# DEVELOPMENT OF IoT BASED NETWORK FOR MONITORING APPLIANCES

BY
S M AZMI HOQUE
16121031
SUVRO DEBNATH
16121036
DEBOKY SAHA
16121049
WASEE AHMED
17321029

A thesis submitted to the Department of Electrical and Electronic Engineering in partial fulfillment of the requirements for the degree of Bachelor of Science in Electrical and Electronic Engineering

Department of Electrical and Electronic Engineering
Brac University
August 2019

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

**Student's Full Name & Signature:**

| | |
|---|---|
| **S M Azmi Haque**<br>16121031 | **Suvro Debnath**<br>16121036 |
| **Deboky Saha**<br>16121049 | **Wasee Ahmed**<br>17321029 |

# Approval

The thesis/project titled “Development of IoT based network for Monitoring Appliances” submitted by

1. S M Azmi Hoque (16121031)
2. Suvro Debnath (16121036)
3. Deboky Saha (16121049)
4. Wasee Ahmed (17321029)

of Summer, 2019 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Department of Electrical and Electronic Engineering on 29-8-2019.

**Examining Committee:**

Supervisor:
(Member)

_____

Dr. S. M. Lutful Kabir
Professor, IICT
BUET

Program Coordinator:
(Member)

_____

Dr.Saifur Rahman Sabuj
Assistant Professor, EEE
Brac University

Departmental Head:
(Chair)

_____

Dr. Shahidul Islam Khan
Chairperson & Professor, EEE
Brac University

# Acknowledgement

We would like to thank and show our immense respect to our honorable supervisor Dr. S. M. Lutful Kabir, Professor, Institute of Information and Communication Technology, BUET; for his enormous contributions, incomparable guidance and tireless support which led us to the completion of this thesis. Without his constant involvement and guidance it would be quiet difficult to achieve this task. We would also like to thank Engr. Rashedul Hasan, for his continuous cooperation throughout the research.

## Abstract

The whole world these days are widely dependent on industries to sustain the necessities of the seven billion people that inhabit the planet and all the industries want to become automated to achieve maximum efficiency. A number of factors determine the level of productivity in an industry. However, the production level does not increase to great extent due to several drawbacks. One of the vital issues that are faced by the industries is that the machine lifetime reduces due to the lack of persistent observation of the appliances. There can be fatigue failure, tolerance failure, and aging failure in instruments. As a result, the industry faces equipment based losses such as Unplanned Stops, Planned Stops, Small Stops, Slow Cycles, Production Rejects, and Startup Rejects. Manual supervising of equipment was prevalent earlier but the effectiveness of this approach was not up to the mark. A solution to this problem would be monitoring the condition of the appliances constantly using the internet which allows access to data from anywhere in the world.

We will be attempting to implement such a mechanism which will allow data from the machine end to be observed and controlled from remote locations using the internet. As a test system we will set up two appliances which will be three light bulbs. We will take into account current flowing through the appliance. Then, we will collect this information from the equipment by varying the load where light bulbs will act as a load and gather the named data. To transfer data to and from the server, we will use mainly two protocols which are MQTT and HTTP. In HTTP protocol, the IP addressing must be held in mind as a dissimilar IP may prevent the machine from interacting with the server, so a real IP is needed at that moment. MQTT is the most popular protocol for IoT and thus we will implement in our project. In MQTT protocol, the issue of changing the IP address with the change of devices can be solved easily. As publish-subscribe-based messaging protocol, it provides one-to-many distribution of messages. Furthermore, it is ideal for use in restricted environments such as Machine to Machine (M2M) communication and Internet of Things (IoT) contexts where a tiny code footprint is needed. Thus, for efficacious execution of our project MQTT protocol is better than HTTP protocol. In addition, the data will be stored in the database on the server. Upon our desire, we can send commands from the server to control the bulbs. The command will be read using microcontroller and it will automatically turn off the light bulbs. Lastly, as smartphones are widely in use these days, we have created a mobile application named “pahara” which will fetch data from the server and as a result we can easily observe the data from anywhere at any time.

**Keywords:** IoT ; HTTP; MQTT; ACS712; ESP8266

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Introduction

Internet of Things (IoT) points to the ever-growing network of physical objects. In IoT, data are collected in different ways via sophisticated sensors which communicates with each other and store this data in the cloud. In IoT, a large number of small blocks from devices, like numerous sensors, transfer across networks [1]. IoT is revolutionizing the way people communicate, work and live. It is a system of interconnected computer networks which allows data transmission without requiring human computer interactions. Moreover, IoT is also M2M communication system which means machines are able to communicate without direct involvement of a person. In this paper, we will discuss the system that we have developed for monitoring appliances remotely. Firstly, a sensor named ACS712 has been chosen for collecting current value which will serve as our data. For communication part we have dealt with two protocols HTTP and MQTT and ESP8266 is used to transfer our data. Furthermore, we have created our own server using PHP and MySQL database. Thus, it gives us the freedom of a remote communication. Finally, a mobile application has been created with an aim to monitor data continuously. So, the corresponding IoT architecture is customized with the mobile application.

## 1.2 Objective

Here our goal is to build a device containing transferring protocol, which will be best for delivering our data. Data transmission needs to be done in such a way that we do not have IP addressing issues. We have chosen two protocols and they are HTTP and MQTT. Both have their own complexity and ways of transferring data. Moreover, a server needs to be created alongside a database to monitor our data. Later a mobile app will be implemented which will make monitoring easier for users.

## 1.3 Motivation

The significance of IoT in every sector, including health monitoring and home automation knows no bounds. In building smart city it is having a crucial role. With the help of IoT, intelligent devices will become convoluted in our life in such a way that devices will take on a day-to-day job that was accomplished by humans conventionally. Previously, in the industries we needed individuals who could work all the time to keep an eye on the equipment for proper maintenance. Thus, it was really difficult for the workers and the cost of production was high. Therefore, an autonomous system should be built which gives flexibility along with control to the users. These devices will be both cost effective and time-saving. Hence, a device with an effective transfer protocol is required for observing. Furthermore, the server needs to be remote so that users can have access from any place in the world.

## 1.4 Literature Review

IoT helps us to handle, manage and store large amounts of data. In recent years, many papers proposed design and architecture for smart monitoring. [2] Presents an energy efficient temperature and humidity monitoring system using HTTP protocol. The measured data are dispatched over the internet using HTTP requests. This whole system is based on a local

network which can be a problem if we go for remote communication. [3] Focuses on home automation using MQTT protocol. Here they have used Adafruit as the MQTT broker and devices are connected with it. Software like IFTTT is used to access appliances through mobile. This approach is not cost efficient as we need to buy this software. [4] Discusses about a system which uses Zigbee based on home automation and focuses on the problem regarding wired connections. However, Zigbee is not secured like WiFi and has a low transmission rate. [5] Describes Bluetooth based home automation system, which contains a mobile host controller and home appliances. The home appliances communicate with the host controller through Bluetooth devices. In contrast, Bluetooth is useful only in short range communication and operating devices needs to be in the range. [6] Analyzes proposal of a system based on automated irrigation by some researchers which will help farmers to monitor crops and reduce wastage of water. They have made the use of Arduino embedded with sensors and WiFi. As Arduino has built in library, it is easier to utilize it. Moreover, they made the system cost effective by using an online based server. However, the use of Thingspeak as the server has made their data vulnerable and hence can be exploited. In this paper our proposed system emphasizes on the reception of current from a sensor and the accumulation of these data in the MySQL database of the server. Remote communication is possible by the use of MQTT and HTTP protocol. We have used our own online based secure server and database. Consequently, it has made our project cost effective and secured. Transmission of data is at a faster rate using ESP8266 and it also consumes less power. Moreover, we made our own mobile app for monitoring which means we do not have to rely upon other software.

## 1.5 Organization of the Thesis

The book is organized in eight chapters which are given as follows:

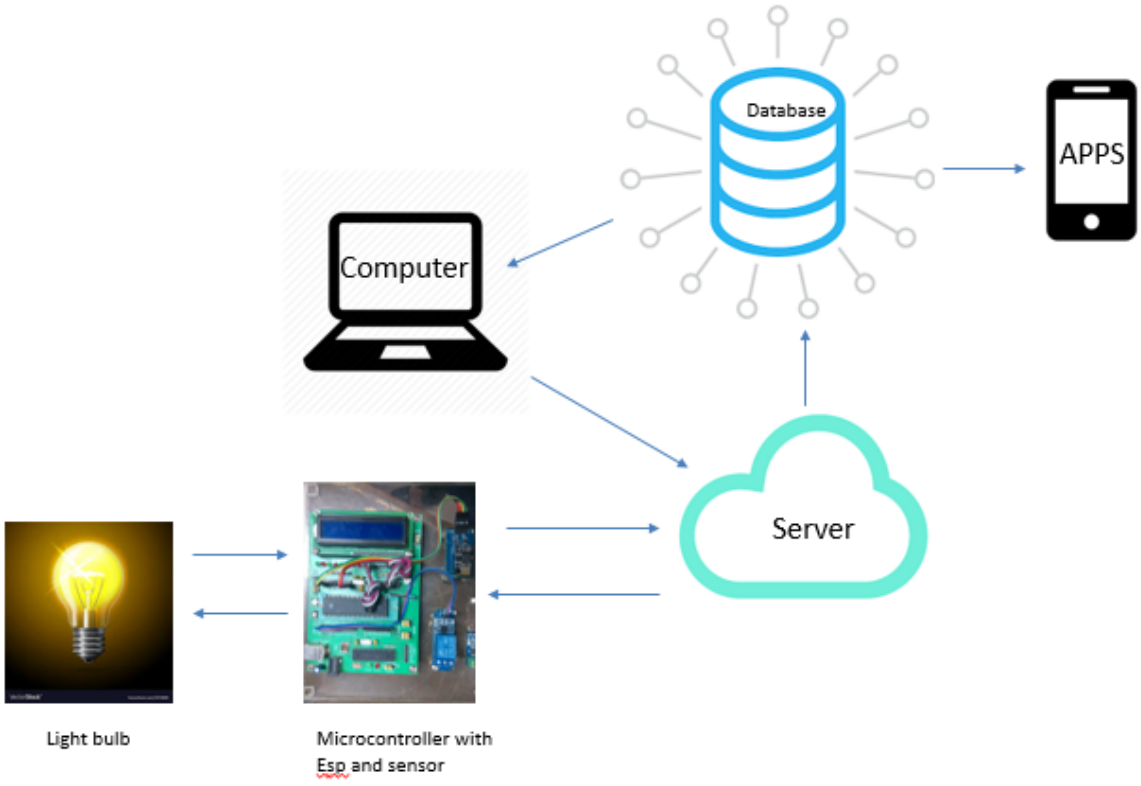
- Chapter-1 is the introduction of our thesis. Along with this, it includes different areas of past research related to our project. It contains motivation and objective of our work.
- Chapter-2 is divided into three sections. In first one, the concept of our project is given. Basically, the approach that we are going to follow and a brief idea of the whole system is described in this section. In the second part, we have discussed about the design or architecture we followed. In addition, last part includes the target output of our project along with the results to be obtained.
- Chapter-3 contains the hardware components that we have used. Each component is divided into different sections and their implantations in the project are explained alongside the codes used.
- Chapter-4 is about communication part of our project where two protocols have their own sections. In this chapter we have also explained the microcontroller side codes developed for both HTTP and MQTT.
- Chapter-5 explains how we have built the server and all other software that has been used to achieve remote communications. The server side code for both MQTT and HTTP protocols are discussed in this chapter.
- Chapter-6 is all about building an app so that we can view data from 000webhost in our smartphones and thus monitoring it continuously.
- Chapter-7 contains the result of Proteus simulation alongside the result of the data transfer rate obtained using both protocols.
- Chapter-8 concludes the paper by discussing the shortcomings and also the future scope of the project, which can be done to make the system more effective.

# Chapter 2

# Overall Design of the Project

## 2.1 Concept

The figure 2.1 depicts the general IoT construction of our system. It is clearly evident from the figure that Atmega32 is the core element of our prototype. To begin with, light bulb is connected to relay which in turn is attached to ACS sensor. Similarly, the ACS sensor is glued with Atmega32. The microcontroller is programmed in such a way that it transfers data to the broker with the help of an ESP8266 WiFi module as we are following MQTT protocol. In addition, the broker sends the data to the server. A computer is used to view and control the turning on and off the device which is essentially our light bulbs. Along with that, a mobile application is built up to effortlessly monitor data. Now, the main notion behind our project is described. The current sensor that we have used follows the principle of Hall Effect. Again, in microcontroller a code has been developed that reads variable DC from sensors. The ADC of Atmega32 uses $V_{ref}$ for calculation purpose. Besides, ESP8266 uses AT commands for communication. Initially, for transferring of data in the server, we have used HTTP protocol. Later, we have switched to MQTT protocol, which follows a different approach than HTTP protocol. In this protocol, a client can be both publisher and subscriber which are true for the server as well. The data published in the broker go to the server and the subscriber connected to the broker in the same topic will receive the message.




*2.1 Overall IoT architecture of our System*

## 2.2 Design of the system/Implementation

The figure 2.2 illustrates the hardware implementation of our project. There are two boxes made of Acrylic board. Inside the boxes there are AVR trainer board, ACS sensor, relay and ESP8266 WiFi module. In addition, there are three light bulbs attached to a wooden board and other three bulbs of light bulbs are joined with plastic board. First of all, a series board containing three light bulbs is assembled. The bulbs represent the appliances and its current values are altered by switching as they are in parallel. The current values have been obtained using the RMS calculation for variable DC wave. At the beginning, we have run simulations in Proteus in search of precise outcome. Then, current values are constantly varied in the code and later attached with Get request to send this data in the server. Furthermore, the AT commands will be checked in real term which is a serial terminal that allows us to see whether our ESP is able to send data in server. Code is set in such a way that if a particular AT command fails it will loop back into that AT command to check. In case of the server we

have connected MySQL with PHP, which will enable us to store current data in database. We will be using a free hosting site to keep our PHP files. Along with it, the hosting site allows phpMyAdmin to run. As a result, we can create a database. Moreover, we need to keep track of date and time, so separate fields have been created. Additional feature has been added in the server webpage which are the buttons that allows the user to turn on and off the light bulbs from anywhere. Finally, a mobile application has been created for observing data from remote locations. For communication between different systems we have used to protocols namely HTTP and MQTT as mentioned earlier. HTTP being an application layer protocol uses port 80/333 for communicating. The fundamental and safe transport layer protocol known as TCP protocol are frequently used for transmission. While working with HTTP protocol it is important to know IP addresses as it changes with WiFi. In contrast, MQTT solves the issue of regular update of IP address with the change of devices. Moreover, the control of appliances using MQTT is easier over HTTP due to the fact that the broker uses different topics for publish and subscription allowing the appliances to receive the message which is subscribed to the topic. Furthermore, the time of execution for data transfer is much lesser in MQTT than HTTP. Thus, we can declare that MQTT is more preferable way of communication than HTTP.

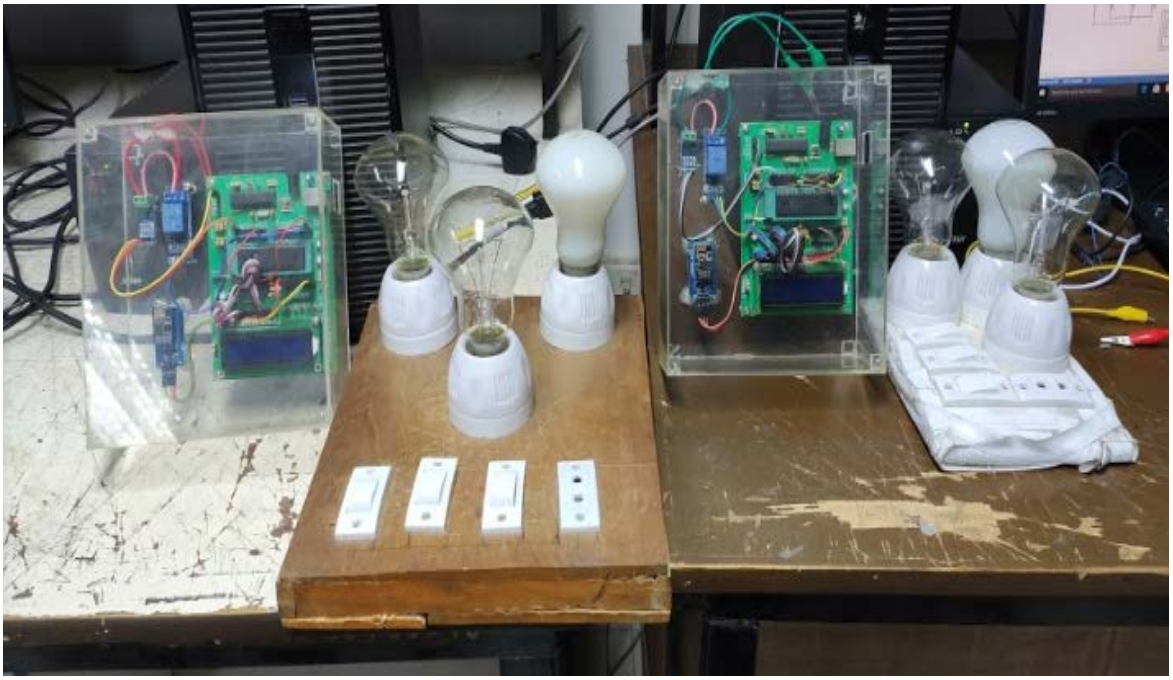


*Figure 2.2: Overall hardware design of our system*

## 2.3 Target output

To start with, our purpose is to look for data with least possible fluctuation in simulation. Besides, as HTTP is a two way communication which are essentially HTTP request and HTTP response, we will look into consideration whether they are being transferred and connections are being established with the server. On the other hand, for MQTT, we will check the connection establishment of the microcontroller with the broker. Our target is to compare the rate of data transfer and efficiency for both the protocols. The change of IP address along with the change in location of the router should be examined for HTTP. In addition, the data transfer rate should be scrutinized with respect to MQTT. Finally, we have to look at the response time of microcontroller to receive data from server. In this way, we will achieve our main objective.

# CHAPTER 3

# Basic Components in The Appliances End

## 3.1 Sensor

Sensors are sophisticated devices responsible for measuring physical quantities. It converts a physical parameter into signals which can be measured electrically. In our case we need to measure current. The sensor that we have used for measuring current was Asc712. It uses the principle of Hall Effect, which was discovered by Dr.Edwin Hall. The principal states that when a current carrying wire is placed in a magnetic field, a voltage is generated which is proportional to the current. This voltage is DC and proportional to the current. The Hall voltage is also proportional to the magnetic field. The module is using a RC filter so that it works fine in low frequency application. The ACS712 comes in three ratings 5A, 20A, and 30A.We have used 30A one in our case.

