

**LAB AUTOMATION SYSTEM WITH PLATFORM SUPPORT
USING THE ASYNCHRONOUS MQTT PROTOCOL**

DISSERTATION

SUBMITTED IN THE PARTIAL FULLFILMENT OF THE REQUIREEMNTS
FOR THE AWARD OF THE DEGREE

OF

MASTER OF TECHNOLOGY

IN

POWER SYSTEM

Submitted by:

JAGRITI SURABHI

(2K17/PSY/07)

Under the supervision of

Dr. M. RIZWAN



DEPARTMENT OF ELECTRICAL ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

2019

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi - 110042

CANDIDTATE’S DECLARATION

I, **JAGRITI SURABHI**, Roll No. **2k17/PSY/07**, student of M.Tech (Power System), hereby declare that the project Dissertation titled “**Lab Automation System with Platform Support using Asynchronous MQTT Protocol**” which is submitted by me to the Department of Electrical Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi

JAGRITI SURABHI

Date: 09.08.2019

Department of Electrical Engineering
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi – 110042

CERTIFICATE

I hereby certify that the Project Dissertation titled “**Lab Automation System with Platform Support using Asynchronous MQTT Protocol**” which is submitted by **JAGRITI SURABHI**, Roll no **2K17/PSY/07** [Department of Electrical Engineering], Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the student under my provision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

Date: 09.08.2019

Dr. M. Rizwan

SUPERVISOR

ABSTRACT

The objective of this project revolves around creating a Lab automation system with the main focus being the usage of a Smart IoT Platform to integrate and control devices capable of communicating over the internet. The system consists of daughter nodes using synchronous or asynchronous protocols for communication, a mother hub hosting a smart platform, a server and front end Web UI. The daughter nodes are microprocessors and smart peripherals like Lighting loads and plug load controllers. Each microprocessor employed hosts an automation circuit that can be controlled cumulatively at one interface, the smart platform.

This platform is hosted by the mother hub, mainly the Raspberry Pi used in this project for that purpose. There is no commonly accepted open standard for interaction with smart devices. This hinders many devices from interacting with each other. A hub is a unit that acts as a tool for all the linked systems to communicate and handle. The R-Pi SoC flamboyant board that has been a choice of enthusiasts for its form-factor, processing, open source integration and the number of GPIO pins it offers for automation. It seeks to deliver local and remote tracking, interoperability, scalability, plug and play, robustness, open protocol, all this being cost-effective.

The open source project Home Assistant, an automation platform running on Python3 is hosted on the mother hub. It has stronger focus on end users with a good selection of software providers and integration services as compared to other beta open source projects. The platform runs in a virtual environment created for its Command line and Graphic user interface. The Internet of Things (IoT) is a fast-growing theme of technical, social and financial importance. IoT's open nature leads to dominant settings being created in which its parts can be rearranged based on the context. This has innovative consequences.

Because IoT is not a consolidated sector where a dominant design guides incremental innovation, instead, innovation arises from connecting components together by concentrating on inter-industry cooperation and user engagement. The further growth and implementation of IoT will be simulated in the coming times.

ACKNOWLEDGEMENT

I would like to express my gratitude towards all the people who have contributed their precious time and effort to help me with this project, without their support it would not have been possible for me to implement this project to its completion.

I would like to thank **Dr.M.Rizwan**, Project Supervisor and Associate Professor, Department of Electrical Engineering for his support, valuable feedbacks and guidance throughout the period of this work.

I would also like to thank my fellow scholars Dr.Priyanka Chaudhary, Astitva Kumar, Tarun Kumar Nirmal and Eklavya Dahiya who encouraged and helped me develop better ideas to keep the challenge alive.

JAGRITI SURABHI

CONTENTS

Declaration	i
Certificate	ii
Abstract	iii
Acknowledgement	iv
Contents	v
List of Tables	x
List of Figures	xi
Symbols, Abbreviation	xiii
CHAPTER 1: INTRODUCTION	1
1.1 : GENERAL	1
1.2 : MOTIVATION: THE INDIAN MARKET SCENARIO	2
1.2.1 : Consumer Applications	3
1.2.2 : Industrial Applications	3
1.2.3 : Public Sector Applications	4
1.3 : ARCHITECTURE	5
1.3.1 : Perception Layer	5
1.3.2 : Network Layer	6
1.3.3 : Processing layer	7
1.3.4 : Application Layer	8
1.3.5.1 : Message Queuing Telemetry Transport	9
1.3.5.2 : Constrained application Protocol (CoAP)	10
1.3.5.3 : Representational State Transfer (REST)	10

1.3.5 : Business Layer	11
CHAPTER 2: LITERATURE REVIEW	12
2.1 : Introduction	12
2.2 : Energy Conservation Schemes in India	13
2.2.1 : Electric Conservation Building Code (ECBC)	13
2.2.2 : Lighting	14
2.2.3 : Renewable Energy Resources	14
2.2.4 : Selection of Appliances	14
2.2.5 : Smart and Green building projects	15
2.3 : Smart Sensors	15
2.4 : IoT Security and Threats	17
2.5 : Smart Building Management System: BEMOSS	18
CHAPTER 3: SENSORS and UI PLATFORM USING MQTT	20
3.1 : Introduction	20
3.2 : Test Circuit Model	20
3.2.1 : Test Circuit Components	20
3.2.2 : Test Circuit Design Models	22
3.3.1 : Cayenne Online Platform	23
3.3.2 : MQTT Protocol	24
3.3.3 : Temperature and Humidity Sensor	28
3.3.4 : Luminosity Sensor	29
3.3.5 : PIR Sensor	31
CHAPTER 4: MODELING OF THE CORE PLATFORM	32
4.1 : Introduction	32
4.2 : Embedded System Hosting	33
4.2.1 : Single Board Computers	33
4.2.2 : Raspberry Pi model 3B	34

4.3 : Performance Analysis	36
CHAPTER 5: HOME ASSISTANT: AN OPEN SOURCE DEVELOPMENT PLATFORM	
5.1: Introduction	40
5.2: The Lovelace UI	42
5.2.1: Smart Light setup	43
5.3: The ESP Tool	44
5.3.1: Device Integration	46
CHAPTER 6: RESULTS AND DISCUSSIONS	49
6.1: Introduction	49
6.2: Results of Test circuit 1	49
6.3: Results of Test circuit 2	50
CHAPTER 7: CONCLUSION AND FUTURE SCOPE	54
7.1: Introduction	54
7.2: Conclusion	54
7.3: Future scope	55
APPENDICES	13
Appendix 1	13
Appendix 2	14
Appendix 3	15
Appendix 4	16
REFERENCES	21

LIST OF TABLES

Table No.	Table	Page No.
Table 1.1	Public sector-wise applications	4
Table 1.2	Comparison between network protocols	7
Table 2.1	Comparison of Lighting Fixtures	14
Table 2.2	A brief comparison of Threats and Challenges of IoT features	17
Table 3.1	MQTT vs HTTP in a constrained environment	27
Table 3.2	A comparison of DHT sensors	28
Table 4.1	Comparison in hardware of selected Single Board Computers	33

LIST OF FIGURES

Figure No.	Figures	Page No.
Figure 1.1	The Present IoT Trend	1
Figure 1.2	The IoT Market in India	2
Figure 1.3	Indian Consumer Expectation from the IoT Sector	3
Figure 1.4	Sensor to Actuator Flow Diagram	6
Figure 1.5	An Event based Middleware Flowchart	8
Figure 1.6	Working of Pub/Sub model in MQTT	9
Figure 1.7	Work-flow of the Client-Server Architecture in CoAPs	10
Figure 1.8:	REST Work-flow Model	11
Figure 1.9:	5-layer Architecture adapted in IoT	11
Figure 2.1	Energy Consumption in the Residential Sector (Source: Planning Commission)	13
Figure 2.2	A CLPD Core-Enabled Hardware Design	16
Figure 2.3	BEMOSS Agents and Work-flow	19
Figure 3.1	ESP8266 NodeMCU Development Kit	21
Figure 3.2	Components Layout of the Test	22
Figure 3.3	PCB Layout of the Test Circuit	22
Figure 3.4	Schematic Diagram of the Test Circuit	23
Figure 3.5	Test Circuits Dashboard on the IoT Platform	24
Figure 3.6	QoS0 Level (at most one message delivery)	25

Figure 3.7	QoS1 Level (at least one message delivery)	26
Figure 3.8	QoS 2 Level (exactly one message delivery)	26
Figure 3.9	MQTT Connections over TCP/IP Protocol	27
Figure 3.10	Temperature Reading from a DHT11 measured every 2 Seconds	29
Figure 3.11	Humidity Readings from a DHT11 measured every 10 Minutes	29
Figure 3.12	Basic Structure and Symbol for LDR	30
Figure 3.13	Luminosity Readings from an LDR measured every 10 Minutes	30
Figure 3.14	Resistances vs. Illumination of an LDR	31
Figure 3.15	The PIR Sensor	31
Figure 4.1	Pin Diagram of Raspberry Pi (Host Embedded System)	35
Figure 4.2	Inserting a Cron Job in Python Script	37
Figure 4.3	Dashboard for Inspecting Health of Host System	37
Figure 4.4	Utilization of the CPU on the Pi	37
Figure 4.5	Network out in kB connected to Local Network	38
Figure 4.6	CPU Temperature of the Pi	38
Figure 4.7	Memory Utilization of the Pi	39
Figure 4.8	All Networks handled by the Pi	39
Figure 5.1	The CLI with Home Assistant running Live	40
Figure 5.2	Desktop and Web UI of successful HA Setup	42
Figure 5.3	TP-Link Smart Light Integration	43
Figure 5.4	ESPHome Setup Script on CLI	44

Figure 5.5	Successful Instant of HA API Integration with ESPHome	45
Figure 5.6	Device Model on Fritzing	46
Figure 5.7	ESPHome Dashboard	46
Figure 5.8	Configuration in ESPHome	47
Figure 5.8	Successful Device Integration	48
Figure 5.10	Web UI with Parameters from Test_Blink on the Dashboard	48
Figure 6.1	Test circuit 1	49
Figure 6.2	Setting Triggers Conditions	50
Figure 6.3	Trigger set and active condition	50
Figure 6.4	Test circuit 2 “test_blink”	50
Figure 6.5	Serial Monitor Output	51
Figure 6.6	Password Protection of HA for Security	51
Figure 6.7	Process manager Htop showing HA on Raspbian	52
Figure 6.8	MQTT Publish Test Result	52
Figure 6.9	MQTT Subscribe Test Result	52
Figure 6.10	Lovelace UI Logbook View	53

CHAPTER 1

INTRODUCTION

1.1 GENERAL

The Internet of Things (IoT) is the physical world's act of digitization. It is a comprehensive term used to indicate a system where wireless and wired connections connect the physical universe to the web. It is not a single technology, rather a tandem agglomeration of different techniques. The Internet of Things industry in India will be worth \$9 billion by 2020, according to "IoT India Congress 2018". In all main industries, including telecommunications, health, agriculture, cars and home, the adoption of IoT technology is set to rise. Globally, the end of 2019, Gartner estimates that 14.2 billion connected things will be in use which will touch 25 billion by 2021.

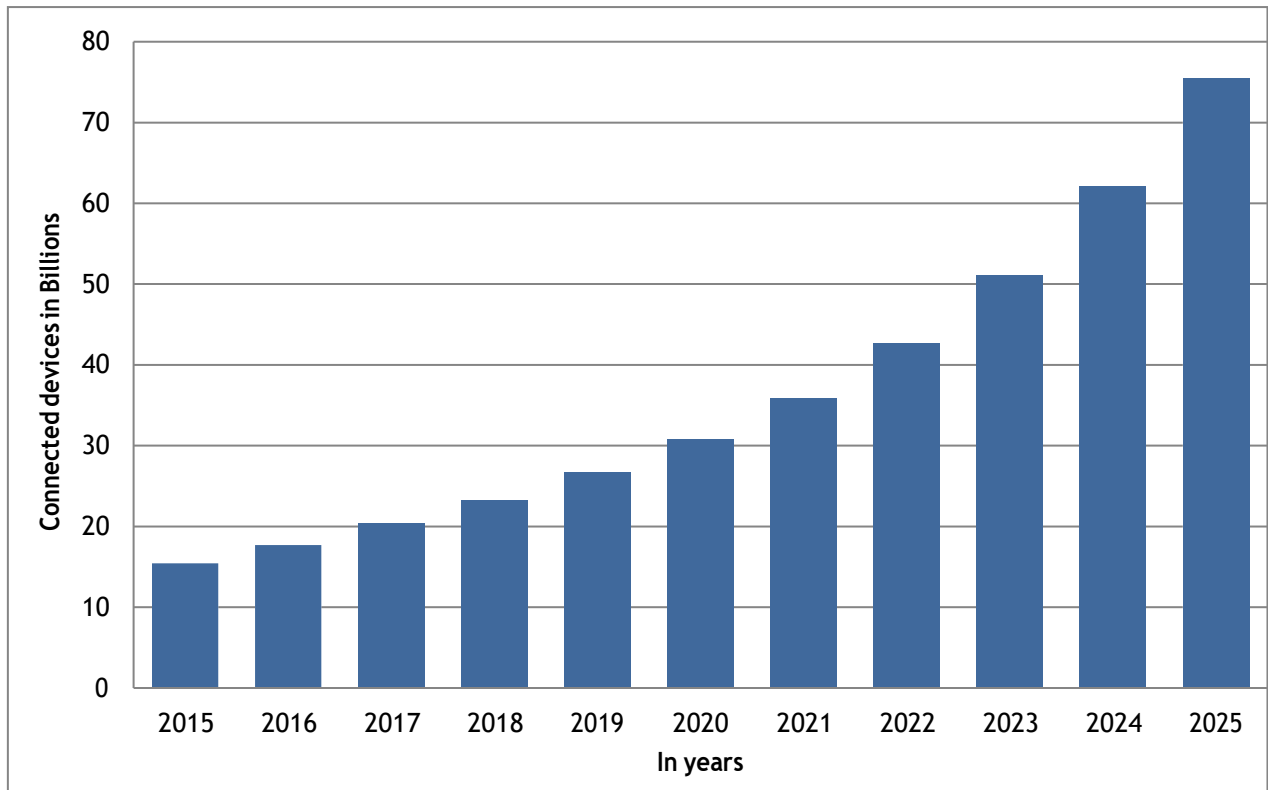


Figure 1.1: The present IoT trend

1.2: MOTIVATION: The Indian Market Scenario

The IoT is well-poised to bloom in the next technological revolution since the advent of the Internet. It has been drawing more and more attention as the age turns increasingly digital employing smart technologies. Although India began its journey into the IoT much later than developed countries, its established hub of connected devices is expected to rise at a rate much faster than them. In 2017, the World Bank pointed out the Indian government’s campaign – “Digital India” as a prominent endeavor to making firms more ambitious by applying IoT technologies and the growth henceforth can be seen in Figure 1.2

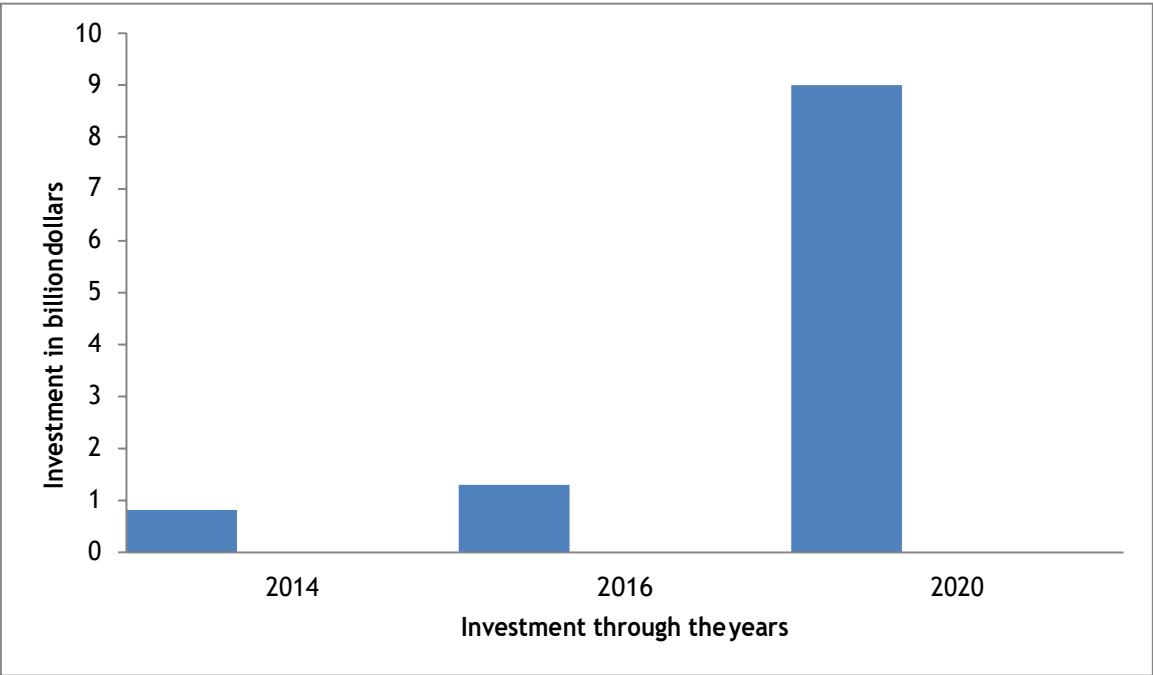


Figure 1.2: The IoT market in India (Source: Deloitte)

The adoption in India is expected to progress across all mainstream industries by the next decade. The prominent sectors driving it currently are transportation, utilities, manufacturing, logistics and automotive. Due to the above listed direct association with Smart city projects, they are expected to see the highest adoption. Healthcare, Retail and Agriculture are the rising sectors adopting smart technologies for solutions. Application across India can be enveloped under three broad classifications- Consumer, Industrial and Public sector.

1.2.1 : Consumer Applications

The expectation of consumers is good living standards and better quality of life as the perception of IoT is limited. Although the applications are not as wide as industrial in this sector, they are opening utilitarian and intimate experiences like wearables, fitness trackers and smart homes. Smartphone applications can now lock houses, set and reset thermostats and even research is ongoing on ways of cooking and monitoring on the basis of ambient temperature.

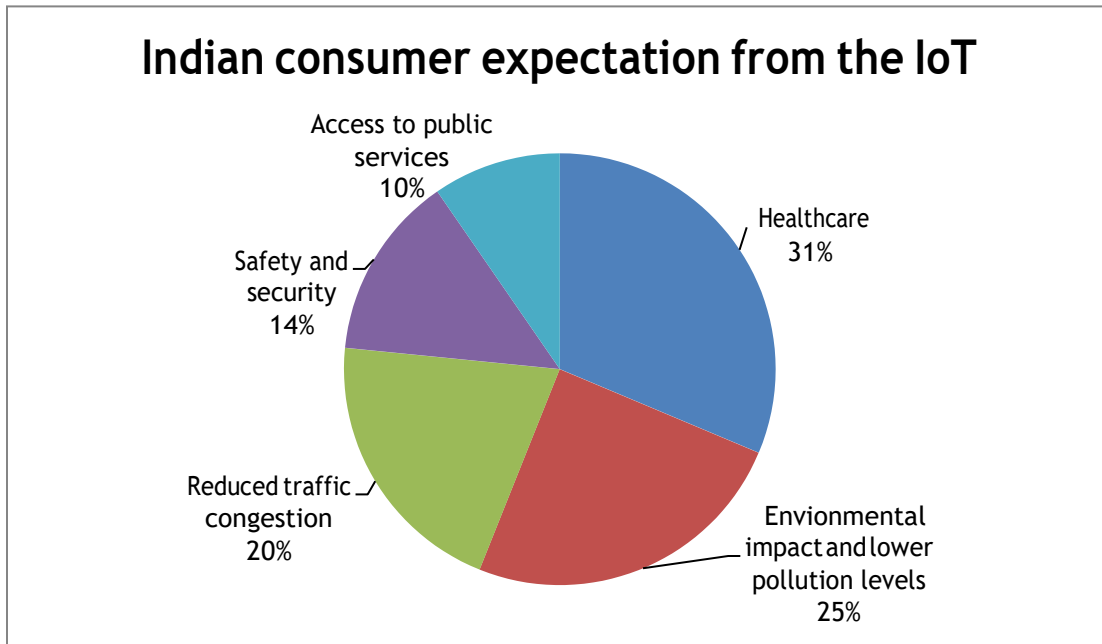


Figure 1.3: **Indian consumer expectation from the IoT** (Source: Tata Communications)

1.2.2: Industrial applications

India as well as globally, the IIoT (Industrial Internet of Things) has far exceeded all applications. The term Industry 4.0 accounts the onset of a new industrial revolution through automation, connectivity and analysis. Some major applications are:

- Supply chain: Goods tracking, connected factories, robotics and improved process automation
- Agriculture: Chemical and fertilizer testing, moisture and pH factor of soil
- Energy: Management and early fault detection
- Transportation: Vehicle tracking, Traffic management

1.2.3: Public Sector Applications:

The chart given in Table 1.1 lists the potential end use in various sectors of the IoT. This is of significance in the coming years as the push for Smart cities is strong has enabled integration of technologies across a wide area.

Table 1.1: **Public sector-wise applications**

Sector	Sub-sector	Examples
Transportation	<ul style="list-style-type: none">• Public Transportation	GPS tracking and real time monitoring of buses to give information on wait time
	<ul style="list-style-type: none">• Traffic	Smarter control of traffic lights to prevent congestion
	<ul style="list-style-type: none">• Public bikes	Tracking by requirement for crowd control and reducing automotives on road
Security	<ul style="list-style-type: none">• Drones	Computer vision, tracking and mapping, surveillance
	<ul style="list-style-type: none">• Fire safety	Smoke detection and automated notification. Sensors in gas pipes can enable early detection
Health	<ul style="list-style-type: none">• Assistance	Single-push button for emergency alert
	<ul style="list-style-type: none">• Medication	Automate medicine supply for Diabetes or high blood pressure and sensor based

		notification for medical devices
Environment	<ul style="list-style-type: none"> • Pollution 	Monitor pollution levels and presence in quantity of polluting particles
Economy	<ul style="list-style-type: none"> • Industry 	Smart infrastructures

1.3: ARCHITECTURE

The IoT has been defined as a paradigm, in which networking and computing capabilities are embedded in a probable object to make it smart. Collaboratively, they can be made to accomplish tasks that require high level of intelligence. For this purpose, IoT systems are fitted with integrated sensors, processors, actuators and transceivers.

