers could be increased in order to convert it to a generic form which serves many FOGHA enabled houses. In this way, the network of these houses would lead to the implementation of modern smart cities.

REFERENCES

1. Business Insider , *Here's how the Internet of Things will explode by 2020*, <http://read.bi/1PXI6FB>, accessed at December 2017.
2. Brush, A., B. Lee, R. Mahajan, S. Agarwal, S. Saroiu and C. Dixon, "Home automation in the wild: challenges and opportunities", *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2115–2124, ACM, 2011.
3. Rafferty, J., C. D. Nugent, J. Liu and L. Chen, "From activity recognition to intention recognition for assisted living within smart homes", *IEEE Transactions on Human-Machine Systems*, Vol. 47, No. 3, pp. 368–379, 2017.
4. Masip-Bruin, X., E. Marín-Tordera, G. Tashakor, A. Jukan and G.-J. Ren, "Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems", *IEEE Wireless Communications*, Vol. 23, No. 5, pp. 120–128, 2016.
5. Chiang, M. and T. Zhang, "Fog and IoT: An overview of research opportunities", *IEEE Internet of Things Journal*, Vol. 3, No. 6, pp. 854–864, 2016.
6. Bonomi, F., R. Milito, P. Natarajan and J. Zhu, "Fog computing: A platform for internet of things and analytics", *Big Data and Internet of Things: A Roadmap for Smart Environments*, pp. 169–186, Springer, 2014.
7. Yi, S., C. Li and Q. Li, "A survey of fog computing: concepts, applications and issues", *Proceedings of the 2015 Workshop on Mobile Big Data*, pp. 37–42, ACM, 2015.
8. Fettweis, G. P., "The tactile internet: Applications and challenges", *IEEE Vehicular Technology Magazine*, Vol. 9, No. 1, pp. 64–70, 2014.

9. Alam, M. R., M. B. I. Reaz and M. A. M. Ali, “A review of smart homes—Past, present, and future”, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 42, No. 6, pp. 1190–1203, 2012.
10. Liu, F. and W. Guo, “The Design and Implementation of MINA-Based Smart Home Data Synchronization System”, *Instrumentation and Measurement, Computer, Communication and Control (IMCCC), 2015 Fifth International Conference on*, pp. 1612–1616, IEEE, 2015.
11. Gu, H., Y. Diao, W. Liu and X. Zhang, “The design of smart home platform based on cloud computing”, *Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011 International Conference on*, Vol. 8, pp. 3919–3922, IEEE, 2011.
12. Dickey, N., D. Banks and S. Sukittanon, “Home automation using Cloud Network and mobile devices”, *Southeastcon, 2012 Proceedings of IEEE*, pp. 1–4, IEEE, 2012.
13. Soliman, M., T. Abiodun, T. Hamouda, J. Zhou and C.-H. Lung, “Smart home: Integrating internet of things with web services and cloud computing”, *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, Vol. 2, pp. 317–320, IEEE, 2013.
14. Ye, X. and J. Huang, “A framework for cloud-based smart home”, *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, Vol. 2, pp. 894–897, IEEE, 2011.
15. Arduino Foundation, *Arduino*, <https://www.arduino.cc/en/Guide/Introduction>, accessed at December 2017.
16. Beagleboard Foundation, *Beagle Board*, <https://beagleboard.org/getting-started>, accessed at December 2017.

17. Intel Corporation , *Intel Edison*, <https://software.intel.com/en-us/iot/hardware/edison>, accessed at December 2017.
18. Aidilab, *Udoo Neo*, <https://www.udoo.org/udoo-neo/>, accessed at December 2017.
19. Raspberry Pi Foundation, *Raspbbery Pi*, <https://www.raspberrypi.org/products/>, accessed at December 2017.
20. Brown, E., *21 Open Source Projects for IoT*, <https://www.linux.com/NEWS/21-OPEN-SOURCE-PROJECTS-IOT>, accessed at December 2017.
21. Gomez, C. and J. Paradells, “Wireless home automation networks: A survey of architectures and technologies”, *IEEE Communications Magazine*, Vol. 48, No. 6, 2010.
22. Sigma Designs, *Z-Wave Protocol*, <http://zwavepublic.com/specifications>, accessed at December 2017.
23. ZigBee Alliance, *ZigBee protocol*, <http://www.zigbee.org/>, accessed at December 2017.
24. Lin, J., W. Yu, N. Zhang, X. Yang, H. Zhang and W. Zhao, “A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications”, *IEEE Internet of Things Journal*, 2017.
25. Siekkinen, M., M. Hienkari, J. K. Nurminen and J. Nieminen, “How low energy is bluetooth low energy? comparative measurements with zigbee/802.15. 4”, *Wireless Communications and Networking Conference Workshops (WCNCW)*, 2012 *IEEE*, pp. 232–237, IEEE, 2012.
26. Al-Fuqaha, A., M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, “Internet of things: A survey on enabling technologies, protocols, and applications”,

