



Do Le Tuan Minh

Building an IoT Application using Amazon Web Service

Technology and Communication
2019

FOREWORD

My 5 years study in Vaasa University of Applied Sciences is going to be concluded soon, so now it is a good time for me to express my gratitude to everyone who has supported me.

Firstly, I would like to show my appreciation to Mr. Jukka Matila, my supervisor, for the understanding I obtained during my studying in Embedded System Specialization and the supports he made during my thesis work, which inspired me significantly to further developing myself.

Moreover, I would give my gratitude to Mr. Vu Anh Truong, my Project Manager, Phan Huu Tuan, Nguyen Doan Tan Anh, my teammates in IoT group and the UX/UI design group from Solution and Technology Unit (STU) of FPT Software for their massive and effective support during the implementation of my project. Without them, I would have hard time in my progression.

Furthermore, I would like to send my love and thanks to my family in Vietnam, they will always be my most significant support and inspiration. I would like to give my thanks to all my classmates and friends I made during my studies, for that time was one of the most wonderful time of my life. Finally, I would like to express my gratitude to my friends Pham Quynh Trang and Phi Thi Thuy Trang for their endless morale support during my hard times.

Vaasa 21.03.2019

Do Le Tuan Minh

ABSTRACT

Author	Do Le Tuan Minh
Title	Building an IoT Application using Amazon Web Service
Year	2019
Language	English
Pages	78
Name of Supervisor	Jukka Matila

The Internet of Things Technology is growing and becoming more popular in this digital system era. This technology is connecting physical world into the Internet in order to boost the productivity or efficiency of countless aspects of life. However, with the growth of demand of the market, Internet of Things is facing challenges such as Variation of Hardware and Platforms, System Connectivity, Security. Amazon Web Service (AWS) IoT offers a wide range of functions to help building IoT solutions for lots of use cases and devices. This thesis focus on the main aspects of an Internet of Things System, Amazon Web Service fundamental and its solution in real IoT Application.

This thesis is built around the project, which was carried out as a research of applying IoT technology in Vietnam, by Solution and Technology Unit (STU) of FPT Software. The main purpose of this thesis was to give understandings of building an IoT infrastructure and building a working system of monitoring and controlling IoT system using AWS IoT, which can be applied in real life and used as study case for later development.

In this project, certain sensor nodes were created in order to obtain data from Ambient Environment. The data was transferred to the IoT Gateway using LORA and Zigbee Technique. Afterward the data was analyzed, stored to a CSV file locally, published on a Cloud Server using Amazon Web Services (AWS) as a JSON file. The main AWS feature used was AWS IoT Core. The data finally can be obtained by using Android application for visualization, tracking the status of nodes and controlling certain devices such as light bulb or water pumping machine.

ABBREVIATIONS

IoT	Internet of Things
AWS	Amazon Web Service
MQTT	Message Queuing Telemetry Transport
HTTP/S	Hypertext Transfer Protocol/Secure
M2M	Machine to Machine
LoRa	Long Range
SDK	Software Development Kit
JSON	JavaScript Object Notation
CSV	Comma-separated Value
XML	Extensible Markup Language
RFID	Radio Frequency Identification
IP	Internet Protocol
OTA	Over the Air
PaaS	Platform as a Service
SaaS	Software as a Service
UART	Universal Asynchronous Receiver-Transmitter
QoS	Quality of Service
WAN	Wide Area Network
REST	Representational State Transfer

API	Application programming interface
TCP/IP	Transmission Control Protocol/Internet Protocol
IDE	Integrated Development Environment
GUI	Graphical User Interface
SRAM	Static Random Access Memory
EEPROM	Electrical Erasable Programmable Read Only Memory
I/O	Input/Output
GPIO	General Purpose Input/Output
ADC	Analog to Digital Converter
IAM	Identity and Access Management
SQL	Structured Query Language
USB	Universal Serial Bus
FPT	Financing and Promoting Technology

TABLE OF CONTENTS

1	INTRODUCTION	10
1.1	Background and Purpose	10
1.2	IoT project for FPT Software	11
2	BACKGROUND FOR THE PROJECT	12
2.1	Internet of Things Definition	12
2.2	History of Internet of Things	13
2.3	IoT Platform	15
2.4	AWS IoT Platform	18
2.4.1	AWS IoT introduction and components	19
2.4.2	How AWS IoT Operate	22
2.5	Protocols and Communication Technique	23
2.5.1	MQTT protocol	23
2.5.2	MQTT libraries	25
2.5.3	LoRa Technology	26
2.5.4	UART protocol	27
2.6	Software	29
2.6.1	Arduino Programming Language	29
2.6.2	Python Programming Language	30
2.6.3	Android	31
2.7	Hardware Components	31
2.7.1	ATmega328P Microprocessor and Arduino UNO R3 development board	31
2.7.2	ESP8266 NODEMCU	32
2.7.3	Raspberry Pi 3 Model B	33
2.7.4	DHT11 temperature and humidity sensor	34
2.7.5	Light Intensity Sensor	35
2.7.6	Soil Moisture Sensor	36
2.7.7	E32-TTL-100 UART Lora SX1278 433Mhz	36
2.7.8	Solar Panel power supply	37
3	IMPLEMENTATION	40
3.1	Infrastructure	40
3.2	IoT device	41
3.2.1	Hardware Circuit Diagram	41

3.2.2	Software.....	44
3.3	Gateway and Cloud	54
3.3.1	AWS IoT Thing set up	54
3.3.2	Gateway	55
3.3.3	Thing Shadow Management and Usage	59
3.4	Android Application.....	64
3.4.1	Setting up the Environment	64
3.4.2	Main Application.....	64
4	TESTING AND RESULT	73
5	CONCLUSION AND FUTURE WORK	75
6	REFERENCES	76

TABLE OF FIGURES

<i>Figure 1: IoT System Structure</i>	12
<i>Figure 2: Stages of IoT Data Management in Cloud Platform (Source: Google Cloud)</i>	13
<i>Figure 3: Number of Devices connected to Internet from 2003 to 2020 (Source: Cisco)</i>	14
<i>Figure 4: IoT application market and trend. (Source: IoT analytic, 2018)</i>	15
<i>Figure 5: Function of IoT Platform (Source: Kaaproject, 2011)</i>	16
<i>Figure 6: functions, features and components of different platform types (Source: First Analytic, 2016, online)</i>	17
<i>Figure 7: AWS IoT main components and how they work (Source: Amazon, online)</i>	19
<i>Figure 8: Example of Thing Shadow structure.</i>	20
<i>Figure 9: Overview of AWS Rule Engine (Source: ProMow)</i>	21
<i>Figure 10: Overview of AWS IoT Authentication and Authorization (Source: Amazon).</i>	22
<i>Figure 11: MQTT Quality of Service (Source: DZone)</i>	24
<i>Figure 12: Example use case of MQTT.</i>	25
<i>Figure 13: LoRa Package Forwarding (Source: Online)</i>	26
<i>Figure 14: Overview of LoraWan Network (Source: iotvigyan)</i>	27
<i>Figure 15: UART data frame (Source: circuitbasics)</i>	28
<i>Figure 16: UART pins connection (Source: Sparkfun)</i>	29
<i>Figure 17: Arduino Uno R3 Microcontroller Board.</i>	32
<i>Figure 18: ESP8266 NODEMCU</i>	33
<i>Figure 19: Raspberry Pi 3B</i>	34
<i>Figure 20: DHT11 temperature sensor and its working principles (Source: how to mechatronics)</i>	34
<i>Figure 21: Connecting LDR</i>	35
<i>Figure 22: Soil Moisture Sensor</i>	36
<i>Figure 23: E32-TTL-100 UART Lora SX1278</i>	37
<i>Figure 24: 9V 2W Solar Panel</i>	38
<i>Figure 25: Power Supply for Node Sensor</i>	39
<i>Figure 26: Data Flow Sequence of the System</i>	40
<i>Figure 27: System Architecture</i>	41

<i>Figure 28: Node sensor circuit.....</i>	<i>42</i>
<i>Figure 29: Node Sensor Short-Range Communication.....</i>	<i>43</i>
<i>Figure 30: Flowchart of the Software of DHT 11 sensor.</i>	<i>44</i>
<i>Figure 31: Wireless Module Setting Software.....</i>	<i>50</i>
<i>Figure 32: AWS IoT Thing Shadow Update Operation</i>	<i>60</i>
<i>Figure 33: AWS IoT Thing Shadow Get Operation.</i>	<i>61</i>
<i>Figure 34: Node Sensor's data in real time</i>	<i>67</i>
<i>Figure 35: GPS checking Tab.....</i>	<i>68</i>
<i>Figure 36: Notification Tab.....</i>	<i>70</i>
<i>Figure 37: DynamoDB Table.....</i>	<i>71</i>
<i>Figure 38: GPS, Graph and Motor Control Tab</i>	<i>72</i>
<i>Figure 39: AWS IoT MQTT testing</i>	<i>73</i>
<i>Figure 40: Thing Shadow Document</i>	<i>74</i>

1 INTRODUCTION

1.1 Background and Purpose

The popularity of Internet of Things is growing dramatically. Its existence is playing an essential role that appears in many aspects of daily life, from connecting homes and cities, to connecting traffics to roads as well as location tracking systems, healthcare, agriculture. Everything around the life can be connected into an enormous infrastructure of Internet of Things, as the result of the huge impact of Internet to mankind.

According to Cisco (2011, online), the number of devices connected to the internet will grow to 50 Billion by 2020. These devices or “Things” include mobile phones, embedded devices, microcontrollers, sensors and actuators. As a result of this rapid development, human beings will receive enormous amount of benefits. The communication between devices becomes better (as known as Machine to Machine Communication or “M2M”), machines communicate with each other automatically without human intervention, which lead to better output and quicker response times. Therefore, time and money are saved in dramatically amounts. Moreover, better M2M communication will lead to more accurate results, which will raise the efficiency of production processes and provide more desirable life quality and convenience for mankind.

However, the rapid growth of internet connected devices, drawbacks and challenges started to increase. Firstly, devices come from different manufactures, which makes difficulty in compatibility between devices and having a common standard. Secondly, IoT infrastructure is complex, as a consequence, more bugs or errors when the system is operating. Furthermore, security and privacy are one of the most challenging issue in the IoT field and data might be exposed during its transition.

This thesis work is carried out in order to archive certain goals:

- Understanding the definition of IoT, its core components and infrastructure.
- Becoming familiar with the AWS IoT platform and its ecosystem when working with an IoT project.
- Developing a working IoT solution, applying state of the arts technology from sensors nodes to IoT cloud server using AWS IoT platform and understanding by researching IoT infrastructure.

1.2 IoT project for FPT Software

With the continuously growth of the IoT technology, its needs in applying to daily life is on high demand. To help gaining better understanding of building an IoT solution, FPT software, a technology-leading company in Vietnam, offered me this project. This project was created to be deployed in the company academy's vegetable field and served as study case for other teams.

Overall, this project was created by deploying an IoT infrastructure including several node sensors in different places, obtaining their data, analysing, monitoring, visualizing and control certain machinery for agricultural purposes.

This project was done using Arduino (C/C++ modified) for nodes, Python for Gateway and Java/XML Android programming languages. AWS IoT platform provides Python and Android SKD.

2 BACKGROUND FOR THE PROJECT

2.1 Internet of Things Definition

With the evolution of the Internet, smartphones and especially sensor systems, Internet of Things (IoT) is becoming a new trend in the world of technology. Internet of Things is defined as “A network of Internet connected objects, which are able to collect and exchange data wirelessly”. This word consists of two parts: “Internet” is the way, method of connectivity and “Things”, which are objects, physical or virtual devices.

In order to collect or share data, “Things” are connected to an IoT platform, which enables straightforward provisioning, management, and automation of connected devices within the Internet of Things universe. It basically connects your hardware, however diverse, to the cloud by using flexible connectivity options, enterprise-grade security mechanisms, and broad data processing powers (Kaaproject 2018.) There are numbers of different aspects that different platforms offer: data storing, protocols, offering features (security, privacy), licenses, price.

An IoT system is separated into three basic components: device (or “Thing”), gateway and cloud.

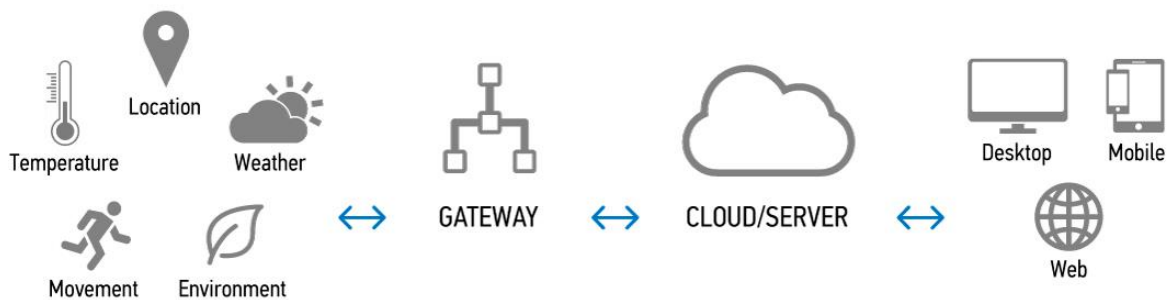


Figure 1: IoT System Structure

- Devices: They contain both hardware and software, which connect to the world directly to obtain data from sensors input. Devices are connected to a common network in order to communicate with each other. Devices might be connected directly or indirectly to the Internet. Example of devices include cars, sensors, smart watches, smart phone.

- Gateway: This central device that processes data from devices which are not directly connected to the Internet, allowing them to connect to the Cloud Service in a secure and easy to manage way. The data from the devices can be transferred to gateway through certain techniques or protocols.
- Cloud: This is a way that data transferred from a gateway is stored, processed and analysed. As the result of these processes, applications (Desktop, Mobile, Web) can be built for specific needs of users. In order for the gateway to communicate with the cloud, an IoT platform is needed.

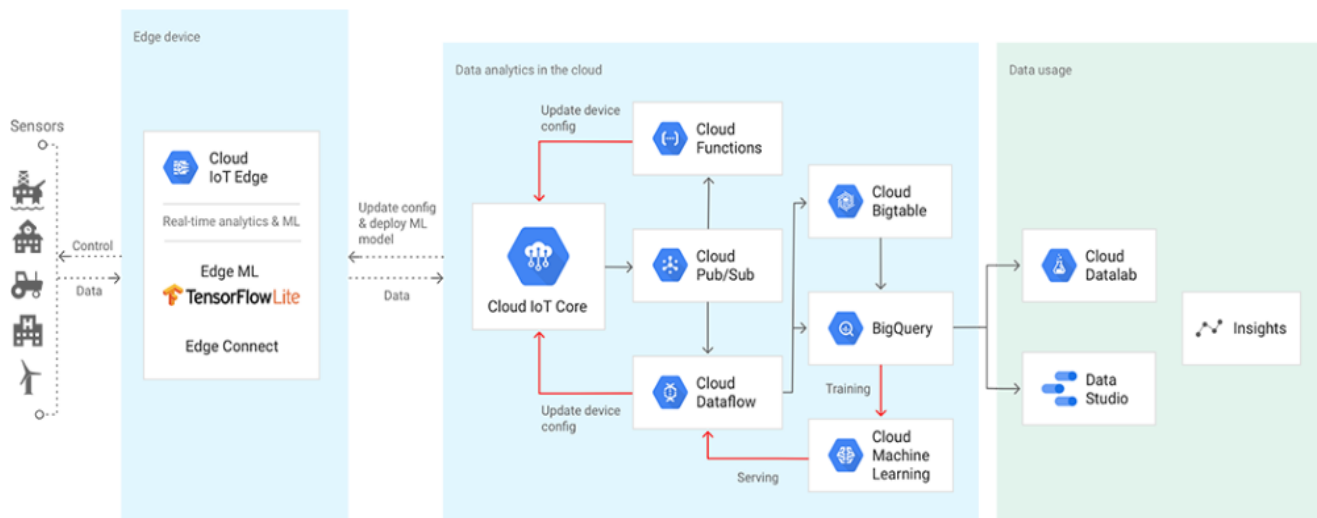


Figure 2: Stages of IoT Data Management in Cloud Platform (Source: Google Cloud)

2.2 History of Internet of Things

The word “Internet of Things” was defined in 1999 by Kevin Ashton, a scientist who was the founder of Auto-ID centre of Massachusetts Institute of Technology (MIT). At that time, he believed that Radio frequency identification (RFID) was the essential part of IoT, which allows computers to manage individual things (Wikipedia, 2011, online). In the same year, his centre also turned RFID into a networking technology by linking devices to the Internet using RFID tag. Cisco Systems (2011, online) estimated that IoT was officially born between 2008 and 2009, when “things” connected to the Internet are more than mankind.

In 2000, LG introduced the first refrigerator that can be connected to the Internet.

In 2008, a group of big companies such as Cisco, Bosch, Ericsson, formed IPSO to promote the Internet Protocol (IP) and research the network of “things”. Smartphones started to connect to the Internet, which made the number of devices connected to Internet grew rapidly.

The things and people ratio raised from 0.08 in 2003 to 1.84 in 2010, which year the number of devices connected to the Internet reached 12.5 billion, which exceeding over the number of human beings on the Earth (Cisco System, 2011, online). The year of 2011 was an explosion for IoT, as the IPv6 protocol was launched. This protocol allows 2218 unique addresses to be created, which solved the problem of IPv4’s small address capacity and lead to shared public IP addresses. This achievement allows even more devices to be connected to the Internet. An estimate said that in 2020 there are going to be 30 billion devices connect wirelessly to the Internet (ABI research, 2013).

Figure 3 shows the number of internet connecting devices from 2003 to 2020.

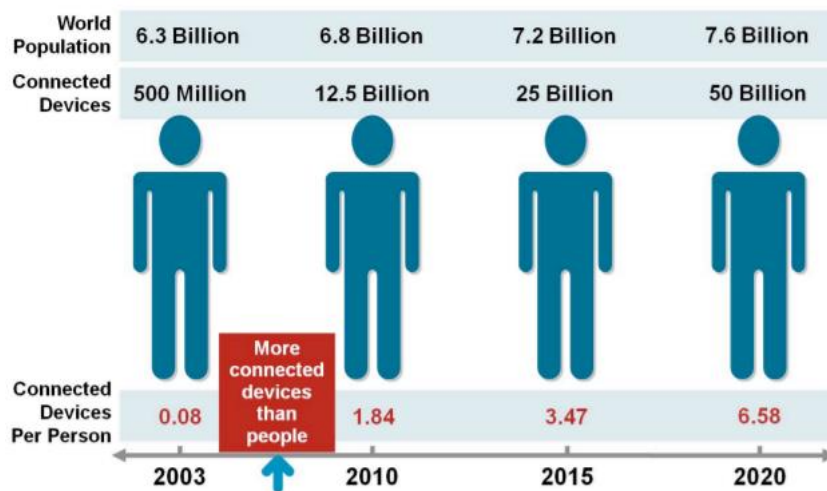


Figure 3: Number of Devices connected to Internet from 2003 to 2020 (Source: Cisco)

With the rapidly gaining of popularity, IoT technology presents in various of market and it is predicted to grow even stronger in the next recent years. Figure 4 shows some of the most popular market of IoT applications and its trend.