The five layer architecture to define the flow of process is given below:

1. **Perception Layer**
2. **Network Layer**
3. **Processing Layer**
4. **Application Layer**
5. **Business Layer**

1.3.1 PERCEPTION LAYER: SENSORS AND ACTUATORS

The Internet of Things cannot be considered without the topic towing towards the new data economy and the information it holds. The devices, sensors and actuators are helpful in interacting with the physical world. The merit of an IoT system is measured by what it can learn from that data. Sensors are the primary source for this. A preferred term for a sensor is a transducer which can be defined as a physical tool that converts one energy form into another, like an electrical impulse which can be interpreted better for reading. To derive helpful inferences, the information gathered by the sensors must be stored and processed intelligibly. A

mobile handset or even a refrigerator can count as a sensor as long as they provide inputs about their present state i.e. internal state and the environment.

An actuator is a device that is used to bring about a change in the environment such as the temperature and humidity controller of an air conditioner. It operates in the reverse direction of a sensor by taking an electrical input and bringing about a physical action as an outcome. It is possible to store and process information on the top of the network itself or on a remote server. If any pre-processing of information is feasible, then either the sensor or any other device in the vicinity is used. The nature of actions can be diverse. Context Awareness is one of them which means change in the physical world is dependent on its state at that point of time. In standard IoT systems, in reaction to a sensed input, a sensor can obtain data and route it to a control center or remote server where decision-making takes place. A corresponding command is sent to an actuator to respond towards the input as depicted in the block diagram in Figure 1.4

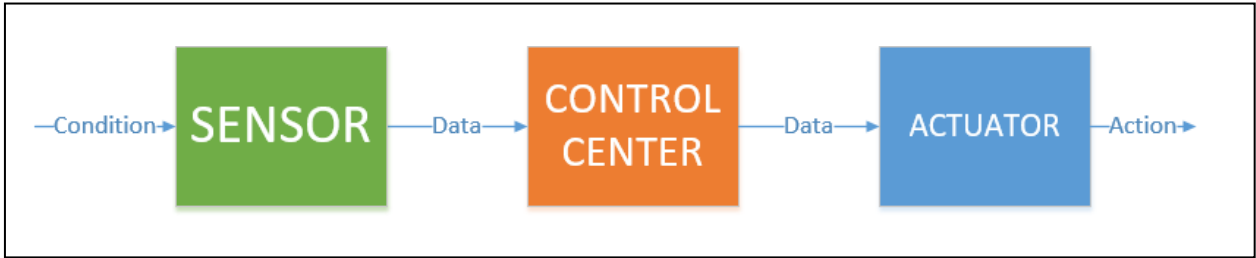


Figure 1.4: **Sensor to Actuator flow diagram**

1.3.2: NETWORK LAYER:

The Network layer transfers sensor data across networks such as wireless, 3G, LAN, 4G, NFC and Bluetooth from the Perception to the Processing layer. It is responsible for the connection over servers and network enabled devices, both wired and wireless and a few parameters for selected protocols are shown in Table 1.2

Table 1.2: Comparison between Network Protocols

Technology	Data Rate	Range	Power Usage	Cost	Frequency
Bluetooth/ BLE	1,2,3 Mbps	~ 300 ft	LOW	LOW	2.4GHz
3G/4G	10-20 Mbps	Several Miles	HIGH	HIGH	Cellular
Wi-Fi	0.1-54 Mbps	< 300 ft	MEDIUM	LOW	subGHz, 2.4GHz
ZigBee	250 Kbps	~ 300 ft	LOW	MEDIUM	2.4 GHz
NFC	424 Kbps	< 200 ft	HIGH	HIGH	2.4 GHz

1.3.3: PROCESSING LAYER: MIDDLEWARE

Software technology is used as a middleware for management, growth and inclusion of various devices and applications in an IoT setting. Here, storage, analysis and processing of huge amount of data take place that comes from the network layer. It provides abstraction to applications from the things and offers various services. Development of middleware in this field is an active area of research and an event based model can be seen in Figure 1.5

The basic features and challenges of a middleware are as follows:

- Resource discovery without human intervention with every device declaring its presence
- Resource management without human intervention
- Data management starting with acquisition, filtering, aggregation and compression
- Event and source code management.
- Scalability
- Security and Privacy
- Ease of Deployment
- Context Awareness
- API for application development

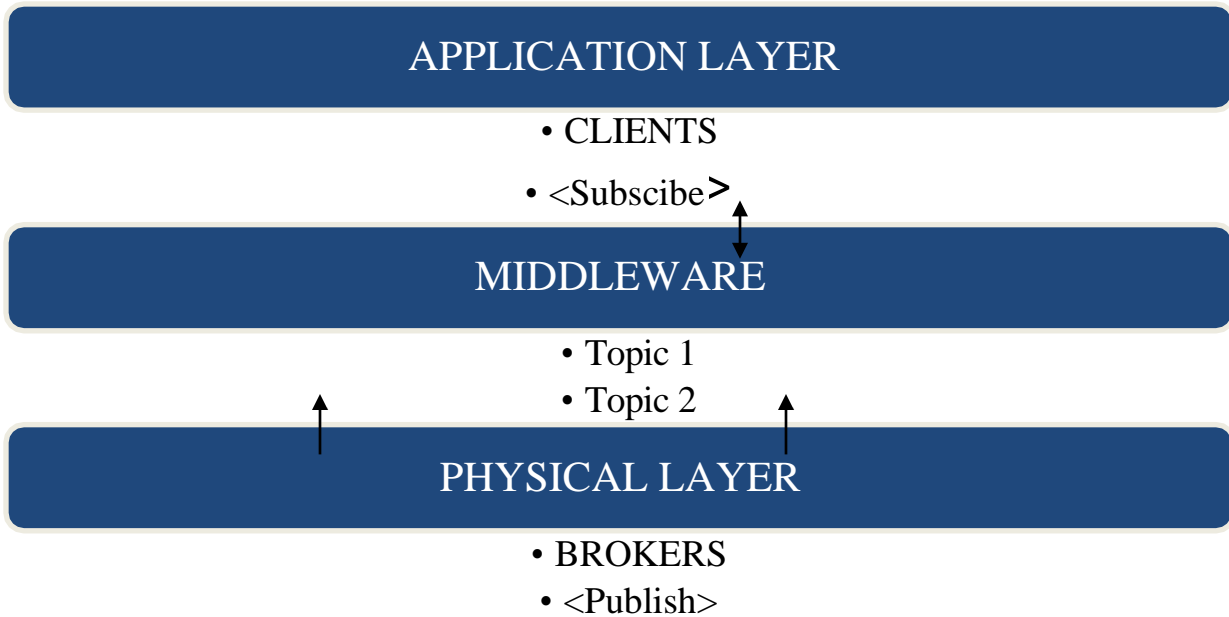


Figure 1.5: An Event based Middleware Flowchart

1.3.4: APPLICATION LAYER

It is the responsibility of this layer to provide users with application oriented services. It defines various applications in which IoT can be deployed such as Smart Cities, Health and Home. The application layer on the Internet is based on HTTP (Hyper Text Transfer Protocol) which is deemed unsuitable in a resource constrained environment because of its verbose nature and large overhead. Many alternate protocols have been developed to suit IoT environments such as MQTT, CoAP, REST, AMQP etc. When selecting protocols to use in IoT, the following factors must be accounted for:

- Data Latency
- Reliability
- Bandwidth Requirement
- Memory and Code Footprint

1.3.5 : APPLICATION LAYER PROTOCOLS

The commonly used protocols are HTTP, OPC UA, MQTT, CoAP, AMQP and many more. We will focus on a few of them for constrained applications.

1.3.5.1 : MQTT (Message Queue Telemetry Transport)

It is an M2M architecture developed to support publish/subscribe architecture over TCP (Transmission Control Protocol) to enable lightweight connectivity. TCP brings in stream simplicity and minimizes the risk of data loss. The publish/subscribe architecture is noteworthy because it annuls the requirement of clients to request updates which minimizes computational requirements, battery and bandwidth. It is suitable for home automation and mobile communication as it lies on a star architecture where all devices connect to a central server which is referred to as the Broker. The communication happens in the following three stages:

1. TCP connection established
2. MQTT connection established data published
3. TCP connection terminated

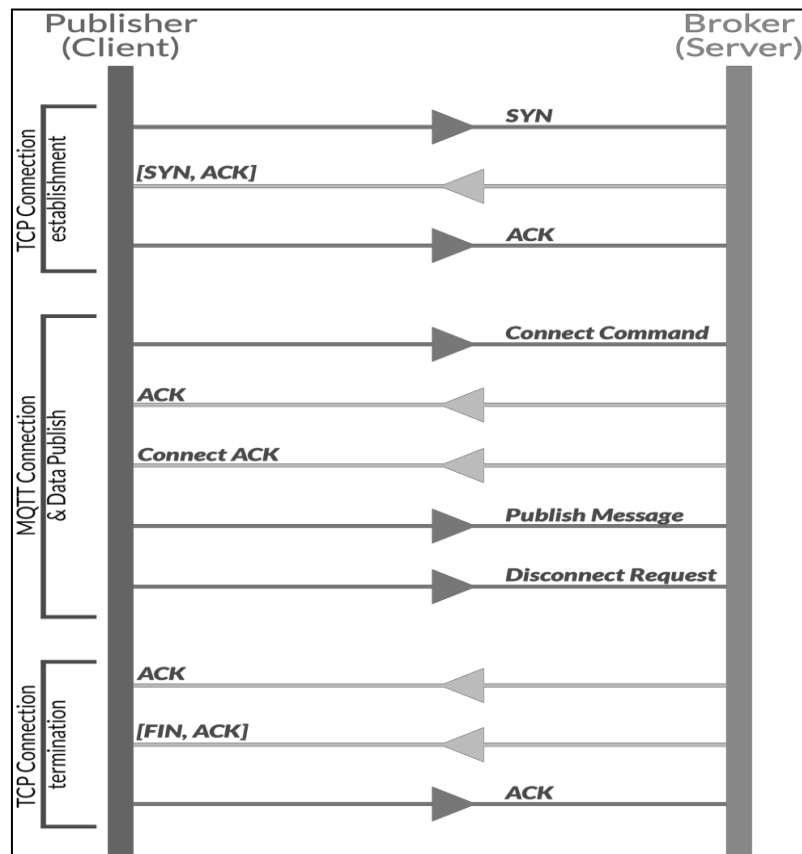


Figure 1.6: Working of Pub/Sub model in MQTT

1.2.5.2: CoAP (Constrained Application Protocol)

As the name suggests, it has been specifically developed for resource-constrained devices as it uses minimal resources for low power applications. Unlike MQTT, this uses client-server architecture over UDP (User Datagram Protocol) which is a document transfer protocol and made use at Smart Energy grids and smart homes. This protocols supports four methods, namely GET, POST, PUT and DELETE. This is a small HTTP protocol but supports encryption and multicasting by enabling communication between multiple devices at one time. A demonstration of using in hospitals for patient health monitoring and data storage can be seen in Figure 1.7

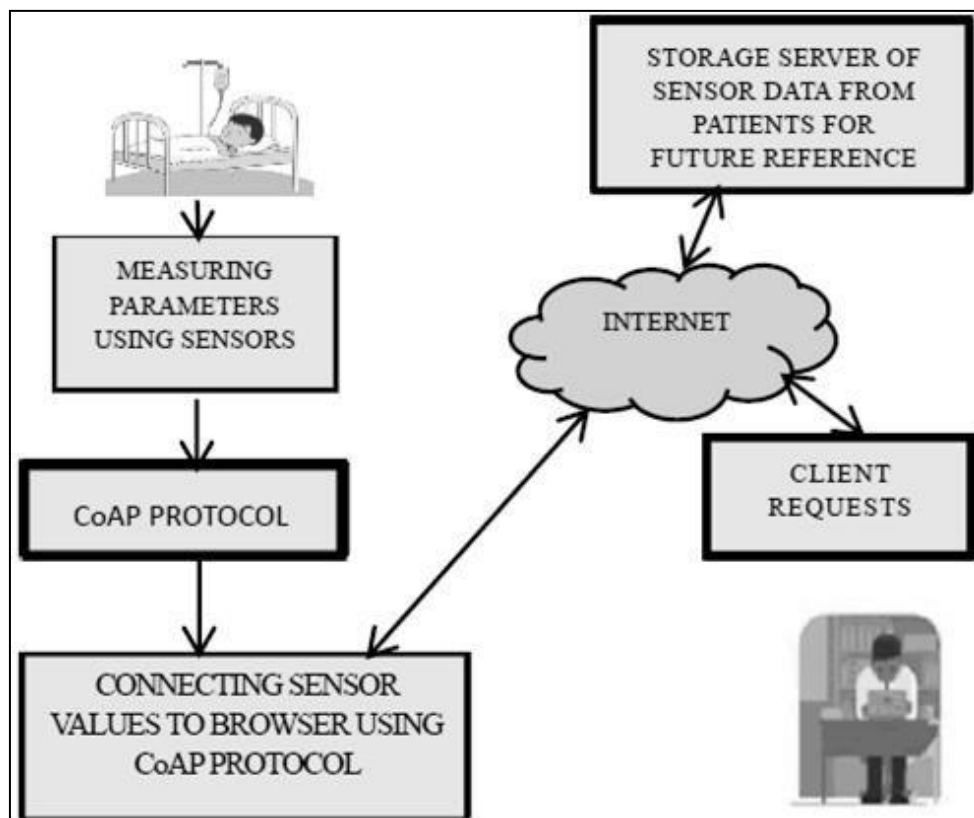


Figure 1.7: Work-flow of the client-server architecture in CoAP

1.2.5.3: REST (Representational State Transfer)

It enables synchronous request/response over HTTP but there is difficulty in implementation due to large overhead and latency. It can be used in M2M, smart phones and tablets and works around JSON and XML data formats.

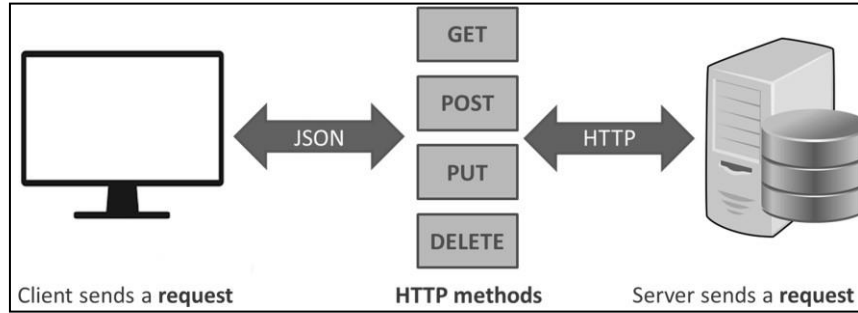


Figure 1.8: **REST work-flow model**

1.2.6: BUSINESS LAYER

The business layer manages the whole IoT system, including applications, business, profit models and users' privacy based on data that has been received from the application layer. This data is molded into a significant service and more of them are created from the current services. The true relevance of the IoT technology is the product of a good business model. Analysis in this layer helps develop strategies to use and automation. Figure 1.9 summarizes all the layers with details.

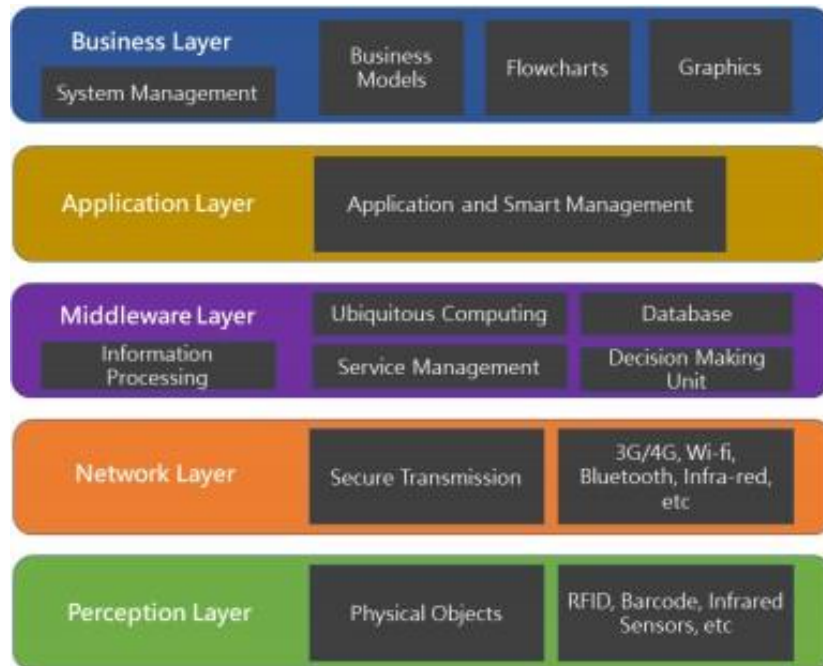


Figure 1.9: **5-layer architecture adapted in IoT**

CHAPTER 2

LITERATURE REVIEW

2.1 : INTRODUCTION

In this endeavor, a deep focus has been maintained on understanding the OSI model, specifically middleware defined to build an IoT environment ranging from three to five layers. Smart Buildings (SB) were originally known as Building Control Systems (BCS), Building Automation Systems (BAS) and Building Management Systems (BMS) among others, but the evolving intricacies of protocols and devices, sensors, actuators, controllers and interconnection with other proficient systems and the internet called for a general term called Smart Buildings. It provides mechanisms, hardware and software to monitor, automate and control not just indoor but outdoor building related tasks as well. For instance:

- Climate control, with HVAC systems including cooling, humidification and air quality
- Visual ambience, with natural and artificial lighting
- Safety, with alarm systems like fire, gas or water leakage; Emergency lighting
- Surveillance and security, with audio and video
- Transportation, with elevators and escalators
- Supply and recycle, like waste management
- Demand response, with energy management

Last decade has seen a significant growth of smart devices, low power consuming chips and SoC's, networking technology and smart sensors. Building sector happens to be one of the largest consumers of electricity in India with residential sector accounting for 22% and commercial sector at 8% of the aggregate energy consumption, still rising at 8% annually in these sectors (Dr Satish Kumar, USAID ECO-II Project, 2011).

As per the NMEEE (National Mission for Enhanced Energy Efficiency) document of 2009, the annual consumption from residential and commercial buildings is expected to rise from 19200 KWh to approximately 89,823 KWh by 2030.

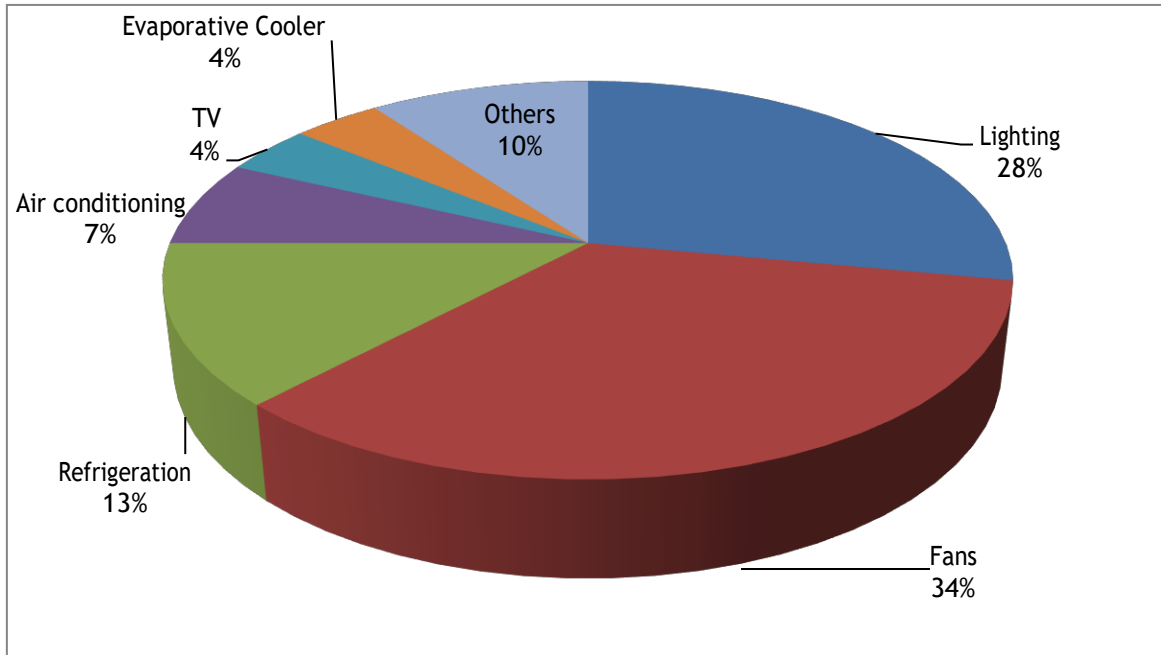


Figure 2.1: **Energy Consumption in residential Sector (Source: Planning Commission)**

The consumption of energy in buildings is dependent on certain factors like:

- Weather Conditions
- Daylight/Sunshine Hours
- Building Design
- Ambient Temperature
- Indigenous efficiency of equipments
- Efficiency of equipments

Hence, reliance on energy driven systems can be co-factored by using climate sensitive designs where the building premises responds in favor of it.

2.2 : ENERGY CONSERVATION SCHEMES IN BUILDINGS

2.2.1 : ECBC (Energy Conservation Building Code)

Under this act, any building with a minimum energy demand of 120 kVA or a connected load of 100kW or greater is covered. Hence, this act sets minimum standards for construction

and design through building systems helping the government to enforce efficient energy use and its conservation.