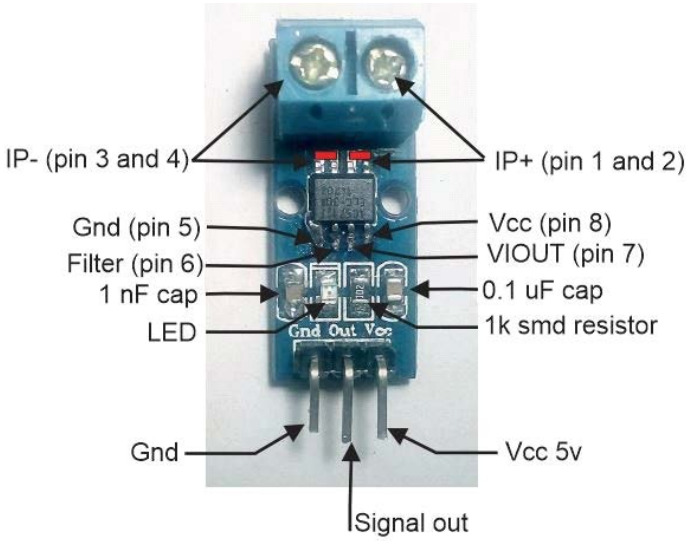



*Figure 3.1: Layout of ACS712 module.*

### 3.1.1 Simulation Code

While performing experiments, we have noticed on the oscilloscope that the output wave shape from the ACS is proportional to the AC current having an offset at $V_{CC}$/2 (2.5V in our case since $V_{CC}$ or supply voltage to ACS is 5V). Since the output is an analog signal, hence we used ADC to convert it to numeric values, thus achieving DC voltage. Here we have used the formula,

$$V_{in} = \frac{V_{ref}}{2^{no.of\ bits}}\ X\ ADCW \text{.....3.1}$$

Here, $V_{in}$ is the voltage inputted in digital format, $V_{rms}$ is the reference voltage, and the number of bits in ATmega32 is 10, which we have used here, and ADCW is the analog voltage received. However the value varies, so we needed to convert the varying DC voltage into its rms value, for it, we have considered the basic AC to RMS conversion formula:

$$V_{rms} = \frac{1}{T}\sqrt{\int_0^T (V_{out})^2\ dt} \text{.....3.2}$$

Here, T is the time period of one cycle, $V_{rms}$ is the average value of the voltage, and $V_{out}$ is the output voltage form the ACS. The value however is still in voltage and we now to convert it to current. To do this we need to divide this by the sensitivity of the device. Sensitivity refers to the ratio of current to voltage of the device provided in the data sheet having a unit of mv/A. Since we have used ACS712 of 30A, and according to the data sheet, sensitivity for it is 66 so we have divided it by 66 to get the value of current flowing through ACS.

## Code: (in Appendix C)

1) ADC conversion means conversion from analog to digital values. We have used preset library for ADC here, and used AVCC as reference voltage for conversion. A 10us delay is

necessary for the accurate conversion of analog signal by the MCU and at the end, the function returns the value that it senses.

2) The function “getCurrent()” will produce the value of current in the variable “cur”, which will be sent to the database. As per equation 3.2, we needed to sum up all the values in one period. The time period is 20ms for the line frequency is 50hz. Since a 10us gap is mandatory, so we have recorded values for 200 times, which we looped 10 times, making it 20000us which in turn is 20ms. The value of the output voltage of ACS has an offset of 2.5Vs, (2.495V in practice) so, a subtraction was needed to get deviation from the mean position. The values are then squared and added. After having gone through loops, we have the summation of the squared values of the voltages. The summation was then square rooted and then divided by the time period to get $V_{rms}$, afterwards this was divided by the sensitivity to get the current. The value of current is stored in the variable “cur”, the rest was part of calibration. We have displayed the result on the LCD which enabled us to compare the value provided by the MCU and ACS to the ammeter as shown in diagram below:

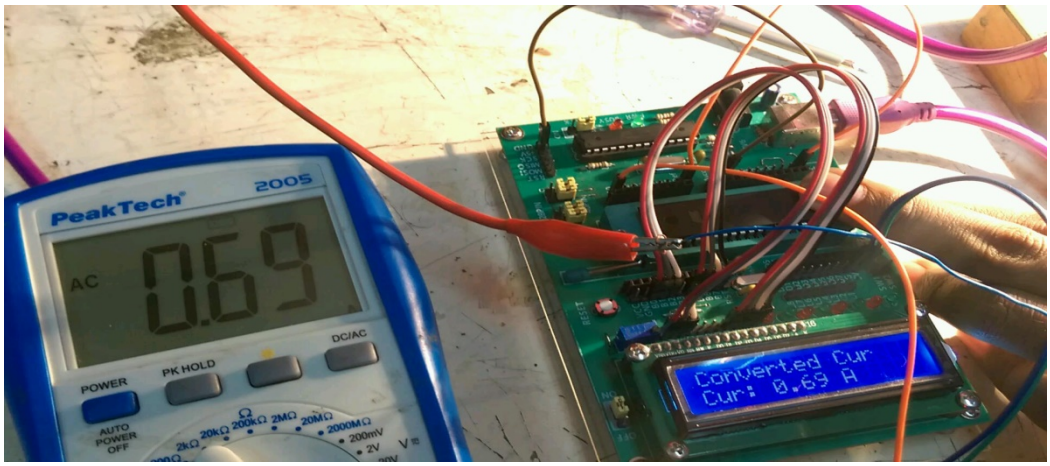



*Figure 3.2: Current reading in Atmega32 compared with multimeter reading.*

Therefore, our Atmega32 can work like an ammeter which proved to be faster than the multimeter and hence data of current is ready and can be sent now.

## 3.2 Communication at The Appliances End

ESP8266 have integrated TCP/IP protocol stack. TCP and UDP are both transport layer protocol, which is responsible for flow control, reliability and correctness of data that is being sent over the network. TCP provides a virtual circuit between sender and receiver, and this protocol helps retransmit damage frame so no data is lost and we therefore get a secure connection [8]. In transmitting end TCP causes fragmentation and attaches a sequence number which is detected by receiver and frames are rearranged according to their order. Thus, any lost frames are tracked. In TCP/IP model application layer is the top most layer and allows user interactions with the application. When two application layers want to communicate, the data is transported in the transport layer. Application layer contains HTTP, MQTT, FTP etc. Network layer is lowest layer and it is a combination of physical and data link layer. It identifies how data will be transmitted throughout the network. This layer carries out mapping of IP addresses into physical ones.

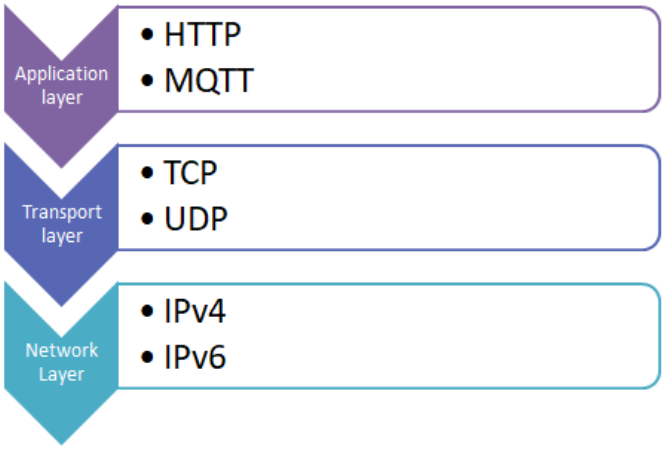



*Figure 3.3: IoT Protocol Stack*

### 3.2.1 ESP8266 Module

ESP8266 contains inbuilt TCP/IP protocol stack that gives any microcontroller access to your WiFi network [9]. All ESP8266 have built in AT commands firmware. Most effective features of ESP8266 are being cost efficient and they have on board processing capability. For ease of using it, ESP8266 is integrated in PCB board, so that we can finally get transmitter, receiver, power and ground. Moreover, it works in 3.3V power supply as 5V power supply may damage the module. In order to get the information of AT commands in detail, we have connected ESP8266 WiFi module with FTDI USB to Serial Converter 3V3 and used Software named RealTerm where we have checked all the AT commands.

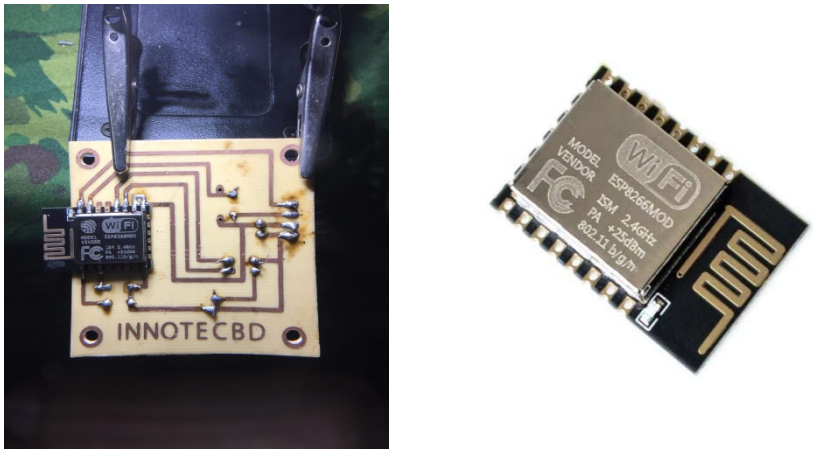



*Figure 3.4: Soldering of ESP8266 in a PCB board.*

ESP8266 specifications are [10]:

- 2.4 GHz Wi-Fi (802.11 b/g/n, supporting WPA/WPA2)
- General-purpose input/output (16 GPIO)
- I²C serial communication protocol
- UART (on dedicated pins, plus a transmit-only UART can be enabled on GPIO2)
- Pulse-width modulation
- Serial Peripheral Interface (SPI) serial communication protocol,

### 3.2.2 AT Commands

AT is the abbreviation of Attention and often used to control modem. Some basic tasks that can be done using AT commands include getting information from mobile phone, getting info of the subscriber, establishing data connection etc. ESP8266 supports wide ranges of AT commands that help to interface ESP8266 with microcontroller through UART. AT commands are involved in controlling operations like connecting to WiFi, restart, mode selection and many others.