- IEEE Communications Surveys & Tutorials*, Vol. 17, No. 4, pp. 2347–2376, 2015.
27. Tan, J. and S. G. Koo, “A survey of technologies in internet of things”, *Distributed Computing in Sensor Systems (DCOSS), 2014 IEEE International Conference on*, pp. 269–274, IEEE, 2014.
 28. Kushalnagar, N., G. Montenegro and C. Schumacher, *IPv6 over low-power wireless personal area networks (6LoWPANs): overview, assumptions, problem statement, and goals*, Tech. rep., 2007.
 29. IETF, *RFC 7252 Constrained Application Protocol*, <https://tools.ietf.org/html/rfc7228>, accessed at December 2017.
 30. Schneider, S., *Understanding The Protocols Behind The Internet Of Things*, <http://www.electronicdesign.com/iot/understanding-protocols-behind-internet-things>, accessed at December 2017.
 31. Sheng, Z., S. Yang, Y. Yu, A. Vasilakos, J. Mccann and K. Leung, “A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities”, *IEEE Wireless Communications*, Vol. 20, No. 6, pp. 91–98, 2013.
 32. Lee, G. M. and J. Y. Kim, “The Internet of Things—A problem statement”, *Information and Communication Technology Convergence (ICTC), 2010 International Conference on*, pp. 517–518, IEEE, 2010.
 33. Liang, N.-S., L.-C. Fu and C.-L. Wu, “An integrated, flexible, and Internet-based control architecture for home automation system in the Internet era”, *Robotics and Automation, 2002. Proceedings. ICRA’02. IEEE International Conference on*, Vol. 2, pp. 1101–1106, IEEE, 2002.
 34. Corcoran, P. M., F. Papal and A. Zoldi, “User interface technologies for home appliances and networks”, *IEEE Transactions on Consumer Electronics*, Vol. 44,

No. 3, pp. 679–685, 1998.

35. Tan, K. K., T. H. Lee and C. Y. Soh, “Internet-based monitoring of distributed control systems-An undergraduate experiment”, *IEEE Transactions on Education*, Vol. 45, No. 2, pp. 128–134, 2002.
36. Swamy, N., O. Kuljaca and F. L. Lewis, “Internet-based educational control systems lab using NetMeeting”, *IEEE transactions on education*, Vol. 45, No. 2, pp. 145–151, 2002.
37. Guinard, D., V. M. Trifa and E. Wilde, *Architecting a mashable open world wide web of things*, ETH, Department of Computer Science, 2010.
38. Jamil, M. M. A. and M. S. Ahmad, “A pilot study: Development of home automation system via raspberry Pi”, *Biomedical Engineering (ICoBE), 2015 2nd International Conference on*, pp. 1–4, IEEE, 2015.
39. Ramlee, R., D. Tang and M. Ismail, “Smart home system for disabled people via wireless bluetooth”, *System Engineering and Technology (ICSET), 2012 International Conference on*, pp. 1–4, IEEE, 2012.
40. Fahim, M., I. Fatima, S. Lee and Y.-K. Lee, “Daily life activity tracking application for smart homes using android smartphone”, *Advanced Communication Technology (ICACT), 2012 14th International Conference on*, pp. 241–245, IEEE, 2012.
41. Gill, K., S.-H. Yang, F. Yao and X. Lu, “A zigbee-based home automation system”, *IEEE Transactions on Consumer Electronics*, Vol. 55, No. 2, 2009.
42. Zhu, J., X. Gao, Y. Yang, H. Li, Z. Ai and X. Cui, “Developing a voice control system for zigbee-based home automation networks”, *Network Infrastructure and Digital Content, 2010 2nd IEEE International Conference on*, pp. 737–741, IEEE, 2010.