2.2.2 : LIGHTING

Three commonly used lighting types in India are Incandescent bulbs, CFL (Compact Fluorescent Lighting) and LED. Conventional incandescent bulbs waste about 90% of their energy emitting heat instead of light while CFL’s can be up to three times more efficient than by using 75% less energy and lasting longer. LED is the fast-growing and widely accepted lighting fixture at present. They make use of only a fourth of the energy provided and last 25 times longer than conventional bulbs. In addition, they come with color variation and dimming features for ambient lighting. Dimmers save electricity when employed to lower light levels as shown in Table 2.1

Table 2.1: Comparison of Lighting Fixtures

Compact Watts (W)	Fluorescent Lumens (Lumens)	Standard Watts (W)	Incandescent Lumens (Lumens)
30	2400	150	2780
22	1900	100	1500
20	1200	75	1170
15	900	60	2780

2.2.3 : RENEWABLE ENERGY SOURCES

Among the many options available, Solar panels are the most preferred form of renewable energy at this day. They find use in electricity and heat generation, along with outdoor and indoor lighting. Small wind turbines for water pumping are also a viable option.

2.2.4 : SELECTION OF APPLIANCES

The BEE (Bureau of Energy Efficiency) has set an energy standard for electrical appliances by giving them star ratings on saving. When we talk of buildings, a substantial amount of energy is consumed by the appliances in it like air conditioners, refrigerators, Iron

boxes, microwaves etc. It is also important to place appliances strategically, for e.g. a refrigerator near a heat source will pull more electricity than required to cool.

2.2.5 : SMART & GREEN BUILDING PROJECTS

The trend of going green and getting smart has popularized itself well over India, which offers opportunities for building automation systems a.k.a smart buildings. The various strategies for which they are adopted for are:

- Lighting management systems incorporated with motion sensor and dimmers
- Safety and security system
- Auxiliary power bank like diesel generator set or from PV cells
- Integration of HVAC
- Resource efficient and Environmentally responsible
- Reduce, Reuse and Recycle

2.3 : SMART SENSORS

The Physical layer as defined in the IoT architecture is the input layer which obtains feedback and is used for measurement and control. The papers included here collaborate to studies on how to make the input parameters more involved in sensing, It also takes in consideration context awareness which studies the input as well as the present ambient state to generate an output to be fed to the core controllers.

Chin-Chi Cheng *et.al* presented a smart air conditioner with a sleep timer optimized the energy consumption by up to 49% with wearable devices as smart sensors. The model procured could detect human temperature and activity during sleep which worked along controlling the air conditioning. The paper also presented control with mobile phones with the intention of improving air conditioner technology.

Dae-Man-Han *et.al* proposed a Home energy control system using IEEE 802.15.4 and ZigBee to demonstrate its implementation using a real test bed. A new on-demand based routing protocol called DMPR (Disjoint Multi Path Routing Protocol) was used in the setup to establish a wireless network between smart nodes.

Yago Luiz dos Santos *et.al* proposed an Iot architecture, applied to a case study for reading Ultra High Frequency Tags with an external UHF antenna for better results. They make use of micro-services and cloud computing for management of the large data that the Radio Frequency Identification Tags generate used in the process system. They made use of a development board called Sparkfun Simultaneous RFID Reader with distance approximation to increase the reading capability of the board. They made use of services provided by AWS (Amazon Web Services) and Docker to run an Iot environment. Emphasis was also on cost reduction in areas where reading distance is a fundamental requirement. The results obtained from the test demonstrated judicious use of the cloud services along with Microsoft Azure for the cloud management and data control on the server side of working.

Quinping Chi *et.al* addressed problems related to sensor calibration like sampling rate and refresh time by proposing a reconfigurable smart sensor design for industrial WSN (Wireless Sensor Networks) in an IoT environment. The core controller is a Complex Programmable Logic Device (CPLD) which can adopt connection and read data simultaneously in real time on multiple sensors as shown in figure 2.2

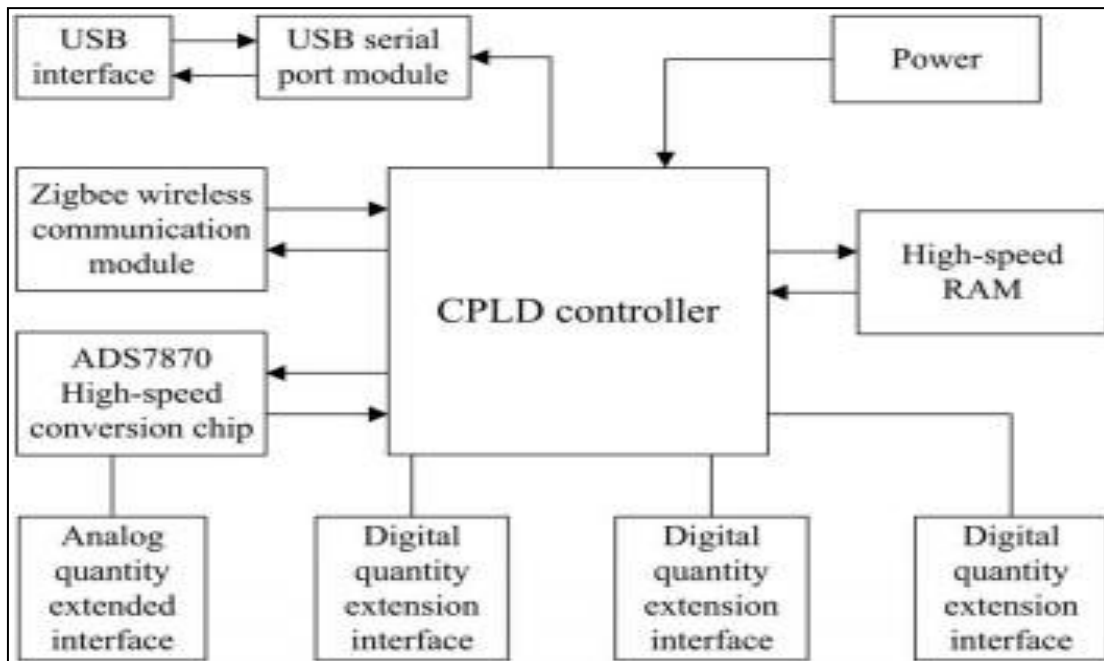


Figure 2.2: A CLPD core-enabled hardware design

2.4: IoT SECURITY AND THREATS

While IoT presents efficiency, convenience and accessibility, it has caused an insightful threat to security and privacy in the recent years. There are research works being carried out actively to address this cause of concern to maintain reliability in this upcoming sector.

Wei Zhou *et.al* presented a concise study on the developing trend on IoT security by investigating the research works carried out from 2013 to 2017 as listed in table 2.2. The majority of works gravitates towards insecure network or protocol problems or privacy breach/disclosure. The lack of security awareness is the leading cause of a vulnerable cloud and web services.

Table 2.2: A brief comparison of Threats and Challenges of IoT features

Feature	Drawback	Challenge	Opportunity
Constrained	Insecure Systems	Lightweight defenses and protocols	Combining biological and physical characteristics
Mobile	Malware Propagation	Cross-domain identification	Dynamic Configuration
Inter-dependence	Bypassing static defenses	Access control and privilege/admin management	Context-based access
Unattended	Remote Attack	Remote Verification	Lightweight Trusted Execution
Diversity	Insecure Protocols	Fragmented	Dynamic analysis simulation platform

Majzoobi *et.al* proposed in their paper the validation and an algorithm for a safe access key generation using a technique called PUF (Physical Unclonable Functions) which makes use of the device structure for identification. This method saves key storage space and makes the key generation algorithm relatively simpler. More work was done on gait and usage habits to collect information for the algorithm.

Chen *et.al* presented a layout for extensive automated firmware dynamic study running on a Linux-based system. The same study for an RTOS (Real Time Operating System) is still in progress to make integration more open sourced.

Sullivan *et.al* presented their work on IDS (Intrusion Detection System) and IPS (Intrusion prevention System) for device protection of their system. Research on heterogeneous devices is still in progress as the system may not function properly when there is an anomaly in incoming data traffic. Different devices integration is possible only by building a gateway for protocol acceptance and conversion. The detection may not be successful if there are unidentified parameters as inputs.

2.5 : SMART BUILDINGS MANAGEMENT SYSTEM: BEMOSS

BEMOSS (Building Energy Management Open System Software) , as it started, was a proposed platform to allow sensing, measurement, processing, filtering and control of Lighting, Plug load and HVAC. It is now a MAS (Multi Agent System) that is capable of integration of multiple smart devices to facilitate grid-interactive and intelligent building operation. The entire architecture comprises of four layers: 1) User Interface (UI); 2) Application; 3) Operating System and Agents; 4) API Translator.

Each layer uses databases to store metadata which is useful for process management. The UI consists of a mobile and Web browser interface with a role-based access control. Possible applications in the application layer are:

- Demand Response
- Behavior pattern analysis
- Planning and Scheduling
- Fault detection and diagnostics
- Price-based management
- Load shape analysis

It supports different communication technologies like Ethernet (IEEE 802.3), Wi-Fi (IEEE 802.11), Zigbee (IEEE 802.15.4) and Serial (RS-232/485); and Data exchange protocols like Modbus, Web, Smart Energy (SE), OpenADR, Zigbee API and BACnet, to name a few.

The third layer is built upon a distributed agent called VOLTTTRON™ whose Information exchange bus (IEB) enable communication among all agents such as Discovery agent, Control agents, Monitoring agents and other Service agents, including the UI.

Every device has a unique API (Application Programming Interface) to which the fourth layer connects BEMOSS to by using translators. They provide a medium of abstraction to obtain readings and send control commands using simple functions like “getDeviceStatus” and “setDeviceStatus”. A standard work-flow on the working and implementation of BEMOSS can be see from figure 2.4. This has been used in several test beds successfully where price and resources are not a constraint. However, we are in a developing nation with an economic limitation. The scope of work beyond this is to provide the same features as the inspired works, but at a lesser cost, adaptable in feature and easy to implement with cloud independent support.

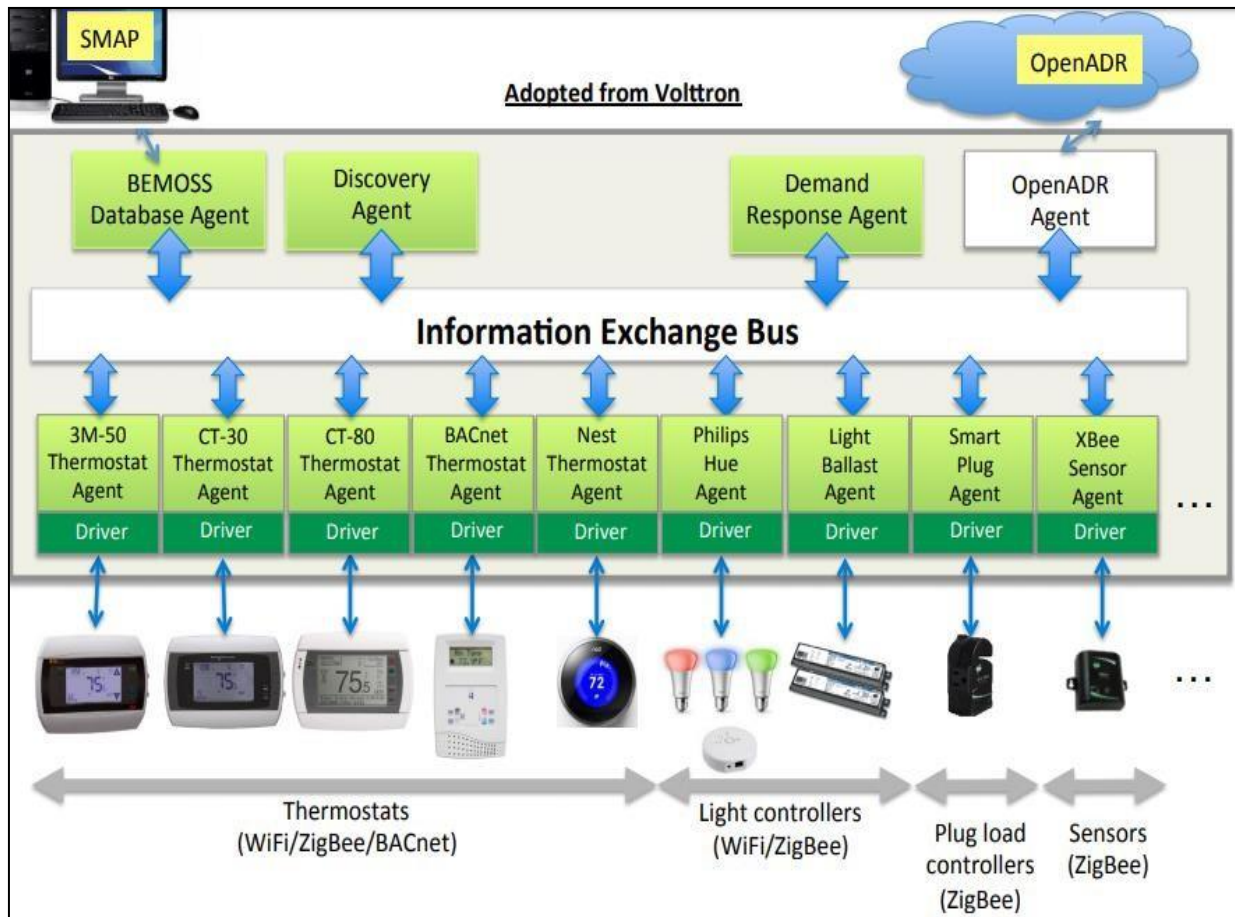


Figure 2.3: **BEMOSS Agents and work-flow**

CHAPTER 3

SENSORS and UI PLATFORM USING MQTT

3.1 : INTRODUCTION

In electronics, it is essentially the application of sensors with/without computer programs that play the most important role i.e. of input. They are being used extensively in consumer electronics for monitoring, measuring, data logging and control. Common sensors are optical, acoustic, radio frequency (RF), proximity, temperature, pressure, ultrasonic, flow and level. Modern sensor technologies enable integration with portable devices, home appliances, smart home automation, robotics, automotive and healthcare.

3.2 : TEST CIRCUIT MODELING

The test circuit used for sensing can measure light intensity, temperature and relative humidity. The actuators in this case are two loads connected to the test circuit through an opto-coupler 4 channel relay. The microcontroller used for this purpose is an Espressif development board called NodeMCU V1.0 (ESP8266 ESP-12) which is an economic, IoT capable, responsive circuit board written in C, LUA or micro-python. The circuit is powered by a 12V DC supply to drive the relays. With 11 GPIO (General Purpose Input Output) pins and one ADC input pin, it can be used to perform actuation tasks for both analog and digital input/output. Components around the header pins are wired on a general-purpose PCB or on a designed layout.

3.2.1 : Test circuit components:

1. Board: ESP8266 NodeMCU development kit (ESP-12)
2. 4N33 opto-couplers
3. Sensor 1: DHT11
4. Sensor 2: LDR (5-mm)
5. Resistors (1 kilo-ohm)
6. Power supply (12 Volts)
7. Software: Arduino IDE, Cayenne IoT Platform, Fritzing, WireShark

The NodeMCU development kit provides access to the features of the board it hosts i.e the ESP8266 processor. The GPIO pins can only be accessed through these, with the pin numbering different from the internal numbering of the ESP8266 as shown in figure 3.1

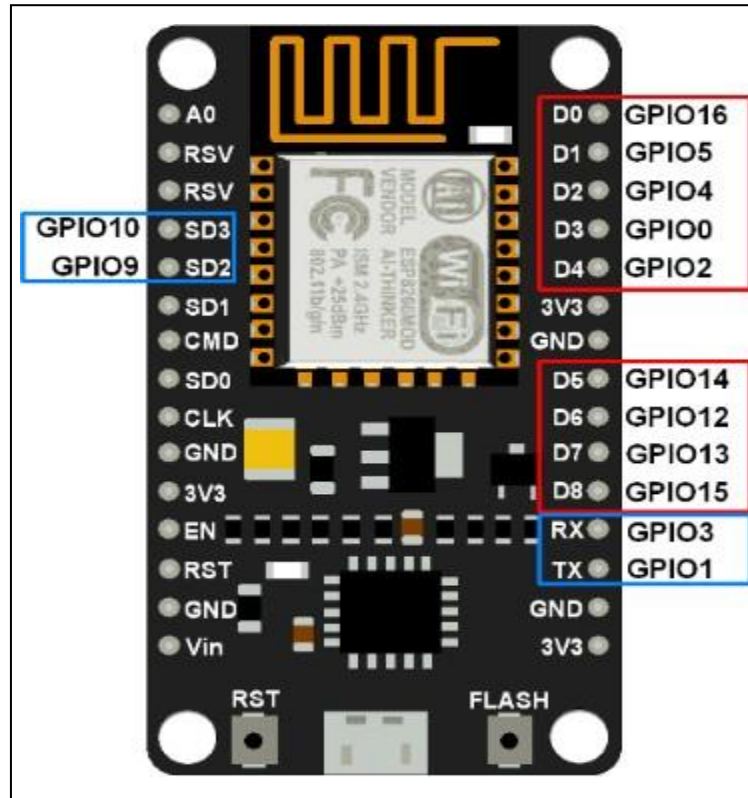


Figure 3.1: ESP8266 NodeMCU development kit

The key features of this board are:’

- Programmable Open source Wi-Fi module
- LUA and C compatible
- 10 GPIO (D0 TO D10), PWM capable, IIC and SPI communication
- USB, TTL, ACM (Abstract control model)
- Arduino IDE supported
- Works both at 5V and 3.3V voltage levels
- Plug and Play with CH340G driver support
- Supports deep sleep to save power
- Can be used both as a Station or Access point

3.2.2 : Test circuit design models:

1.

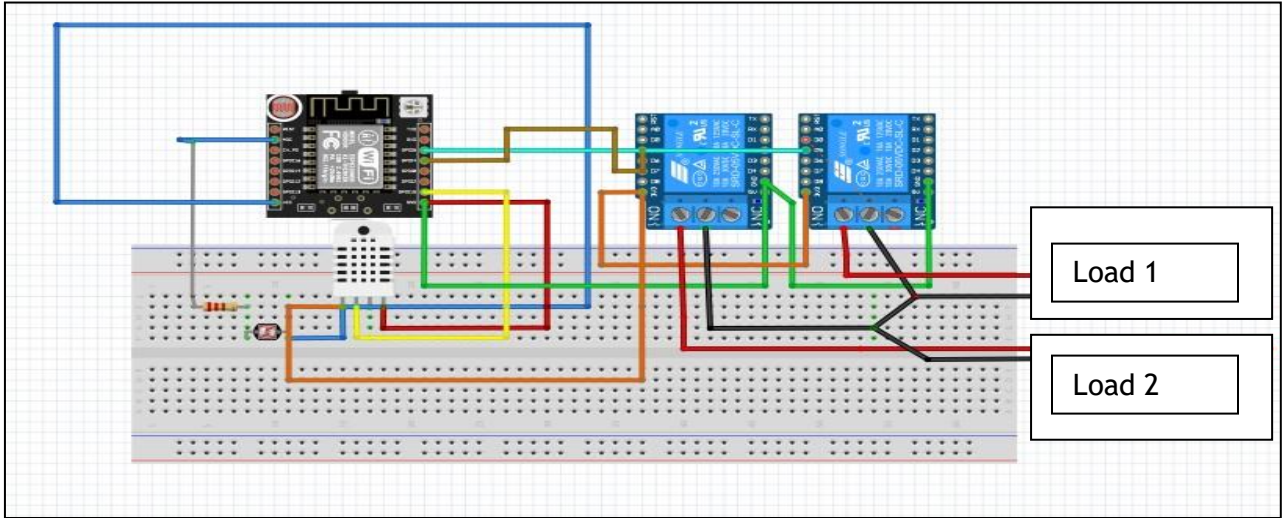


Figure 3.2: Components layout of the Test

2.

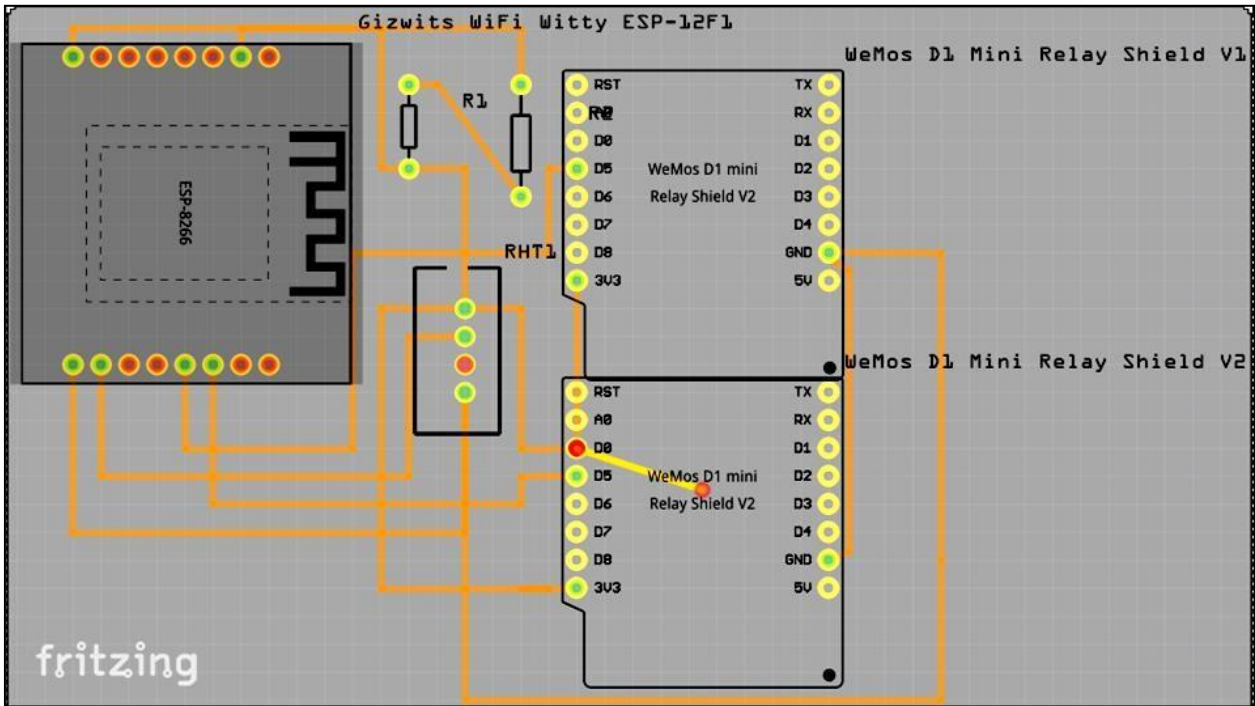


Figure 3.3: PCB layout of the Test circuit

3.