1. AT+CWMODE_CUR=3

This command is used to acquire the existing WiFi working modes. Here, 3 states that it will work both in Station mode and Access Point mode. AP mode enables us to develop our own network and connect to it with other devices where Station mode mode enables the ESP8266 to connect to a Wi-Fi network which is generated by our wireless router.

1) Station
2) AP
3) Station AP

This configuration will not be stored in flash.

2. AT+CWJAP_CUR (JOINT ACCESS POINT) =”ssid”,”password”

This command connects to WiFi router. We need user ID, password and channel for this. In our project, we have used ssid as “STS” and password “tkd9663sts”.

3. AT+CWLAP

This command will list all the surrounding access points. Finding AP with specific ssid is important.

4. AT+CIPMUX = 0

This command sets the module to multiple connections. CIPMUX=0 means it has single connection and CIPMUX=1 will represent multiple connections. We have used 1 in our code.

5. AT+CIPSTART = “Type”, “address”, port

This command connects to the website and establishes TCP (type) connection. IP address of receiving end the server and port number is also important here. The IP address used in this project is “waseeserver.000webhostapp.com” and port number 80.

6. AT+CIPSEND=”data length”

This command is used to send data from ESP8266 module where number of characters is specified in this case. In our case, the number of characters is 73.

7. AT+CIPCLOSE

It closes TCP or UDP connections.

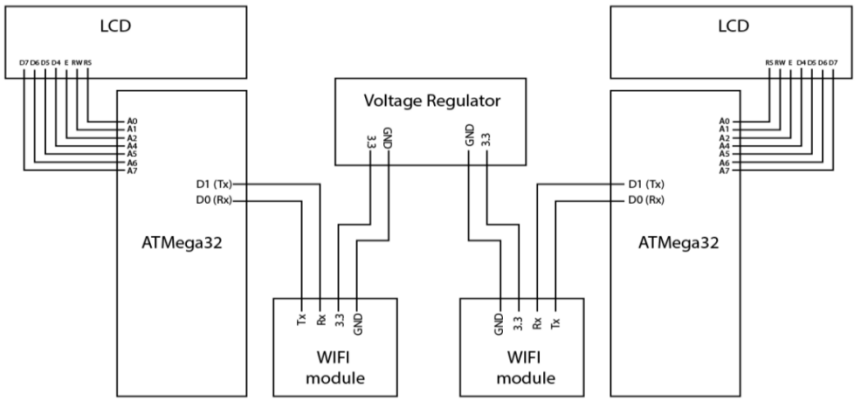



*Figure 3.5: Interfacing ESP8266 with ATmega32*

## 3.3 Relay

Relay is an electromagnetic switch which helps to make or break contact of a circuit. They achieve this property with help of signal which makes circuit on or off. Relay is used for controlling high power circuit with the help of low power signal. Relay operates in either NC (normally closed) or NO (normally open) condition. Normally open means when no signal given the circuit contacts will be open as relay not energized. Our purpose of using relay is to control our appliances remotely. From the server side we will send on or off command and microcontroller will receive signal and act accordingly. We will check for on or off in "search" function. If any of the condition matches then actions will be taken accordingly. For example if OFF is matched, then microcontroller will send a signal to relay to open the circuit.

*Figure 3.6: Relay*

# Chapter4

# Remote Communication

## 4.1 Introduction

The application layer directly interacts with the end users, so protocols used here is of great importance. In TCP/IP model application layer exists on the uppermost part from the client end which imparts the interface between different applications and the network. Moreover, this layer performs data formatting and presentation. Application layer being implemented by browser, contains protocol like HTTP, AMQP, MQTT, CoAP and many others. We also need a protocol that allows remote communication without using a fixed real IP.

## 4.2 HTTP

The configuration of the system under HTTP protocol is depicted in figure 4.1 below

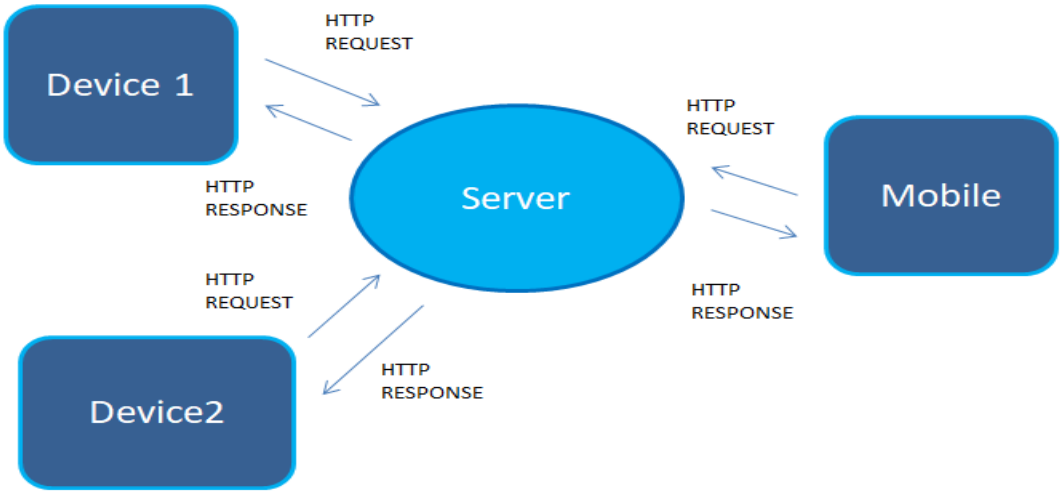



*Figure 4.1: Configuration of system using HTTP*

The HyperText Transfer Protocol (HTTP) is used to define how client-server programs can be written to retrieve web pages from the Web. An HTTP client sends a request and thus an HTTP server returns a response. The server uses the port number 80. Similarly, the client uses a temporary port number. HTTP uses the services of TCP which is a connection oriented and reliable protocol.

### 4.2.1 The Protocol

HTTP is the foundation of data communication for the World Wide Web. It follows a client server structure, message size is usually high. Usually two types of message are prevalent in HTTP and they are request message and response message. The first section in the request message is called request line. There are three fields in this line separated by one space and carriage return. The fields are called method, URL and version. The method defines the type of message. The client uses GET method to send request where the body of the message is empty. The post method is used to send some information to the server, which is usually used to add or modify the web page. After the request line, there are zero or more request header lines followed by the body. The body contains the command to be sent or the file to be published on the website. Similarly, the response message consists of a status line, header lines, a blank line and sometimes a body. The first line in a response message is called the status line. There are three fields in this line separated by one space and carriage return. The first field defines the version of HTTP protocol, which is currently 1.1. The status code field defines the status of the request. It consists of three digits. The codes in the range 100 are only informational; the codes in the 200 range indicate a successful message. In addition, the codes in the range 300 range redirects the client to another URL and the codes in the range

400 indicate an error on the client site. Finally, the codes in the 500 range indicate an error on the server site. For response message there are and zero or more response header lines next to status line. Each header line sends additional information from the server to the client. In addition, the body contains the document to be sent from the server to the client, which is present unless there is an error message. Additionally, HTTP comes in handy when we go for big data collection.  In contrast, while working with HTTP the IP addressing needs to be kept in mind as a different IP might stop device from communicating with the server, hence a real IP is required at that time.

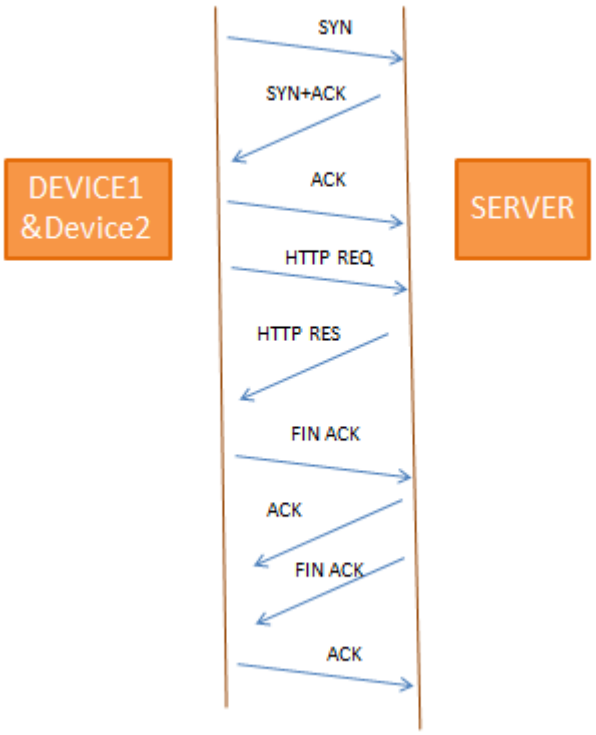



*Figure 4.2: Communication Sequence in HTTP*

HTTP follows the three-way TCP handshake. This helps to synchronize segment number. Therefore, no packets will be lost giving a secure data transfer. The device (client) sends a synchronize packet to the server. The server responds using synchronize/acknowledge packet. The device (client) site receives the SYN/ACK packet and replies with Acknowledgement [11].