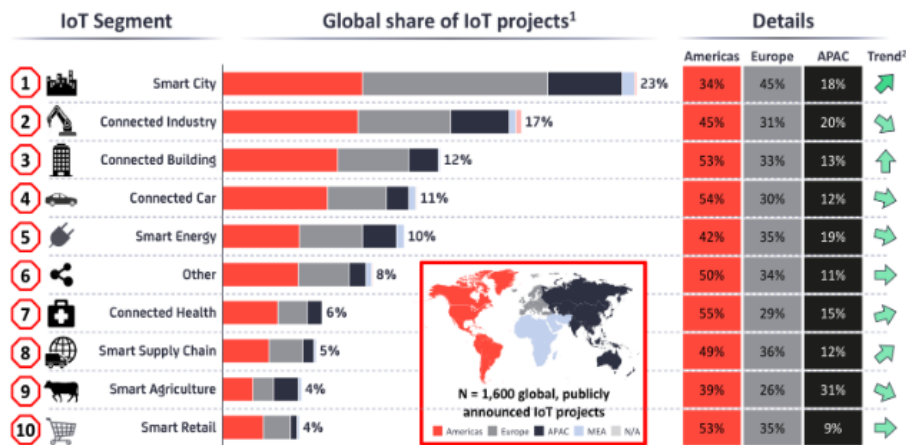


Figure 4:IoT application market and trend. (Source: IoT analytic, 2018)

According to the Figure 4, Smart City is taking the trend as the most popular application with 23% in IoT market. Every entity of the City joins the huge ecosystem of the Internet, from surveillance, automatic transport, energy and water distribution, security to ambient environment monitoring. Some challenges that Smart City is facing and solving are traffic jams, pollution and vandalism. Together with Smart City, Smart Home is a big trend of IoT as the houses become smarter with Automatic air conditioner, heater, light switching on when required. These products help saving a reasonable amount of money, time and energy in an efficiency way. On the other hand, the following categories are earning more reputation for the IoT market: Agriculture, Health Care, Industrial, Smart Car and Wearable Gadgets.

2.3 IoT Platform

An IoT platform usually acts as a supporting software, which functions as a middleware between the hardware (things, objects) and the applications. Its tasks include collecting data from devices via certain protocols and network topologies, configuring and controlling devices remotely, visualizing data and over the air firmware update for them.

An IoT platform can be separated into layers. At the bottom is the infrastructure layer, which enables the functions of the platform. The communication layer allows the devices

to connect to the cloud to operate certain functions that the platform offers. The final layer contains core features provided by the platform. These features include container management, data collection, message and OTA software updating. Figure 5 demonstrates the block diagram architecture and functions of the IoT platform.

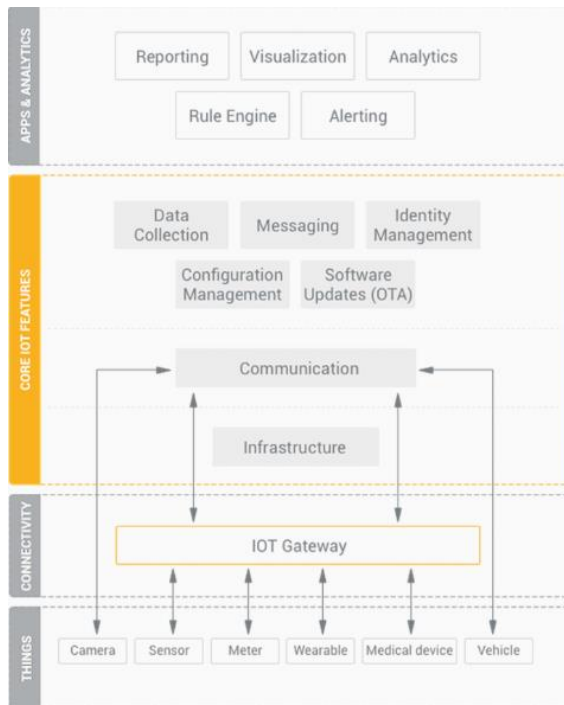


Figure 5: Function of IoT Platform (Source: Kaaproject,2011)

In the market there are various IoT platforms. Choosing the right platform is one of the most challenging issues in an IoT solution. There are three main types of platforms (Link Lab, 2016, online):

- Application enablement and development: Platforms that offer templates, modules or widget-based framework to establish end user applications.
- Network, data and subscriber management: platforms that simplify the connecting cellular machine to machine (M2M) data, in order to reduce building of data infrastructure.
- Device management: Platforms that monitor, troubleshoot and administrate the status of endpoints.

Figure 6 shows the functions, features and components of different platform types:

Functionalities, Features, Components by Platform Type

	Device Management (DM)	Network Management (NM)	Application Enablement/Dev (AEP/ADP)
Basic, Fairly Standard*	<ul style="list-style-type: none"> Dashboard/user interface Libraries (device drivers, protocols, adapters) Device profiling ("birth certificate," assign units to customer, device-to-asset/assoc./application/network association) Messaging (bi-directional) Device provisioning, activation, configuration Remote device decommissioning Bulk functions, actions Alerts, notifications Reporting (automated; standard or customized) History Device status check (on/off, sleeping, utilization) Location tracking/awareness (associations, neighbors), mapping Firmware check, version tracking Over-the-air (OTA) firmware and software updates, admin (scheduled/automatic, manual) Hierarchy control (by device, user, geography, technology) Search Security (settings; authentication, registration, role-based permissions, encryption, audit) User account mgmt, privileges enablement APIs/third-party s/w integration (incl. platforms) 	<ul style="list-style-type: none"> Dashboard/user interface Profile creation Provisioning, configuration Rate plan selection, plan changes Plan activation, de-activation, renewal Connection, subscription agreements Subscription, communication status Coverage map view with diagnostics Bulk functions, actions Alerts, notifications Reporting (automated, standard or customized) History (airtime usage metering, admin changes), auditing Hierarchy control (by device, user, network, geography) Search, database queries APIs/third-party software integration Message/data brokerage (support) Security (settings; authentication, registration, role-based permissions, encryption, audit, reporting) User/role administration, management SIM order (inventory) management IP Routing Check Network monitoring/performance analysis (signal strength) Connection audit (analyze, test, diagnostics) Hardware testing (remote reset, device properties overview) Troubleshooting tools Traffic routing, limiting (intelligent connection selection, optimization) Roaming network control, selection tool 	<ul style="list-style-type: none"> Dashboard/user interface Libraries (device drivers, protocols, adapters), abstraction layer Messaging (bi-directional) Data collection/capture/aggregation Data publishing to applications Data visualization (charts, graphs, gauges) Mobile apps Application hosting Runtime, rule, event processes engine(s) Technical support (built-in resources, tools) Alerts/notifications (text, email, platform messages) Reporting (automated, standard or customized) Data storage (history), database, backup APIs (open, RESTful); integration frameworks Software development kits (SDKs) Developer tools (incl. drag-and-drop data, widgets) Application management (supporting updates/changes) Security (authentication, authorization, access control) Hierarchy control (range of dimensions) User/role administration, management Edge application logic Supports integration/data sharing (outgoing) with third-party applications (via APIs) Data integration (incoming) with third-party sources (incl. enterprise systems), "mashups" with structured/unstructured data Solution templates Forms
More Advanced, Not Standard*	<ul style="list-style-type: none"> Dynamic asset grouping Inventory management, deployment workflow Remote diagnostics/device health Troubleshooting tools, debugging Multi-network communication support Automatic check-in with device, auto config Jamming detection Third-party device support Predictive maintenance, performance analysis Analytics 	<ul style="list-style-type: none"> Billing, rating, etc. Risk/demand management Automation rules (for life-cycle mgmt, incident resolution) Multi-network communication support, legacy protocol support Analytics 	<ul style="list-style-type: none"> Turnkey applications Dashboard replication Collaboration Sandbox, prototyping and app testing Compliance (monitoring, analyzing access security logs; auditing) Multi-network communication support, legacy protocol support Quality of Service (QoS) validation Cloud-to-cloud integration (ERP, MRP, MES, SCADA) Analytics (data processing, filtering, mining, querying)

Figure 6: functions, features and components of different platform types (Source: First Analytic,2016, online)

When choosing IoT platforms, these categories should be taken in consideration:

- **Deployment:** How the solution architecture should be deployed. There are two main deployments: the whole platform is powered by cloud computing or by a local computer. Cloud computing offers Platform as a Service (PaaS) or Software as a Service (SaaS). PaaS provides essential tools for the users to develop and manage the applications without taking care of configuration and maintenance of the infrastructure. SaaS is a central hosted software, which offers used product software without purchasing it.

- Licence: There are Commercial Platforms (AWS IoT, Azure) and Open Source Platform (ThingSpeak). Open Source Platforms usually offer most of their features for free and depend on the licences, users are allowed to modify the source code. On the other hand, Commercial Platforms offer limited number of services, the limitations often are time periods, data storage, number of devices. For instance, AWS IoT charged for each 1 million messages have been sent to the Cloud.
- Protocols: Most of the platform support Hypertext Transfer Protocol/ Secure (HTTP/S), which requires the devices to be powerful enough to be capable of supporting Web Protocols. Some platforms offer low-weight protocol such as Message Queuing Telemetry Transport Protocol (MQTT).
- Development Tools: Platforms bring various of libraries and Software development kits (SKDs) for specific programming languages and devices hardware.
- Scalability: Most IoT project start with small number of devices in testing stage. As the demand grows, the number of devices increases as consequence, which turns Scalability into a crucial issue. On the other hand, the Platform should support various different hardware through different networks.
- Security: With the rising of IoT devices, every aspects of these devices are vulnerable to high number of attacks, hacks and viruses. Privacy of the users is also important, as the data transmission might carries sensitive information that must not be compromised. Security must be taken in consideration as a critical issue when using an IoT Platform.

2.4 AWS IoT Platform

AWS IoT is a cloud platform that allows connected devices to interact with the provided AWS service and other devices efficiency and securely, offered by American electronic commerce, cloud computing and artificial intelligent company Amazon. Today AWS is one of the biggest cloud service providers in the world, with 90 services spreading in a wide range of features include computing, networking, storage, database and analytics

(Wikipedia, online). AWS IoT is a paid service based on the number of delivered messages. Free Tier of AWS IoT is offered for 12 months after a free sign up and the user can use up to 250,000 free messages per month.

2.4.1 AWS IoT introduction and components

Figure 7 demonstrates the architecture of an IoT system powered by AWS IoT:

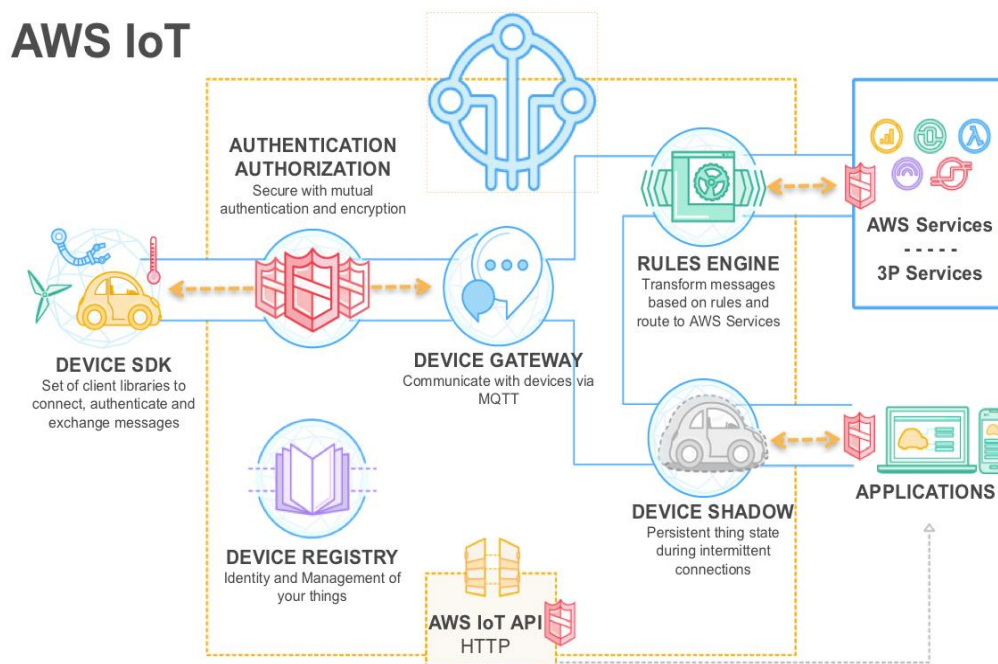


Figure 7: AWS IoT main components and how they work (Source: Amazon, online)

AWS IoT contains the following components (Amazon, online):

- **Device SDKs:** Software Development Kits that offer tools for devices to connect and exchange data with AWS IoT using either HTTP or MQTT protocols. The SDK includes C, JavaScript, Python, Android, Arduino and necessary libraries, guides for the developers.
- **Device Gateway:** central device enables devices to communicate with AWS IoT efficiently and securely. Gateway exchanges messages in publish/subscribe model. The Device Gateway supports HTTP, MQTT and WebSocket Protocols.

- Thing Registry: location where resources of devices are stored, organized. Thing is a unique identity to each device regardless of type of devices or connection method. Each Thing consist of attributes such as state of devices (Thing Shadow), Certificate, Rule.
- Thing Shadow: latest state or information of the devices, allows applications or other devices to read and interact with the main device, even when it is offline. Thing Shadow is represented as JSON file and severed as a database.

```
{  
  "state": {  
    "desired": {  
      "color": "red"  
    },  
    "reported": {  
      "color": "green"  
    }  
  }  
}
```

Figure 8: Example of Thing Shadow structure.

- desired is where users update their wanted information to the current state of the device without directly connect to it.
- reported is where the device writes in order to update its new state. Application read this area to determine the current state of the device.
- Message Broker: Secure component for Devices and AWS IoT to subscribe/publish messages with each other. HTTP or MQTT protocols are provided by the Broker for this Communication.
- Rules engine: Rules give the devices the ability to use services from AWS. Some AWS services include:
 - Amazon Simple Storage Service (S3): provides scalable storage in AWS cloud.
 - Amazon DynamoDB: NoSQL database.
 - Amazon Kinesis: collect massive amount of data and process it in real time.

- AWS Lambda: allows running code on a virtual server.
- Amazon Simple Notification Service (SNS): notification service which used mostly for mobile phone cases, allow sending or receiving messages to distributed systems or mobile devices.
- Amazon Simple Queue Service (SQS): service which orders the messages, put them into queues.

Figure 8 shows the overview and work flow of data obtained from AWS IoT to be routed to other AWS services.

AWS IoT Rules Engine

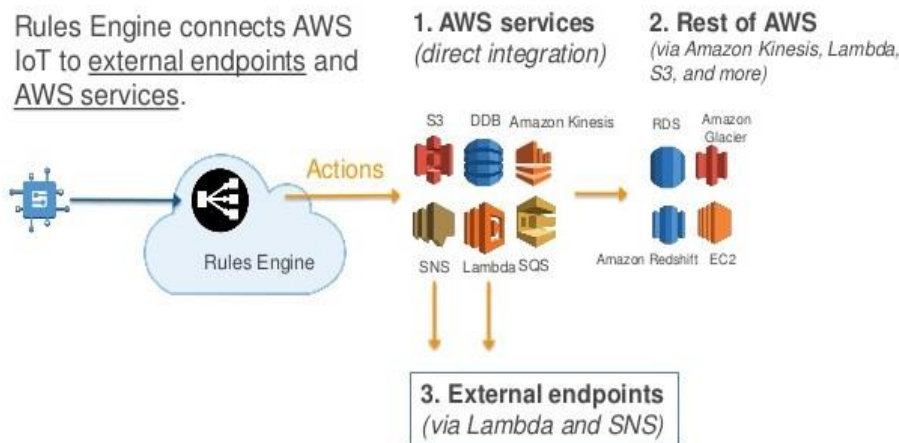


Figure 9: Overview of AWS Rule Engine (Source: ProMow)

- **Authentication and Authorization:** each device connected must have a credential in order to use the Services provided by AWS. Without proven identity, data cannot be exchanged between devices and AWS IoT. AWS IoT uses X.509 certificate-based authentication. “X.509 certificates are digital certificates that use the X.509 public key infrastructure standard to associate a public key with an identity contained in a certificate. X.509 certificates are issued by a trusted entity called a certification authority (CA). The CA maintains one or more special certificates called CA certificates that it uses to issue X.509 certificates. Only the certification authority has access to CA certificates.” (Amazon, online). These certificates can be assigned manually into each device. Policies are used by the

Message Broker in order to authenticate devices, giving devices permissions. With AWS IoT, certificates can be created, and attach policies into created certificates.

Figure 10 shows the authentication and authorization process of AWS IoT:

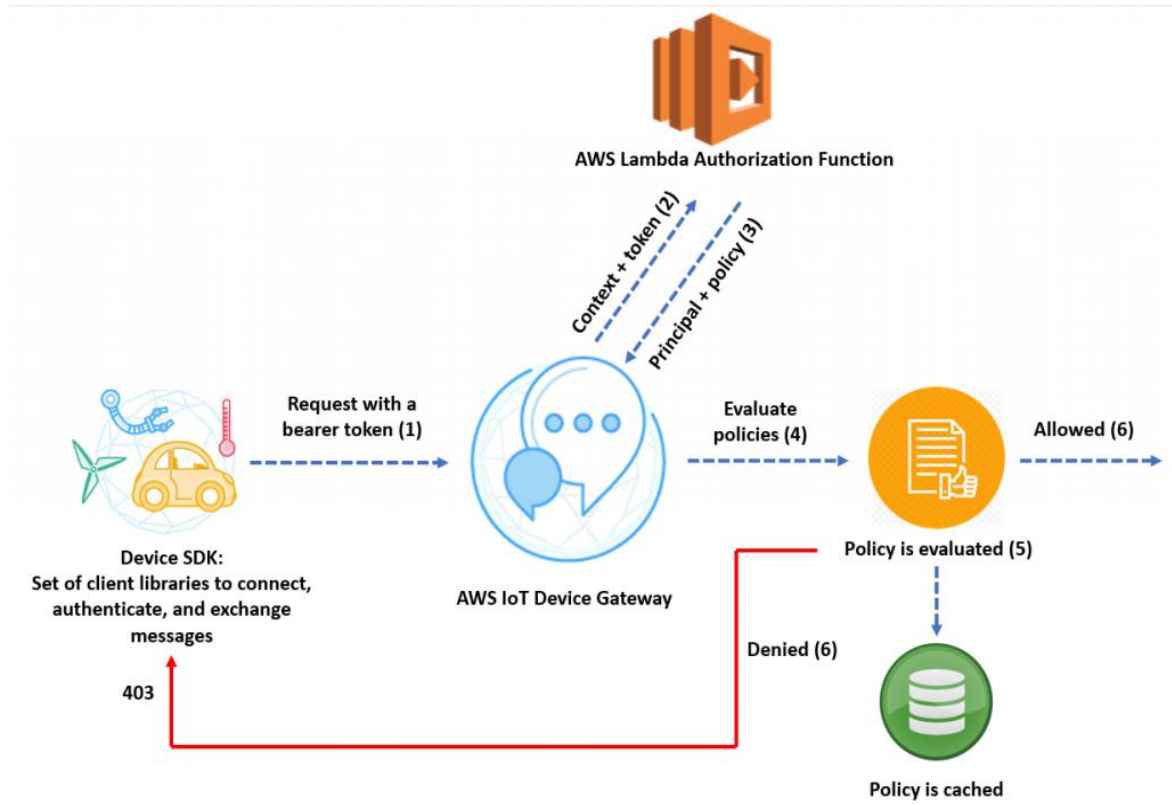


Figure 10: Overview of AWS IoT Authentication and Authorization (Source: Amazon).

2.4.2 How AWS IoT Operate

AWS IoT allows Internet-Connected devices associate with AWS Cloud, thus services in the Cloud collaborate with connected devices. These applications collect or process information from devices or allow users remotely to control the devices.

This communication is protected by a X.509 certificate. When registering a Thing, this certificate is generated by AWS IoT. The certificate files in format of .crt files needed to be downloaded and attached in the devices. When communicating with AWS IoT, these certificate files must be presented as a credential.