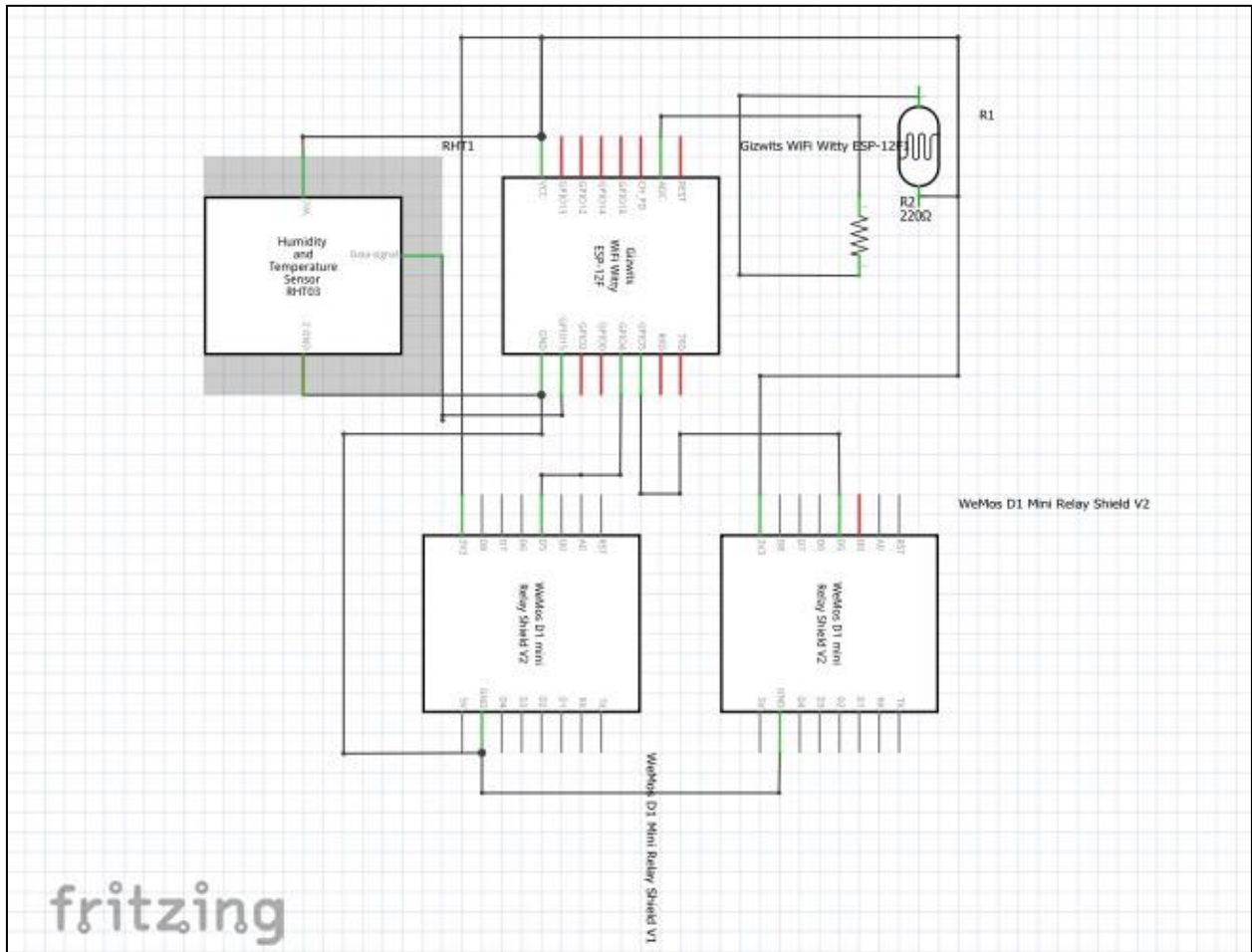


Figure 3.4: Schematic diagram of the Test circuit

3.3.1 : CAYENNE ONLINE PLATFORM

It is an online IoT platform with a dashboard and a mobile application supporting its web interface. For hardware-oriented programming, it makes easier for measure and control of devices. Along with its drag and drop feature, building programs get easier by standardizing the connection of devices such as motors and sensors by making use of their respective drivers. It also supports the light-weight MQTT protocol for data logistics in lieu with the Pub/Sub architecture. A dashboard featuring sensors of the test circuit can be seen in Figure 3.4



Figure 3.5: Test circuits dashboard on the IoT platform

3.3.2 : MQTT PROTOCOL

This is a quick emerging standard for IoT nowadays due to features like lightweight overhead, less bandwidth, low latency, publish-subscribe model, small code footprint and bi-directional capabilities on top of the TCP/IP model. They are preferred for networks like cellular or satellite which are volatile and do not require high availability or bandwidth to initiate connections. Data transmission is widely distributed which is beneficial for remote devices with limited memory and processing power. Hardware manufacturers prefer this protocol for scalability, security and efficiency. Some prominent features of this protocol are:

- Open source, easier to adopt and implement
- One to many distribution, Pub/Sub model
- Simple commands
- Small message headers
- Multiple QoS (Quality of Service) levels

The QoS are efficiency levels set for the transmission of information. It decides how each message will be delivered by assigning a mandatory value for every single message unit. It validates the guarantee of transmission between the sender and the receiver, being set on the client side. The broker will send the message to the subscribers with the same QoS level set originally.

There are three QoS levels, mainly 0, 1 and 2 as follows:

- QoS 0:

This is the simplest layer, with low overhead as client publishes a message but there is no acknowledgement by the broker. A message in this format can be lost if either the client disconnects or the server fails. There is no possibility of PUBACK duplicate messages as message gets delivered only once. It still stands to be the fastest way to send a message using MQTT as only PUBLISH command is used here to maintain the flow as shown in figure 3.5. This can be applied where loss of message can be tolerated now and again, such as in an IoT environment where a device is monitoring and sending cumulative readings.

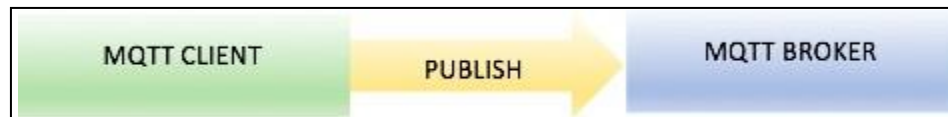


Figure 3.6: **QoS0 level (at most one message delivery)**

- QoS 1:

This level comes with a guarantee that the message will be attempted to be delivered once, but may be eligible to be delivered more than once. Hence, there is a possibility of duplicating and multiple attempts by acknowledgement PUBACK sent by the broker to the sender. If this feedback is not received, sender can publish again with a duplicate bit set called DUP. Hence, flow and affirmation is maintained, although the message may reach the broker multiple times. When a PUBLISH occurs, message is stored in an abstract layer like a disc, and removed when a PUBACK is confirmed as depicted in figure 3.6. This format can be used when the IoT device in use can tolerate receiving a message more than once; the way of doing around is by using a unique timestamp attached to every publication.

Security comes at the cost of battery/processor utilization and communication overhead. But, on application level, using Client ID, password and username as credentials, device authentication can be done as in case of HTTP as shown in table 3.1

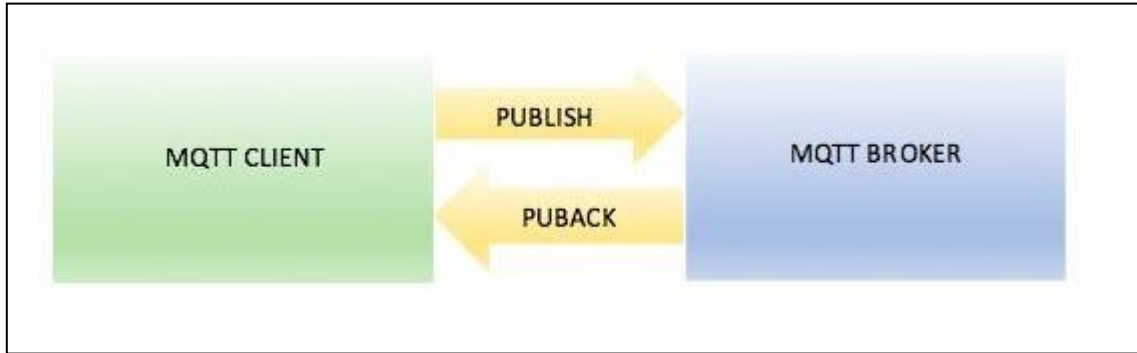


Figure 3.7: **QoS1 level (at least one message delivery)**

- **QoS 2:**

This is an additional layer to the QoS 1 layer, which ensures that the message gets delivered only once and at once. There is a sequence of four messages between the client and subscriber which is handshake to confirm that the original data has been sent and an acknowledgement has been received. PUBLISH flow takes the message and stores in the abstract layer which is responded by the broker with PUBREC (Publish Received). On getting this, client sends PUBREL (Publish Reliance) which is acknowledged and main data is sent to subscribers by PUBCOMP (Publish Completed). This does not compromise on delivery but the cost of data transfer is relatively high.

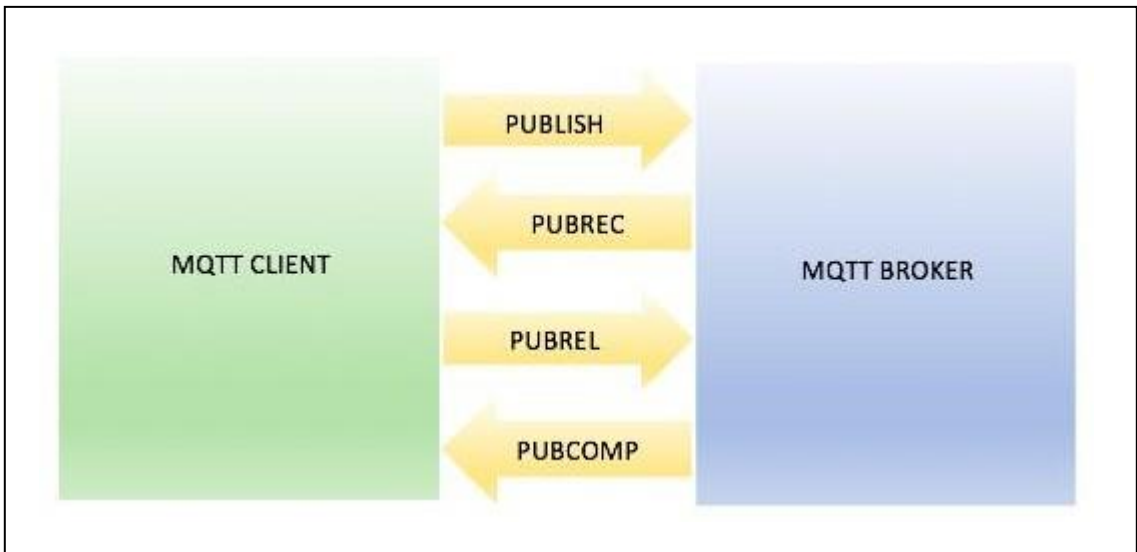


Figure 3.8: **QoS 2 level (exactly one message delivery)**

Table 3.1: MQTT vs HTTP in a constrained environment

Characteristics	MQTT	HTTP
Design	Data-focused	Document-focused
Base Model	Publish/Subscribe	Request/Response
Complexity	Simple use commands	Complex
Message size	Small headers Binary format, up to 2 bytes	Large headers Text format
Service Layers	QoS 0 QoS 1 QoS 2	Uniform service level for all messages
Distribution	One to many	One to One

(ip.addr eq 127.0.0.1 and ip.addr eq 127.0.0.1) and (tcp.port eq 49705 and tcp.port eq 49706)						
No.	Time	Source	Destination	Protocol	Length	Info
49	10.392789	127.0.0.1	127.0.0.1	TCP	55	49706 → 49705 [PSH, ACK] Seq=3 Ack=1 Win=2053 Len=1
50	10.392835	127.0.0.1	127.0.0.1	TCP	54	49705 → 49706 [ACK] Seq=1 Ack=4 Win=2048 Len=0
51	10.393103	127.0.0.1	127.0.0.1	TCP	55	49706 → 49705 [PSH, ACK] Seq=4 Ack=1 Win=2053 Len=1
52	10.393130	127.0.0.1	127.0.0.1	TCP	54	49705 → 49706 [ACK] Seq=1 Ack=5 Win=2048 Len=0
54	10.769539	127.0.0.1	127.0.0.1	TCP	55	49706 → 49705 [PSH, ACK] Seq=5 Ack=1 Win=2053 Len=1
55	10.769577	127.0.0.1	127.0.0.1	TCP	54	49705 → 49706 [ACK] Seq=1 Ack=6 Win=2048 Len=0
56	10.772021	127.0.0.1	127.0.0.1	TCP	55	49706 → 49705 [PSH, ACK] Seq=6 Ack=1 Win=2053 Len=1
57	10.772047	127.0.0.1	127.0.0.1	TCP	54	49705 → 49706 [ACK] Seq=1 Ack=7 Win=2048 Len=0
59	10.973510	127.0.0.1	127.0.0.1	TCP	55	49706 → 49705 [PSH, ACK] Seq=7 Ack=1 Win=2053 Len=1
60	10.973546	127.0.0.1	127.0.0.1	TCP	54	49705 → 49706 [ACK] Seq=1 Ack=8 Win=2048 Len=0
64	11.130327	127.0.0.1	127.0.0.1	TCP	55	49706 → 49705 [PSH, ACK] Seq=8 Ack=1 Win=2053 Len=1
65	11.130368	127.0.0.1	127.0.0.1	TCP	54	49705 → 49706 [ACK] Seq=1 Ack=9 Win=2048 Len=0
66	11.130763	127.0.0.1	127.0.0.1	TCP	55	49706 → 49705 [PSH, ACK] Seq=9 Ack=1 Win=2053 Len=1
67	11.130792	127.0.0.1	127.0.0.1	TCP	54	49705 → 49706 [ACK] Seq=1 Ack=10 Win=2048 Len=0
89	16.079670	127.0.0.1	127.0.0.1	TCP	55	49706 → 49705 [PSH, ACK] Seq=10 Ack=1 Win=2053 Len=1
90	16.079710	127.0.0.1	127.0.0.1	TCP	54	49705 → 49706 [ACK] Seq=1 Ack=11 Win=2048 Len=0
> Frame 90: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0 > Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00) > Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1 > Transmission Control Protocol, Src Port: 49705, Dst Port: 49706, Seq: 1, Ack: 11, Len: 0						
0000	00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00E.				
0010	00 28 31 8c 40 00 80 06 00 00 7f 00 00 01 7f 00	.(1.@.....				
0020	00 01 c2 29 c2 2a c4 0a 64 59 ba 6d a9 57 50 10	...)*...dY·m·WP·				
0030	08 00 99 54 00 00	...T..				

Figure 3.9: MQTT connection over TCP/IP protocol

3.3.3 : Temperature & Humidity Sensor

These sensors are digital in nature that take input to read values of temperature and humidity. Relative humidity accounts for both temperature and pressure, as the electrical permittivity of the dielectric material changes with humidity. There are many variants available, such as DHT11, DHT22, DHT21 and RHT04 and table 3.2 is a study of two. The DHT11 is a low-cost digital device made up of a capacitive humidity sensor and a thermistor to measure the surrounding and produce a digital signal on the data pin of the microcontroller. It is capable of generating data every 2 seconds as shown in figure 3.8 and 3.9

Capacitive RH (Relative Humidity) sensors are made of dielectrics whose dielectric constant varies when subjected to humidity i.e. amount of water present in the air. A DHT11 has a hygroscopic polymer film as its dielectric with layers of electrodes on its either sides. The advantages of using a capacitive RH sensor are:

- Linear output voltage
- Stable results
- Detects wide range of RH
- Used in HVAC, weather stations, automobiles
- Good accuracy
- Low cost and replaceable

Table 3.2: A comparison of DHT sensors

Specification	DHT11	DHT22	DHT21	RHT04
Temperature Range	[0] – [50] degree Celsius	[-40] ~[125] degree Celsius	[-40] ~ [80] degree Celsius	[-40] ~ [100] Degree Celsius
RH Range	20 - 80% (accuracy 5%)	0 - 100 (accuracy 2%)	0 - 100 (accuracy 3%)	0 - 100 (accuracy 2%)
Sampling rate	1 Hz	0.5 Hz	0.5 Hz	0.5 Hz

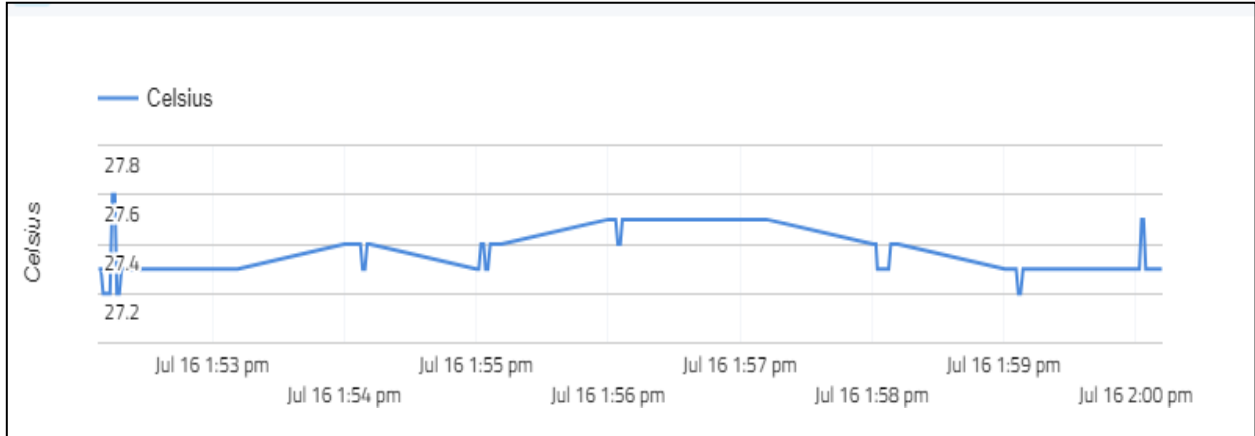


Figure 3.10: **Temperature reading from a DHT11 measured every 2 seconds**

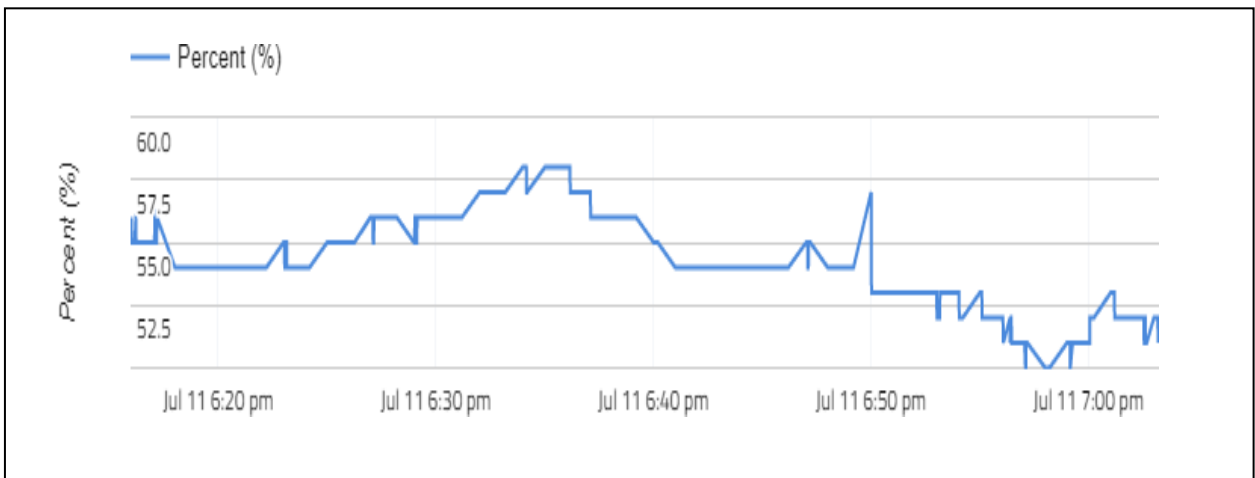


Figure 3.11: **Humidity readings from a DHT11 measured every 10 minutes**

3.3.4 : Luminosity Sensor:

LDR (Light Dependent Resistor), also known as Photoresistor, is a component that has a variable resistance which changes with the incoming light intensity and is therefore used in light sensing circuits. It has a serpentine track which is a cadmium sulphide (CdS) film passing through the sides. The structure is enveloped in a resin case to allow unrestricted access to light. They require small power and voltage for operation but are highly inaccurate with a response time of about tens or hundreds of milliseconds. This optoelectronic device is preferably used in light

varying sensor circuit, and light and dark activated switching circuits. Some of its applications include street lighting, fire alarm, night security light, light activated switch circuit etc.