### 4.2.2 Microcontroller side code

This part explains the code running on the microcontroller (AtMega32), which primarily will send and receive data from the server, in other words communicate with the server. We have used a switch case structure for this code. Since our task is sequential, this structure provided us a system where the flow of instructions can only move forward if the previous task was successful. In main code section, we will explain the cases and their purpose, afterwards in library section we will explain what the functions do in much more detail.

### Main code: (in Appendix D)

1) The library file was added at the beginning, and PORTC.0 will be used to turn on or off the bulb, it was initially set to 1. The MCU shows “Welcome” at the start where the LCD, ADC, USART are initialized for proper functionality. We have applied the switch case structure to carry forward our work. We took PORTC.0 as the led control pin.

2) In each case, specific “AT Commands” will be sent to ESP8266 and the reply will be read. This is how we communicate with the ESP. From here on “Initializing” will be shown on the LCD which depicts that the ESP module is being initialized and made ready for us to start communication. At the start, we send “AT” and if the module is connected, we expect the reply to be “OK”. If this is such the case, we can move to case 2 otherwise, we will keep repeating this, which indicates that the module is not detected by the MCU. In case 2, we set the mode at 3. If this is successful, we will match this by comparing the reply

(expected to be OK in this case) using the search function (explained in library section). Then, we can move on to case 3 which will mainly deal with connection with the WiFi. Here we have to send the command “AT+CWJAP_DEF?” which asks the module if it is connected with the WiFi or not. If the reply is “No AP” then we will move to case 4 which will search and connect to the WiFi. However, if the module is already connected to the WiFi, it will skip case 4 and move to case 5. In case 4, “AT+CWJAP_DEF=‘user id’, ‘password’ ” was sent to ESP, which will use the information to connect to the WiFi. If “WIFI CONNECTED” was the return message from the ESP it means that the module has connected with the WiFi and we can move on to case 5. Our ultimate goal is to reach case 5 and onwards, hence if case 3 is successful, it directly can move to case 5, skipping case 4 which makes the procedure faster.

3) In case 5, we need to set the mux using “AT+CIPMUX=1” and the value to be set is 1 which allows multiple connections. For all of the steps till now, we will only see the words “Initializing”, and a few dots afterwards to indicate the code moving forward. In Case 6, we would see the word “Sending”, which means all the previous steps have been completed and now we can move on to the actual data transfer. This does several tasks, starting with getting the value of the current flowing through using the “getCurrent” function (as explained in chapter 3.1.1), the value of the current will be stored in a float variable “cur”. The function “apiKeyValueUpdate” uses this to generate an API key which will be used to send data. This case sends the AT command “AT+CIPSTART” and in it we have sent the type of connection (TCP), the remote IP (domain name of the web server) and the port number (80). This starts the connection. If the reply is “CONNECT”, then we can move on to case 7 which basically sends the information about the number of bytes to be sent using the command “AT=CIPSEND”. This has to be highly specific, and our API key is about 73 bytes hence 73 was sent. Then we move on to case 8. In the

circumstance where we did not find “>” symbol (this will be returned in the case where the command was successful), we would consider that the connection has been broken, thereby going straight to case 9 (explained later), restarting the process.

4) With the API key being ready in stage 6 and the number of bytes that would be send is specified in case 7, in stage 8, we have to send data with the “USART_println(char *send)” function to the server, where it stores the data and lastly, if this too was successful, we must close the connection. Closing the connection is what is in case 9 where we have used the command “AT+CIPCLOSE” which closes the connection.

Thus, one objective has been met which is sending data to the server. The other objective was to control the light bulbs from the server end. Since our device is in a remote location with a remote IP, hence sending data from the server is difficult. Here we have used a special technique to make this possible. As we know, after sending data via HTTP protocol, a return message is sent from the server side as a response that it has received data which is termed as “echo”. We have used this to send data from the server. When we wanted to turn on the bulb, we send “eqp1ON” and “eqp1OFF” if we wanted it off (eqp2 for the other equipment). So at stage 8 after sending the value of current, we have immediately started to search for the keywords, and if “eqp1ON” was found, then PORTC.0 is made high, which in other words would turn on the lamps, and vice versa if “eqp1OFF” was found. After this, in order to send and receive new data, we have to start from stage 6, that is by re-establishing connection.

## Library Files: (in Appendix E)

1) The “void USART_println(char *send)” is one of the most important functions for using in all the stages and its purpose is to send commands to ESP8266. The while loop will continue till the data is sent. However, before that it checks if the USART data register is

empty or not and if it is empty, it loads the data in UDR, which sends the information to the ESP module.

2) The “int Search(char *search)” function is probably the other most used functions. Its job is to check whether the value passed through to the function exists in the array “receive”. This array holds the reply that the MCU gets from the ESP. When the MCU receives data, an interrupt is called, and the data from the UDR is stored in the receive array. The procedure used here is that initially, we count the number of characters that the “search” function has. Then we match each of the characters of “search” individually and if the character is the same as in the receive array, “matchFound” variable is incremented. This goes on till the entirety of the receive array is checked through, and in the end if “matchFound” is of the same number as “numberofmMatch” then 1 is returned, otherwise 0 is returned, where a return of 1 would mean that the variable passed on exists in the data send back by the ESP module.

3) The “apiKeyValueUpdate(float value)” function is called on stage 6 and its purpose is to update the “API_key” array. In the array only 4 values change namely the value of the current in variable “cur” (00.00) as shown in the array “char API_key[74]”. The value of the current is initially made an integer and then each of the digits placed individually. For example, suppose the value of current is 12.09, the first line converts it to 1209. In “API-key [18]” we need “1”, so 1209 divided by 1000 gives “1” and we have added 48 in each of the lines since we want the value in ASCII format. The other digits have been converted to fit each of the digits in the positions required.

## 4.3 MQTT

### 4.3.1 The Protocol

The configuration of the system under MQTT protocol is illustrated in figure 4.3 below

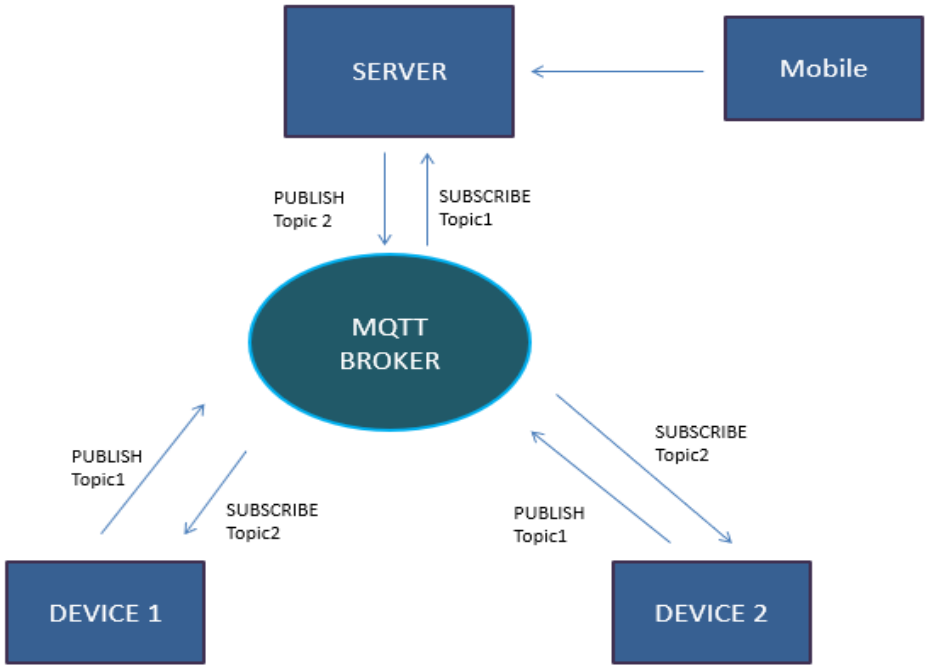



*Figure 4.3: Configuration of system using MQTT*

MQTT stands for Message Queuing Telemetry Transport protocol, which is featherweight broker based publish/subscribe protocol. MQTT is designed to give advantages to small processing devices by communicating in low bandwidth [12]. In MQTT protocol broker uses different topic for publishing and subscribing. Appliances subscribed to the topic will receive the message. MQTT separates publisher and subscriber in space, time and synchronization. In this protocol, the publisher or subscriber do not know about each other's IP address or port. Moreover, there is no need of the subscriber or publisher to be actively present at the same time. The speed of sending or receiving messages can be different. This makes control easier as we can control not only one device like HTTP. Also the time frame to execute task

decreases. Furthermore, MQTT has built in Quality of Service (QoS) level for message delivery. It is an agreement between the sender of a message and the receiver of a message which defines the assurance of transportation for a particular message. There are 3 QoS levels in MQTT and they are given below:

- **At most once (0):** The lowest position of QoS is zero where delivery of a message is not guaranteed. The receiver does not recognize acknowledgment of the message and the sender does not store and re-transmit the message. QoS level 0 is often referred to as "fire and forget," providing the exactly similar assurance as the TCP protocol underlying it.
- **At least once (1):** QoS level 1 ensures that the receiver receives a signal at least once. The sender will store the message until the receiver receives a PUBACK packet that recognizes receipt of the message. Moreover, sending or delivering of a message is possible numerous times.
- **Exactly once (2):** In MQTT, QoS 2 offers the highest level of service. This level ensures that the intended recipients receive each message only once. QoS 2 is the most secure, but it is slow. The guarantee is given between the sender and the receiver that there will be at least two request / response flows (a four-part handshake) among them. The sender and receiver use the original PUBLISH message packet identifier for coordination in delivery of the message.