Devices publish messages to the IoT Gateway to report their current state in JSON format or MQTT topics. Each MQTT topics has a name that identifies the devices with an updated state. These MQTT topics are sent to MQTT Message Broker in order to be delivered to clients which subscribed to these topics. Each device has a Thing Shadow, which stores the devices 's current state. There are two entities in the Shadow: the latest reported state and the desired state requested by the applications, which is the way the applications control the devices. The Shadow responses to these requests by providing JSON format with the state information (desired and reported). After processing with the new state, the shadow sends the message to indicate that the state has been changed, which is received by the devices, which change theirs state and report back to AWS IoT.

2.5 Protocols and Communication Technique

2.5.1 MQTT protocol

Message Queuing Telemetry Transport (MQTT) is a light weight protocol, which is used to publish/subscribe for Machine to Machine (M2M) communication with low bandwidth environment. MQTT is capable of sending data at maximum of 256 MB.

MQTT protocol consists of the following components:

- Client: devices that using MQTT libraries and connected to MQTT Broker over a network. MQTT libraries are available in various of languages such as Python, C++, Java.
- Messages: the information needed to be exchanged between devices.
- Topics: hierarchy structure similar to folder, are the location where Clients can publish/subscribe messages. Topics can be nested under another Topic. Each Topic level is separated by a slash (/).
- Broker: The location where messages are received, filtered, afterward determines the subscribed clients and transfers the messages to these clients. Each client identify themselves to the Broker using ClientID. MQTT uses TCP/IP to connect to the broker.
- Quality of Service (QoS): overall performance of the MQTT service. MQTT has 3 levels of QoS.

- At most once (0): a message is sent only once and receivers do not acknowledge the delivery. This value is default.
- At least once (1): a message is resent by the publisher multiple times until the delivery is acknowledged by the receiver.
- Exactly once (2): Senders and receivers using two-level handshake to guarantee that the message can be received only once.

Figure 11 demonstrates the MQTT QoS's level and their features.

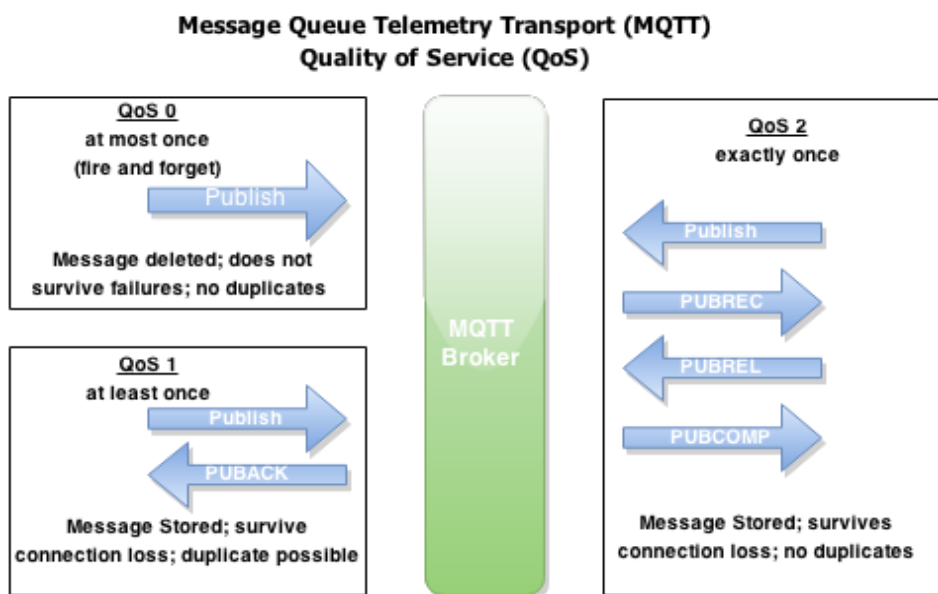


Figure 11:MQTT Quality of Service (Source: DZone)

MQTT is a publish/subscribe protocol. When a device connects to the network, it becomes a client of MQTT network. The Client can publish or subscribe messages into a topic. When the Client publishes to a topic, it is transferred to the Broker and afterward distributed to all clients which subscribed to that topic. A client can publish to multiple topics, in addition it can subscribe to a topic and publish to other topics at the same time. The subscribing client listens to incoming message and react according to the received message.

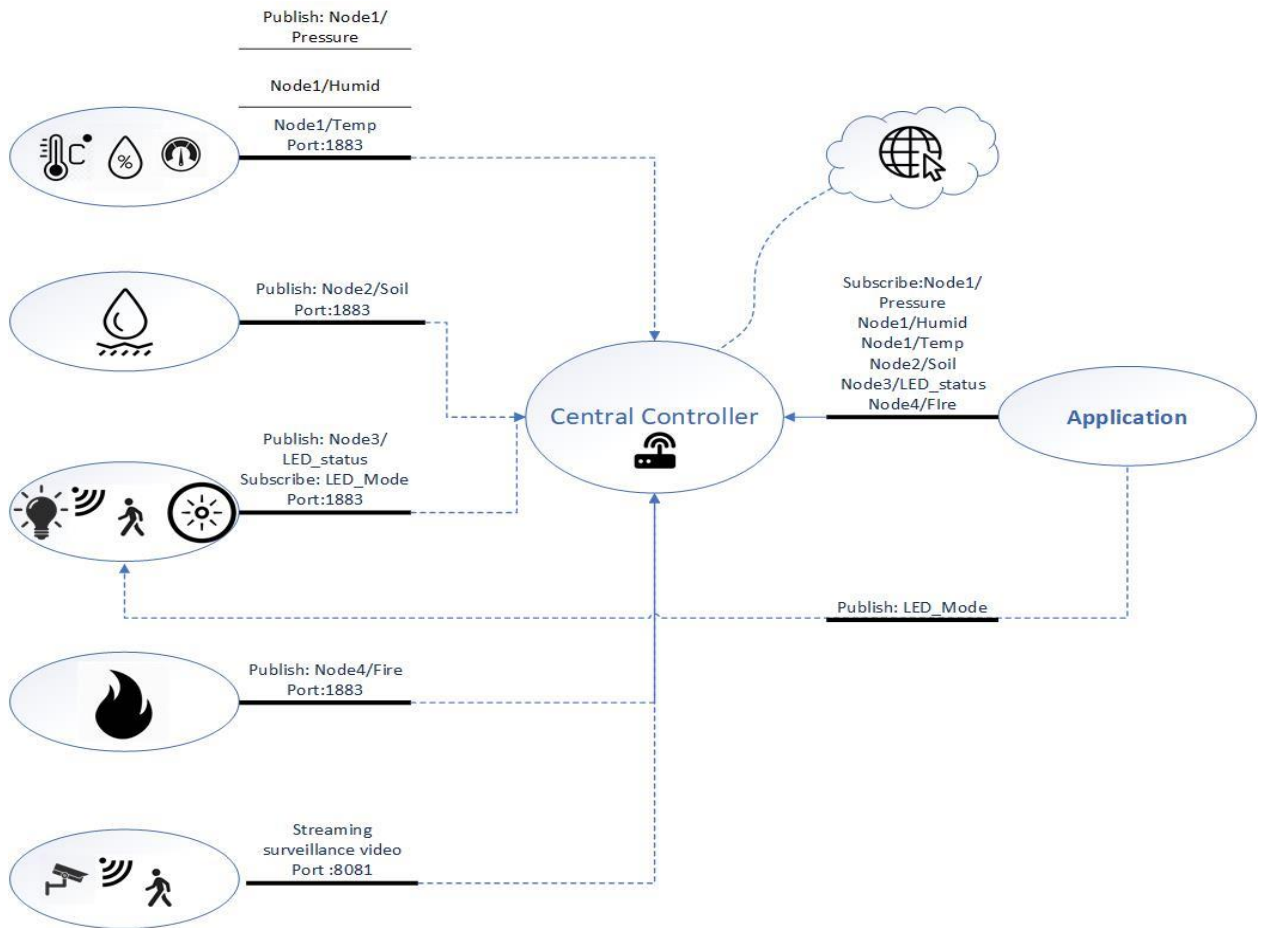


Figure 12: Example use case of MQTT.

2.5.2 MQTT libraries

In order to use MQTT, the following libraries are essential. Mosquitto MQTT is an open source which enable creation of Broker and Clients of MQTT protocol. Paho MQTT is library for MQTT features and functions implementation. Both of libraries are available in various programming languages. In this thesis, both of the libraries are used in Python Language and ran in Linux Operating System of Raspberry Pi 3.

Adafruit created PubSubClient in order Arduino-based Microcontrollers to connect and exchange data using MQTT protocol. The Microcontrollers supported include Arduino Yun, ESP8266, regular Arduino with Ethernet or WIFI shield and needed to be connected to the Internet in order to use the library. This library is written in C++ and using Arduino Language (modified C/C++).

2.5.3 LoRa Technology

LoRa stands for “Long Range”, a spread spectrum modulation technique derived from Chirp Spread Spectrum (CSS), researched and developed by Cyleo before being bought by Semtech in 2012. LoRaWAN is a media access control (MAC) protocol for wide area networks. LoRa modulation technique allows data transferring up to several of kilometres without power amplifier technology. Lora offers protected wireless communication, low power consumption, wide area network (WAN) communication protocol, ideal for various of IoT Application. However, this technique is facing numerous amount of challenges such as low data rate, radio can be blocked by certain obstacle (walls, buildings, forests).

LoRa uses Chirp Spread Spectrum modulation technique, which multiplies the data signal with a chip sequence to produce signals with bandwidth than the original data ‘s. This principle helps reducing the complexity and precision required for receiver. Many LoRa devices can exchange data on multiple channels simultaneously by using the chirp signal, LoRa signals have different chirp rates to operate in the same area without interfering with each other.

A LoRa payload consist of the following components:

- Preamble: detects LoRa Signal
- Sync Message: Detects starting of the LoRa payload
- Payload: Contains MAC command and messages
- CRC: check block errors

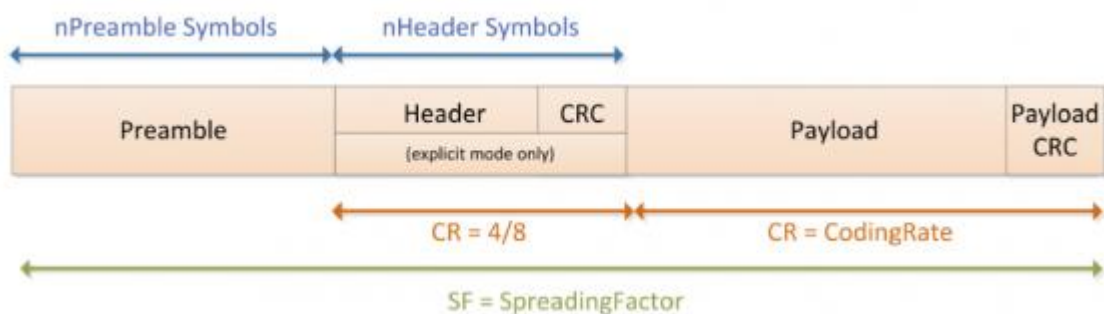


Figure 13: LoRa Package Forwarding (Source: Online)

LoRaWAN network is a star of stars topology network, developed by LoRa Alliance, enables low power, wide area communication between remote sensors and gateways connected to the network. Lora Network is consist of gateway, Network servers and End devices. Gateway is responsible for gathering data from devices (Sensors, Microcontrollers) using LoRa technique, afterward transmits the data to computers or server via TCP/IP protocol. Network Servers can be cloud based platforms such as The Thing Network, can be used for up link (from end points to application) or down link (application to end points)

Lora Network is consist of gateway, Network servers and End devices.

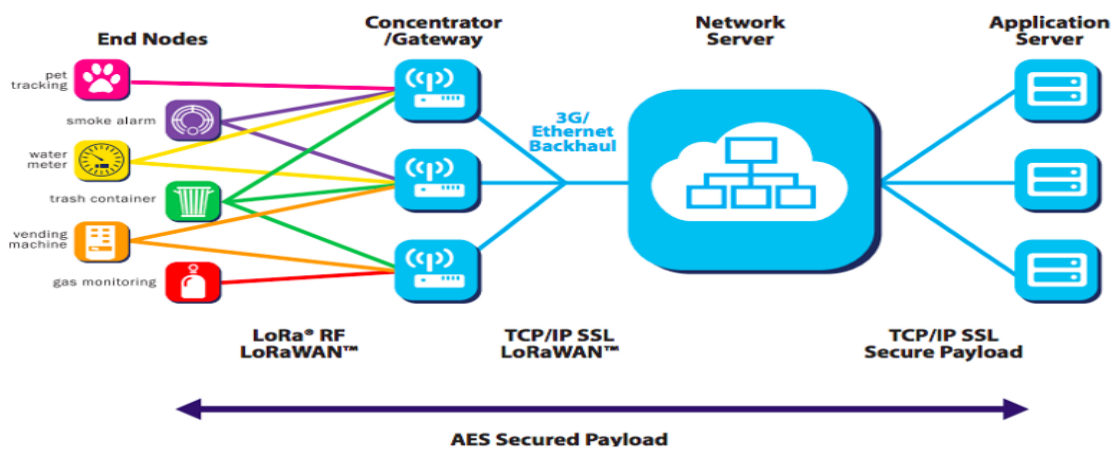


Figure 14: Overview of LoraWan Network (Source: iotvigyan)

Depending on the region, LoRa chips work in different frequencies: 433MHz (Asia), 915MHz (USA) and 868MHz (Europe).

2.5.4 UART protocol

Universal asynchronous receiver-transmitter (UART) is a hardware component that handles asynchronous serial communication. Serial communication transfers a single bit, one by one at a time. This communication uses little wiring and requires small amount of pins from the hardware to operate. UART is an Asynchronous Serial Communication, which does not use external clock signal in its data transferring process. This method is ideal for minimizing the wiring requirement, instead, the transmitting UART appends start and stop bits into its data frame.

Important aspects of UART during data transferring include:

- Data bits: the data needed to be transferred. The amount of data in a package is 5 to 9 bits.
- Synchronization bits: include start bit and stop bit. These bits define the beginning and end of the data frame in order the receiving UART acknowledges when to start reading. There is only one start bit, while the number of stop bit can be either one or two.
- Parity bits: a low-level error checking bit. There are some factors affect the data during its transferring for instance different Baud Rate between two UARTs, long distance transferring or electromagnetic radiation. To produce parity bit, the number of “1” is counted and summed. If the number is odd, the parity is set to 1, on the other hand, if the number is even, the parity is set to 0. If the parity bit matches the data, the receiver acknowledges that the transmission was error free.
- Baud Rate: define the data rate is transferred over the serial line, expressed in bits per second (bps). Both Transmitter and Receiver need to operate at the same Baud Rate. If both UARTs do not run at the same Baud Rate, data misinterpreting or missing occur.

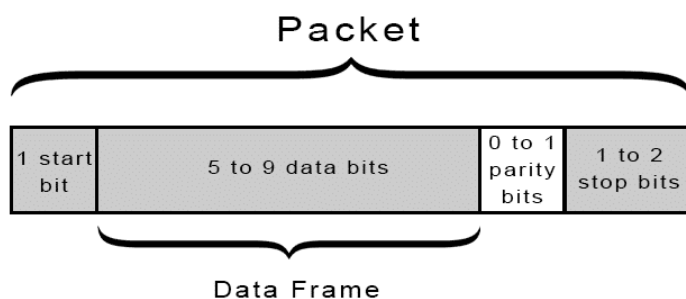


Figure 15: UART data frame (Source: circuitbasics)

Serial bus consists of two wires: sending and receiving data bus. As a result, each UART devices have two serial pins: transmitting (TX) and receiving (RX).

The transmission starts when the data is transferred from the data buses to transmitting UART in parallel. Afterward, the transmitting UART creates a package by appending

synchronous bits, parity bit altogether with the data bits. The package is transferred serially from TX pin of the transmitting UART to RX pin of the receiving UART. The receiving UART unpacks the package by removing the synchronous bits and parity bit, retrieve the data bits. Finally the data bits are transferred parallelly to the data buses.

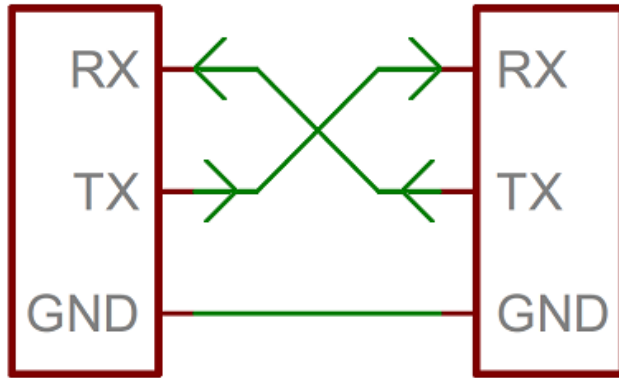


Figure 16: UART pins connection (Source: Sparkfun)

UART communication is ideal for solutions that require small wiring, which reduces the complexity of the circuits or hardware. In addition, this method does not require clock signal or parity bit to check errors. However, due to its simplification, UART does not support multiple transmitters or receivers system. Furthermore, the size of the data package is limited to 9 bits, however, data received can be put in a buffer, which will be retrieved by the receiver with first in first out (FIFO) method. Finally, both UARTs need to be the same or close Baud Rate in order to receive the data accurately.

2.6 Software

The following programming languages and tools were used in order to fulfill the software development tasks in the project.

2.6.1 Arduino Programming Language

Arduino is an open source tool for developing software for simple microcontroller boards using ATMEGA microprocessors or Arduino development kits. The software is written based on C or C++ programming language and supports all features of C/C++. Arduino

Language has an enormous number of libraries depends on the demand of users such as sensors, wiring, networking, motor controlling, timing.

Arduino IDE is used as development environment for all Arduino-based boards. This IDE simpler to develop software, compile and upload code to the board in real time. The IDE is free and can be found in Arduino Website. When uploading the code to the board, the code is translated into C language, later transferred to avr-gcc compiler, which translates into language that microcontroller able to understand. (Banzi 2011, 20-21). This IDE is available in many Operating Systems.

Arduino programs can be divided into three main components:

- Structure: includes Setup() and Loop() function
 - Setup() function is used to initialize variables, pin modes, using libraries. This function runs only once and will be reset each time the board is powered up or reset.
 - Loop() function is used to loop forever until the board is reset or powered off, control the board.
- Variable Declaration
- Functions

2.6.2 Python Programming Language

Python is a scripting, high level and widely-used programming language which excels in the development speed and code simplification, which helps programmers express their idea in much fewer lines of code and more clearly. Python is used in a wide range of application such as Web Development, Scientific, Mathematic Computing, Machine Learning, Artificial Intelligent, Graphical User Interface. Python earns prestige in some aspects such as a free and open source language, easy to learn, object oriented, cross platform portability, support applications require high performance, wide range of libraries in various fields of applications. With the assist from Python SDK from Amazon, Python is an ideal solution for developing IoT gateway connects to AWS services.

2.6.3 Android

Android is a mobile application developed by Google, based on Linux Kernel and other open source software, primarily used for touch screen smart phone, tablet. Android application can be written in Kotlin, Java and C++ languages.

In this project, Android was used to build a GUI application which visualizes the data obtain from Amazon Cloud Server in real time, displays the status of each sensor nodes either if it is operating or not, tracks the location of each nodes and controls certain mechanism such as lightning bulb, water pumping machine. Amazon provides the Android SDK, which contains certain useful features such as push notification, publish/subscribe service between application and cloud server or the nodes. Android application development was done using Android Studio IDE. This IDE supports multiple languages such as Kotlin, Java, C++, fast build time, auto generate XML files. Android Studio is one of the most popular tools to build applications for Android using devices.

2.7 Hardware Components

Node sensors and gateway were built based on the following hardware components.