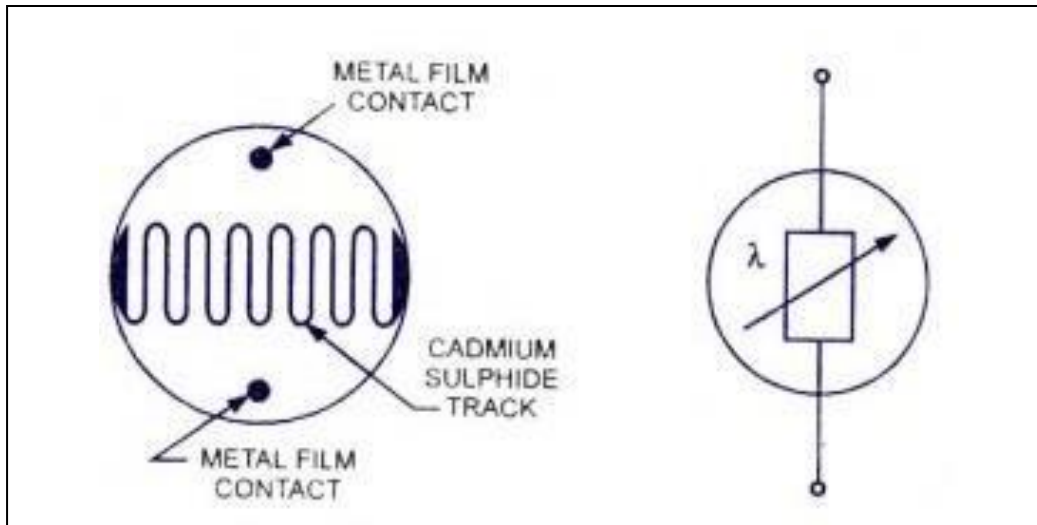


Figure 3.12: **Basic structure and symbol for LDR**

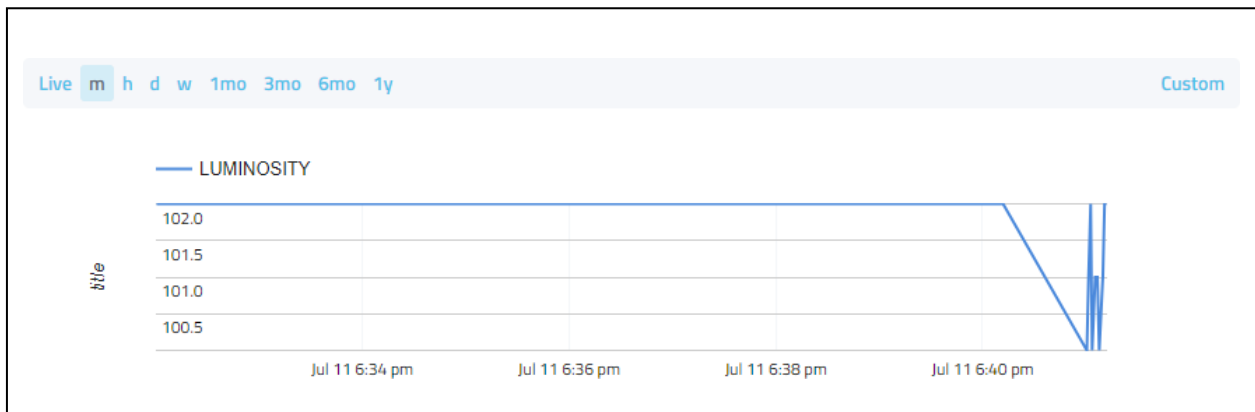


Figure 3.13: **Luminosity readings from an LDR measured every 10 minutes**

LDR devices are light dependent whose resistance vary accordingly. It decreases when light falls on them and increases in the dark. The separate terms for them are dark and light resistance, the former being very high in value, around $10^{12} \Omega$. The resistance starts decreasing when the device starts absorbing light. Photocells are non-linear devices, the sensitivity of which are material based as it has different spectral response to certain wavelengths of light. It takes about 8-12 ms for the change in resistance to take place when light is incident on it, while more seconds for resistance to recover back up to its initial value after the removal of light.

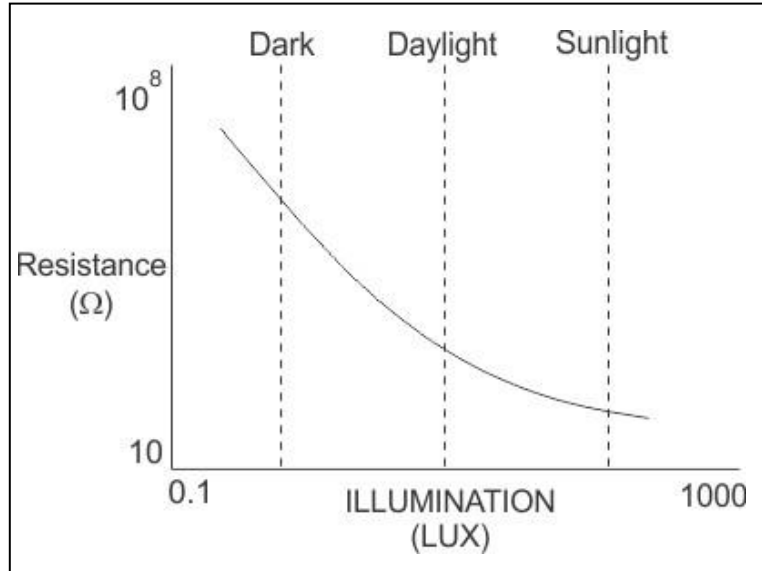


Figure 3.14: **Resistance vs. Illumination of an LDR**

3.3.5: Passive Infra-Red Sensor

The PIR sensor detects incoming infrared radiation from the human body and converts it into an electrical charge. This charge is proportional to the detected level, and the signal is further conditioned by an in-built FET. The output pins connect to an external circuitry for further amplification. The linear range is up to 10 meters; however it depends on the sensitivity of detection. There are two potentiometers that are designed to vary the sensitivity and alter time settings. The time setting knob decides the interval for which the output pin should stay high to an incoming radiation. Sensitivity decides the linear and angular range of the detection range spread.

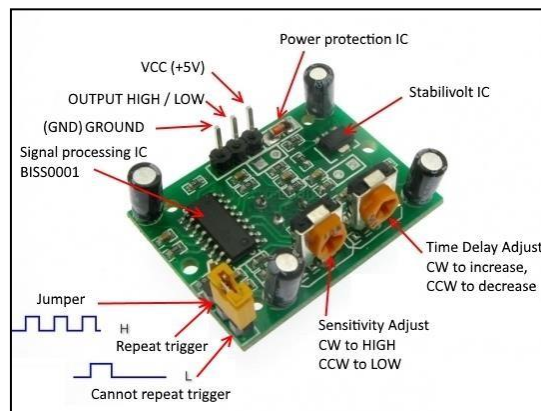


Figure 3.15: **The PIR sensor**

CHAPTER 4

MODELLING OF A CORE PLATFORM

4.1 : INTRODUCTION

An open source platform is a one step solution to help buildings save energy and operate more efficiently. The aim of the platform is to first target the building sizes along with their uses ie. residential, industrial or commercial. In India, Small buildings take up to 5000 sq. ft and Medium Buildings take up to 5000 – 50,000 sq. ft of space. These buildings constitute a majority in this country and aren't mostly equipped with building automation systems. The features for an efficient platform are:

- *Interoperability*: Capable of communicating with multiple IoT devices, not necessarily from the same manufacturer, using different communication technologies and data exchange protocols
- *Open Architecture*: Seamless interfacing of hardware
- *Open Source*: Designed to let software developers contribute to the platform by including additional functionalities, applications and add-ons
- *Plug & Play*: By making use of a dedicated Device discovery agent, the platform should be able to automatically discover nearby compatible devices and integrate it smoothly
- *Remote control & monitor*: Making use of a built-in web server for remote access to the front end UI in real time without being on the home/local network. Also enable remote controlling from anywhere with optimized latency.
- *Advanced control*: Use of intelligent algorithms by machine learning and implementing DR for reducing energy consumption. This can contribute to our carbon footprint and offer comfort and control of our daily electricity usage.

The features listed demand a considerable computation prowess increasing the hardware investment. The merits can only be enabled by keeping an excellent performance at a minimum cost along with being adaptive to host machines.

4.2 : EMBEDDED SYSTEM HOSTING

An embedded system platform comprises of all the necessary building blocks to get a microcontroller up and running in a short span of time. The idea behind this is that libraries, frameworks, drivers, schedulers and source code with patches are already inbuilt so that focus can be on the task in hand. The advantages to embedded platform development are:

- Firmware robustness
- Banking on existing software
- Potential to bring down the overall development task
- Faster time to launch/use

Although Cloud Computing has become immensely popular, it still isn't economic enough. For eg. Web services offered by AWS (Amazon Web Services) EC2, equivalent to 2GB memory and a virtual CPU costs more than 13,000 INR annually. Thus, open-source platforms are specified to run on embedded systems enabling a cost-effective BEMS solution. Many SBC's (Single Board Computers) are available for hosting depending on their performance and run time.

4.2.1 : SINGLE BOARD COMPUTERS

A single board computer is differentiated from a conventional development kit/evaluation board because in single package it is a complete computing platform; having all the hardware and software needed to operate properly. A basic comparative study is listed in Table 4 as shown

Table 4.1: Comparison in hardware of selected Single Board Computers

	RASPBERRY Pi 3B	RASPBERRY Pi 4	ODROID C2	ODROID XU4	Beagle Board X15
Processor	Broadcom BCM 2837 Based on ARM Cortex-A53	1.5 GHz Quad-core 64-bit RAM Cortex-A72 CPU (3x performance)	Amlogic S905 (4x Cortex-53 @ up to 1.5GHz)	Samsung Exynos5422 (4x Cortex-A15 at 2.0GHz & 4x Cortex-A7 at 1.4GHz)	TI AM5728 2×1.5-GHz ARM Cortex-A15

	RASPBERRY Pi 3B	RASPBERRY Pi 4	ODROID C2	ODROID XU4	Beagle Board X15
GPIO	40	40	40	40	157
On-Board Storage	No	No	Ext Flash	Ext Flash	4 GB 8-bit eMMC flash storage
RAM	1GB DDR2	4GB DDR4	2GB DDR3	2GB LPDDR3	2GB DDR3
Power	2.5A, 5V	3A, 5V	2A, 5V	4A, 5V	210-460 mA, 5V
Pricing	2200 INR	2500 INR	9500 INR	11,700 INR	18,500 INR

4.2.2 : RASPBERRY PI- MODEL 3B

This is a system on chip being widely used in hosting many platforms mostly aimed at home automation and security control. It is the safest and most economic bet to implement projects consisting of dedicated functions and a scope of further expansion. In this platform system, we are making use of the Pi to host our core Home Assistant. The features match our demand from the IoT environment and do not require a considerable load on the processor. Hence, available RAM (Random Access Memory) is enough to accommodate all the batch jobs it has to perform in keeping the system running in the background and script jobs that it has to perform in the HA virtual environment. The setup of the system, additionally requires:

- A display module, working as an output to access the Pi GUI (Graphical User Interface)
- A keyboard and mouse, as inputs
- Power supply to the board, micro USB supported, 2.5A
- HDMI connection to the display module
- Ethernet/Wi-Fi support
- SD card and Casing, for protection

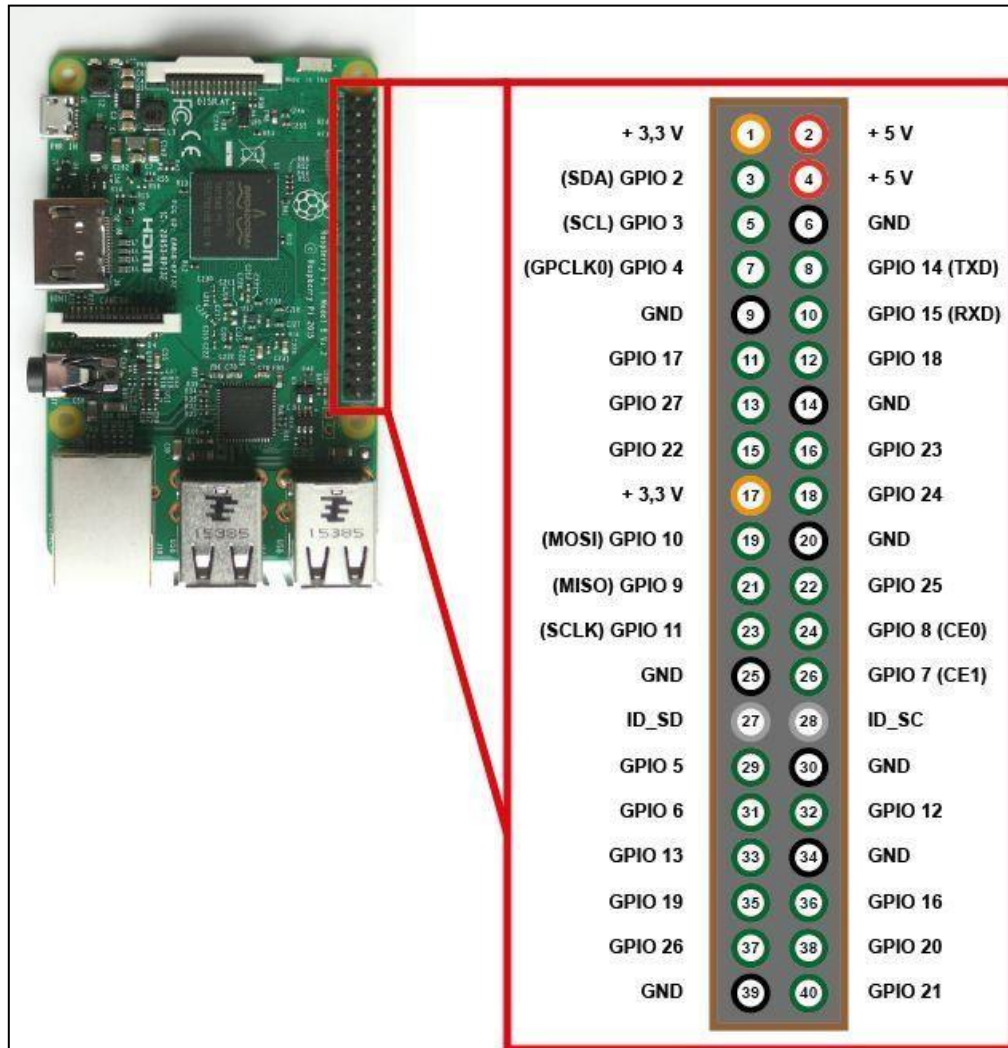


Figure 4.1: Pin diagram of Raspberry Pi (Host Embedded System)

The host system software is called Raspbian which has been specifically dedicated to the Pi. It is a free operating system based on Debian optimized for the Raspberry Pi hardware, all versions included. Raspbian images are available in both 32 and 64 bits and can be flashed in a micro SD card using Balena Etcher, a cross-platform tool for flashing OS images to the SD cards and USB drives. It supports both .iso and .img files as well as zipped folders onto storage media to create live drives. Since we are hosting HA, we will install HASSBIAN, a version to suit our requirements of dedicated functions, preconfigured for Home Assistant.

4.3: PERFORMANCE ANALYSIS

To test the working of our embedded board, a platform is hosted on it with smart integrations working with both synchronous and asynchronous protocols. The testing is done to review the processing power, energy consumption, performance as well as operating temperature gradient to scale. The software used for this purpose is called Remote-IoT which uses a secure AWS (Amazon Web Service) IoT cloud platform to connect to networked devices remotely, from anywhere. The added feature is encryption by the SSH (Secure Shell Access) tunnel which is a web hooking tool to access network ports securely. This same feature is also used by the banking systems which provide 64 ~ 128 bits encryption to all passwords and vulnerable data. This is also works on the TCP/IP stack on which the MQTT is based as well. Key features of this tool are:

- Full device management
- Secure Remote access
- Software-dedicated solutions
- Web-management console
- No dedicated VPN (Virtual Private Network) or Firewall required
- Uses web tunnel
- Runs Cron-jobs

Cron is a Linux feature which arranges a command or script on the server to run automatically at a specified date and time. A scheduled job that it performs to do so is called a Cron job. Scripts executed as Cron jobs are mostly used to alter files or databases. The software tool Remote-IoT is programmed to run a job on our host system to monitor it's health and performance at all times. The limitation to this, however, is the refresh rate. It allows a minimum gap of 15 minutes before the next update. It has three main components:

- Script
- Command
- Action/output

```
curl -s -L https://remote-iot.com/install/remote-iot-install.sh | sudo -s bash

sudo /etc/remote-iot/services/setup.sh 'your_login_email' 'your_password' 'device_name' 'device_note'

echo "0 2 * * * curl -s -L https://remote-iot.com/install/upgrade.sh | sudo -s bash" | crontab -|
```

Figure 4.2: **Inserting a Cron job in python script**

Status	Device Name	Device Id	Note	Group	Connect URL	Internal IP	External IP
✔	raspberrypi	5D153B6CC61E55F4	Hosting_Home_Assistant			172.16.40.116	14.139.251.100

Monitoring Execute Script							
							Execute New Script
Job Name	Script	Status	Submit Time	Schedule Time	Execute Time	▲	Execute User
update		Executed	2019-07-09 12:02:49	2019-07-09 12:03:27	2019-07-09 12:03:54		JAGRITI SURABHI
time_date		Executed	2019-07-09 12:01:16	2019-07-09 12:01:51	2019-07-09 12:02:04		JAGRITI SURABHI
TIME		Executed	2019-07-09 11:57:13	2019-07-09 11:58:18	2019-07-09 11:59:05		JAGRITI SURABHI

Figure 4.3: **Dashboard for inspecting health of host system**

The results of the inspections are as shown in the following figures:

1.

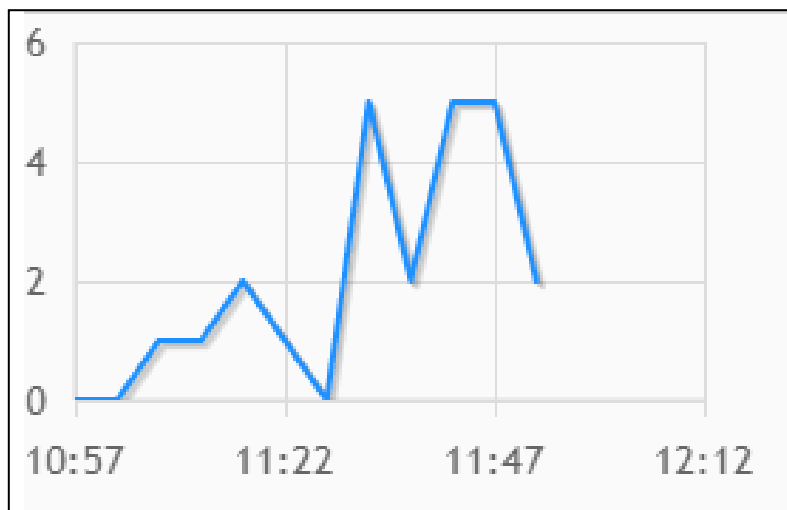


Figure 4.4: **Utilization of the CPU on the Pi**

2.

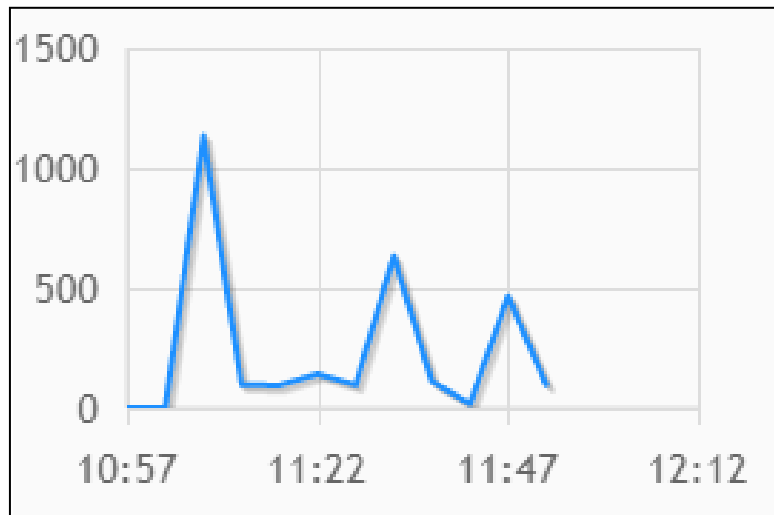


Figure 4.5: Network out in kB connected to local network

3.

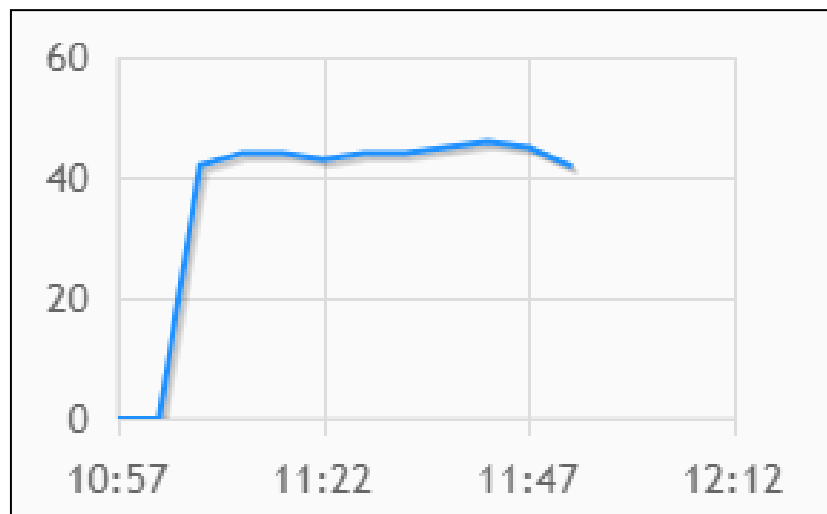


Figure 4.6: CPU Temperature of the Pi

4.