In addition, MQTT support intermittent connectivity which means the session may last for weeks or even months. It uses automatic “keep alive” messages. This means that if a message is sent using QoS 1 and QoS 2 and the device is not connected, then the message will be queued for delivery when the device regains connectivity. Furthermore, it supports retained message which is automatically delivered when a client subscribe to a topic. A single message per topic is distributed automatically when one subscribes to the topic for the first

time if one stays subscribed to the topic, then any changes in the message will be pushed to the subscriber automatically.

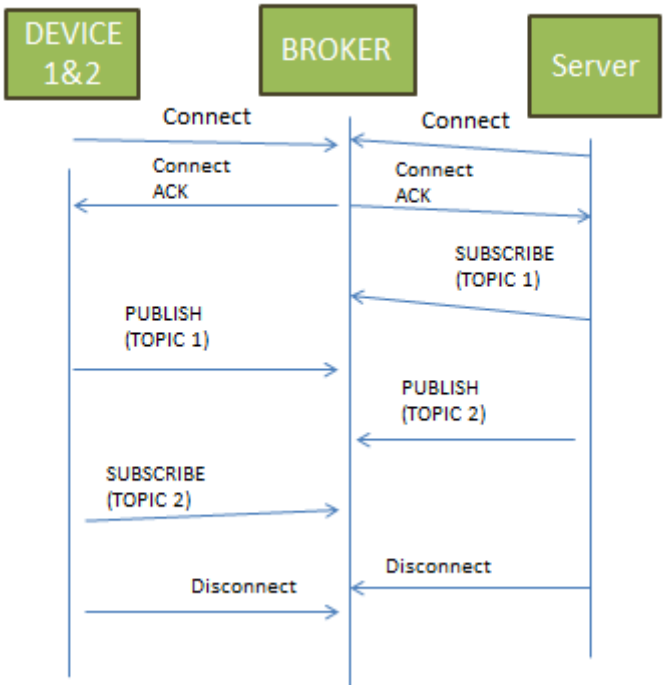



*Figure 4.4: Communication Sequence in MQTT*

In the above figure, both devices are publishing to Topic1 and the server is subscribed to the same topic. So whenever the broker receives a data it immediately sends it to the server. Moreover, for controlling purpose the server publishes to Topic2 and our devices are subscribed to the same topic. When microcontroller receives any command for turning on or off, it will act according to it.

### 4.3.2 MQTT Packets

As like during the communication during HTTP, the MCU needed to be coded to allow us to communicate with the broker, we will have used the MCU as both publisher and subscriber. To use the MQTT protocol, we had to use special packets that the protocol uses which is described below.

**The connect packet:**

This section will cover the connect packet. It is actually a single packet which we have broken it down to three tables for better understanding. Upon connecting to the broker, the connect packet must be the first packet that is to be sent from the client to the server, which should only be sent once over a connection.

| CONN | RL | PLEN | | PNAME | | | | | | LVL | FLAG | KA | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 16 | 44 | 00 | 06 | M | Q | I | s | d | p | 03 | 194 | 0 | 60 |

*Table 4.1: Connect Packet (byte 0 to 13)*

The above shows the first 13 bytes of the connection packet. The full form of byte 0 is shown below.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | MQTT Control Packet type (1) | | | | Reserved | | | |
| | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

*Table 4.2: Byte 0 of Connect Packet*

In byte 0, the first four LSB bits are reserved with value 0 and the four MSB bits are for MQTT control packet type. The connect packet being control packet type 1 is fixed and the lowest bit of the MSB that is the $4^{th}$ bit is 1 and reset bits are zero. This results in the value of CONN byte having the value of 16.

Byte 1 named as RL stands for Remaining Length which is in the number of bytes that prevails in the packet. The total length of our connect packet is 46 but the first 2 bytes are being used up so the remaining bytes is 44 and hence, in the RL field we provide 44.

Byte 2 and 3 is for the length of the protocol, which is about 6 bytes in our case because the protocol we have used is “MQIsdp”. This is because since our broker is CloudMQTT and

they use this protocol which is the reason for using “MQIsdp” here and the name itself takes up takes up the bytes 4 to 9.

Byte 10 represents the revision level of the protocol used by the client. The level of protocol used was 3 by CloudMQTT and hence we have put 3 here. Byte 11 is the flag byte, and its full form is shown below:

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Byte 11 | User Name Flag | Password Flag | Will Retain | Will QoS | | Will Flag | Clean Session | Reserved |
| | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |

*Table 4.3: Byte 11 of Connect Packet*

This in decimal is 194 and hence it is our input. This byte is essentially the flag bytes which will specify the behavior of the MQTT connection.

Bit 1 is for clean session and is used to control the lifetime of the session state. With this bit being 0, the server must resume communications with the client based on state from the current session, in a case where there is no session, the server, must create a new session. If it is 1, the client and server must start a new session, and discard any previous sessions. The new session will continue as long as the network connection does not fail. We have put 1 here for continuing the session as long as disconnection does not occur.

The 2$^{nd}$ bit is for will flag. With this being 1, it indicates that if the connect request gets accepted, a “will message” must be stored on the server and associated with the network connection. This also must be published when the connection closes unless being deleted by the server on receipt of a disconnect packet. Along with that, the Will QoS and Will retain fields will be used by the server, and hence the fields must be present in the payload. With 7$^{th}$ bit being 0, the “will retain” and “will QoS” must be 0 and must not be present in the payload, and also, a will message will not be published when this network connection ends.

The 4th and 3rd bit specifies QoS level during publishing. If the “will flag” is 0 then “the will QoS” must be 0 as well, however with “will flag” set to 1, QoS flag can be 1 or 2, however cannot be 3.

The 5th bit is for “will retain” which specifies whether the “will message” is to be retained during publishing. When “will flag” is 0, then the will retain flag must also be 0. If “will flag” is 1 and “will retain” is 0. The server publishes the “will message” as non-retained message, and when “will retain” is 1, the will message will be published as a retained message.

The 6th bit is for password, if it is 0, meaning we have no password, and hence a password should not be present in the payload, however when it is 1, password must be present in the payload.

Lastly, the 7th bit is for User name flag, when 0. User name should not be present and vice versa. If the 7th bit is 0, the password field must be 0 as well.

Bytes 12 and 13 represent keep alive flag, which essentially is a 16-bit word which represents the maximum time between the client finishes transmitting one control packet and starts sending the next one. We have used 60 here to keep the connection alive for a minute.

This concludes the basic connection packet; the rest of the packet is referred to as the “payload” as it contains information that is specific to the use. In our case we have the client identifier, a username and a password, and our payload will comprise of three parts. As a part of the protocol it is mandatory that the order is, client identifier, will topic, will message, username and password. We have not used the “will topic” and “will message” here which is why those will not remain a part of our payload.

| CIDLEN | | CID | | | | | | ULEN | | UNAME | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 6 | M | I | C | R | 0 | 1 | 00 | 08 | n | C | i | h | n | t | a | k |

*Table 4.4: Connect Packet (byte 4 to 31)*

| PWLEN | | PWD | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 |
| 0 | 12 | C | O | 7 | A | c | c | S | W | r | z | Z | F |

*Table 4.5: Connect Packet (byte 32 to 45)*

So the 14th and 15th byte represents the length of the Client ID which is 6 in our case for "MICRO1" (MICRO2 for the other module), so the value has been put as such and the bytes 16 to 21 covers the Client ID. In the similar way the Bytes 22 and 23 represents the length of the user name. Upon setting up the account on CloudMQTT, they have by default provided us with a user name and a password which goes up here. So with our user name being "nCihntak" which is of 8 characters, the length was 8 in ULEN field and bytes 24 to 31 was allotted for this. The password works similarly and our password was "C07AccSWrzZF" which is of 12 characters so, bytes 32 and 33 represents 12 and bytes 34 to 45 was used up for the password.

**The publish packet:**

The publish packet will be used by the MCU to send data to the server, hence data will be the value of current. The packet as a whole is given below. Here our topic name was "Publish1" and the message was the value of current (represented here by xx.xx).

| PFL | RL | TLEN | | TOPIC | | | | | | | | Message | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 48 | 15 | 0 | 8 | P | u | b | l | i | s | h | 1 | X | X | . | X | X |

*Table 4.6: Publish Packet*

The full form of byte 0 is as follows

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | MQTT Control Packet type (3) | | | | DUP flag | QoS level | | Retain |
| | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

*Table 4.7: Byte 0 of Publish Packet*

For publish packet, it is Control packet type 3 hence, the 4$^{th}$ and 5$^{th}$ bit are 1. Here, bit 3 is termed as the "DUP Flag" which means duplication flag. It indicates whether the packet is sent for the first time or a re-delivery. In the instance where the packet will be redelivered is when the bit should be 1 otherwise it will always be 0.