2.7.1 ATmega328P Microprocessor and Arduino UNO R3 development board

The Arduino UNO is a microcontroller board based on the Microchip ATmega328P microcontroller. ATmega328P is a Microcontroller chip created by Atmel in megaAVR family. This microcontroller contains the following features: 32KB of flash memory (memory which stores the code program), 1KB of EEPROM (separated data space, whose content can be reset and reprogrammed by pulsed voltage), 2KB of SRAM (location which stores data as long as it is powered by power supply), 1 pair of serial programmable UARTs, 14 digital I/O pins(includes 6 pins for 8-bits PWM output), 6 analog input pins, 3.3V or 5V output power pins, 2 Ground pins, 16MHz crystal oscillator, USB connection, power jack and a reset button. Arduino UNO operates by using 3.3V or 5V power supply. The board can be powered by either of the following methods: USB cable (500mA at 5V) , DC barrel plug 5.5mm/2.1mm (2A at 9-12V), power I/O pins (9-12V). In this project Arduino boards are served as sensor nodes.

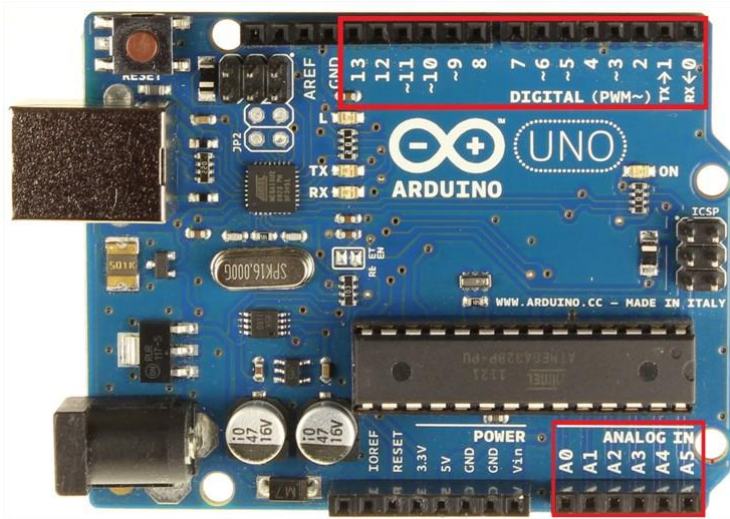


Figure 17: Arduino Uno R3 Microcontroller Board.

2.7.2 ESP8266 NODEMCU

NODEMCU is an open source microcontroller platform which includes NODEMCU firmware runs on the ESP8266 WiFi System on a Chip (SoC, circuit that integrates all components of a components or other electronic systems). NODEMCU can be programmed using LUA language or Arduino IDE. Key features of ESP8266 NODEMCU include: programable WiFi module, can be programmed by using LUA or Arduino IDE, 10 GPIO pins with PWM, I2C and SPI communication, one Analog to Digital Converter pin, micro USB port for 5V power supply, programming and debugging, integrated PCB antenna. ESP8266 NODEMCU is ideal for IoT projects, as it can be connected with external hardware via GPIO and connected to the Internet. Through WiFi and certain communication protocols, NODEMCU can served as sensor nodes, slaves to IoT gateways, or it can be controlled remotely by IoT applications. In networking, NODEMCU can be used as access point and station, hosting web server or connecting to The Internet to fetch or upload data. In this project the NODEMCU is used to control certain devices remotely using the application through MQTT protocol. In addition, NODEMCU is programmed using Arduino IDE, in order to do this process, certain firmware and libraries needed to be install.



Figure 18: ESP8266 NODEMCU

2.7.3 Raspberry Pi 3 Model B

Raspberry Pi 3 Model B is the latest product of Raspberry Pi, a popular series of micro-computer developed by Raspberry Pi Foundation. The Raspberry Pi 3B came up with appealing features: Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz Processor, 1 GB SDRAM, wireless communication includes Wifi, LAN, Bluetooth, 4 USB ports, HDMI port, 40 GPIO pins, micro SD card storage, 5V/ 2.5A power supply using micro USB port. Raspberry Pi using Raspbian, which based on Debian Linux Distribution, as its Operating System. In addition, the community contributed Ubuntu, Windows 10 IoT Core as alternative choices of Operating System. Raspberry Pi 3 supports numerous programming languages such as Python, Scratch, C/C++. Raspberry Pi is ideal for education related to embedded system or electronic, embedded system or IoT projects for what it offered. In this thesis the Raspberry Pi 3B served as an IoT gateway, receives data from sensor nodes through LORA, stores data locally under csv format, pushes data to Amazon Cloud Server through MQTT protocol and uses some services from Rule Engine. These activities were developed using Python SDK of AWS IoT.



Figure 19: Raspberry Pi 3B

2.7.4 DHT11 temperature and humidity sensor

DHT11 is a low-cost sensor that measure ambient environment 's temperature and humidity. DHT11 contains NTC (Negative Temperature Coefficient) temperature sensor (thermistor) and humidity depending resistor which has two electrodes with moisture holding substrate between them. When humidity changes, the resistance between these electrodes changes, this change is measured and processed by an IC, ready to be read by the microcontroller. To measure the temperature, when the resistance of the NTC decreases with the temperature increment, the resistance and the temperature proportional with each other in a hyperbolic curve.

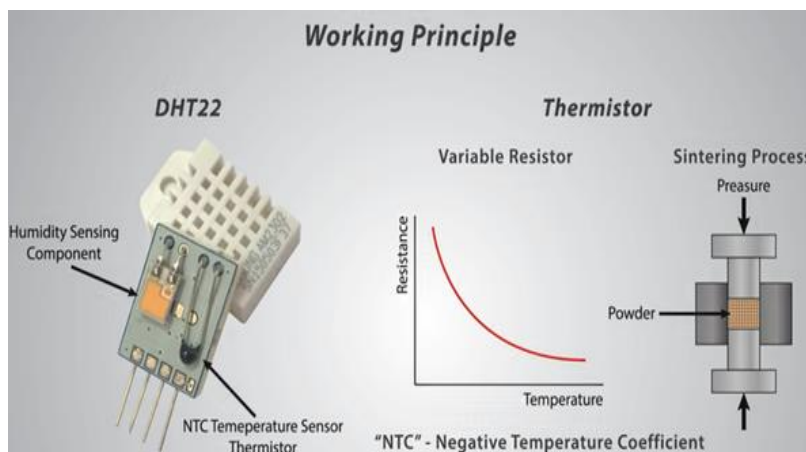


Figure 20: DHT11 temperature sensor and its working principles (Source: how to mechatronics)

DHT11 has the following technical specifications: powered by 3 to 5V power supply, great for 20-80% humidity with 5% accuracy and 0-50 °C temperature with 2 °C accuracy. In IoT or embedded projects, DHT11 is a popular choice with its low-cost and accurate measurement, the downside of this sensor is it can only read the digital signal every 2 seconds.

2.7.5 Light Intensity Sensor

Light Intensity sensor is a device that convert radiant energy from the light into electrical signal. In this project, light sensor is a Light Depending Resistor (LDR) or Photoresistor. This component changes its resistance from high in the dark to low in the light. LDR usually used by connecting it in series with a resistor and use DC voltage supply.

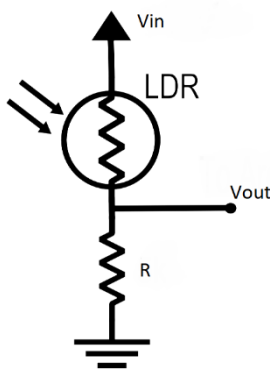


Figure 21: Connecting LDR

Using Voltage Divider we calculate the output voltage as:

$$V_{out} = V_{in} * \frac{R}{R + R_{LDR}}$$

Where V_{out} is the output voltage, R_{LDR} is the resistance of LDR and R is the resistance of the resistor connected with the LDR. From this equation R_{LDR} is calculated as:

$$R_{LDR} = \frac{R * V_{in}}{V_{out}} - R$$

However, the microcontroller read Analog values only, the output voltage is converted to Analog value as (for 10 bits ADC):

$$AnalogValue = \frac{V_{out}}{1024}$$

Finally, the illuminance of the light is calculated as:

$$Lux = \frac{500}{R_{LDR}} = \frac{500}{\frac{R * V_{In}}{V_{Out}} - R} = \frac{500}{\frac{1024 * R * V_{In}}{AnalogValue} - R}$$

2.7.6 Soil Moisture Sensor

FC-28 Soil Moisture was chosen for this project for its low-cost and simple usage. This Sensor consists of two probes to measure the volumetric of the water. The probes are designed to be embedded into soil, allow current to current pass through the soil and gain the resistance value to measure the moisture. When more moisture detected, the soil conducts more current, as the result lower the resistance and increase the moisture level. This sensor can be connected in two mode: Analog mode and Digital Mode. In this project, Analog mode was used, which used the Analog pin of the sensor. The sensor output is the value between 0-1023 because of Arduino's 10 bits ADC.

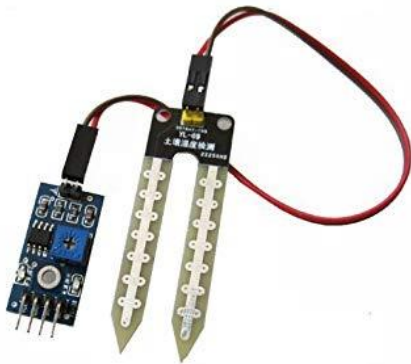


Figure 22: Soil Moisture Sensor

2.7.7 E32-TTL-100 UART Lora SX1278 433Mhz

E32-TTL-100 is a wireless transceiver module based on SX1278 chip created by SEM-TECH, which is a RF transceiver that uses LoRa spread spectrum modulation technology, provides a wide coverage range, low power consumption and good noise resistance. This module transformed the default SPI protocol of SX1278 chip into UART for simpler connection with microcontrollers, in addition this module includes UART to USB module in

order to connect with computers such as Raspberry PI or PC. However, additional software using USB configurations are required, or changing the registers inside the software manually.



Figure 23: E32-TTL-100 UART Lora SX1278

This module contains the following features:

- Main processor: SX1278
- Power supply: 2.3-5.5V
- Maximum transmission distance: 3km (ideal environment without obstacle)
- Transferring rate: 2.5Kbps
- Working frequency: 410MHz-441MHz (default 433 MHz)
- Communication Interface: UART, 1200-115200 Baud Rate

2.7.8 Solar Panel power supply

When building an IoT solution, power supply is one of the most important aspects, as the devices need to run constantly for long time, consume less power and they do not need to continuously change or charge the battery. As such, Solar Panel rise as one of the best options, since Vietnam is a tropical country with long day light and high intensity sunlight. The Solar Panel 9V 2W has a transparent surface, good light conduction by heat-

resistant glass, battery material made of silicon for high conversion efficiency, especially effective in low light environments.



Figure 24: 9V 2W Solar Panel

This panel has the following features:

- Output Voltage: 9V
- Load Current: 170~220mA
- Maximum Power Output: 2W
- Dimension: 115 x 115 x 2mm

This Solar Panel was used to charge numerous 3.7V chargeable Lithium Battery to be used at night. In order to archive this goal, TP4056 standalone Linear Li-Ion Battery Charger was used. This module has the following specifications:

- Input voltage - 5V via microUSB or solder pads on left-hand side of module
- Full charge voltage - 4.2V
- Charging current - 1A by default.

The following figure demonstrates the connecting between Solar Panel, TP4056 and the battery.

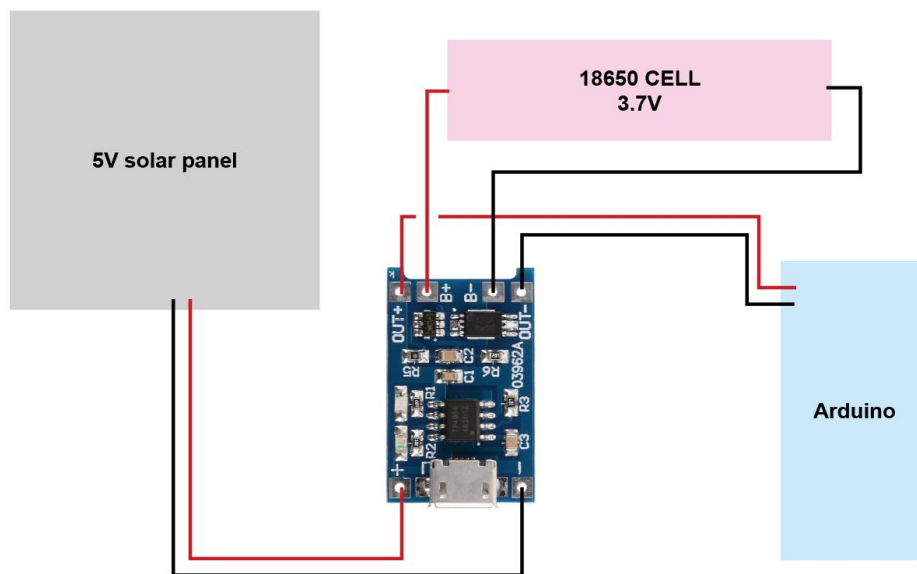


Figure 25: Power Supply for Node Sensor

The charging of the Lithium batteries using TP4056 divided into two stages:

- The batteries are charge using the current-limited power supply. At the end of this stage when they reach the full voltage, the battery charges are 70-80%.
- At their full voltage stage, the charges severed as voltage-limited power supply, the voltage remains at maximum, while the current drop, until it reaches 10%, the batteries peaked their full capacity.

3 IMPLEMENTATION

3.1 Infrastructure

In this project, similar sensor nodes were created and placed in different places. Each node contained different kinds of sensors (Temperature and Humidity, Soil Moisture, Light Intensity). These nodes obtained data of ambient environment, encrypted, put them into data frames appropriate for the requirement of the wireless transmitting method, and pushed these data frames into the gateway. LoRa method was used for long-ranged and WIFI for short-ranged communication between the nodes and the gateway. In the gateway, data frames were decrypted, managed that they were the same as the sensors 's. In addition, Data was saved locally into CSV files. Through MQTT protocol, the data was transported to the Cloud Server of AWS IoT platform. In there it is analysed, visualized by graph in real time, saved into JSON files called "Thing Shadow". Finally, an Android Application was created with monitoring purpose on the data obtained from the Cloud server. Moreover, this application also checks the status of each node, for instance, if the node was online or not, if the value of the sensors exceeds the threshold, GPS allocation and device control such as Water Pumping Machine.

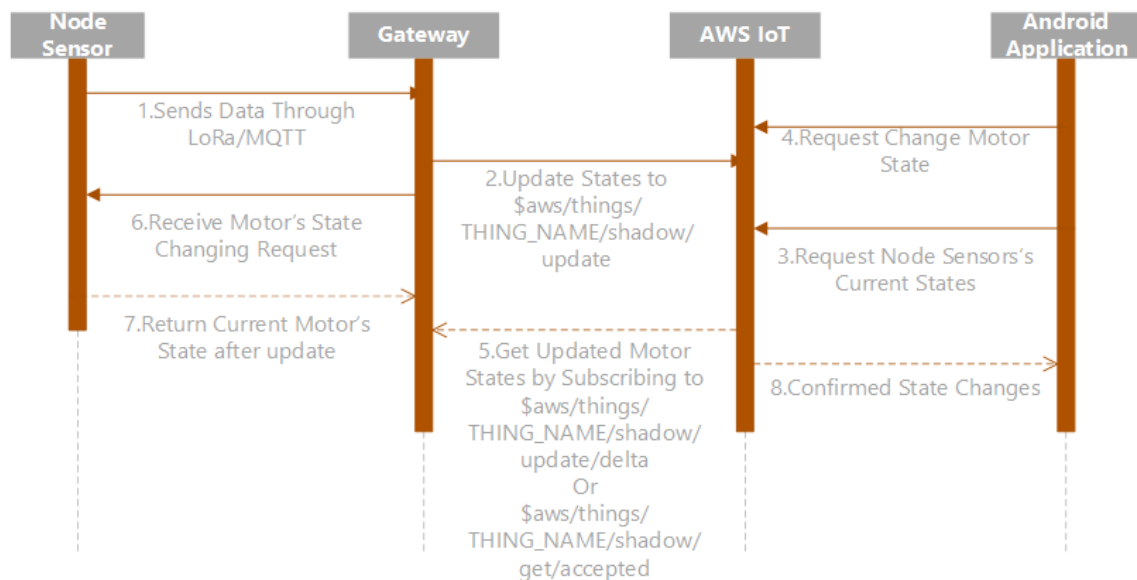


Figure 26: Data Flow Sequence of the System

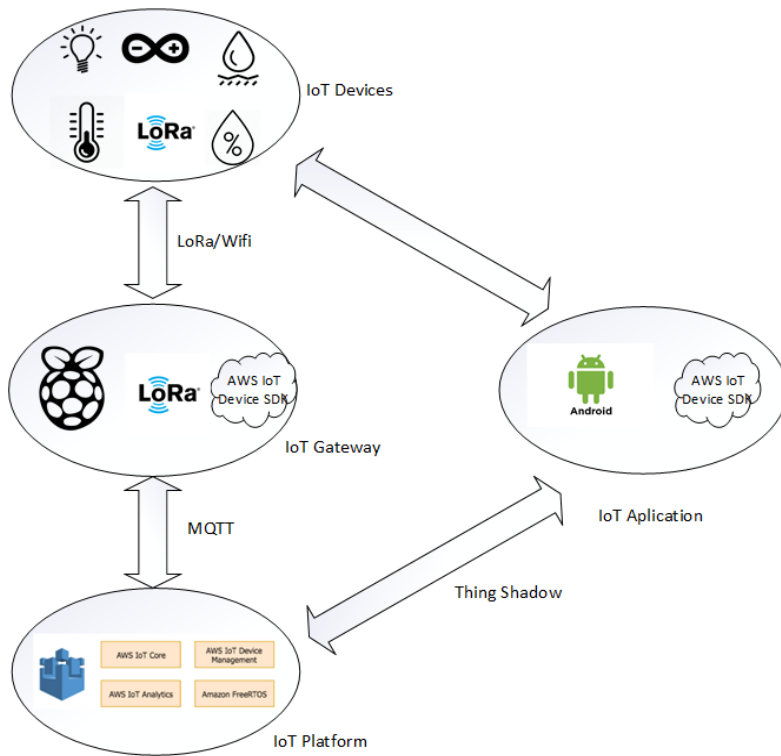


Figure 27: System Architecture

3.2 IoT device

3.2.1 Hardware Circuit Diagram

Table 1 demonstrates the connection between Arduino Microcontroller and E32-TTL-100 LoRa Module:

Arduino Pins	E32-TTL-100 Pins
Digital Pin 3	M01
Digital Pin 4	M1
Digital Pin 6	RX
Digital Pin 7	TX

Table 1: Pin connection between Arduino board and LoRa Module.