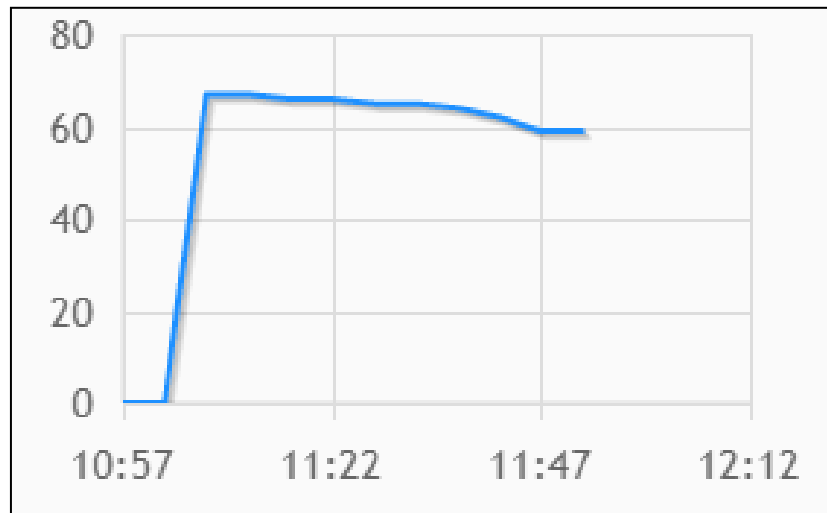


Figure 4.7: **Memory utilization of the Pi**

5.

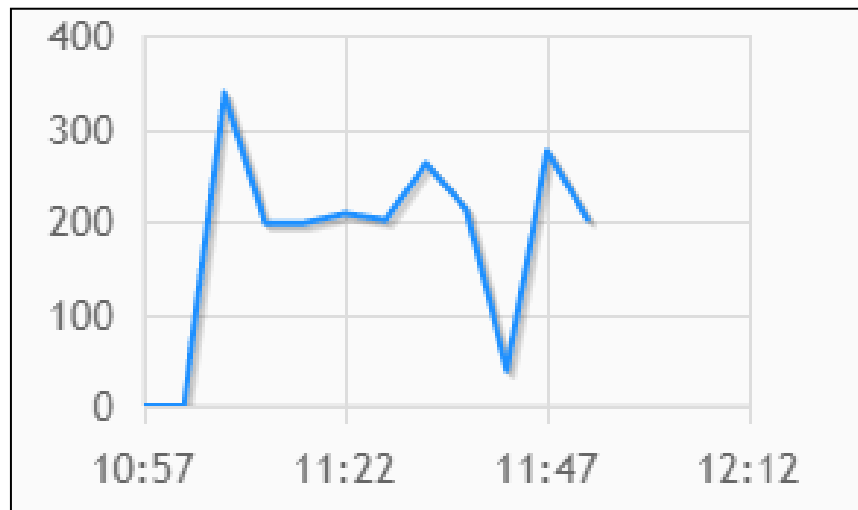


Figure 4.8: **All networks handled by the Pi**

CHAPTER 5

HOME ASSISTANT

AN OPEN SOURCE DEVELOPMENT PLATFORM

5.1: INTRODUCTION

Among the many development platforms available, our project has been designed on the development of Home Assistant (HA) as a middleware for device management, control and automation. A lot of IoT products send their data to the servers and are managed by cloud computing. This does create a stream of well management but at a risk of data theft, privacy and indeed, security. With Home Assistant, data can be stored and managed locally, with security settings to allow only those users with access. Alongside, it supports a variety of features like communication protocols, automation and trigger platforms, voice assistance and useful plug-ins. The figure 5.1 shows HA active once the OS has been flashed on the Pi.

```
login as: pi
pi@192.168.1.219's password:
[ ASCII art logo ]

system info:
Distro.....: Raspbian GNU/Linux buster/sid
Kernel.....: Linux 4.14.98-v7+

Uptime.....: up 13 minutes
Load.....: 0.00 (1m), 0.04 (5m), 0.07 (15m)
Processes...: 88 (root), 8 (user) | 96 (total)

CPU.....: ARMv7 Processor rev 4 (v7l)
Memory.....: 85Mi used, 628Mi free, 875Mi in total

services:
home-assistant@homeassistant: active
Last login: Mon Jul 15 20:00:30 2019 from 192.168.1.205

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

pi@hassbian:~ $ █
```

Figure 5.1: The CLI with Home Assistant running live

The commendable features for hosting this platform are:

- **Community built being open-source in nature:**

An off the market solution is ideal and easy for automation, but it creates dependencies. The users of smart homes have to be dependent on the dedicated applications for their smart devices. For example, If the user has an Apple homePod setup, it can only be controlled by Apple and its updates or software patches to be compatible with other devices. While they do offer support for a variety of devices, its rarely in favor of niche devices like microcontrollers. Open source HA(Home Assistant) offers tons of support for devices and the libraries are open to be updated regularly with new builds.

- **Control over local data:**

Today, every device linked to the Internet has cloud connectivity and stores their information on the servers. This leads to records of present states of devices, history, access timings, frequency of use, power consumption etc. While servers make tasks easier to perform by independent control, one no longer has full control on their device. Apart from that, a huge of security underlies with internet connectivity. Home Assistant lets one configure its own setup and is a beneficial tool for learning.

- **Work in progress:**

This project is a work in progress, as it still is running on the beta versions. However, the libraries of python such as pi-wheel, pip, tcpdump etc. are regularly updated being open source. A lot of contributors add their libraries to help build this platform every day. Hence, it is a learning curve to understand automation systems better.

- **Customization:**

While there are many open source platforms to develop such as OpenHAB, Pytotion, OpenRemote, Calaos, OpenMotics, and Domoticz, this project resides around the HA because of its familiar language support i.e. Python as well as easy to configure YAML script. It is a human-readable-data-serialization language, generally used for configuration only but can be extended to be used in applications where data is being stored or transmitted. It trumps over other data formats like XML and JSON, because of the ease of activity.

5.2: The Lovelace UI

It is the name of the HA user interface, which is fast, customizable and a powerful way to manage IoT environments, working both on mobile and desktop. Figure 5.2 show our web and mobile interface.

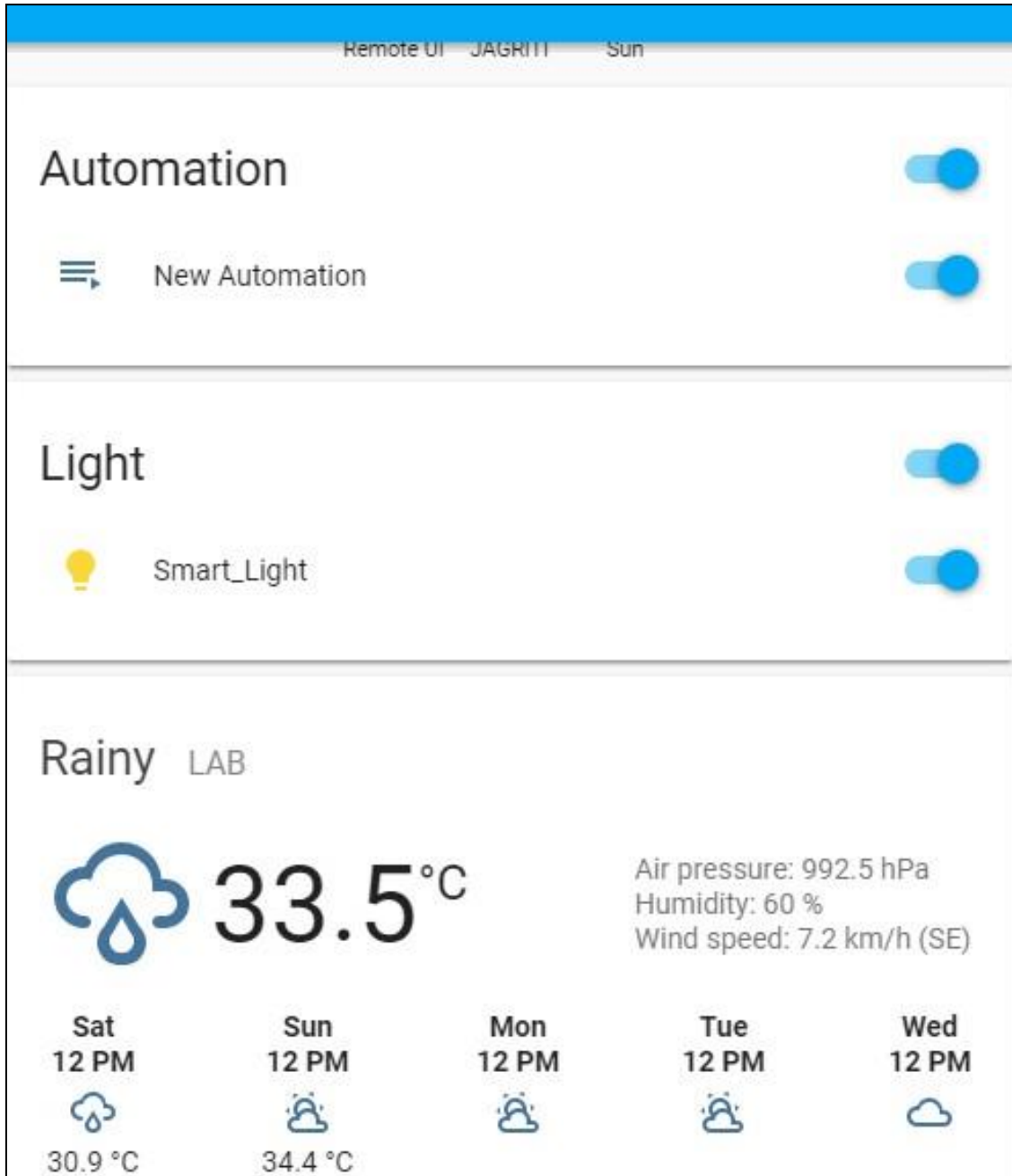


Figure 5.2: Desktop and Web UI of successful HA setup

5.2.1: Smart Light setup:

Light bulbs have become exceptionally smarter in the previous years. There are a variety of connected solutions available to replace the conventional bulbs, which can be directly controlled just by a few taps from cellphones and desktops. Smart bulbs make use of LEDs and various other technologies combined, hence they are expensive. But they also use less energy and last much longer, adding to the savings.

Smart bulbs offer a degree of control over other bulbs such as scheduled timers and remote control. They also make use of geofencing, which means working with the GPS of your smart device to detect position and automate the bulbs when in the vicinity. The various smart bulbs available in the Indian markets are Philips Hue, TP-Link smart bulb series, Wipro Garnet, Syska LED, D-Link smart bulbs etc. For economic purpose and the ease of use, we have chosen TP-Link LB110 smart light as one of our nodes with its successful integration as shown in figure 5.3. The key features are:

- No hub required for connection and protocol conversion
- Brightness control
- Scheduling possible
- Voice-control equipped
- Energy saving
- Bayonet holder provided
- Compatible with 2.4 GHz Wi-Fi band

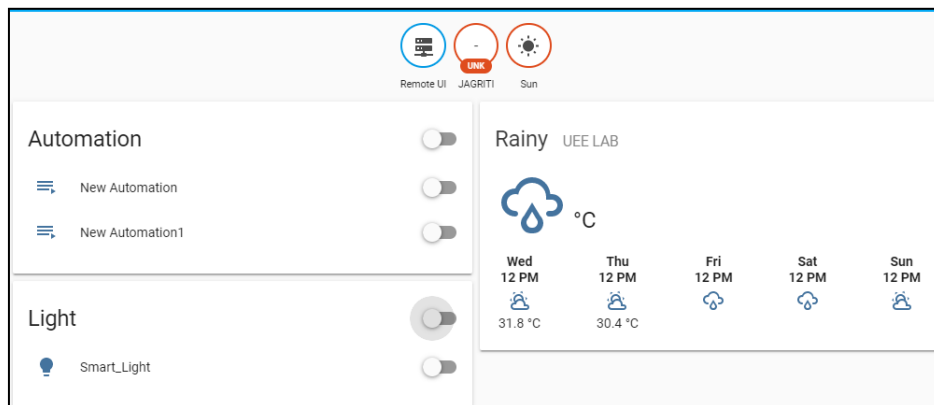


Figure 5.3: TP-Link Smart Light integration

5.3 : THE ESP (Extra Sensory Perception) TOOL

The ESP8266 and ESP32 are one of the cheapest and yet efficient micro-controlling devices out there. They are being used for countless home automation projects and have also made their way in the Smart home concept. Even though they are niche devices, many purposes of automation such as light and fan control, measurement of set parameters through sensors, actuating a response through the output GPIO pins or output devices like switches and servomotors. Lead manufacturers like Sonoff make use of these chips because of their availability and dirt cheap pricing.

Setting up microcontrollers has become an easy task over the years, with the Arduino library readily available for programming in C language for ESP boards and the indigenous compiling tool ESPlorer for programming in the LUA language. However, automation with customization over communication protocols requires more than just flashing firmware onto the boards. This is where the ESP tool comes in handy with its front-end ESPHome. It requires simple yet powerful configuration files to fix the API between the HA hub and the microcontrollers to start the handshake of information. To top that up, MQTT protocol can make communications easy by offering its limited bandwidth for exchange of binary information between the daughters and the mother hub. Since this is also an open source project, the tool only runs on Python2 for now. Future updates may include the latest language updates.


The source code for installation of ESPHome is as shown in figure 5.4

```
1 pip2 install esphomeyaml
2 esphomeyaml config/ dashboard
3 # Alternative for docker users:
4 docker run --rm -p 6052:6052 -p 6123:6123 -v "$PWD":/config ottowinter/esphomeyaml /config dashboard
```

Figure 5.4: ESPHome setup script on CLI

The dashboard interface is on the same server that is hosting Home Assistant but at a different port number. This hosts itself on port 6052, the set port for all ESP boards micro-controlling through the hub front-end. After connection is established, sensor readings and switches need to be added as features to the source code- YAML configuration file.

```

[18:43:16][I][application:097]: You're running esphomelib v1.10.1 compiled on Jul 17 2019, 18:37:34
[18:43:16][C][wifi:341]: WiFi:
[18:43:16][C][wifi:240]:   SSID: 'Doorway'
[18:43:16][C][wifi:241]:   IP Address: 192.168.1.208
[18:43:16][C][wifi:243]:   BSSID: 00:1E:A6:D4:87:68
[18:43:16][C][wifi:245]:   Hostname: 'test_blink'
[18:43:16][C][wifi:250]:   Signal strength: -66 dB 
[18:43:16][C][wifi:251]:   Channel: 1
[18:43:16][C][wifi:252]:   Subnet: 255.255.255.0
[18:43:16][C][wifi:253]:   Gateway: 192.168.1.1
[18:43:16][C][wifi:254]:   DNS1: 0.0.0.0
[18:43:16][C][wifi:255]:   DNS2: 0.0.0.0
[18:43:16][C][switch.gpio:049]: GPIO Switch 'test'
[18:43:16][C][switch.gpio:050]:   Pin: GPIO13 (Mode: OUTPUT)
[18:43:16][C][switch.gpio:066]:   Restore Mode: Restore (Default to OFF)
[18:43:16][C][logger:099]: Logger:
[18:43:16][C][logger:100]:   Level: DEBUG
[18:43:16][C][logger:101]:   Log Baud Rate: 115200
[18:43:16][C][api:072]: API Server:
[18:43:16][C][api:073]:   Address: 192.168.1.208:6053
[18:43:16][C][ota:129]: Over-The-Air Updates:
[18:43:16][C][ota:130]:   Address: 192.168.1.208:8266
[18:43:16][C][ota:132]:   Using Password.
[18:43:16][I][application:114]: Running through first loop()
[18:43:16][I][application:141]: First loop finished successfully!
[18:44:01][D][api:531]: Client 'Home Assistant 0.94.4 (192.168.1.219)' connected successfully!

```

Figure 5.5: Successful instant of HA API integration with ESPHome

The API integration shown in figure 5.5 is the successful integration of an ESP8266 microcontroller to the web front of the HA. The process takes roughly about 10 minutes which is a latency this system affords. These issues arise mainly out of weak connectivity signal over the Wi-Fi. Ethernet enabled hub has yielded faster results and hence low latency. It is a three step process listed as:

- Type of API:

There are three types of API in use:

1. SOAP (Simple Object Access Protocol)
2. REST (Representational State Transfer)
3. RPC (Remote Procedural Call)

Home Assistant offers the “REST”ful API for device integration on the same IP address as that of the web frontend.

- Data Payload: This is the format of data sent from the API process flow. HA offers and accepts data only in the JSON format.

- Documentation: A robust documentation is needed for popular use of any platform.

5.3.1: DEVICE INTEGRATION INSTANT: “test_blink”

1. Device software model:

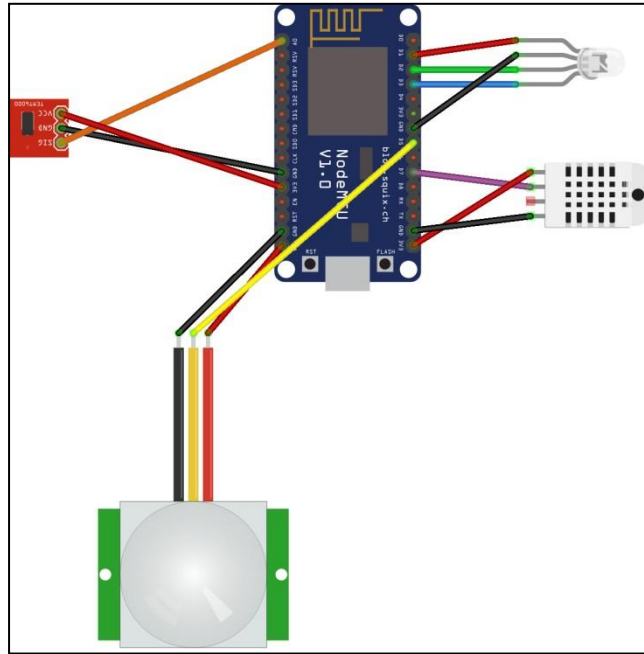


Figure 5.6: **Device Model on Fritzing**

2. ESPHome configuration:

The source code for ESP can be flashed on the board through the ArduinoIDE, any of choice. For our test here, the code is that of an LED control with a web server to do so. The script is both in C for back-end and HTML for front-end. The dashboard and simple yaml script used are shown in figure 5.7 and 5.8 respectively.

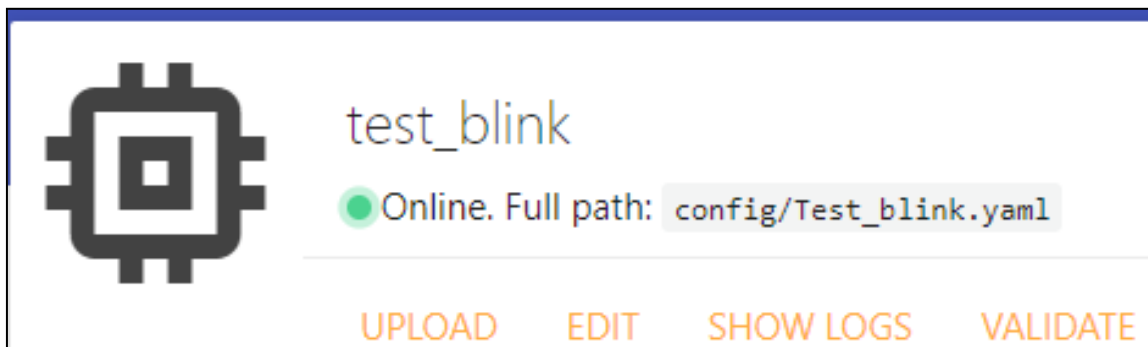


Figure 5.7: **ESPHome Dashboard**

```

esphomeyaml:
  name: test_blink
  platform: ESP8266
  board: nodemcu2
wifi:
  ssid: '[redacted]'
  password: '[redacted]'
  manual_ip:
    static_ip: 192.168.1.[redacted]
    gateway: 192.168.1.1
    subnet: 255.255.255.0
switch:
  - platform: gpio
    pin: GPIO4
    name: switch1
  - platform: gpio
    pin: GPIO5
    name: switch2
sensor:
  - platform: adc
    pin: A0
    name: Brightness
    unit_of_measurement: lux
    filters:
      - lambda:
          return (x / 10000.0) * 2000000.0;
  - platform: dht
    pin: GPIO13
    temperature:
      name: Lab Temp
    humidity:
      name: Lab Hum
    update_interval: 20s
binary_sensor:
  - platform: gpio
    pin: D5
    name: Motion
    device_class: motion
api:
  password: 'automato'
ota: password: '[redacted]'

```

Figure 5.8: **Configuration in ESPHome**

3. Front-End device link:

Once the code is compiled and validated, the configuration is to be reflected for users to access easily. Since customization is possible, the gpio pins of the board can be accessed directly from the Lovelace UI as shown in figure 5.9. The pins can be programmed to be used as sensors, actuators (output) or input/toggle pins. The states have to be defined.



Figure 5.9: Successful device integration



Figure 5.10: Web UI with parameters from test_blink on the dashboard

CHAPTER 6

RESULT AND DISCUSSIONS

6.1 : INTRODUCTION

The implementation of IoT devices with a platform support in an environment is discussed in this chapter. We have first implemented a test circuit to work with sensors and automate switches and devices with the help of a online platform called Cayenne. Then we have automated by using triggers designed in the platform, such as time or sensor based scheduling. After that, we have established connection using the MQTT protocol and analyzed it using a hardware tool called WireShark. It provides information about data packets, loss in transmission, bit length with overhead, QoS, protocol as well as the protocol commands being used. All these information are handy for working around the study of various protocols.