Bit 1 and 2 are to represent QoS, which indicates the assurance level for delivery of an application message. The table below shows the definitions of the bits with regards to QoS settings.

| **QoS value** | **Bit 2** | **bit 1** | **Description** |
|---|---|---|---|
| 0 | 0 | 0 | At most once delivery |
| 1 | 0 | 1 | At least once delivery |
| 2 | 1 | 0 | Exactly once delivery |
| - | 1 | 1 | Reserved – must not be used |

*Table 4.8: QoS settings*

We have used the at most once delivery here.

Lastly, the 0 bit is used for "retain". If the flag is 0, the server will not store the message, and must not remove or replace any existing retained message. If the flag is 1, the server must store the message, so that it can be sent to any future subscribers of the same topic name. In the case where a QoS is 0, but the retain flag is 1, the server must discard any messages retained for that topic and stores the new QoS message as the new retained message.

The next byte like in the connect packet is for Remaining Length, and our publish packet size is 17 bytes, and since 2 have been used up here, so the remaining is set as 15 here.

In addition, the TLEN, which represents Topic Length, since our topics are “Publish1” for the first module and “Publish2” for the second, the length being of 8 bytes, we have used 8 here. After this, using the next 8 bytes (from 4 to 11 bytes) we will set the topic name and using the next 5 bytes (12 to 16 bytes) the message would be sent.

**The subscribe packet:**

The MCU will also need to be subscribed to specific topics to receive information from the server side. And to initiate subscription, the subscribe packet needs to be used. The full packet with the data we have sent is given below.

| SUB | RL | PKTID | | TLEN | | TOPIC | | | | | | | | QS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 130 | 13 | 0 | 1 | 0 | 8 | S | E | R | V | E | R | 0 | 1 | 0 |

*Table 4.9: Subscribe Packet*

The details of the first byte is given below

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | MQTT Control Packet type (8) | | | | Reserved | | | |
| | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

*Table 4.10: Byte 0 of Subscribe Packet*

The reserve bits are specific for the subscribe packet and the packet type for subscription is 8, and thus our input is as it is. The decimal value is 130, which was our input. This is preset in the protocol.

Just like the previous packets, the next byte is for Remaining Length. The total size of the packet will be 15 so we will set the RL field to 13 as 2 bytes have been used up already here.

Byte 4 and 5 are used for “TPLEN” meaning Topic Length, which like in publish packet, represents the length of the Topic. The name of our topic is “SERVER01” for equipment 1, and “SERVER02” for equipment 2 and since both are of length 8, so we set the TLEN as 8. So the next 8 bytes (from byte 6 to byte 13) will be used to send the topic name.

In the subscribe packet, we will also set the QoS level using the last byte, so we used the last byte (byte 14) to set it.

### 4.3.3 Microcontroller side code

As like during the communication during HTTP, the MCU needed to be coded to allow us to communicate to the broker, we will have used the MCU as both publisher and subscriber. To use the MQTT protocol, we had to use packets as described previously that the protocol uses.

### Main code: (in Appendix F)

1) We have again used the switch case structure, however, a bit unconventionally. We have used “go” as the variable that the switch case variable, and the cases defined initially. The variable “go” has been set to “TCPConnect”, “Nextgo” is set to “connectCIP”. So, to start off, the program starts from the case “TCPConnect”.

2) From the TCPConnect case, we will see the word “connecting” appear. Here the MCU would first use “AT+CIPSTART” at command to establish connection to CloudMQTT using the 14944 port (as provided by the broker). We will search for the word, “CONNECTOK” (if the connection is first time, this should be the reply from the ESP), and for "ALREADY CONNECTED" (if connection was established earlier) using the “search” function we have created. (will be discussed at the end of this chapter). If connection is successful, we should get either of the two messages and can move on to the next case. Otherwise the program will keep trying to connect and hence remain in this case.

Here the next case would be “ClosedOrNot” and this is going to be the most used, and important step of the code. The purpose of this case is simple but very important as this check whether connection is broken or not. Here, we would search for the word “CLOSED” for after closing of the connection and this word is sent back by the ESP module. So if this has occurred, means we need to reconnect hence we need to go back to “TCPConnect” case and so the value of “go” will be “TCPConnect”. If the value of “lastgo” is “SearchAndOutput” it means that the flow of the code has come here from the “SearchAndOutput” case (described later) and we need to go to “connectCIP” case after “TCPConnect”. If this did not occur it means that the connection is stable till now and we can move on to the next stage, which is the name of the case held by the variable “nextgo”. The value of “nextgo” was initialized as “connectCIP” in the beginning the so we will move to that case.

The case “connectCIP” is the initial step of sending the connection packet, using which we can start the communication. Using the AT command “AT+CIPSEND=46”, where 46 represents the length of the data that will be sent. The value of “nextgo” is set to “sendConnectPacket” , and go is set to “ClosedOrNot”. This will be the structure of every case where “nextgo” will be set to the case that it should move to, and “go” will always be set to “ClosedOrNot” to recheck connectivity. This rechecking is performed to ensure stability of the system. If the connection is stable, at the end of the “ClosedOrNot” case the value of “go” will be set to “nextgo”, that is the next case. Otherwise, “go” shall be set to “TCPConnect” to restart the connection.

3) In the “sendConnectPacket” case, we send the connection packet using the “connectSend()” function (described later). We have given it a time of 1 second to be sent and then after rechecking connection and finding it to be stable, the program moves to the “subscribeCIP” stage. With the connection packet sent, means we can now subscribe to

publish to the broker. We start off by subscribing first. So to subscribe we need to send the subscribe packet, and in this case, we send the length of the data to be received using the AT+CIPSEND command, which is 15 in this case.

After this, we move on to "sendSubscribePacket" case. Here the subscription packet will be sent using the "subscribeSend()" function (described later), and again wait for 1 second. We have now successfully subscribed to the topic, and will now be receiving data. After this, we move on to the next case, namely "searchAndOutput" case where we will search data, and take necessary action.

4) Till now, "Connecting" will be shown on the LCD, however, when we move onto "searchAndOutput" case this shall not be the case. Notice that in the previous case ("sendSubscribePacket" case) the variable "subdisplayFlag" has been set to 1 which indicates that we have sent the subscribe packet, and hence the device is now subscribed to the broker and to reflect this, "Subscribed" will be shown on the MCU. So after this, we will search for the keywords used to control the bulbs and thus used the search function. When we send the command to turn on the blubs we send "SERVER01ON1" and "SERVER01OFF1" when we want to turn off the bulbs which are the two words that are searched for ("SERVER01ON2" and "SERVER01OFF2" for the other equipment). Upon finding "SERVER01ON1", PORTC.0 will be made high and vice versa. After this case itself runs 100 times, defined in the "If" statement in line 74, we move onto publishing data. A 10ms delay is provided and with the case running 100 times means that we publish data every 1 second. So after every second "nextgo" would be set to "publishCIP" (the next stage) and the program moves to the "publishCIP" case.

5) The last two cases are for publishing data to the broker. In “publishCIP” case, we set the value of “nextgo” to “sendPublishPacket” and use the “getCurrent” function to calculate the value of the current. The function “publishValueUpdate(cur)” updates the publish packet that is to be sent (described later). Lastly, the number of bytes to be sent (17 in this case) is sent using “AT+CIPSEND” command.

   With connection still being viable, we move onto the case “sendPublishPacket”. Its purpose is simply to send the publishing packer using the “publishSend()” function. During this time, along with “Subscribed”, we would see the words “Publishing” on the LCD letting us that it is now a data has been published to the database. With this, publishing would be complete, and after this, ideally (considering the connection to still be stable) we would move back to the search “searchAndOutput”, so continually search of there is any controlling commands (turning the light on or off) coming from server.

## Library Files: (in Appendix G)

Most of the library functions, that we have used have already been discussed in the previous chapter (HTTP MCU code), so, here we will only discuss the new functions created for MQTT protocol.

1) The “connectSend()”, “publishSend()”, and “subscribeSend()” functions each sends their respective array. For connecting, “connectSend()” function sends the “connect” array which is the connect packet. Similarly “publishSend()” function sends the publish array and “subscribeSend()” sends the subscribe array. Here the “connect” and the “subscribe” packets will remain the constant, however for publish, the value of current will have to be updated (described later).

```
unsigned char connect[46]={16,44,0,6,'M','Q','I','s','d','p',3,194,0,60,0,6,'M','I','C','R','O','1'
           ,0,8,'n','c','i','h','n','t','a','k',0,12,'C','O','7','A','c','c','S','W','r','z','Z','F'};
unsigned char publish[17]= {48,15,0,6,'d','e','v','i','c','e','E','1','0','0','.','0','0'};
unsigned char subscribe[15]={130,13,0,1,0,8,'S','E','R','V','E','R','0','1',0};
```

*Figure 4.5: Packets in MCU code*

In the connectSend() function, we first cleared the buffer, which was necessary to avoid data corruption or overwriting data and afterwards, loading the connect array into UDR which would send the data. Here the while loop goes on for 46 times, for the connect packet of length 46. The “publishSend” and “subscribeSend” functions work exactly in the same manner with the difference in the number of times the while loop will run which will be 17 and 15 for “publishSend” and “subscribeSend” functions respectively.

2) The “publishValueUpdate(float value)” function is like the “apiKeyValueUpdate()” function and it updates the new current value into the publish array, and works exactly in the same fashion as previously discussed (in the “apiKeyValueUpdate()” function in chapter 4.2.2).