43. Piyare, R. and M. Tazil, “Bluetooth based home automation system using cell phone”, *Consumer Electronics (ISCE), 2011 IEEE 15th International Symposium on*, pp. 192–195, IEEE, 2011.
44. Kamilaris, A. and A. Pitsillides, “Towards interoperable and sustainable smart homes”, *IST-Africa Conference and Exhibition (IST-Africa), 2013*, pp. 1–11, IEEE, 2013.
45. Wenbo, Y., W. Quanyu and G. Zhenwei, “Smart home implementation based on internet and WiFi technology”, *Control Conference (CCC), 2015 34th Chinese*, pp. 9072–9077, IEEE, 2015.
46. Navya, M. and P. Ramachandran, “Development of secured home automation using social networking sites”, *Indian Journal of Science and Technology*, Vol. 8, No. 20, 2015.
47. Jain, S., A. Vaibhav and L. Goyal, “Raspberry Pi based interactive home automation system through E-mail”, *Optimization, Reliability, and Information Technology (ICROIT), 2014 International Conference on*, pp. 277–280, IEEE, 2014.
48. Birajda, G. and S. Mahindrakar, “Embedded webserver based home automation using raspberry pi”, *International Journal of Modern Trends in Engineering and Research*, Vol. 1, No. 5, 2014.
49. Upadhyay, Y., A. Borole and D. Dileepan, “MQTT based secured home automation system”, *Colossal Data Analysis and Networking (CDAN), Symposium on*, pp. 1–4, IEEE, 2016.
50. Celebre, A. M. D., A. Z. D. Dubouzet, I. B. A. Medina, A. N. M. Surposa and R. C. Gustilo, “Home automation using raspberry Pi through Siri enabled mobile devices”, *Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM), 2015 International Conference on*, pp. 1–6, IEEE, 2015.

51. Sarkar, C., S. A. U. Nambi, R. V. Prasad and A. Rahim, “A scalable distributed architecture towards unifying IoT applications”, *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pp. 508–513, IEEE, 2014.
52. Gonnot, T. and J. Saniie, “User defined interactions between devices on a 6LoWPAN network for home automation”, *Technology Management Conference (ITMC), 2014 IEEE International*, pp. 1–4, IEEE, 2014.
53. Korkmaz, I., S. K. Metin, A. Gurek, C. Gur, C. Gurakin and M. Akdeniz, “A cloud based and Android supported scalable home automation system”, *Computers & Electrical Engineering*, Vol. 43, pp. 112–128, 2015.
54. Pavithra, D. and R. Balakrishnan, “IoT based monitoring and control system for home automation”, *Communication Technologies (GCCT), 2015 Global Conference on*, pp. 169–173, IEEE, 2015.
55. David, N., A. Chima, A. Ugochukwu and E. Obinna, “Design of a home automation system using arduino”, *International Journal of Scientific & Engineering Research*, Vol. 6, No. 6, pp. 795–801, 2015.
56. Sehgal, T. and S. More, “Home Automation using IOT and Mobile App”, *International Research Journal of Engineering and Technology*, Vol. 4, pp. 694–698, 2017.
57. Patil, S. A. and V. Pinki, “Home Automation Using Single Board Computing as an Internet of Things Application”, *Proceedings of International Conference on Communication and Networks*, pp. 245–253, Springer, 2017.
58. Lohokare, J., R. Dani, A. Rajurkar and A. Apte, “An IoT ecosystem for the implementation of scalable wireless home automation systems at smart city level”, *Region 10 Conference, TENCON 2017-2017 IEEE*, pp. 1503–1508, IEEE, 2017.
59. Qiu, T., R. Qiao and D. O. Wu, “EABS: An event-aware backpressure scheduling

- scheme for emergency Internet of Things”, *IEEE Transactions on Mobile Computing*, Vol. 17, No. 1, pp. 72–84, 2018.
60. Yi, S., Z. Hao, Z. Qin and Q. Li, “Fog computing: Platform and applications”, *Hot Topics in Web Systems and Technologies (HotWeb), 2015 Third IEEE Workshop on*, pp. 73–78, IEEE, 2015.
 61. Openstack Project, *Openstack*, <https://www.openstack.org/>, accessed at December 2017.
 62. Aazam, M. and E.-N. Huh, “Fog computing and smart gateway based communication for cloud of things”, *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*, pp. 464–470, IEEE, 2014.
 63. Li, J., J. Jin, D. Yuan, M. Palaniswami and K. Moessner, “EHOPES: Data-centered Fog platform for smart living”, *Telecommunication Networks and Applications Conference (ITNAC), 2015 International*, pp. 308–313, IEEE, 2015.
 64. Hong, K., D. Lillethun, U. Ramachandran, B. Ottenwälder and B. Koldehofe, “Mobile fog: A programming model for large-scale applications on the internet of things”, *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, pp. 15–20, ACM, 2013.
 65. Zao, J. K., T. T. Gan, C. K. You, S. J. R. Méndez, C. E. Chung, Y. Te Wang, T. Mullen and T. P. Jung, “Augmented brain computer interaction based on fog computing and linked data”, *Intelligent Environments (IE), 2014 International Conference on*, pp. 374–377, IEEE, 2014.
 66. Hu, P., H. Ning, T. Qiu, Y. Zhang and X. Luo, “Fog computing based face identification and resolution scheme in internet of things”, *IEEE Transactions on Industrial Informatics*, Vol. 13, No. 4, pp. 1910–1920, 2017.
 67. Evans, D., “The internet of things: How the next evolution of the internet is