6.2 : Results of the Test circuit 1

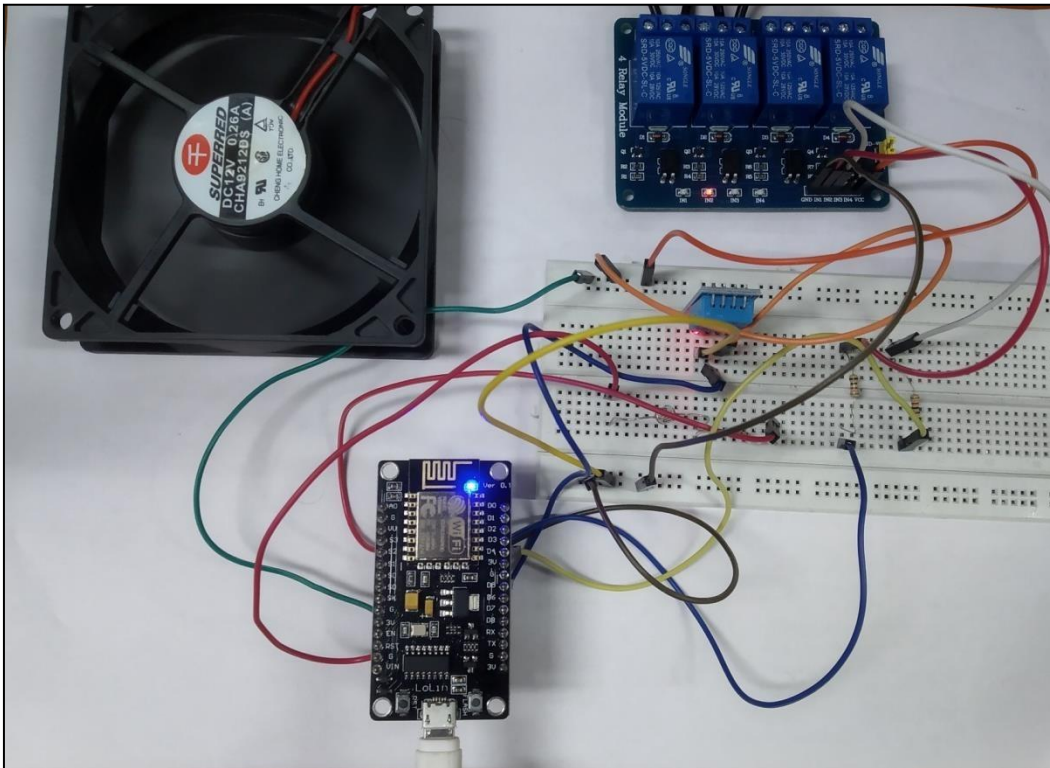


Figure 6.1: Test circuit 1

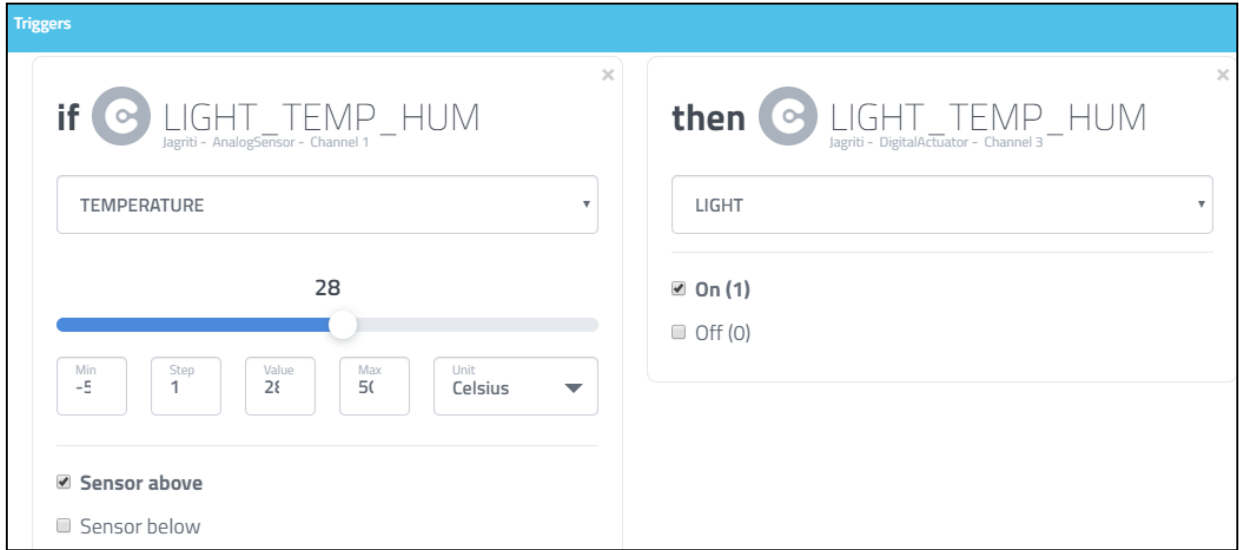


Figure 6.2:

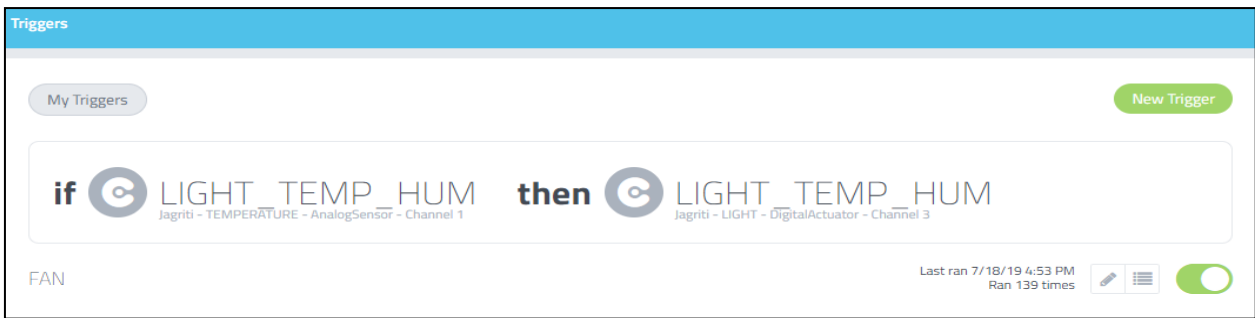


Figure 6.3: Trigger set and active condition

6.3: Results of the circuit 2- “test_blink”

1. The test circuit is designed to sense motion, irradiance, temperature and humidity and control devices such as lights and fans.

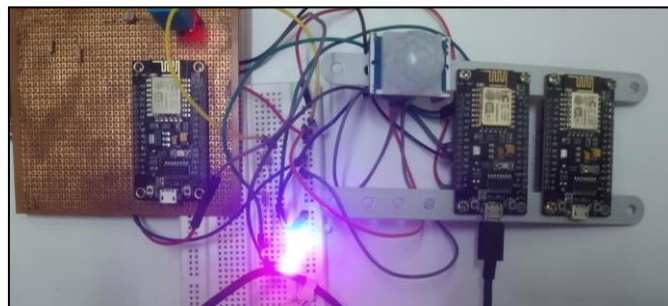


Figure 6.4: Test circuit 2 “test_blink”

```

07:41:08.061 -> WiFi connected
07:41:08.061 -> IP address:
07:41:08.061 -> 192.168.1.208
07:41:08.061 -> Ready
07:41:08.061 -> IPess: 192.168.1.208
07:41:08.061 -> Attempting MQTT connection...connected
07:41:08.061 -> Setting LEDs:
07:41:08.061 -> r: 0, g: 0, b: 0
07:41:08.061 -> {"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"0.00","motion":"","ldr":"0","temperature":"0.00","heatIndex":-10.30}
07:41:08.096 -> {"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"0.00","motion":"standby","ldr":"0","temperature":"0.00","heatIndex":-10.30}
{"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"0.00","motion":"standby","ldr":"1024","temperature":"0.00","heatIndex":-10.30}
{"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"0.00","motion":"motion detected","ldr":"1024","temperature":"0.00","heatIndex":-10.30}
{"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"0.00","motion":"standby","ldr":"1024","temperature":"0.00","heatIndex":-10.30}
{"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"0.00","motion":"motion detected","ldr":"1024","temperature":"0.00","heatIndex":-10.30}
{"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"0.00","motion":"standby","ldr":"1024","temperature":"0.00","heatIndex":-10.30}
{"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"0.00","motion":"motion detected","ldr":"1024","temperature":"0.00","heatIndex":-10.30}
{"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"0.00","motion":"motion detected","ldr":"1024","temperature":"84.02","heatIndex":"79.42"}
07:42:06.845 -> {"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"72.00","motion":"motion detected","ldr":"1024","temperature":"84.02","heatIndex":"91.18"}
{"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"72.00","motion":"standby","ldr":"1024","temperature":"84.02","heatIndex":"91.18"}
{"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"72.00","motion":"standby","ldr":"1024","temperature":"83.30","heatIndex":"89.58"}
{"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"72.00","motion":"standby","ldr":"1024","temperature":"83.84","heatIndex":"90.77"}
{"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"72.00","motion":"standby","ldr":"1024","temperature":"84.20","heatIndex":"91.59"}
{"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"72.00","motion":"standby","ldr":"1024","temperature":"84.56","heatIndex":"92.42"}
{"state":"OFF","color":{"r":255,"g":255,"b":255},"brightness":255,"humidity":"72.00","motion":"standby","ldr":"1024","temperature":"84.20","heatIndex":"91.59"}

```

Figure 6.5: Serial Monitor output

2.

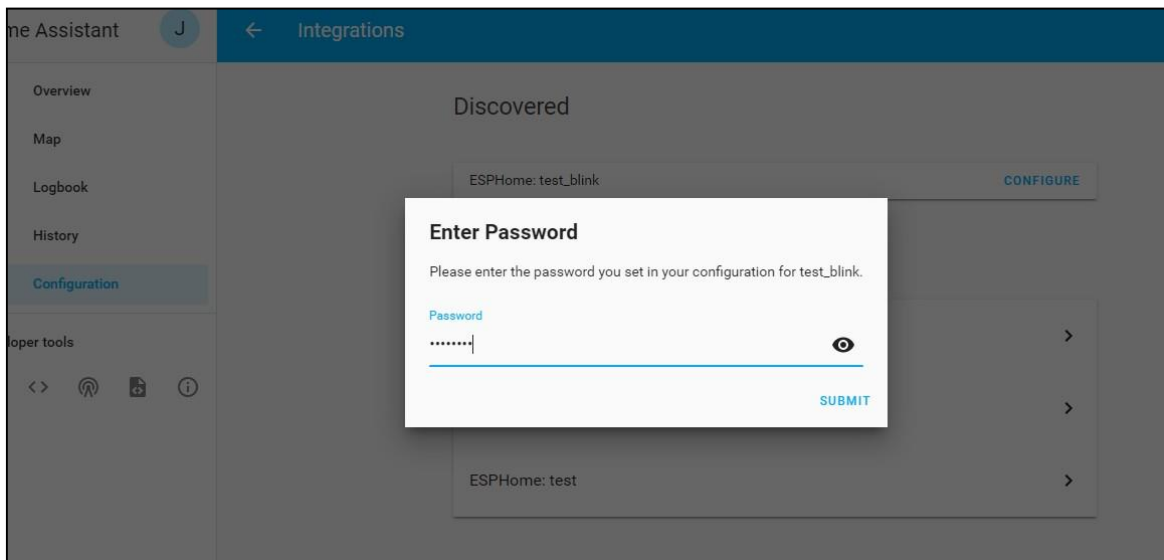


Figure 6.6: Password protection of HA for security

```

1  ||| 0.7%] Tasks: 36, 66 thr; 1 running
2  [ 0.0%] Load average: 0.09 0.04 0.01
3  [ 0.0%] Uptime: 02:22:40
4  ||| 2.0%]
Mem[|||||] 148M/876M
Swp[ ] 0K/100.0M

```

PID	USER	PRI	NI	VRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
3372	pi	20	0	8304	3152	2436	R	2.0	0.4	0:00.48	htop
538	homeassis	20	0	360M	66936	11668	S	0.7	7.5	2:02.33	/srv/homeassistant/bin/python /srv/homeassistant/bin/hass
420	root	20	0	330M	44364	11896	S	0.7	4.9	1:42.30	java -jar /etc/remote-iot/services/remotetiot.jar
442	root	20	0	307M	20916	10456	S	0.7	2.3	0:04.37	java -jar /etc/remote-iot/services/remotetiot.jar
548	ntp	20	0	7852	2752	2396	S	0.0	0.3	0:01.24	/usr/sbin/ntpd -p /var/run/ntpd.pid -g -u 109:114
3303	pi	20	0	12100	3364	2568	S	0.0	0.4	0:00.09	sshd: pi@pts/0
426	root	20	0	307M	20916	10456	S	0.0	2.3	0:06.10	java -jar /etc/remote-iot/services/remotetiot.jar
441	root	20	0	330M	44364	11896	S	0.0	4.9	0:04.36	java -jar /etc/remote-iot/services/remotetiot.jar
582	homeassis	20	0	360M	66936	11668	S	0.0	7.5	0:07.29	/srv/homeassistant/bin/python /srv/homeassistant/bin/hass
310	root	20	0	27572	72	0	S	0.0	0.0	0:00.61	/usr/sbin/rngd -r /dev/hwrng
1	root	20	0	33600	7928	6384	S	0.0	0.9	0:03.86	/sbin/init
101	root	20	0	18532	7228	6392	S	0.0	0.8	0:02.14	/lib/systemd/systemd-journald
128	root	20	0	17720	3684	2884	S	0.0	0.4	0:00.74	/lib/systemd/systemd-udev
357	root	20	0	25428	2728	2428	S	0.0	0.3	0:00.14	/usr/sbin/rsyslogd -n -iNONE
358	root	20	0	25428	2728	2428	S	0.0	0.3	0:00.00	/usr/sbin/rsyslogd -n -iNONE
359	root	20	0	25428	2728	2428	S	0.0	0.3	0:00.21	/usr/sbin/rsyslogd -n -iNONE
282	root	20	0	25428	2728	2428	S	0.0	0.3	0:00.40	/usr/sbin/rsyslogd -n -iNONE
375	root	20	0	66728	10068	8368	S	0.0	1.1	0:00.00	/usr/lib/udisks2/udisksd
383	root	20	0	66728	10068	8368	S	0.0	1.1	0:00.00	/usr/lib/udisks2/udisksd
393	root	20	0	66728	10068	8368	S	0.0	1.1	0:00.00	/usr/lib/udisks2/udisksd
408	root	20	0	66728	10068	8368	S	0.0	1.1	0:00.00	/usr/lib/udisks2/udisksd

Figure 6.7: Process manager htop showing HA on raspbian

```

pi@hassbian:~ $ mosquitto_pub -d -u JAG -P automate -t dev/test -m "AUTOMATION PROJECT"
Client mosqpub|744-hassbian sending CONNECT
Client mosqpub|744-hassbian received CONNACK (0)
Client mosqpub|744-hassbian sending PUBLISH (d0, q0, r0, m1, 'dev/test', ... (18 bytes))
Client mosqpub|744-hassbian sending DISCONNECT
pi@hassbian:~ $

```

Figure 6.8: MQTT Publish test result

```

pi@hassbian:~ $ mosquitto_sub -d -u JAG -P automate -t dev/test
Client mosqsub|734-hassbian sending CONNECT
Client mosqsub|734-hassbian received CONNACK (0)
Client mosqsub|734-hassbian sending SUBSCRIBE (Mid: 1, Topic: dev/test, QoS: 0)
Client mosqsub|734-hassbian received SUBACK
Subscribed (mid: 1): 0
Client mosqsub|734-hassbian sending PINGREQ
Client mosqsub|734-hassbian received PINGRESP
Client mosqsub|734-hassbian received PUBLISH (d0, q0, r0, m0, 'dev/test', ... (18 bytes))
AUTOMATION PROJECT
Client mosqsub|734-hassbian sending PINGREQ
Client mosqsub|734-hassbian received PINGRESP

```

Figure 6.9:

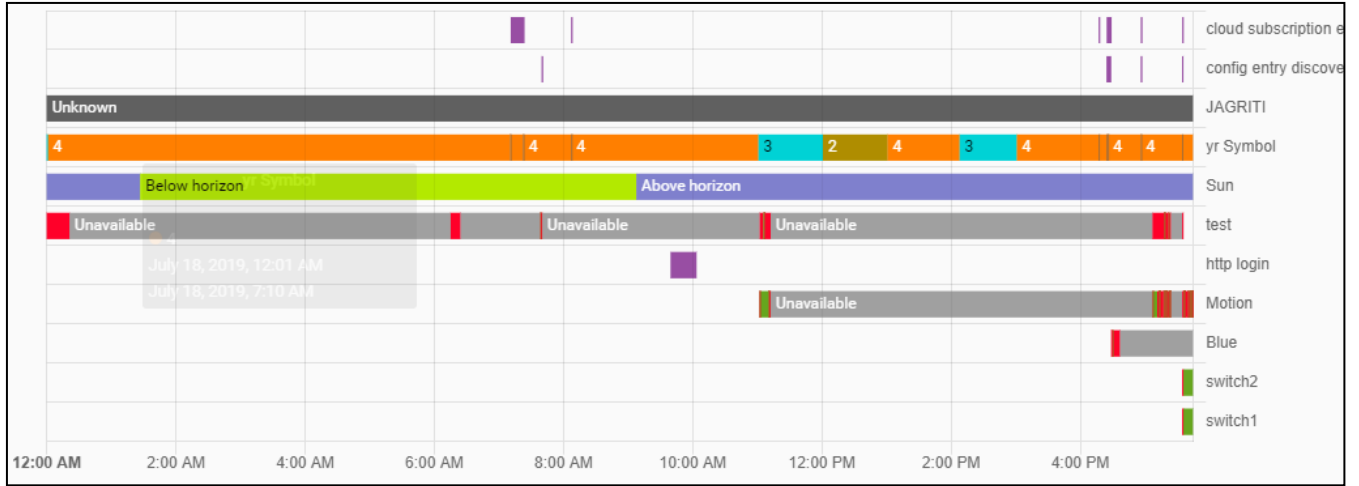


Figure 6.10: Lovelace UI logbook view

CHAPTER 7

CONCLUSION & FUTURE SCOPE

7.1: INTRODUCTION

After discussing the various aspects of this project from conceptualization to design and results, the next step is to draw repercussions from this research work. This chapter discusses the conclusions arrived and outlines the scope for future work in this area.

7.2: CONCLUSION

The objective of this project was to impart the knowledge pertaining to the various communication protocols and the use of one in successful integration with physical devices around us. The MQTT protocol works better than the REST (HTTP) in niche device integrations as it publishes only the subscribed topics and works in limited bandwidth. There is no background running process to cause weakness in connections and the response time is 13 seconds better than the latter by testing. The advantages of this have already been listed in earlier chapters, but it has immense future use and scopes. MQTT is a secure protocol that requires user authentication from both client and server sides. Both the subscription and publishing is validated from both sides by a user protected password. Further modifications can be addition of SSL to the protocol to make it more secure.

The secure shell access adds more security to the connection service. They can be easily made use in smart projects of home automation and smart cities where connection failure has to be actively avoided. However, since this is still a project under development, a few drawbacks do exist that have to be addressed in the work to be done following this. The API of integration between devices is not an easy task, depending upon the bandwidth of the internet connection being used. The micro-controllers being used in the test beds are subjected to refresh frequently due to API validation every 5 minutes. Once successful, the system does run smoothly, but the process to get there is an ongoing integration fight.

Even though the hub works in a wireless environment, it is better to keep it connected to direct cable Ethernet to avoid loss and latency in transmission. The sensors can be replaced with better

upgrades with a guarantee of accuracy, resolution and sensitivity. The readings obtained from the binary sensors are promising but ambient sensors need care with the power supply and current entering the input/output pins. They all work smoothly with the used protocol but have shown latency with HTTP.

The Home Assistant interface makes automation easier once it has been equipped with a good configuration.yaml file and integrations are done smoothly. Results have shown ease of use on both the desktop dashboard and mobile UI.