Figure 28 and 29 demonstrates the overall diagram of the device node:

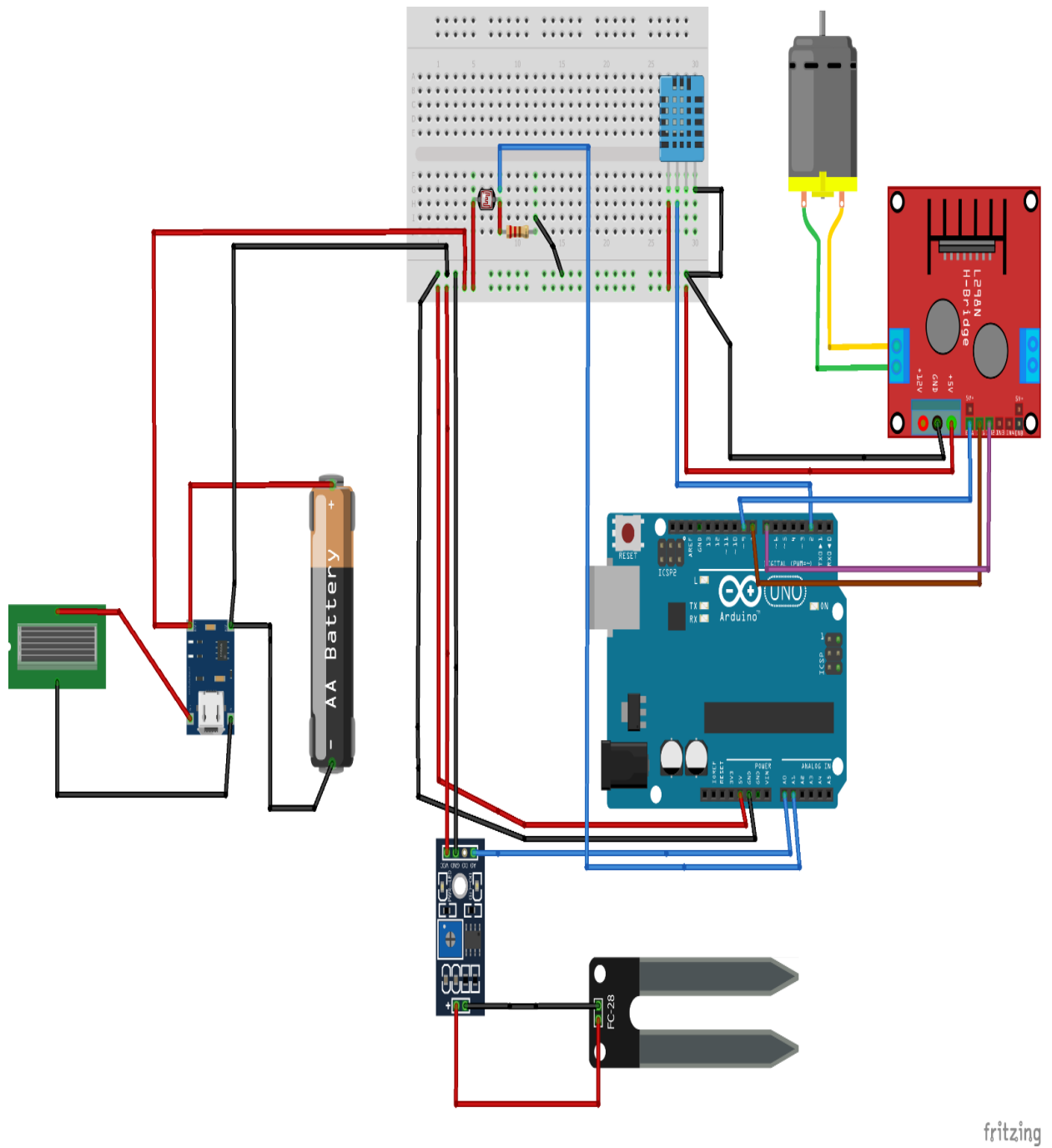
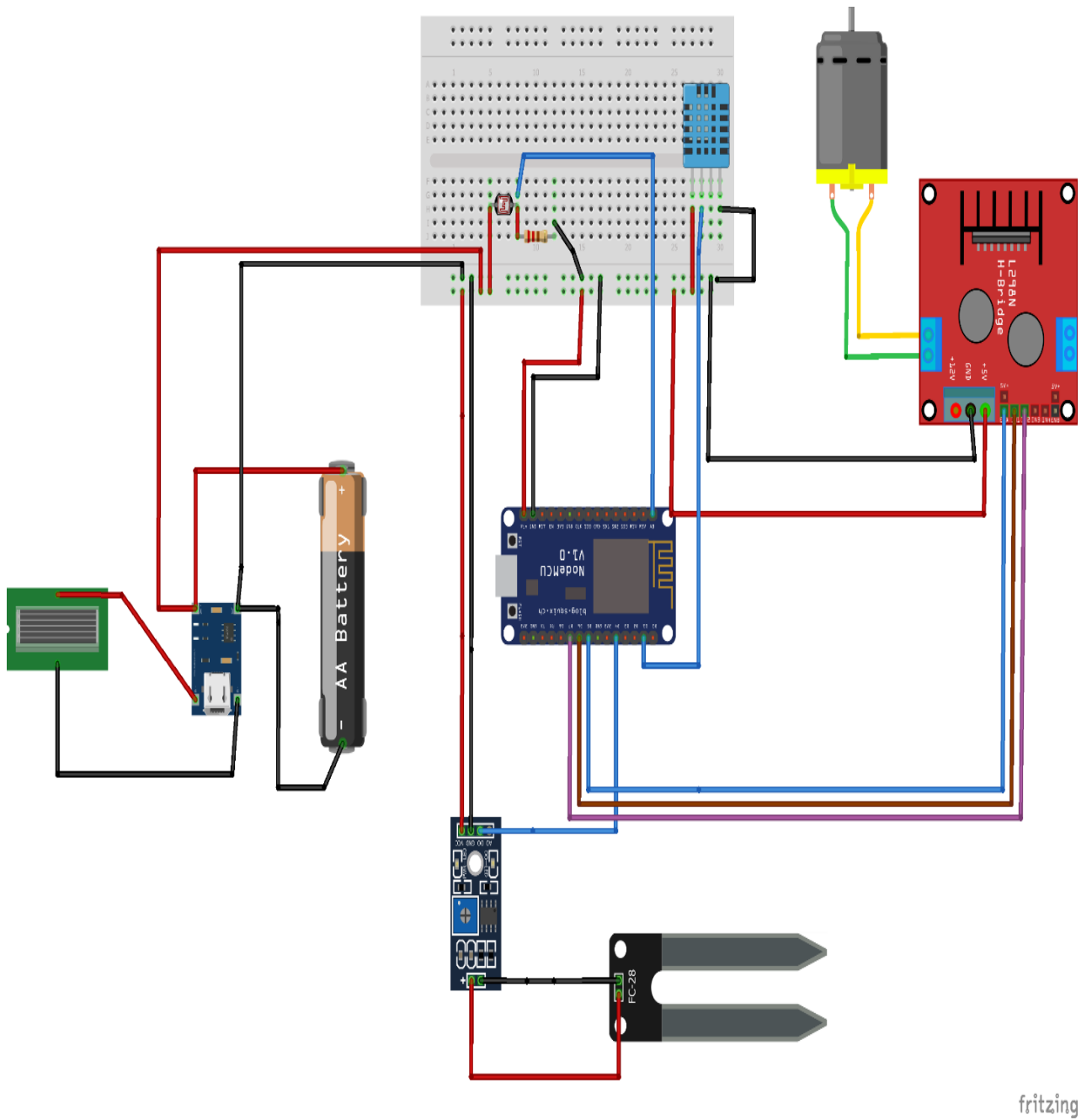


Figure 28: Node sensor circuit



fritzing

Figure 29: Node Sensor Short-Range Communication

3.2.2 Software

The software for the device nodes contains the following functions:

3.2.2.1 Obtaining Temperature and Humidity sensor 's data using its datasheet.

The DHT11 has 4 pins, which two are power supply, one is used for data transferring.

Figure 30 demonstrates the working principle of the sensor:

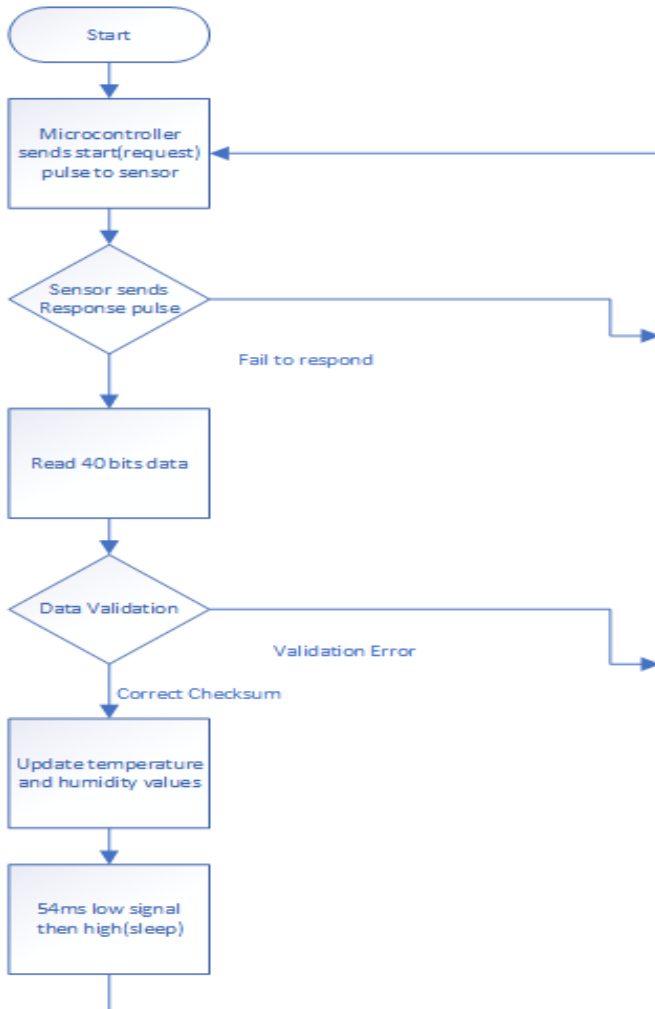


Figure 30: Flowchart of the Software of DHT 11 sensor.

The communication between this sensor and the microcontroller is divided into three phases:

- **Start Pulse (Request Phase):** The Microcontroller sends start pulse to the DHT11 sensor, which is produced by setting the data pin to low for 18 to 20 us and then high afterward.

- **Response Phase:** After getting the Start Pulse, DHT 11 sends Response Pulse which notifies the Microcontroller that DHT11 has received the Start Pulse, ready to transfer data. The Response Pulse is set to low for 54us and goes high for 80us
- **Data Transferring:** After sending Response Pulse, DHT11 sends data frame, which contains 40 bits long, divided into five 8 bits segments of the Temperature, Humidity and Checksum. The first two segments are the Humidity value in relative percentage value, the following two segments are the Temperature value in Celsius. The final segment is the Checksum bit (Parity bit), which is compared with the sum of remaining segments, if the result is equal, therefore the data is correct and being transferred to the microcontroller, else it is discard and the communication restarts from the beginning. Once the data is received, DHT11 pin goes to low power consumption mode until next start pulse DHT11 sends 54us and afterward goes high. After this time frame, DHT11 enters sleep mode until next communication.

To accomplish the communication, the software carries out certain tasks.

Define a pin as input for the sensor, in this case PD2 of PORT D.

- Create Request() function, which cycles selected pin from low to high in 20ms.

```
DDRD |= (1<<DHT_PIN);
PORTD &= ~(1<<DHT_PIN); //set to low pin
_delay_ms(20);           // wait for 20ms
PORTD |= (1<<DHT_PIN); // set to high pin
```

- Create Response() function, which receive response pulse from the DHT11

```
DDRD &= ~(1<<DHT_PIN);
while(PIND & (1<<DHT_PIN));
while((PIND & (1<<DHT_PIN))==0);
while(PIND & (1<<DHT_PIN));
```

- Create Receive_Data() function to receive 40 bits data from the sensor. . During the communication, “0” and “1” bits are received in the way, that pulse goes low

for 54us, afterward goes high. If the pulse goes high for more than 30us, bit “1” is received, otherwise bit “0” is received. Each time this function is called , one segment of 8 bits data is received.

```
for (int q=0; q<8; q++) //loop for each 8 bits segments
{
    while((PIND & (1<<DHT_PIN)) == 0); // check if received bit 0 or
1
    _delay_us(30);
    if(PIND & (1<<DHT_PIN))// if high pulse is greater than 30ms
    c = (c<<1) | (0x01); // "1" bit received
    else // "0" bit received
    c = (c<<1);
    while(PIND & (1<<DHT_PIN));
}
return c;
```

- Finally compare the sum of the data with the checksum to see if its correct, otherwise return an error.

In Arduino, this sequence is simplified and contained inside the DHT library. This Library is available to be used for both DHT11 or DHT22, which is a slightly improve version of DHT11. To receive the data, simply create an object of DHT class, define two variables of Temperature and Humidity data, and call the method to receive the data within the class.

```
h = dht.readHumidity();
t = dht.readTemperature();
```

3.2.2.2 Obtaining Soil Moisture and Light Intensity Sensor ‘s data using ADC register.

These two sensors work similar to each other: Each sensors has an output pin to communicate with the microcontroller. The output voltage can be processed through an ADC and used according to need of application.

The Atmega328P has 10 bits ADC. Converted output binary data is held in two special function 8-bit register ADCL (result Low) and ADCH (result High).

To enable the ADC of the microcontroller, certain registers are needed to be set:

- **ADCH:** ADC data higher byte
- **ADCL:** ADC data lower byte
- **ADMUX:** ADC Multiplexer selection register. The first 2 bits define Reference Selection Bits, in this case these bits are set to 01 (Vcc 5V). The last 5 bits defined the channel for the ADC conversation. For reading Soil Moisture sensor, these bits are set to 00000 (ADC channel 0), while reading Light sensor they are set to 00001 (ADC channel 1)
- **ADCSRA:** ADC Control and status register. Important bits needed to be used for the conversation: ADSC (second bit) start the conversation, ADIF (forth bit) enable ADC interrupt flag, is set when the conversation is completed and the Data Registers are updated.

To accomplish the communication, the software carries out certain tasks.

- Create Init() function to enable the ADC

```
DDRA=0x00;           // Make ADC port as input
ADCSRA = 0x87;       // Enable ADC, fr/128
```

- Create Analog_Read() to read the data from the sensor

```
Int low,high,out;
ADMUX = 0x40;           // Vref: Avcc, ADC channel: 0 for soil moisture
sensor
// ADMUX=0x41, ADC channel 1 for light sensor
ADCSRA |= (1<<ADSC);   // start conversion
while ((ADCSRA &(1<<ADIF))==0); // monitor end of conversion interrupt
flag
ADCSRA |=(1<<ADIF);    // set the ADIF bit of ADCSRA register
low = (int)ADCL;       // Read lower byte
high=(int)ADCH*256;    // Read high byte
out=low+high;
return(out);
```

- Use the Analog_Read() to convert the raw value into desired value.

```
adc_value = ADC_Read(); // Copy the ADC value
moisture = 100-(adc_value*100.00)/1023.00; // Calculate moisture in %
light=500/((1024*R*Vin/adc_value)-R); //discussed in chapter 2
```

In Arduino, the communication is simplified and contained inside `analogRead()` function. To convert ADC value to % format, `map` function was used. In order to change to 0-100% format, Arduino's `map` function was used. The function is used as following:

$$\text{map}(x, in_{min}, in_{max}, out_{min}, out_{max})$$

Where x is the current value read by Analog pin, in_{min} to in_{max} is from 0 to 1023 in our case, the current's range, out_{min} to out_{max} is the desired range we want to convert to, in this case from 0 to 100. The final function is written as:

```
l=analogRead(LDR);           //LDR pin A1, set as input
m=analogRead(moist);         //moist pin A0, set as input
m = constrain(m,485,1023);
m = map(m,485,1023,100,0);
```

3.2.2.3 Controlling the Water Pumping (DC motor).

In this project, as an experiment, the real Water Pumping was replaced by a DC Motor for simplification. DC motor converts Electronic energy in to Mechanical energy through electromagnetic field inside the motor. In order to control the motor, Pulse Width Modulation technique (PWM) was used to control the speed of the motor by adjusting the average voltage applied to the DC motor by means of square pulse. The speed of the motor is determined by the Pulse Width, the large the Width the higher of the speed, this value can be determined by the Duty Cycle. In this project Fast PWM mode was used. Certain Registers needed to be adjusted to enable PWM:

- TCCR0: Timer Counter Control Register 0. Generate Fast PWM by setting bit 6 and 3 (WGM00 and WGM01) to 1. For Example: set fast PWM mode with non-inverted output

```
TCCR0 = (1<<WGM00) | (1<<WGM01) | (1<<COM01) | (1<<CS00);
```

- OCR0: setting the duty cycle. Atmega328P has 8 bits PWM, as the result the maximum value is 255. For Example 25 % duty cycle:

```
OCR0=64;
```

The rotation direction can be changed by altering the voltage polarity. This can be done by using L293D H-bridge motor driver IC. The L293D IC receives signals from the microprocessor and transmits the relative signal to the motors. Its two voltage pins, one of which is used supply voltage for itself and the other is used to apply voltage to the motors.

The L293D has 3 more pins to control the Motor: 2 pins are for Motor's Rotation Direction and the other pin receive PWM signal from the Microcontroller, controlling Motor's Speed. The L293D switches its output signal according to the input received from the microprocessor. The L293D can supply the desired voltage for the DC motor to operate, despite the fact microcontrollers might supply insufficient for the motor.

In Arduino, to create PWM signal, assign a pin as output and use function `analogWrite()`. From the figure 29, enable pin of L293D is connected to Digital pin 10, 2 motor control pins are connected to Digital pin 9 and 8. To control the direction of the motor set these 2 pins to HIGH and LOW corresponding by using `digitalWrite()`. In this project, the motor is controlled by Android Application, when pressing "On" button, the motor start running at 50% Duty Cycle, when "OFF" button is pressed, the motor stops.

3.2.2.4 Preparing Data Frame, Data transferring, waiting for the signal from the gateway to be enabled to send data.

The data frame needed to be transferred consisted of the following components:

- The first five bytes are the checking bytes of the Node. The Node is allowed to send data if these bytes match the format from the gateway.
- The following byte holds the ID of the Node.
- The next 8 bytes contain the value of corresponding sensors: Light Intensity, Soil Moisture, Humidity and Temperature.
- The last byte is the status of the motor.

In order to enable UART communication, certain registers needed to be considered:

- **UDR:** USART Data Register, contain 2 register Tx for transmitting and Rx for receiving. Buffer uses FIFO shift register to transmit the data.
- **UCSR:** UART Control and Status Register. Enable the communication by setting RXEN and TXEN bit of UCSRB register to 1. To enable 8 bits data, URSEL of UCSRC is set to 1.

- **UBRR:** UART Baud Rate Register

To transmit the package, monitor the UDRE bit from UCSRA. When UDRE flag becomes one, which indicates transmitting buffer is empty and ready to accept another byte. To receive the data, monitor the RXC bit in UCSRA, RXC bit indicates receive complete status.

In this project, a pair of E32-TTL-100 LoRa module was used to wirelessly transfers data from a Node Sensor to the Gateway. Before the transmission, E32-TTL-100 LoRa module need to be set up, by using Wireless Module Setting Software. This software is used to modify parameters of the LoRa module, in order to be functional as desired of the user.

Figure 31 demonstrates the overview of the user interface of Wireless Module Setting Software for configuring LoRa Modules.

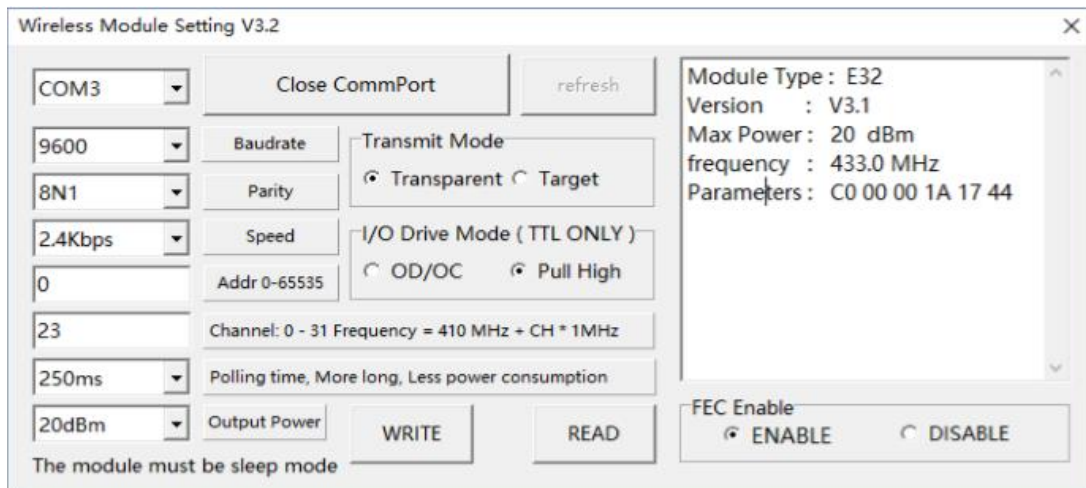


Figure 31: Wireless Module Setting Software

The parameters include:

- **HEAD:** frame data control controlling commands. This parameter can be either 0xC0 to save other parameters to EEPROM when power is cut, or 0xC2 to not save other parameters when power-down.
- **ADDH and ADDL:** High and Low address byte of module.
- **SPED:** rate parameter, include Baud Rate, air data rate.
- **CHAN:** operation frequency.