- changing everything”, *CISCO white paper*, Vol. 1, No. 2011, pp. 1–11, 2011.
68. Raspberry Pi Foundation, *Raspbian Operating System*, <https://www.raspbian.org/>, accessed at December 2017.
 69. SSH Communications Security Corp., *SSH Protocol*, <https://www.ssh.com/ssh/protocol/>, accessed at December 2017.
 70. Real VNC Ltd., *Real VNC*, <https://www.realvnc.com/en/raspberrypi/>, accessed at December 2017.
 71. DockerInc., *Docker*, <https://www.docker.com/>, accessed at December 2017.
 72. Docker Inc., *Docker Swarm*, <https://docs.docker.com/engine/swarm/>, accessed at December 2017.
 73. Docker Hub, *Official Docker Node-Red Image*, <https://hub.docker.com/r/nodered/node-red-docker/>, accessed at December 2017.
 74. Docker Hub, *Customized Docker Node-Red Image*, <https://hub.docker.com/r/hanilozman/node-red-docker/>, accessed at December 2017.
 75. JS Foundation, *Node Package Manager*, <https://www.npmjs.com/>, accessed at December 2017.
 76. InfluxData Inc., *Chronograf Visualization Tool*, <https://docs.influxdata.com/chronograf/v1.4/>, accessed at December 2017.
 77. Redis Labs, *Redis*, <https://redis.io/>, accessed at December 2017.
 78. Docker Hub, *Redis Docker Image for Arm Processors*, <https://hub.docker.com/r/alexellis2/redis-arm/>, accessed at December 2017.
 79. Eurotech, *MQTT Protocol*, <http://mqtt.org/>, accessed at December 2017.

80. Crontab Project, *Crontab*, <http://www.adminschoice.com/crontab-quick-reference>, accessed at December 2017.
81. Scaleway Inc., *Scaleway Cloud Provider*, <https://www.scaleway.com/>, accessed at December 2017.
82. Angular.js Project, *Angular.js Framework*, <https://angularjs.org/>, accessed at December 2017.
83. Amazon.com Inc., *Amazon Echo*, <https://www.amazon.com/Amazon-Echo-And-Alexa-Devices/>, accessed at December 2017.
84. JS Foundation, *Alexa Home Skill Plugin of Node-Red Framework*, <https://flows.nodered.org/node/node-red-contrib-alexa-home-skill>, accessed at December 2017.
85. Cade, M. and S. Roberts, *Sun Certified Enterprise Architect for J2EE technology study guide*, Prentice Hall Professional, 2002.
86. OpenSSL Software Foundation, *OpenSSL*, <https://www.openssl.org/>, accessed at December 2017.
87. Particle Company, *Particle Photon*, <https://www.particle.io/>, accessed at December 2017.
88. Raspberry Pi Foundation, *Raspberry Pi SenseHAT*, <https://www.raspberrypi.org/products/sense-hat/>, accessed at December 2017.
89. Intel Research, *OpenCV library*, <https://opencv.org/>, accessed at December 2017.
90. Princeton University, *WordNet Database*, <https://wordnet.princeton.edu/wordnet/>, accessed at December 2017.

91. NPM, *Node-Red Wordpos Wrapper library*, <https://www.npmjs.com/package/node-red-contrib-wordpos>, accessed at December 2017.
92. IEEE IoT Initiative, *IoT scenarios of IEEE IoT Initiative*, <https://iot.ieee.org/iot-scenarios.html>, accessed at December 2017.
93. ICT CityPulse, *Aging population - home monitoring*, <https://scenarios.organicity.eu/scenarios/8fee1c04-2f7c-4078-99e5-a5290e9c1eac>, accessed at December 2017.
94. ICT CityPulse, *Continuous care*, <https://scenarios.organicity.eu/scenarios/8aafc206-40f4-4e62-8591-300b15ef87a8/>, accessed at December 2017.
95. ICT CityPulse, *Bringing the cloud or network structures closer to communal scales*, <https://scenarios.organicity.eu/scenarios/8d9eead4-f28a-45d9-b276-0262631d7aed>, accessed at December 2017.
96. GreenIQ Ltd., *GreenIQ: IoT Water Saving*, <https://iot.ieee.org/iot-scenarios.html?prp=24>, accessed at December 2017.
97. Zalevski, Z., *FabriXense*, <https://iot.ieee.org/iot-scenarios.html?prp=23>, accessed at December 2017.
98. Krylovskiy, A., *MQTT Benchmark*, <https://github.com/krylovsk/mqtt-benchmark>, accessed at December 2017.
99. Kopytov, A., *Sysbench Benchmark*, <https://wiki.gentoo.org/wiki/Sysbench>, accessed at December 2017.
100. Iperf Project, *Iperf*, <https://iperf.fr/>, accessed at December 2017.