7.3 : FUTURE SCOPE

The extension to the present work is possible in the following ways:

- Using MQTT protocol more extensively in small lab projects to generate awareness
- Smart cities projects can be implemented with/without cloud support on HA
- Hardware tools like WireShark can be used to study the transmission packets and forecasting models vcan be developed.
- MATLAB integration of IoT projects for statistical analysis

APPENDIX 1

SCHEMATICS OF CIRCUIT BOARDS USED

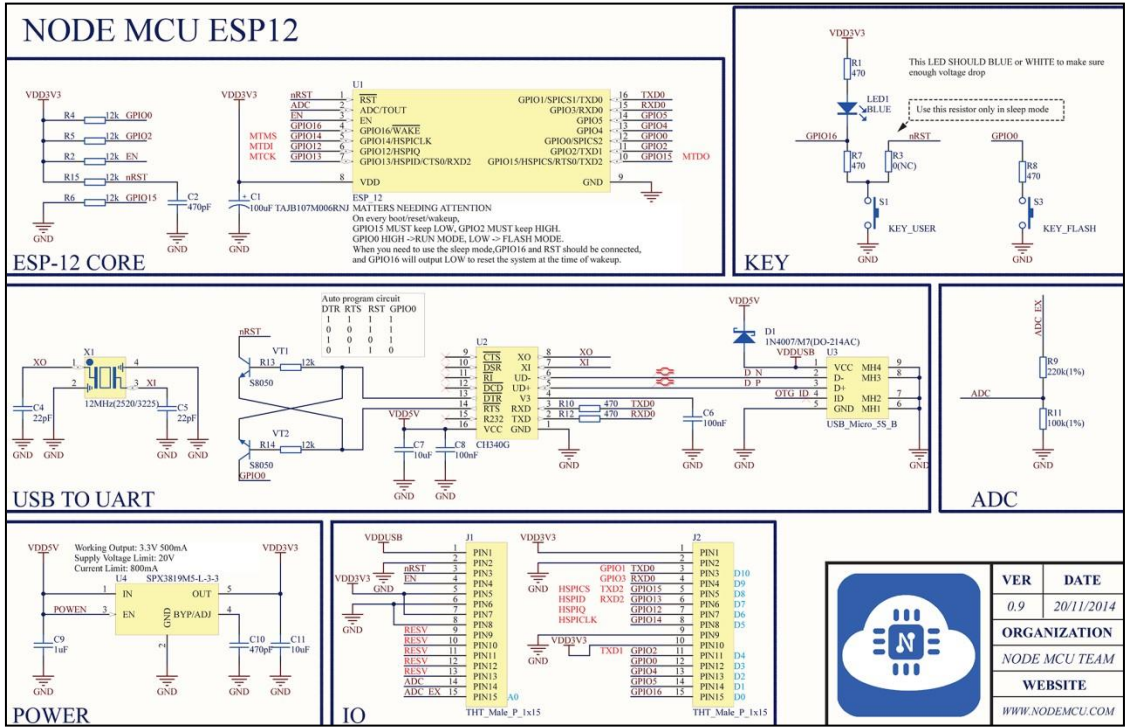


Figure 7.1: NodeMCU v1.0 dev kit schematics

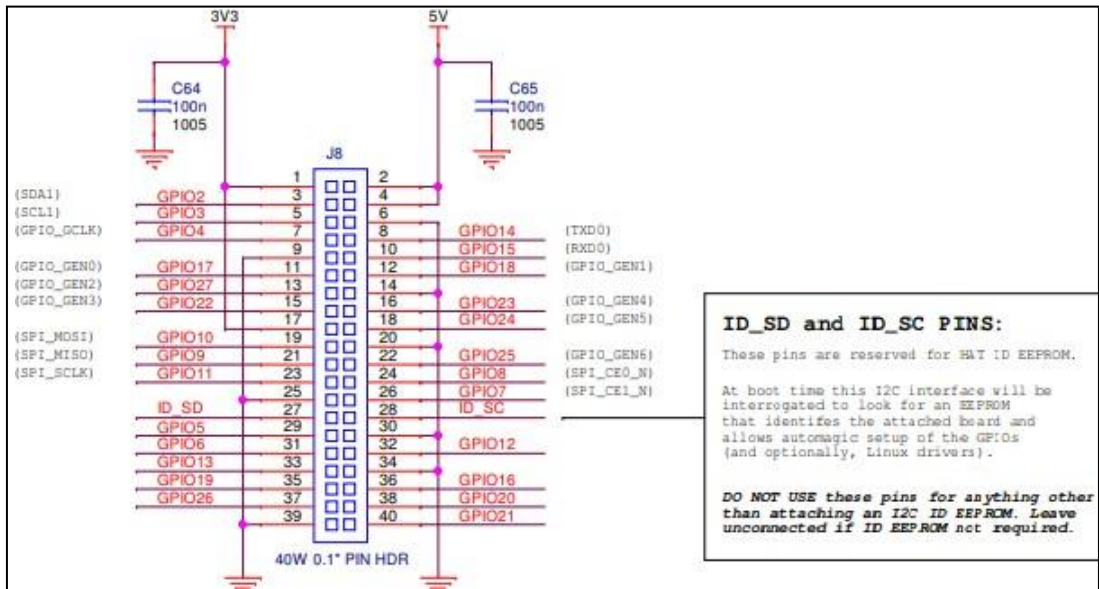


Figure 7.2: Schematics for Raspberry Pi 3 Model B

APPENDIX 2

LB110 SMART WI-FI LED BULB SPECIFICATION SHEET

Specifications

Working Status


- Typical Lumen Output: 800lm
- Input Power (Actual power draw in Watts): 10W typical
- Stand-by (Light off) Power: 0.5W Max
- Color Temperature: 2700K
- Rated Input Voltage: 220-240VAC 50/60Hz
- Beam Angle: 180 Degree typical
- Dimmable: Yes (via app and cloud only)

General

- Package Contents: Smart Wi-Fi LED Bulb LB110, Quick Installation Guide
- Bulb Lifetime (to 50% brightness level): 25000 Hrs min @Ta=25C
- Lamp Base : E27
- Certifications: RoHS, CE
- Operating Temperature: -15°C to + 40°C (5°F ~ 104°F)
- Operating Humidity: 10%-90%RH
- Weight: 170g

Network

- Protocol: IEEE 802.11b/g/n
- Wireless Type: 2.4GHz, 1T1R
- System Requirements: Android 4.4 or higher, iOS 10 or higher



TP-Link LB110 Smart Wi-Fi LED Bulb with Dimmable Soft White Light

Figure 7.3: LB110 specification sheet

APPENDIX 3

DHT11 TECHNICAL DATA SHEET

Parameters	Conditions	Minimum	Typical	Maximum
Humidity				
Resolution		1%RH	1%RH	1%RH
			8 Bit	
Repeatability			± 1%RH	
Accuracy	25 °C		± 4%RH	
	0-50 °C			± 5%RH
Interchangeability	Fully Interchangeable			
Measurement Range	0 °C	30%RH		90%RH
	25 °C	20%RH		90%RH
	50 °C	20%RH		80%RH
Response Time (Seconds)	1/e(63%)25 °C , 1m/s Air	6 S	10 S	15 S
Hysteresis			± 1%RH	
Long-Term Stability	Typical		± 1%RH/year	
Temperature				
Resolution		1 °C	1 °C	1 °C
		8 Bit	8 Bit	8 Bit
Repeatability			± 1 °C	
Accuracy		± 1 °C		± 2 °C
Measurement Range		0 °C		50 °C
Response Time (Seconds)	1/e(63%)	6 S		30 S

Figure 7.4: DHT11 Technical specifications

APPENDIX 4

CODE SNIPPETS

1. Test circuit code for Chapter 3 test model:

```
#define CAYENNE_DEBUG
#define CAYENNE_PRINT Serial
#include <CayenneMQTTESP8266.h>
#include <DHT.h>

char ssid[] = "*****";
char wifiPassword[] = "*****!";

char username[] = "d222bc00-1d7d-11e9-809d-0f8fe4c30267";
char password[] = "9a2b4d42d97f90733e3c4b68bdd6110be98d7fd6";
char clientID[] = "8ecd8d90-a3d8-11e9-9636-f9904f7b864b";

DHT dht(14, DHT11);

void setup() {
  delay(1000);
  int D3=0;
  int D4=2;
  int D6=12;
  int D7=13;
  pinMode(D3, OUTPUT);
  pinMode(D4, OUTPUT);
  pinMode(D5, OUTPUT);
  pinMode(D7, INPUT);
  Serial.begin(115200);

  Cayenne.begin(username, password, clientID, ssid, wifiPassword);
}

void loop() {
  Cayenne.loop();
  Cayenne.virtualWrite(0, analogRead(A0)/10);

  float temp = dht.readTemperature();
  float hum = dht.readHumidity();

  if (isnan(temp) || isnan(hum)) {
    Serial.println("Failed to read from DHT sensor!");
  }
  Cayenne.virtualWrite(1, temp, TYPE_TEMPERATURE, UNIT_CELSIUS);
  Cayenne.virtualWrite(2, hum, TYPE_RELATIVE_HUMIDITY, UNIT_PERCENT);
  if(digitalRead(D3)==0) Cayenne.virtualWrite(3,0);
  else Cayenne.virtualWrite(3,1);
  if(digitalRead(D4)==0) Cayenne.virtualWrite(4,0);
  else Cayenne.virtualWrite(4,1);
}

CAYENNE_IN(3)
{
  digitalWrite(D3, getValue.asInt()); //'asInt' for either '0' or '1' value }
}
```

2. Test circuit “test_blink” code for Chapter 5 model

```

float ldrValue;
int LDR;
float calcLDR;
float diffLDR = 25;

float diffTEMP = 0.2;
float tempValue;

float diffHUM = 1;
float humValue;

int pirValue;
int pirStatus;
String motionStatus;
char message_buff[100];
int calibrationTime = 0;
const int BUFFER_SIZE = 300;

#define MQTT_MAX_PACKET_SIZE 512

byte red = 255;
byte green = 255;
byte blue = 255;
byte brightness = 255;

byte realRed = 0;
byte realGreen = 0;
byte realBlue = 0;

bool stateOn = false;
bool startFade = false;
unsigned long lastLoop = 0;
int transitionTime = 0;
bool inFade = false;
int loopCount = 0;
int stepR, stepG, stepB;
int redVal, grnVal, bluVal;

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");

  char message[length + 1];
  for (int i = 0; i < length; i++) {
    message[i] = (char)payload[i];
  }
  message[length] = '\0';
  Serial.println(message);

  if (!processJson(message)) {
    return;
  }

  if (stateOn) {
    // Update lights
    realRed = map(red, 0, 255, 0, brightness);
    realGreen = map(green, 0, 255, 0, brightness);
    realBlue = map(blue, 0, 255, 0, brightness);
  }
  else {
    realRed = 0;
    realGreen = 0;
    realBlue = 0;
  }

  startFade = true;
  inFade = false; // Kill the current fade
  sendState();
}

#include <ESP8266WiFi.h>
#include <DHT.h>
#include <PubSubClient.h>
#include <ESP8266DNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#include <ArduinoJson.h>

#define IsFahrenheit true
#define wifi_ssid "****"
#define wifi_password "*****"
#define mqtt_server "192.168.1.***"
#define mqtt_user "****"
#define mqtt_password "*****"
#define mqtt_port 1883

#define light_state_topic "bruh/sensornode1"
#define light_set_topic "bruh/sensornode1/set"

const char* on_cmd = "ON";
const char* off_cmd = "OFF";

#define SENSORNAME "sensornode1"
#define OTApasword "automate"
int OTAport = 8266;

const int redPin = D1;
const int greenPin = D2;
const int bluePin = D3;
#define PIRPIN D5
#define DHTPIN D7
#define DHTTYPE DHT11
#define LDRPIN A0

bool flash = false;
bool startFlash = false;
int flashLength = 0;
unsigned long flashStartTime = 0;
byte flashRed = red;
byte flashGreen = green;
byte flashBlue = blue;
byte flashBrightness = brightness;

WiFiClient espClient;
PubSubClient client(espClient);
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  pinMode(PIRPIN, INPUT);
  pinMode(DHTPIN, INPUT);
  pinMode(LDRPIN, INPUT);
  Serial.begin(115200);
  delay(10);
  ArduinoOTA.setPort(OTAport);
  ArduinoOTA.setHostname(SENSORNAME);
  ArduinoOTA.setPassword((const char *)OTApasword);
  Serial.print("calibrating sensor ");
  for (int i = 0; i < calibrationTime; i++) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println("Starting Node named " + String(SENSORNAME));
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);
  ArduinoOTA.onStart([]() {
    Serial.println("Starting");
  });
  ArduinoOTA.onEnd([]() {
    Serial.println("\nEnd");
  });
}

```

```

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");

  char message[length + 1];
  for (int i = 0; i < length; i++) {
    message[i] = (char)payload[i];
  }
  message[length] = '\0';
  Serial.println(message);

  if (!processJson(message)) {
    return;
  }

  if (stateOn) {
    // Update lights
    realRed = map(red, 0, 255, 0, brightness);
    realGreen = map(green, 0, 255, 0, brightness);
    realBlue = map(blue, 0, 255, 0, brightness);
  }
  else {
    realRed = 0;
    realGreen = 0;
    realBlue = 0;
  }

  startFade = true;
  inFade = false; // Kill the current fade

  sendState();

  flashBlue = blue;
}

flashRed = map(flashRed, 0, 255, 0, flashBrightness);
flashGreen = map(flashGreen, 0, 255, 0, flashBrightness);
flashBlue = map(flashBlue, 0, 255, 0, flashBrightness);

flash = true;
startFlash = true;
}
else { // Not flashing
  flash = false;

  if (root.containsKey("color")) {
    red = root["color"]["r"];
    green = root["color"]["g"];
    blue = root["color"]["b"];
  }

  if (root.containsKey("brightness")) {
    brightness = root["brightness"];
  }

  if (root.containsKey("transition")) {
    transitionTime = root["transition"];
  }
  else {
    transitionTime = 0;
  }
}

return true;
}

bool processJson(char* message) {
  StaticJsonBuffer<BUFFER_SIZE> jsonBuffer;

  JsonObject& root = jsonBuffer.parseObject(message);

  if (!root.success()) {
    Serial.println("parseObject() failed");
    return false;
  }

  if (root.containsKey("state")) {
    if (strcmp(root["state"], on_cmd) == 0) {
      stateOn = true;
    }
    else if (strcmp(root["state"], off_cmd) == 0) {
      stateOn = false;
    }
  }

  // If "flash" is included, treat RGB and brightness differently
  if (root.containsKey("flash")) {
    flashLength = (int)root["flash"] * 1000;

    if (root.containsKey("brightness")) {
      flashBrightness = root["brightness"];
    }
    else {
      flashBrightness = brightness;
    }

    if (root.containsKey("color")) {
      flashRed = root["color"]["r"];
      flashGreen = root["color"]["g"];
      flashBlue = root["color"]["b"];
    }
    else {
      flashRed = red;
      flashGreen = green;
    }
  }

  void sendState() {
    StaticJsonBuffer<BUFFER_SIZE> jsonBuffer;
    JsonObject& root = jsonBuffer.createObject();
    root["state"] = (stateOn) ? on_cmd : off_cmd;
    JsonObject& color = root.createNestedObject("color");
    color["r"] = red;
    color["g"] = green;
    color["b"] = blue;
    root["brightness"] = brightness;
    root["humidity"] = (String)humValue;
    root["motion"] = (String)motionStatus;
    root["ldr"] = (String)LDR;
    root["temperature"] = (String)tempValue;
    root["heatIndex"] = (String)dht.computeHeatIndex(tempValue, humValue, IsFahrenheit);
    char buffer[root.measureLength() + 1];
    root.printTo(buffer, sizeof(buffer));
    Serial.println(buffer);
    client.publish(light_state_topic, buffer, true);
  }

  void setColor(int inR, int inG, int inB) {
    analogWrite(redPin, inR);
    analogWrite(greenPin, inG);
    analogWrite(bluePin, inB);

    Serial.println("Setting LEDs:");
    Serial.print("r: ");
    Serial.print(inR);
    Serial.print(", g: ");
    Serial.print(inG);
    Serial.print(", b: ");
    Serial.print(inB);
  }
}

```

```

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect(SENSORNAME, mqtt_user, mqtt_password)) {
      Serial.println("connected");
      client.subscribe(light_set_topic);
      setColor(0, 0, 0);
      sendState();
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

bool checkBoundSensor(float newValue, float prevValue, float maxDiff) {
  return newValue < prevValue - maxDiff || newValue > prevValue + maxDiff;
}

void loop() {
  ArduinoOTA.handle();
  if (!client.connected()) {
    // reconnect();
    software_Reset();
  }
  client.loop();
  if (!inFade) {
    float newTempValue = dht.readTemperature(IsFahrenheit);
    float newHumValue = dht.readHumidity();
    pirValue = digitalRead(PIR_PIN); //read state of the
    if (pirValue == LOW && pirStatus != 1) {
      motionStatus = "standby";
      sendState();
      pirStatus = 1;
    }
  }

  if (startFade) {
    if (transitionTime == 0) {
      setColor(realRed, realGreen, realBlue);
      redVal = realRed;
      grnVal = realGreen;
      bluVal = realBlue;
      startFade = false;
    }
    else {
      loopCount = 0;
      stepR = calculateStep(redVal, realRed);
      stepG = calculateStep(grnVal, realGreen);
      stepB = calculateStep(bluVal, realBlue);
      inFade = true;
    }
  }
  if (inFade) {
    startFade = false;
    unsigned long now = millis();
    if (now - lastLoop > transitionTime) {
      if (loopCount <= 1020) {
        lastLoop = now;
        redVal = calculateVal(stepR, redVal, loopCount);
        grnVal = calculateVal(stepG, grnVal, loopCount);
        bluVal = calculateVal(stepB, bluVal, loopCount);
        setColor(redVal, grnVal, bluVal); // Write current values to LED pins
        Serial.print("Loop count: ");
        Serial.println(loopCount);
        loopCount++;
      }
      else {
        inFade = false;
      }
    }
  }
}

else if (pirValue == HIGH && pirStatus != 2) {
  motionStatus = "motion detected";
  sendState();
  pirStatus = 2;
}
delay(100);
if (checkBoundSensor(newTempValue, tempValue, diffTEMP)) {
  tempValue = newTempValue;
  sendState();
}
if (checkBoundSensor(newHumValue, humValue, diffHUM)) {
  humValue = newHumValue;
  sendState();
}
int newLDR = analogRead(LDR_PIN);
if (checkBoundSensor(newLDR, LDR, diffLDR)) {
  LDR = newLDR;
  sendState();
}
}
}
if (flash) {
  if (startFlash) {
    startFlash = false;
    flashStartTime = millis();
  }
  if ((millis() - flashStartTime) <= flashLength) {
    if ((millis() - flashStartTime) % 1000 <= 500) {
      setColor(flashRed, flashGreen, flashBlue);
    }
    else {
      setColor(0, 0, 0);
    }
  }
  else {
    flash = false;
    setColor(realRed, realGreen, realBlue);
  }
}
}

int calculateStep(int prevValue, int endValue) {
  int step = endValue - prevValue;
  if (step) {
    step = 1020 / step;
  }
  return step;
}

int calculateVal(int step, int val, int i) {
  if ((step) && i % step == 0) {
    if (step > 0) {
      val += 1;
    }
    else if (step < 0) {
      val -= 1;
    }
  }
  if (val > 255) {
    val = 255;
  }
  else if (val < 0) {
    val = 0;
  }
}

return val;
}

void software_Reset() // Restarts program from beginning but does not reset the peripherals and registers
{
  Serial.print("resetting");
  ESP.reset();
}
}

```

REFERENCES

- [1] M. Pipattanasomporn, Murat Kuzlu, Warodom Khamphanchai and Avijit Saha
“BEMOSS: An agent platform to facilitate grid-interactive building operation with IoT devices” IEEE Conference on Innovative Grid Technologies-Asia (ISGT ASIA)
2015.7387018
- [2] Leonardo Albernaz Amaral, Everton de Matos, Ramão Tiago Tiburski, Fabiano Hessel, Willian Tessaro Lunardi and Sabrina Marczak, Book on “Middleware Technology for IoT Systems: Challenges and Perspectives Toward 5G”, Internet of things (IoT) in 5G Mobile Technologies, pp 333-367, 2016
- [3] Abhinav Prashant, Rohan Gupta, presentation on “Middleware for Internet of Things: A Survey”
- [4] Pandesswaran C, Surender S and Karthik KV “Remote Patient monitoring System based Coap in wireless sensor networks” International Journal of Sensor Networks and Data Communications 5:145 2016
- [5] Catarinucci L, Donno DD, Palano L (2015) An IoT Aware Architecture for Smart Healthcare Systems, IEEE journal 2: pp 515-526
- [6] Mohammad Aazam and Imran Khan and Aymen Abdullah Alsaffar and Eui-nam Huh
“Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved” Proceedings of 2014 11th International Bhurban Conference on Applied Sciences & Technology (IBCAST) Islamabad, Pakistan, 14th - 18th January, 2014, pp 414-419
- [7] Nagaraju Kaja “A Review of Energy Consumption in Residential Sector in India; Possibilities for energy conservation”, IJRHAL, Vol.6, Issue 6, Jun 2018, 375-384
- [8] Energy Statistics 2018 (25th issue), Central Statistics Office, Ministry of Statistics and Programme implementation, Government of India, New Delhi
- [9] Zhou, W., Jia, Y., Peng, A., Zhang, Y., and Liu, P. (2018) “The Effect of IoT New Features on Security and Privacy: New Threats, Existing Solutions, and Challenges Yet to Be Solved” IEEE Internet of Things Journal, 1–1. doi:10.1109/jiot.2018.2847733
- [10] Carlos M.S. Rodrigues, Bruno S.L.Castro “A Vision of Internet of Things in Industry 4.0 with ESP8266”, IJECET, Vol.9, Issue 1, Jan-Feb 2018, pp 1-12

- [11] NodeMCU, “Lua based interactive firmware for mcu like esp8266”. [Online]. Available: <https://github.com/nodemcu/nodemcu-firmware>. Last access: 10/12/16
- [12] Espressif, “Low-power, highly-integrated Wi-Fi solution”. [Online]. Available: <http://espressif.com/products/hardware/esp8266ex/overview/>
- [13] ESP8266 NodeMCU, “Comparison of ESP8266 NodeMCU development boards”. [Online]. Available: <http://frightanic.com/iot/comparison-of-esp8266-nodemcudevelopment-boards/>
- [14] A. Varghese and D. Tandur, “Wireless requirements and challenges in industry 4.0,” in Contemporary Computing and Informatics (IC3I), 2014 International Conference on. IEEE, 2014, pp. 634–638.
- [15] Manan Mehta, “ESP 8266: A Breakthrough In Wireless Sensor Networks And Internet Of Things”, in International Journal of Electronics and Communication Engineering & Technology (IJECE) Volume 6, Issue 8, Aug 2015.
- [16] Wi-Fi Module, “ESP8266EX Datasheet”. [Online]. Available: <http://www.adafruit.com>
- [17] Dong-Ying Li and Shun-dao Xie and Rong-jun Chen and Hong-Zhou Tan, “Design of
- [18] Internet of Things System for Library Materials Management using UHF RFID” 2016 IEEE International Conference on RFID Technology and Applications (RFID-TA),2016, pp. 44-48
- [19] S. Shubhangi, N.Pooja, S. Shubhangi, S. Vrushali, J. Yogesh, “MQTT- Messge Queuing
- [20] Telemetry Transport protocol”, Internation Journal of Research , ICRRTET, Vol.3, Issue 3, pp. 240-244
- [21] Andy, Stephen Clark. "MQTT for Sensor Networks (MQTT-SN) Protocol Specification Version 1.2" (PDF).
- [22] <https://www.raspberrypi.org/documentation/>
- [23] <https://www.home-assistant.io/>
- [24] <https://code.wireshark.org/review/gitweb?p=wireshark.git;a=tree>
- [25] <https://www.arduino.cc/en/IoT/HomePage>
- [26] <https://cayenne.mydevices.com/cayenne>