- **OPTION:** include transmission mode, I/O drive mode, Wireless Wake up time.

By default these parameter was set to C0 00 00 1A 17 44, which means 433MHz frequency, address is 0x0000, using channel 0x17, having 2.4kbps air data rate, 9600 Baud Rate, 8N1 parity and 100mW Transmitting power. In this project these parameters were kept the same as default. The two modules were set to be both Read and Write, Target transmission mode, as such the modules exchange data within the desired address and channel. In order another Node Sensor desired to communicate with the Gateway, addition pair of LoRa module in different address and channel need to be deployed, as such prevent data collision between Node Sensor's data transmission.

The 2 pins M0 and M1 of the LoRa chip needed to be set to 00 in order to work in normal mode, which Serial port open, wireless channel open, transparent transmission. When these pins are set to 11(Sleep Mode), the module turns into sleep mode, transmission stops and certain parameters can be modified.

In Arduino, UART communication is done by SoftwareSerial Library. First of all, Tx and Rx pins needed to be set by invoking mySerial() method from the library as follow:

```
SoftwareSerial mySerial(6, 7); // RX, TX
```

Transmission Baud Rate is set to 9600 using begin() method of the library. sendData () function was created in order to transmit the data, by focus on the print() method of the library

```
for(int i = 0; i < len; i++)
{
    mySerial.print(buff[i]);
    mySerial.print(' ');
}
mySerial.println();
```

UART Receiver Feature was used for two purposes: controlling the motor and control the data transmission. LoRa Module might need to shut down in Sleep Mode to change certain parameters for desired Operations of the users. readSerialString() function was created in order to read the signal from the gateway and determine certain actions:

- **ENABLE_DEVICE:** if this flag is received, the transmission work as normal by setting M1 and M0 pins of LoRa Module to 0 and 0

- `DISABLE_DEVICE`: if this flag is received, the transmission halted by setting M1 and M0 pins of LoRa Module to 1 and 1 (Sleep Mode)
- 1: this flag enables controlling the motor in manual mode, turn on the motor with 50% duty cycle.
- 0: this flag enables controlling the motor in manual mode, turn off the motor.

`Serial.read()` function only read a single bit at a time, it should be put into a loop to read the whole string, by making `readSerialString()` function:

```

if(mySerial.available()) {
    while (mySerial.available()){
        sb = mySerial.read();
        serInString[serInIndx] = sb;
        serInIndx++;
    }
    return serInString;
}

```

3.2.2.5 Short range communication case

For shorter range (300-500m), the solution was simpler than the longer range. The Node works the same as the long-range solution, however the communication protocol changed to be MQTT instead of UART. The Node use ESP8266 NODEMCU in place of AT-mega328P for its WiFi connection. In this case Sensor Node and Gateway need to be in the same network.

In order the NODEMCU can be programmed in Arduino IDE, certain library needed to be included. First of all in Arduino IDE, select File>Preference, In “Additional Boards Manager URLs” field, put in http://arduino.esp8266.com/stable/package_esp8266com_index.json and confirm. The next step is navigating to Tools>Board>Board Management, find "esp8266 by ESP8266 Community” and install. Finally choose the appropriate board by selecting Tools> board, choose NodeMCU 0.9 (ESP-12 Module).

To start programming the NodeMCU and MQTT service, `ESP8266WiFi.h` and `PubSubClient.h` library must be included. In order to connect to WiFi, certain parameters are required

- ssid: the name of the WiFi
- password: the WiFi's password
- mqtt_server: the IP address of the gateway(or MQTT broker)
- Optional: mqtt_username and mqtt_password if the broker set those parameters.

Setupwifi() function was created to setting and connecting to the WiFi with some important parameters:

```
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
```

Reconnect() function connects the nodes and gateway through MQTT protocol. If the connection failed, it enters an infinite loop every 5 seconds and indicate an error, until the connection is successful.

```
void setup() {
  client.setServer(mqtt_server, 1883);
}
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    WiFi.mode(WIFI_STA);
    if (client.connect("ESP8266",mqtt_username,mqtt_password)) {
      Serial.println("connected");

    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

Callback() function was created to display the incoming message from the gateway through MQTT by subscribing to “Motor_Status” topic from the gateway , check the bit of the payload:

- If “1” is received, the motor runs at 50% duty Cycle
- If the bit is “0”, the motor is turned off.

```

void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message arrived [");
    Serial.print(topic);
    Serial.print("] ");
    for (int i = 0; i < length; i++) {           //read message arrived from MQTT
        Topic
        Serial.print((char)payload[i]);
    }
    if((char)payload[0] == '1'){
        analogWrite(Motor_Pin,127); //motor runs at 50%DC
        status="1" //update motor's status
    }
    if((char)payload[0] == '0'){
        analogWrite(Motor_Pin,0);
        status="0" //update motor's status
    }
}
}

```

Data gathering from the Sensor's functions are the same as the regular case, using Arduino programming language. The Node put the data obtained from the sensors and put data into "Topics" with the name format: "nodename/sensordata" and using publish() method from the PubSubClient library to transfer the data to the gateway. For instance, sending temperature from "node2":

```
client.publish("node2/temp",
String(temp).c_str())
```

3.3 Gateway and Cloud

3.3.1 AWS IoT Thing set up

Before the gateway is ready to be operated, a Thing must be created on Thing Registry of AWS IOT. Thing represents the device. This will be where the state of the device is stored (Thing Shadow. Each Thing will be associated with one or more Certificate and Rule. In order to create a Thing, log in to AWS IoT Console, choose Manage to expand the choice, afterward select Things and Create a single Thing, finally name the Thing. In this project, Node Sensor using Arduino as Microcontroller has "Node1" as Thing's Name, "Node2" is Thing's Name of Node Sensor using ESP8266 as Microcontroller.

The certificate is where the authentication certificates (*.crt files) will be managed. Devices need to use these certificates with the SDK to connect to Thing Shadow via MQTT / RestFul. A Certificate will be associated with one or more Thing / Policies that allow devices customize the operation, as well as policy-based permissions. To create a certificate, logging to AWS IoT console, select Secure>Certificate>One-click certificate creation. This action generates some certificate files: certificate for things, public key, private

key and root CA. These files needed to be download and attach as necessary parameters inside gateway's software. Afterward return to Certificate tab and active the Certificate in order it can be used.

Policies allow devices permissions to access AWS IoT resources, including MQTT Topics, Device Shadows and other Things. Using Policies controls the data from devices and strengthen the security of the IoT solutions. To create a policy, logging to AWS IoT console, select Secure>Policy>Create a policy and give the policy a name. In this setting, under Add statements, there are two parameters which need to be filled, which define the types of actions can be performed by a resource:

- Action: certain policies action and respective resources
- Resource ARN: depends on the action, Resource can be one of the following
 - **Client** – arn:aws:iot:<region>:<accountId>:client/<clientId>
 - **Topic ARN** – arn:aws:iot:<region>:<accountId>:topic/<topicName>
 - **Topic filter ARN** – arn:aws:iot:<region>:<accountId>:topicfilter/<topicFilter>

While creating the Policy, “*” can be specified as wildcard of “anything” or “everything” to use all of the resources allowed. The final policy should be:

```
{
  "Version": "2018-04-20",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*"
    }
  ]
}
```

Finally, in order to be used, Policy need to be attached into a certificate, as well as attaching the certificate to the Thing. This action can be done by navigate to Secure>Certificate, select the activated Certificate, choose Attach Policy. The same procedure can be applied for attaching Certificate to Thing.

3.3.2 Gateway

After the Thing on AWS IoT is prepared, the gateway is ready to function. First of all, certain libraries need to be included:

- serial: Serial communication library.
- csv: csv file activities library.
- json: json file activities library.
- paho.mqtt.client: mqtt broker and client library.
- ssl: Secure Sockets Layer, library which creates secure connection between client and server

In order to use MQTT service to communicate with AWS IoT, some parameters needed to be filled:

- End Point: the host that gateway desired to communicate with. To retrieve the End Point, log into AWS IoT console, select device>interact. The End Point is in the format of: XXXXXXXXXXXX.iot.<region>.amazonaws.com
- keyPath, certPath, caPath: certificate files downloaded during certificate creation, including corresponding Thing_Name.private.key, Thing_Name.cert.pem, root-CA.crt
- ClientID or ThingName: name of the Thing registry on AWS IoT.
- Port number for MQTT connection. In this case 8883 default port was used.

After all required parameters are present, the Gateway is ready to connect to AWS IoT through MQTT protocol by using connect() function from paho MQTT library:

```

mqttc = paho.Client ()
mqttc.tls_set(caPath,certfile=certPath,keyfile=keyPath,cert_reqs=ssl.CERT_REQ
UIRED,tls_version=ssl.PROTOCOL_TLSv1_2,ciphers=None) # pass parameters
mqttc.connect(awshost, awsport, keepalive=60) # connect to aws server
mqttc.loop_start ()

```

The E32-TTL-100 UART Lora SX1278 module interacts with the Gateway through the Raspberry Pi 's USB Interface. In order to receive data transferred from Sensor Nodes, the Gateway reads incoming message of any USB port available using Serial(port_name, Baud Rate) method from serial library. Each Node communicates with the gateway through different USB ports, as such avoiding data collision during transportation. For

instances Node1 sends data through ttyACM1 port, while Node2 through ttyAMA0. Afterward the Gateway reads bytes one by one, if the first 5 bytes are the same with the desired number's combination, the transportation carries on with the remaining package, otherwise the transportation stops and returns to the beginning, until the first 5 checking bytes match. The data is retrieved as the following pseudo code:

```
port = serial.Serial("/dev/ttyACM1", baudrate=38400, timeout=0.5) #read USB
port
def split(str): #function to split string
x=str.split(" ")
return x
data=str(port.readline()) #read USB port's data line by line
x=split(data)
ID=x[6] #ID of node
light=(int(x[7])*256+int(x[8])) #light
moist=(int(x[9])*256+int(x[10])) #soil moisture
humid=(int(x[11])*256+int(x[12])) #humidity
temp=(int(x[13])*256+int(x[14])) #temperature
status=int(x[15]) #status of motor
```

The data is saved locally into the Raspberry Pi in csv format by using csv library:

```
csvresult = open("/home/pi/data.csv","a")
csvresult.write("{0},{1},{2},{3},{4},{5},{6}\n".format(strftime("%Y-%m-%d
%H:%M:%S"),ID, light, moist, humid, temp, status))
```

These actions open an empty csv file in the desired path, afterward write the data in 7 columns with the first column is the date and time, following by Node ID, light intensity, soil moisture, humidity, temperature and status of the motor.

Finally, the Gateway transfers the data to AWS IoT using MQTT protocol by publishing topic that contains messages(data) and AWS IoT subscribe to this topic to fetch the data.

Two Topics are published, one is the regular Topic, the other is the Topic which update the Thing Shadow. In order to update the state of Thing Shadow, message should be written in JSON format, and published to Topic \$aws/things/THING_NAME/shadow/update:

```
shadow_update = $aws/things/THING_NAME/shadow/update"
message = '{"state": {"reported": {"groupID": {"ID": "'+ str(ID)
+'"}, "Device-0046": {"LightIntensity": '+ str(light) +', "SoilMoisture": '+
str(moist) +', "Temperature": '+ str(temp) +', "Humidity": '+ str(humid)
+', "Status": '+ str(status)+'}}}}'
publish_aws(shadow_update, message)
```

The other Topic is published without using JSON format

```
time = datetime.utcnow()
time_str = now.strftime('%Y-%m-%dT%H:%M:%SZ')
payload = '{"timestamp": "' + now_str + '"ID": "' + str(ID) + "', "Device-0046": "LightIntensity": '+str(light) + ', "SoilMoisture": '+ str(moist) + ', "Temperature": '+ str(temp) + ', "Humidity": '+ str(humid) + ', "Status": '+ str(status) }'
```

```
myMQTTClient.publish("node1/data", payload, 0)
```

In Short-Ranged Communication Case with ESP8266 NODEMCU as Microcontroller, instead of using Serial Communication between Node Sensor and Gateway, this case invokes MQTT protocol, which has significantly shorter delivery time than tradition UART protocol. Moreover, its communication is low-weight, as such lower power consumption. Addition Python Script was used in order to listen to Topics Published from the NODEMCU, unlike the regular case transferring all sensor data into one data frame, NODEMCU publishes each sensor data into different Topics. In this Short-Ranged case, Node ID is not transferred in the Topic, instead this parameter is retrieved by splitting it from the Topic's name under "nodename/sensordata" format, by using string split method. The Gateway listens to the Topic by using subscribe() method from Paho MQTT library. on_message() function was created to splitting Node ID and retrieving messages from Topics:

```
mqtt_topic=[("node1/temp",0), ("node1/humid",0), ("node1/pressure",0),
("node1/soil", 0), ("node1/status",0), ("node1/light",0)] #topic
subscribed from NODEMCU
def on_connect( client, userdata, flags, rc):
    print ("Connected with Code :"+str(rc))
    client.subscribe(mqtt_topic) #subscribe to Topics

def on_message(client, userdata, msg):
    ID = msg.topic.split('/')[0] #taking Node ID
    if (msg.topic == "node1/status"): #retrieving data from Topics
        light_stat = int(msg.payload)
    else:
        data = float(msg.payload)
```

Data transferring to AWS IoT in this case remains similar to regular case by using MQTT protocol.

After the Data is successfully transferred into AWS IoT, there are two options to further investigations and usages. Firstly, Data is delivered into Thing Shadow, in order End Point Android Application can retrieve its states for certain actions such as Visualization,

Node Status Tracking or Motor Controlling. The other choice is using AWS IoT Rule Engines to be transformed and used by other AWS's services.

3.3.3 Thing Shadow Management and Usage

Thing Shadow is a persistent virtual clone of the Thing created in AWS IoT Thing Registry. Thing Shadow is a JSON state document, which is used to store and retrieve current state of the Thing. This document can be interacted using either MQTT protocol or REST API. In this project, MQTT was used. One of the biggest advantages of Thing Shadow is can be interact, regardless of internet connection or not. The Gateway or Devices interact with Thing Shadow by publishing to Shadow Operations 's Topics (Update, Get, Delete), and Subscribe to Response (Accept, Reject). In this project, Update and Get Operations were used for the interaction between Gateway and AWS IoT.

Update Topic creates a Thing Shadow if there were none, otherwise updates the information of Thing Shadow with the desired/reported state data provided in the messages sent by the publisher. Update Topic is under format: `$aws/things/THING_NAME/shadow/update`. Thing Shadow publishes accepted Topic to report/desired part of the document when accepting the update request, or return errors into rejected topic when rejects update request. After being accepted updating, AWS IoT publishes to document Topic with previous and current state of Thing Shadow. Using Delta Topic, Messages are sent to all subscribers with the difference between "desired" or "reported" state.

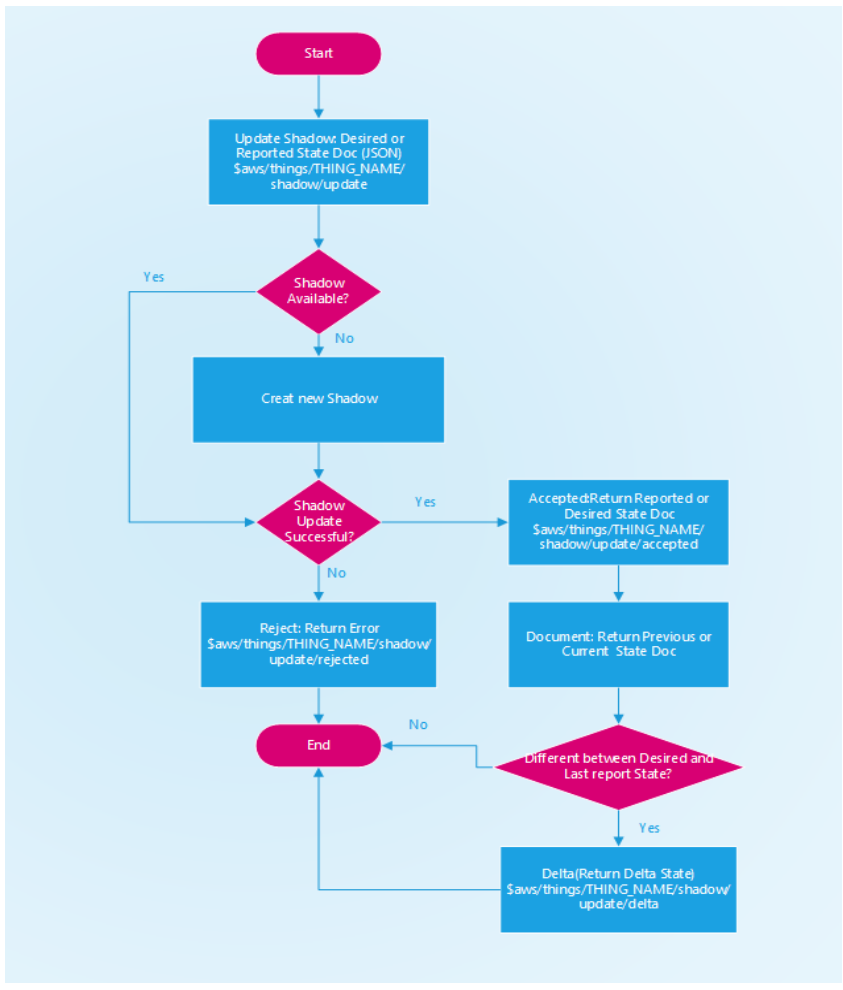


Figure 32: AWS IoT Thing Shadow Update Operation

Get Operation returns the latest state of the Thing Shadow as a JSON document. The Gateway publish to this Topic an empty message in order to retrieve data from Thing Shadow. Get Topic is in format of: \$saws/things/**THING_NAME**/shadow/get. The Thing Shadow publishes either accepted Topic, which triggered when devices retrieved the document, or rejected Topic when the Thing Shadow failed to return its documents to devices.

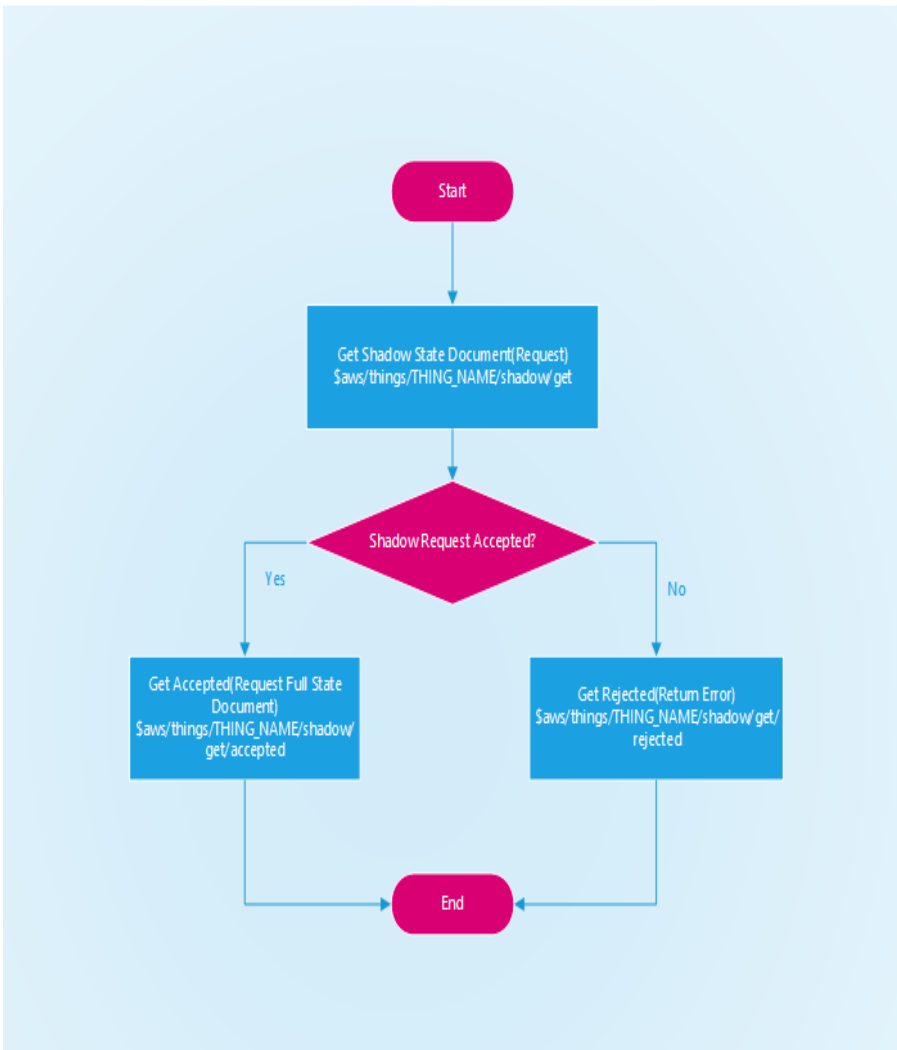


Figure 33: AWS IoT Thing Shadow Get Operation.

In order to check Thing Shadow State documents, log into AWS IoT Console, navigate into Registry>Things, choose desired Thing> Shadow. In this project, the Gateway continuously publishes messages of sensor data every 1 second into \$aws/Thing/Thing_Name/shadow/update, this message(s) contain Node Sensor (Thing) 's states inside its "reported" section. After the update Operation successful, End Point Android Application publishes messages to Thing Shadow get Topic to retrieve its data for visualization. Status of the motor is the only parameter need to be acquired by the gateway from Thing Shadow, which is under control by Android Application. In Android Application, buttons were adopted to control the motor using Thing Shadow. Using "ON" or "OFF" button changes the "state" parameter either to 0 (turn off motor) or 1(turn on the motor with 50% Duty Cycle), publishes update of Thing Shadow State, by putting the status into "desired" section of Thing Shadow. The Thing Shadows service responds by

sending a message to the accepted and delta topic. The status of the Motor can be retrieved, and published to the Node Sensor by the Gateway, using the following code:

```

THING_NAME = ""
SHADOW_UPDATE_ACCEPTED="$aws/things/"+THING_NAME+"/shadow/update/accepted"
SHADOW_UPDATE_DELTA="$aws/things/"+THING_NAME+"/shadow/update/delta"
SHADOW_GET="$aws/things/"+THING_NAME+"/shadow/get"
SHADOW_UPDATE = "$aws/things/" + THING_NAME + "/shadow/update"
#update current status of motor to reported section of Thing Shadow
SHADOW_STATE_DOC_MOTOR_ON = """"{"state": {"reported": {"status": "ON"}}}""""
SHADOW_STATE_DOC_MOTOR_OFF = """"{"state": {"reported": {"status": "OFF"}}}""""
def MOTOR_Status_Change(Shadow_State_Doc,Type): #function to change motor
status
    DESIRED_MOTOR_STATUS = ""
    SHADOW_State_Doc = json.loads(Shadow_State_Doc)

    #retrieving motor status
    if Type == "DELTA":
        DESIRED_MOTOR_STATUS= SHADOW_State_Doc['state']['status']
    elif Type == "GET_REQ":
        DESIRED_MOTOR_STATUS=
        SHADOW_State_Doc['state']['desired']['status']

    if DESIRED_MOTOR_STATUS == "1":
        Serial.write(DESIRED_MOTOR_STATUS)#send Serial message to control
        the motor
        client.publish(SHADOW_UPDATE, SHADOW_STATE_DOC_MOTOR_ON,qos=1)
        #update the status to Thing Shadow
        #client.publish(Motor_Status,DESIRED_MOTOR_STATUS,qos=1)for short
        range case, send signal to motor control topic
    if DESIRED_MOTOR_STATUS == "0":
        Serial.write(DESIRED_MOTOR_STATUS) #send Serial message to
        control the motor

        client.publish(SHADOW_UPDATE, SHADOW_STATE_DOC_MOTOR_OFF,qos=1)
        #update the status to Thing Shadow
        #client.publish(Motor_Status,DESIRED_MOTOR_STATUS,qos=1)for short
        range case, send signal to motor control topic

def on_connect(client, userdata, flags, rc):
    client.subscribe(SHADOW_UPDATE_ACCEPTED, 1)
    client.subscribe(SHADOW_UPDATE_DELTA, 1)
    client.subscribe(SHADOW_GET_ACCEPTED, 1)

def on_message(client, userdata, msg):
    if str(msg.topic) == SHADOW_UPDATE_DELTA:
        SHADOW_STATE_DELTA = str(msg.payload)
        MOTOR_Status_Change(SHADOW_STATE_DELTA, "DELTA")

    elif str(msg.topic) == SHADOW_GET_ACCEPTED:
        SHADOW_STATE_DOC = str(msg.payload)
        MOTOR_Status_Change(SHADOW_STATE_DOC, "GET_REQ")

```

The Gateway subscribes to delta or get accepted topic, receives its new status, publishes new status for the Node Sensor to control the Motor and publishes new status back to

Thing Shadow update Topic. In response, in update accepted Topic, the Thing Shadow changed the current status and set desired to NULL.

The Thing can be detected if is connecting to AWS IOT using MQTT 's Last Will and Testament (LWT) messages that set the connected parameter to false if a device is disconnected. Each Thing represent a Node Sensor. LWT is a feature in MQTT protocol, which notifies other clients about an ungracefully disconnected client. The client specifies its last will message when connecting to the broker. The last will message is a MQTT message with a topic, retained message flag, QoS, and payload. The broker stores the message until it detects that the client has disconnected ungracefully. In order to send LWT message from the Raspberry Pi Gateway, Paho MQTT's will_set() function is used:

```
lwm=" {"state":{"reported":{"connected":"false"}}}" # Last will message
regular=" {"state":{"reported":{"connected":"true"}}}" #regular message
client is connecting to broker normally
client.will_set("Thing_Name/Thing_Status",lwm,QoS1,retain=False)
client.publish("Thing_Name/Thing_Status",regular)
```

After creating the LWT message, a republish rule is invoked, by register an LWT message to a non-reserved topic and create a rule that republishes the message on the reserved topic (Topics start with \$ sign). In order to create a Rule, log into AWS IoT Console, select Act>Create Rule and fill in the parameters as following code:

```
{"rule": {
  "ruleDisabled": false,
  "sql": "SELECT * FROM ' Thing_Name/Thing_Status '",
  "description": "",
  "actions": [{
    "republish": {
      "topic": "$aws/things/Thing_Name/shadow/update",
      "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish"
    }
  ]
}
```

When the Thing connects, it publishes to the update topic in AWS IoT and Thing_Status, set connected state to true. If graceful disconnection occurs, Thing publishes a message to the update Topic and sets its connected state to false. The Android Application subscribe to this topic to determine the state of the Thing and display.

3.4 Android Application

3.4.1 Setting up the Environment

In Android studio, after creating a new project, navigate to `app>build.gradle`, add the dependencies, download the .jar files and put them into lib directory :

```
implementation files('libs/aws-android-sdk-core-2.6.22.jar')
implementation files('libs/aws-android-sdk-iot-2.6.22.jar')
implementation files('libs/aws-android-sdk-ddb-2.6.22.jar')
```

In order to develop the application using AWS, AWS credentials must be obtained using Amazon Cognito Identity, which is a credentials provider. With this service, other services can be accessed without inserting private credentials into the application. AWS credentials can be obtained by creating an identity pool, which stores user 's identity data. This identity pool has configurable IAM roles, which allow specifying services can be accessed. TO create an identity pool, log into AWS Cognito, select Manage Federated Identities>Create new identity pool, choose Allow to create 2 default roles: unauthenticated users and authenticated users. Under unauthenticated role, IAM policy can be used, allows the application to publish/subscribe to the Thing. The pool's ID is needed to request Cognito for a temporary credential and register as a parameter in the code of application.

```
// Initialize the AWS Cognito credentials provider
credentialsProvider = new CognitoCachingCredentialsProvider(
    getApplicationContext(), // context
    COGNITO_POOL_ID, // Identity Pool ID
    MY_REGION // Region
);
```

3.4.2 Main Application

The application is divided into 3 Tabs:

- Dashboard for overview of the whole system with the number of devices, devices that are currently connecting or disconnecting from AWS IoT, quick access menu to control and data logging
- Devices Tab lists all available devices, accessing to Sensor's data view including real time number, overall graph by hour, date and month, controlling Motor menu and Map

- Notification Tab displays notification when any Sensor's data exceed limited value.

In order to fulfil the purpose of the application, PubSub API from AWS IoT is the core feature, provides cloud-based message-oriented middleware. Import the following classes to use the MQTT features in the app:

```
import com.amazonaws.mobileconnectors.iot.AWSIoTmqttManager;
import com.amazonaws.mobileconnectors.iot.AWSIoTmqttNewMessageCallback;
import com.amazonaws.mobileconnectors.iot.AWSIoTmqttQos;
```

In the configuration, using AWSIoTManager class to provide client ID and MQTT End Point, which is the same End Point retrieved from Thing Registry. To use PubSub with AWS IoT, IAM policies is needed and attach to Amazon Cognito Identity. Create the same policy when creating a Thing allows full access to all the Topics. To attach the policy into the user's Cognito Identity ID:

```
import com.amazonaws.services.iot.AWSIoTClient;
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.services.iot.model.AttachPolicyRequest;

AttachPrincipalPolicyRequest policyAttachRequest =
new AttachPrincipalPolicyRequest ();
policyAttachRequest.setPolicyName(AWS_IOT_POLICY_NAME);
policyAttachRequest.setPrincipal(createKeysAndCertificateResult.getCertificateArn());
mIoTAndroidClient.attachPrincipalPolicy(policyAttachRequest);
```

After the setting was completed, MQTT connection needed to be established as the following code:

```
// MQTT Client
mqttManager= new AWSIoTmqttManager(clientId, CUSTOMER_SPECIFIC_ENDPOINT);
try {
    mqttManager.connect(AWSMobileClient.getInstance(), new
AWSIoTmqttClientStatusCallback() {
        @Override
        public void onStatusChanged(final AWSIoTmqttClientStatus status,
final Throwable throwable) {
            Log.d(LOG_TAG, "Connection Status: " +
String.valueOf(status));
        }
    });
} catch (final Exception e) {
    Log.e(LOG_TAG, "Connection error: ", e);
}
```

The application is capable of subscribing to Thing Shadow get accepted topic to retrieve data from AWS IoT. Thing Shadow is a JSON document, as such JSON classes are needed.

```
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
```

Messages from get accepted Topic is retrieved by creating subscribeNumber() function:

```
public static void subscribeNumber(String topic) {
    Log.d(LOG_TAG, "topic = " + topic);
    final String[] message = new String[1];
    try {
        mqttManager.subscribeToTopic(topic, AWSIotMqttQos.QOS0,
            new AWSIotMqttNewMessageCallback() {
                @Override
                public void onMessageArrived(final String topic,
                    final byte[] data) {
                    try {
                        message[0] = new String(data, "UTF-8");
                        Log.d(LOG_TAG, "AWSMessage arrived:");
                        Log.d(LOG_TAG, "    Topic: " + topic);
                        Log.d(LOG_TAG, "    AWSMessage: " + message[0]);
                        JSONdeviceNumber(message[0]);
                        JSONnoti(message[0]);
                    } catch (UnsupportedEncodingException e) {
                        Log.e(LOG_TAG, "AWSMessage encoding error.",
                            e);
                    }
                }
            });
    } catch (Exception e) {
        Log.e(LOG_TAG, "Subscription error.", e);
    }
}
```

checkJSONint(), checkJSONstring() and checkJSONdouble() were used to check if the document has the appropriate String and get the data, for instance taking Temperature data, the other data are taken similar to this code:

```
JSONObject json = new JSONObject(string);
JSONObject state = json.getJSONObject(STATE);
JSONObject reported = state.getJSONObject(REPORTED);
JSONObject device = reported.getJSONObject(deviceName);
Integer temperatureVal = checkJSONint(device, TEMPERATURE);
```

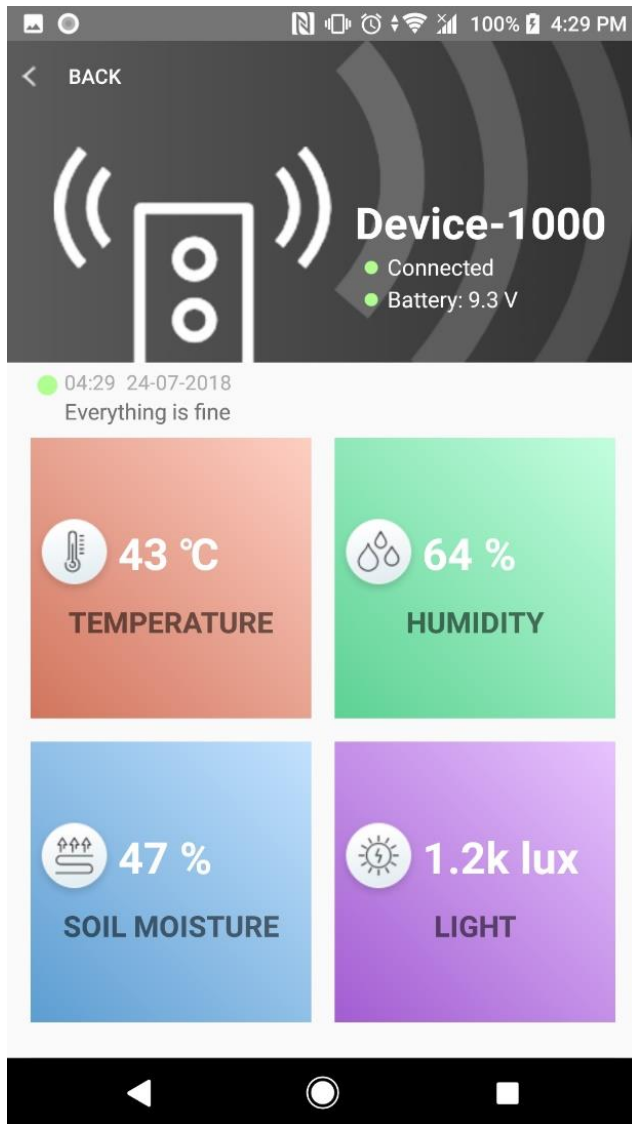


Figure 34: Node Sensor's data in real time

As a test case, Nodes 's location were pushed into Thing Shadow with fixed Longitude and Latitude of the Node where they were currently. Using Google Map API, the map was created based on the Longitude and Latitude obtained from Thing Shadow:

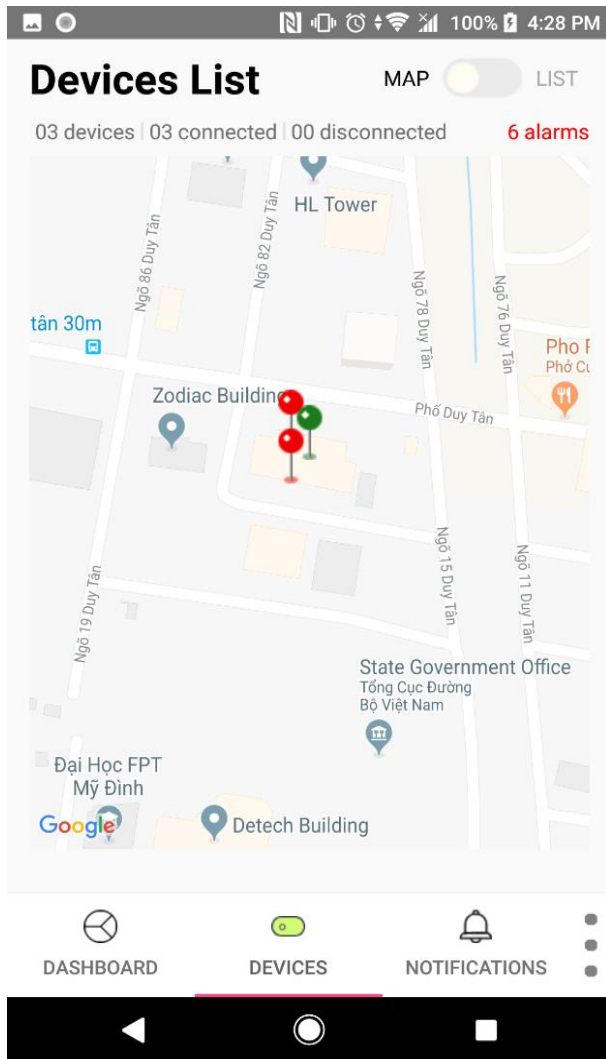


Figure 35: GPS checking Tab

```

public void onMapReady(GoogleMap googleMap) {
    int marker;
    mMap = googleMap;
    locationText.setText("Location ");
    LatLng fville = new LatLng(values.getDouble(PubSubJSON.LAT),
        values.getDouble(PubSubJSON.LNG));
    mMap.clear();
    if (alarmedDevice.getBoolean(deviceNameValue)) {
        marker = R.drawable.map_marker_red; //mark red to Node
        //having problem
    } else {
        marker = R.drawable.map_marker_green; //mark green to normal
        //Node
    }
    mMap.addMarker(new
        MarkerOptions().position(fville).title(deviceNameValue).icon(Bitmap
           DescriptorFactory.fromResource(marker)));
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(fville,
        18.0f));
}

```

Buttons were created to control Motor Status. When pressing the button, the Status changed to the opposite Status, publish the new Status to the Thing Shadow update Topic, using publishString() method:

```
public static void publish(String topic, String msg) {
    try {
        mqttManager.publishString(msg, topic, AWSIoTmqttQos.QOS0);
    } catch (Exception e) {
        Log.e(LOG_TAG, "Publish error.", e);
    }
}

publish("$saws/things/Thing_name/shadow/update", '{"state": {"desired": + sta-
tus +}}')
```

Notifications are based on the limited value of each data types and will be sent if the value exceed the threshold. The Notifications are displayed under Bundle Format. In addition, notifications are displayed in each Node Sensor's own tabs. Format of a Notification message is displayed as the following code:

```
switch (string) {
    case TEMPERATURE:
        min = 10;
        max = 45;
        notiSub = "The temperature of this area is " +
            String.valueOf(value) + " \u2103";
        ImageResource = R.drawable.temperature_alarm;
        break;
}

if (value < min || value > max) {
    //take data that exceeded limit, based on cases
    JSONObject json = new JSONObject(message);
    JSONObject metadata = json.getJSONObject(METADATA);
    JSONObject reported = metadata.getJSONObject(REPORTED);
    //get device name
    JSONObject device = reported.getJSONObject(devices[position]);
    JSONObject deviceValue = device.getJSONObject(string);
    //take the time
    long timestamp = checkJSONint(deviceValue, TIMESTAMP) * 1000L;
    SimpleDateFormat format = new SimpleDateFormat("hh:mm dd-MM-yyyy");
    //set the alarm event to true
    alarmedDevice.putBoolean(devices[position], true);
    NotiList.notiIcon.putInt(NOTIICON + String.valueOf(numberOfNoti), ImageResource);
    //increase number of notification
    numberOfNoti++;
}
```

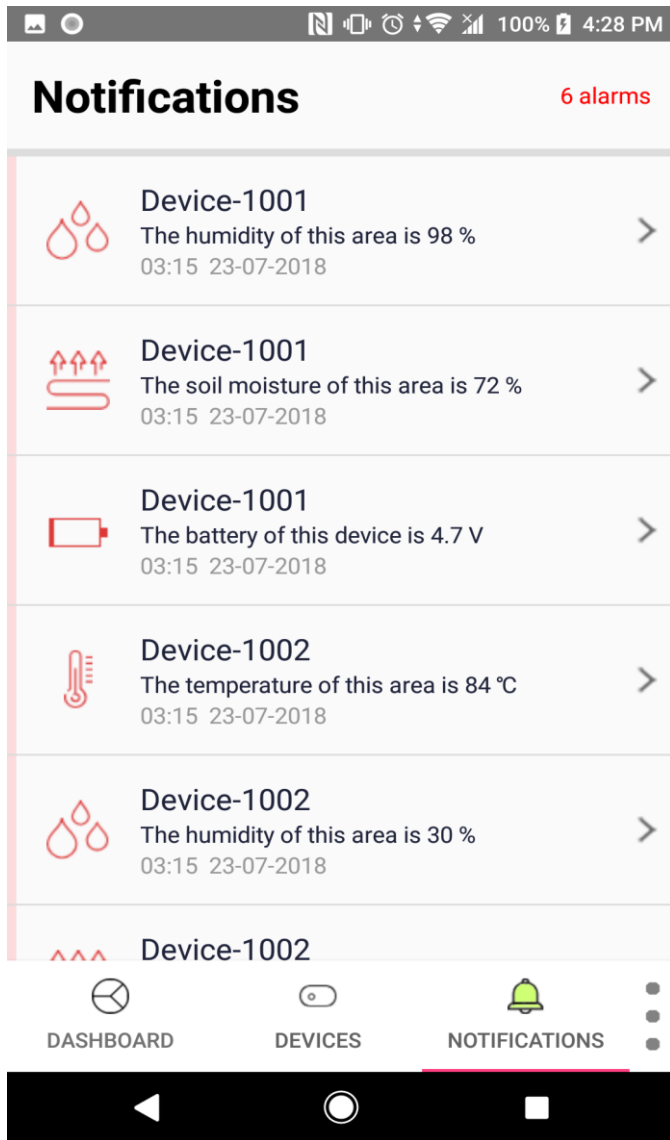


Figure 36: Notification Tab

Finally, data should be stored into a database before being demonstrated by line graph. The Application obtains Data and display the graph, based on time interval (Hour, Date, Month), and average value of desired data. AWS provides DynamoDB for this task. Graph package used in this project was GraphView. Amazon DynamoDB is a fully managed NoSQL database that supports both document and key-value store models, offers security, back up, restores and in memory caching, suitable for mobile and IoT purposes. In order to redirect message into DynamoDB, log into AWS IoT console, navigate to Act>Create a Rule, under Rule query statement, input `SELECT Humidity, Temperature, Light, Soil, Status FROM 'Node1/data'` , this action redirects messages from Node1/data Topic, which was created in Section 5.3.2 to the Database. Under Select an

Action tab, choose “Insert a message into a DynamoDB table” to create a table in DynamoDB console, select “ID” Partition key, “timestamp” as Sort Key. The messages in the payload can be spitted into separated columns using DynamoDBv2 rule. The following figure demonstrated the final DynamoDB table:

GreenHouseTable [Close](#)

Overview **Items** Metrics Alarms Capacity Indexes Global Tables Backups Triggers Access control Tags

Create item Actions ▾

Scan: [Table] GreenHouseTable: Row, PositionInRow ...

Scan ▾ [Table] GreenHouseTable: Row, PositionInRow ▾ ^

+ Add filter

Start search

<input type="checkbox"/>	Row ⓘ	PositionInRow ▾	payload ▾
<input type="checkbox"/>	Node_1	1483468613	{"Humidity": {"S": "60"}, "Light": {"S": "200"}, "Soil": {"S": "40"}, ...
<input type="checkbox"/>	Node_1	1483468614	{"Humidity": {"S": "60"}, "Light": {"S": "200"}, "Soil": {"S": "40"}, ...
<input type="checkbox"/>	Node_1	1483468615	{"Humidity": {"S": "64"}, "Light": {"S": "200"}, "Soil": {"S": "40"}, ...
<input type="checkbox"/>	Node_1	1483468630	{"Humidity": {"S": "64"}, "Light": {"S": "200"}, "Soil": {"S": "40"}, ...
<input type="checkbox"/>	Node_1	1483468650	{"Humidity": {"S": "64"}, "Light": {"S": "200"}, "Soil": {"S": "50"}, ...
<input type="checkbox"/>	Node_1	1483468669	{"Humidity": {"S": "64"}, "Light": {"S": "200"}, "Soil": {"S": "50"}, ...

Figure 37: DynamoDB Table.

DynamoDBManager class were used to scan the Database and retrieve the data from the Table. The following code demonstrates the data receiving process:

```
@DynamoDBHashKey(attributeName = "timestamp ")
public Long getTime1() {
    return timestamp;
}

public void setTime1(Long timestamp) {
    this.time_load = timestamp;
}

@DynamoDBAttribute(attributeName = "Humidity")
public String getHumidity() {
    return humidity;
}

public void setHumidity(String humidity) {
    this.humidity = humidity;
}
```

The Application receives data from the DynamoDBManager class to display by graph:

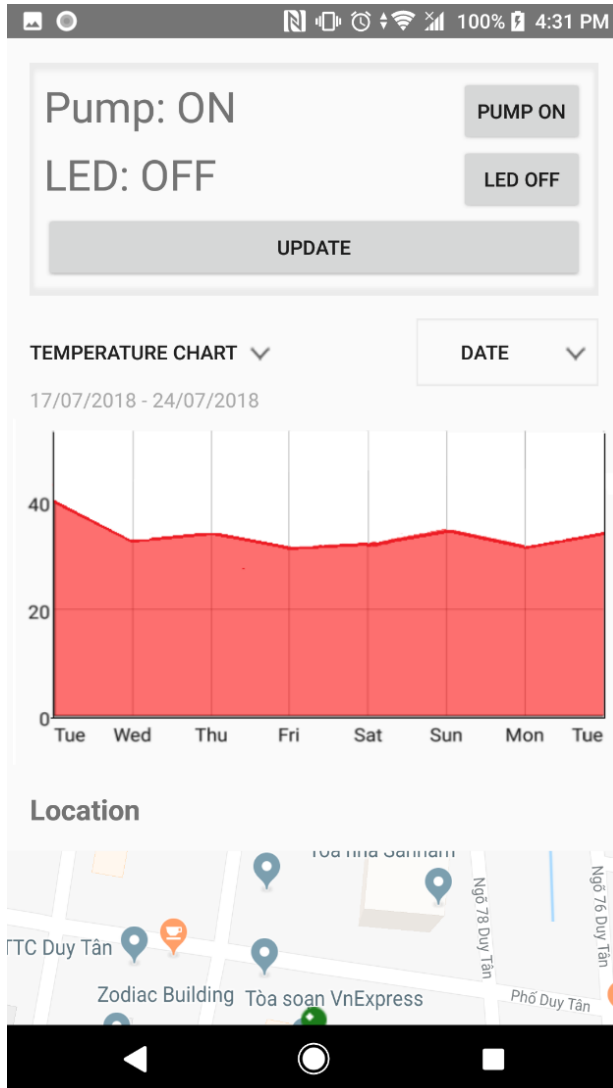


Figure 38: GPS, Graph and Motor Control Tab

```
itemDB = DynamoDBManager.getIotHistory();  
for (DynamoDBManager.IotHistory up : itemDB) {  
    timestampDB.add(up.getTime());  
    temperatureDB.add(up.getTemperature());  
    humidityDB.add(up.getHumidity());  
    soilMoistureDB.add(up.getSoilMoisture());  
    lightIntensityDB.add(up.getLightIntensity());  
}  
dtdDate = ddt.format(calendar.getTime());
```


4 TESTING AND RESULT

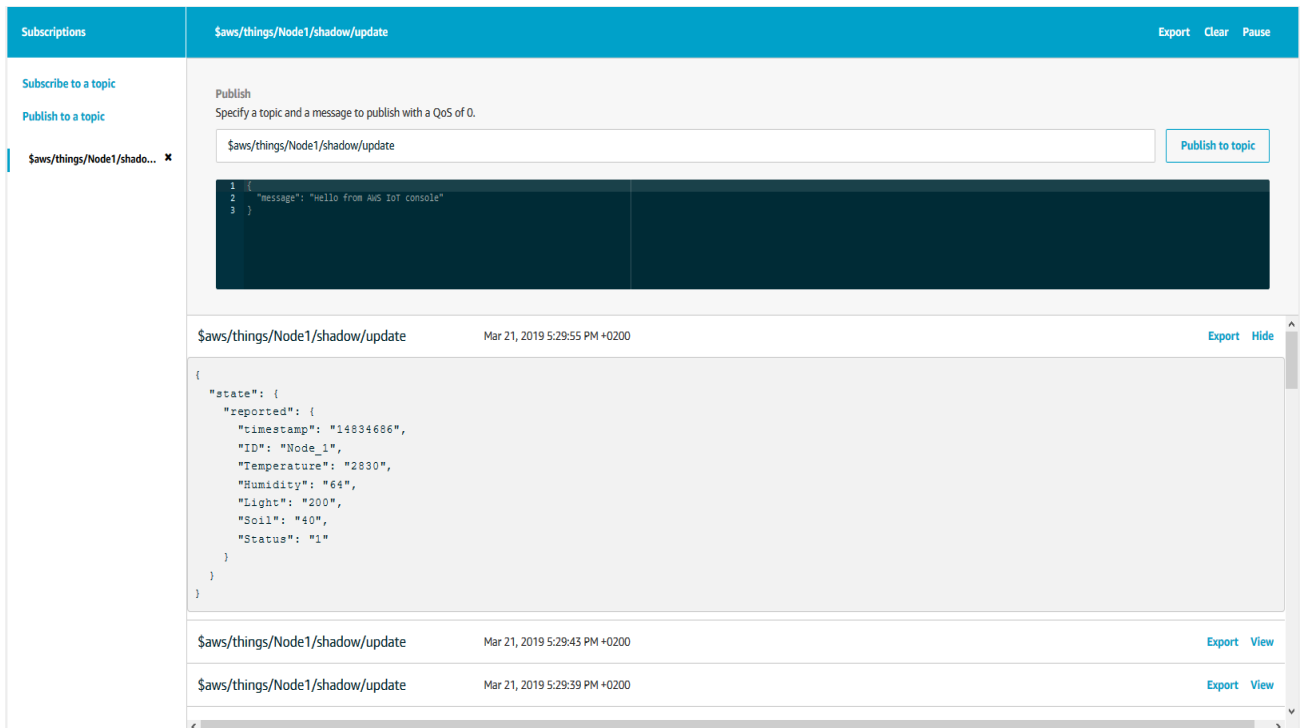
The testing phase was divided into 3 parts:

Node Sensor Testing: Circuit Testing by using Proteus 8 Simulation to simulate the Theory Circuit before constructing real circuit. Data received from sensor is monitored by Serial Monitor from Arduino IDE. Serial Monitor is used to check MQTT connection between NODEMCU and Gateway in Short Ranged Communication. Messages exchanged between two LoRa Module can be seen by using HERCULES software, which is used to check UART communication data.

Gateway and Cloud Testing: The Data Receive from the Node Sensor can be monitored using print() method of Python. Csv files are stored internally inside Raspberry Pi. MQTT connection between Gateway and AWS IoT can be checked inside on_connect() function

```
def on_connect(client, userdata, flags, rc): # func for making
connection
    global connflag
    print ("Connected to AWS")
    connflag = True
    print ("Connection returned result: " + str(rc) )
```

Messages arrived to the Gateway can be detected using Test Section inside AWS IoT Console



The screenshot displays the AWS IoT Console interface for testing MQTT. On the left, there's a 'Subscriptions' sidebar with a search bar and a list of subscriptions. The main area shows a 'Publish' section where a topic '\$aws/things/Node1/shadow/update' is entered. Below this, a code editor shows a JSON message: `{ "message": "hello from AWS IoT console" }`. The bottom section shows a message history table with columns for topic, timestamp, and actions (Export, Hide, View).

Topic	Timestamp	Actions
\$aws/things/Node1/shadow/update	Mar 21, 2019 5:29:55 PM +0200	Export Hide
\$aws/things/Node1/shadow/update	Mar 21, 2019 5:29:43 PM +0200	Export View
\$aws/things/Node1/shadow/update	Mar 21, 2019 5:29:39 PM +0200	Export View

Figure 39: AWS IoT MQTT testing

The Thing Shadow States and full Document can be inspected inside Shadow Tab in Thing Management.

The screenshot displays the Shadow Tab interface. On the left, there is a sidebar with navigation options: 'Jobs', 'Violations', and 'Defender metrics' (with a 'new' badge). The main content area is divided into two sections: 'Shadow state' and 'Metadata', each showing a JSON document.

Shadow state:

```
{
  "desired": {
    "color": "green"
  },
  "reported": {
    "color": "red",
    "timestamp": "14834686",
    "ID": "Node_1",
    "Temperature": "2830",
    "Humidity": "64",
    "Light": "200",
    "Soil": "40",
    "Status": "1"
  },
  "delta": {
    "color": "green"
  }
}
```

Metadata:

```
{
  "metadata": {
    "desired": {
      "color": {
        "timestamp": 1553112561
      }
    },
    "reported": {
      "color": {
        "timestamp": 1553182105
      },
      "timestamp": {
        "timestamp": 1553182196
      },
      "ID": {
        "timestamp": 1553182196
      },
      "Temperature": {
        "timestamp": 1553182196
      },
      "Humidity": {
        "timestamp": 1553182196
      },
      "Light": {
        "timestamp": 1553182196
      }
    }
  }
}
```

Figure 40: Thing Shadow Document

Android Application Testing: Data Visualization in Real Time, Graph, Notification messages and Node Status, Motor Controlling were tested and fulfilled their Roles. However, the Graph might take long time to be displayed.

In Addition, Node Sensor's Range Test were established. The Node using LoRa Module longest range to receive data was approximately 2 km without obstacle and 100 m with obstacles. The Node using NODEMCU has approximately 40-60m with obstacle to receive data.

5 CONCLUSION AND FUTURE WORK

When working with this project, I have gained understanding about an actual IoT system in real life, how to build it and how it can be applied. I learned how to build a system from scratch, choosing the right components, testing my skills with my existing knowledge and learn the new programming language. For the customer, which is the higher branch of the company itself, this was a good opportunity to obtain more understanding of IoT, invest further into this field and apply this technology into their products.

The challenge of this project were the difficulties in finding appropriate components, learning the AWS IoT features, architectures and its appliances.

The Project has potential to be expanded. First of all, the location of Node Sensor can be checked by installing a GPS module into each Nodes. Secondly, more machinery mechanisms can be applied such as a light bulb, and each mechanisms can be operated in a manual mode or a automatic mode, based on certain condition. For instances, Water Pump can be in automatic mode based on Soil Moisture value. The Motor Speed's options can be implemented into the application. Finally, more AWS services can be applied. In this project, under ongoing development, the data from DynamoDB can be redirected to S3(Simple Storage Service), which stores data in the large amounts and provides scalability, durability, and security, by using AWS Pipeline, which moves data between totally different AWS calculation and storage services. The data from S3 can be used by AWS Quicksight for Visualization, Analytic of Machine Learning. Moreover, AWS Greengrass can be used when the system expands further with more devices connected, which extends AWS to edge devices so they can operate locally on the data they generated, while using cloud services and resources. Using AWS Greengrass makes devices run Lambda functions and communicate with other devices in groups, even when not connected to the Internet.

6 REFERENCES

Internet of Things definition

https://en.wikipedia.org/wiki/Internet_of_things

Internet of Things Overview

<https://cloud.google.com/solutions/iot-overview>

History of Internet of Things

<https://www.forbes.com/sites/gilpress/2014/06/18/a-very-short-history-of-the-internet-of-things/>

IoT application in the market in 2018

<https://iot-analytics.com/top-10-iot-segments-2018-real-iot-projects/>

AWS IoT main page

<https://aws.amazon.com/iot>

AWS IoT documentation

<https://docs.aws.amazon.com/iot/latest/developerguide/iot-dg.pdf>

AWS IoT components Overview

<http://www.promow.be/resources/amazon-web-services-iot/>

<https://aws.amazon.com/iot-core/features/>

AWS IoT Rule Engine Document

<https://docs.aws.amazon.com/iot/latest/developerguide/iot-rules.html>

AWS IoT Security

<https://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html>

LoRa Information

<https://www.semtech.com/lora/what-is-lora>

PubSubClient library for ESP8266

<https://learn.adafruit.com/mqtt-adafruit-io-and-you/arduino-plus-library-setup>

AWS IoT pricing

<https://aws.amazon.com/iot-core/pricing/>

How AWS IoT works

<https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html>

Dave Evan (April 2011, online) The Internet of Things How the Next Evolution of the Internet Is Changing Everything

https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

IoT Platform

<https://www.kaaproject.org/what-is-iot/>

<https://www.link-labs.com/blog/what-is-an-iot-platform>

Arduino Platform

<https://www.arduino.cc/en/Guide/Introduction>

Python Programming Language

<https://www.python.org/>

AWS IoT Tutorial For Python and Raspberry Pi

<https://techblog.calvinboey.com/raspberrypi-aws-iot-python/>

End to End IoT application Deployment

https://d1.awsstatic.com/Projects/P4113926/aws-tutorial_deploy-iot-application.pdf

Solar Panel Battery Charging Process.

<https://www.arduino-lab.net/solar-li-ion-charger-and-power-supply-for-arduino/>

MQTT protocol

<http://mqtt.org/>

<https://en.wikipedia.org/wiki/MQTT>

<https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/>

Python and Paho MQTT with AWS IoT

<https://www.hackster.io/mariocannistra/python-and-paho-for-mqtt-with-aws-iot-921e41>

Mosquitto MQTT client and server

<https://mosquitto.org>

Paho MQTT library for Python

<https://pypi.org/project/paho-mqtt/>

UART protocol

<https://learn.sparkfun.com/tutorials/serial-communication>

<http://www.circuitbasics.com/basics-uart-communication/>

ESP8266 communicate with Raspberry Pi using MQTT

<https://www.hackster.io/ruchir1674/raspberry-pi-talking-to-esp8266-using-mqtt-ed9037>

Android tutorial

<https://developer.android.com/guide/>

Fraden, J. 2010. Handbook of Modern Sensors: Physics, Designs, and Applications.

DHT11 temperature and humidity sensor

<https://howtomechatronics.com/tutorials/arduino/dht11-dht22-sensors-temperature-and-humidity-tutorial-using-arduino/>

Chengdu Ebyte Electronic Technology Co.,Ltd .Online. E32-TTL-100 Datasheet v1.2

AWS IoT Android Development Guide

<https://aws-amplify.github.io/docs/android/start?ref=amplify-android-btn>

Thing Shadow Usage

<https://iotbytes.wordpress.com/device-shadows-part-1-basics/>