3) **Search Function**

When data arrives into MCU it arrives in the receive buffer of the microcontroller UART. Let us assume that we have a receive buffer named “Receive” of size 11 that means Receive[11];

If information called “Hello world” arrives into our empty receive buffer it will look like something shown in below :

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| H | e | l | l | o | | W | o | r | l | d |

If we want to search a word like “world” it will work as shown below.

| | 0 | Receive[11] | | | | | | | | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | H | e | l | l | o | | W | o | r | l | d |
| Step1 | W | o | r | l | d | | | | | | |
| Step2 | | W | o | r | l | d | | | | | |
| Step3 | | | W | o | r | l | d | | | | |
| Step4 | | | | W | o | r | l | d | | | |
| Step5 | | | | | W | o | r | l | d | | |
| Step6 | | | | | | W | o | r | l | d | |
| Step7 | | | | | | Match Found | W | o | r | l | d |

When it finds the match it returns “1” otherwise “0” .This helps us to enter and exit from any condition and make decisions. That is how program flow can be controlled of the state machine with the help of search function.

When “Receive” buffer receives data it places the data into next array variable by replacing the existing. When array reaches its maximum value, it starts from the beginning and continue the process. That means it stores data by circulating. This mechanism of data storing creates a situation like below. If we want to search “Hello” from the array below, we will not be able to find it following the old searching method.

| o | | W | o | r | l | d | H | e | l | l |
|---|---|---|---|---|---|---|---|---|---|---|

That’s why we have introduced a circular array searching mechanism. It is nothing but adding the same “Receive” buffer side by side twice into a new array and search from the new array. The mechanism that it follows is given below:

1. Elements of new array (0-21) can hold 22 elements. It has been constructed by putting two “Receive” buffer side by side.
2. Our search keyword is “Hello”, it will search step by step like before.
3. Match has been found in step 9.

| | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | Receive[11] | | | | | | | | | 10 | 0 | Receive[11] | | | | | | | | 10 | |
| | 0 | NewArray[22] | | | | | | | | | | | | | | | | | | | 21 | |
| | o | | W | o | r | l | d | H | e | l | l | o | | W | o | r | l | d | H | e | l | l |
| Step-1 | H | e | l | l | o | | | | | | | | | | | | | | | | | |
| Step-2 | | H | e | l | l | o | | | | | | | | | | | | | | | | |
| Step-3 | | | H | e | l | l | o | | | | | | | | | | | | | | | |
| Step-4 | | | | H | e | l | l | o | | | | | | | | | | | | | | |
| ---------- | ---------- ---------- | | | | | | | | | | | | | | | | | | | | | |
| Step-9 | | | | | | | | H | e | l | l | o | Match Found | | | | | | | | | |

Here the concept of “NewArray” is theoretical. In reality, it has been developed with controlled program and searching the “Receive” buffer twice. This is how we can resolve the issue and make our search function work properly.

In “clear section” of the search function everything remains as it is, but when clear is “1” (It can be either “1”or “0”). It clears the search key from “Receive” array. As example if search “Hello” when clear “1”

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | W | o | r | l | d | | | | |

“Hello” has been removed from the “Receive” buffer. Thus “Search with clear” has many great applications regarding output and control of the state machine.

# Chapter 5

# Server

## 5.1 000webhost

In order to view real time data from sensor, we need a live server with a database. To develop this, initially we had created a local server for ourselves to test out the concept before going remote. For this we are using XAMPP which is an Apache distribution containing Perl, PHP and MariaDB. XAMPP acts as a local server, allowing us to create database with phpMyadmin and operate the MySQL database [13]. After testing everything on local host we uploaded the PHP files on 000webhost which is a free domain hosting website, hence getting space for free. 000webhost also supports to PHP and MySQL database. Thus using this, we had created our very own remote server.

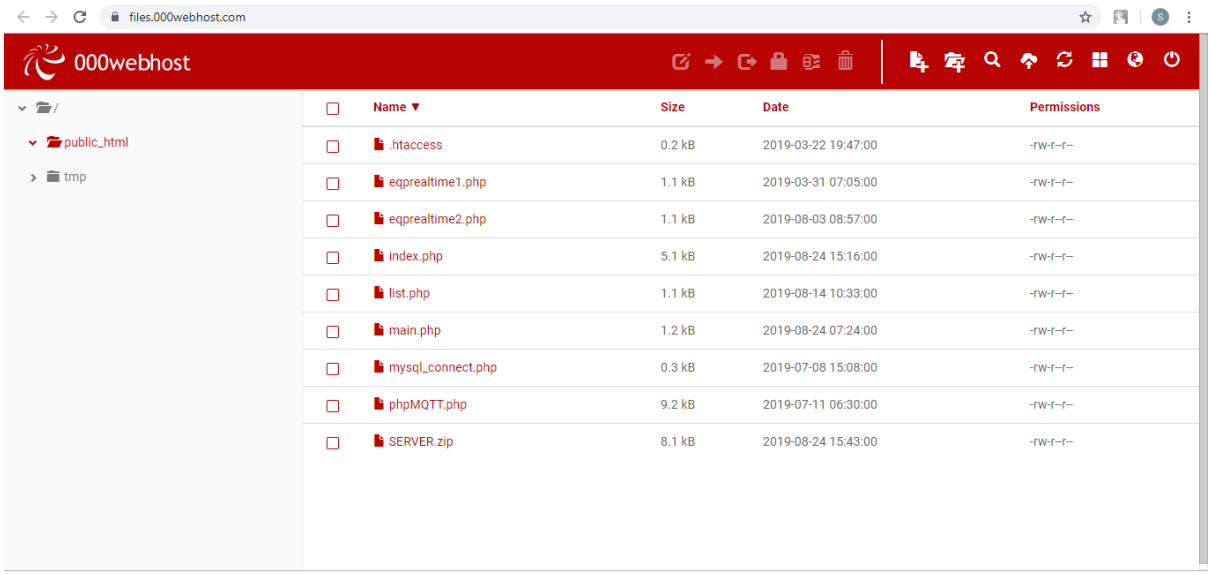



*Figure 5.1: Php files stored in 000webhost*

## 5.2 CloudMQTT Broker

To create the system of remote communication with MQTT protocol we were in need of an online broker with real IP. CloudMQTT is such an online broker, which was available for us to use which assures us with high probability of message transfer. It is a mosquito-based broker, which means that our work now is not only limited to CloudMQTT itself, rather we can use it on any mosquito-based broker. CloudMQTT can operate in 3 modes which includes fire-and forget, exactly once and at least once [14]. We can even use websocket client of CloudMQTT to view publish messages on different topics directly, without the need of any other devices. This enables us to do test runs and whether our broker is connected with other devices.

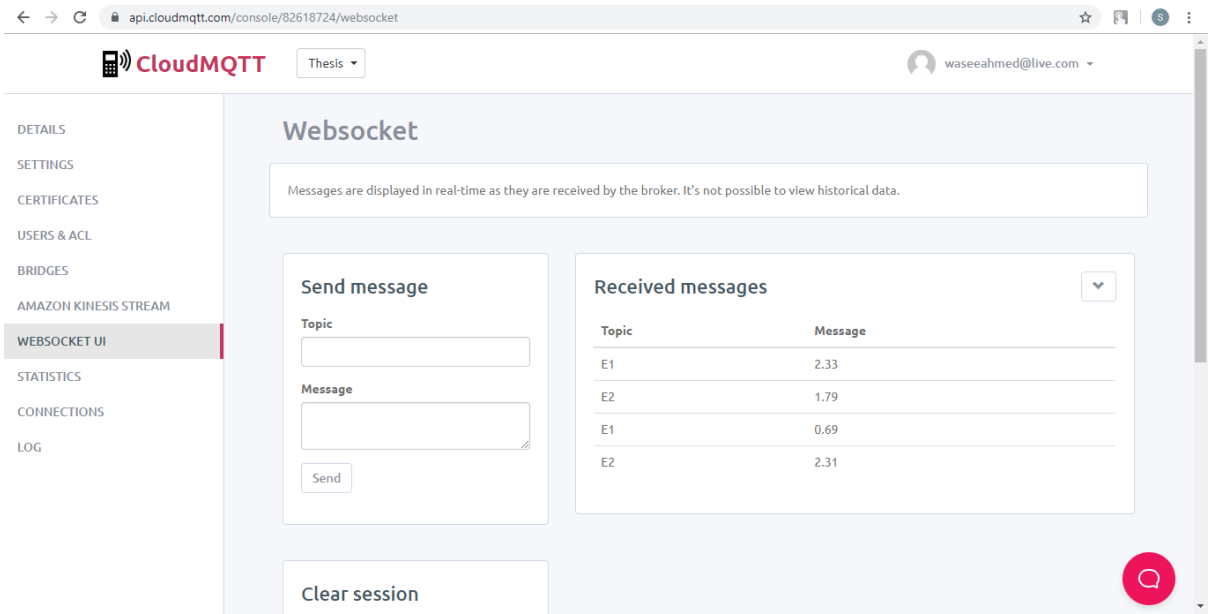



*Figure 5.2: CloudMQTT WebSocket*

## 5.3 Website interface

Since we have an online server, we have created a webpage that will allow us to handle things as we with graphical user interface (GUI) and basically will be the introductory page on our server. This page contains various utilities such as, the manual override for the bulbs, we have the option of turning on or off any of the two equipment using the buttons available. We can view the data of the two devices with the buttons “Equipment1” or “Equipment2” this would show the all of the readings of current that arrived and is stored on the database of the device. In addition, we have added a search option to allow us to see particulate data. We can directly search data by either ranges of “current” of by “date and time”. The website can not only showing data but also print it as well using the print option. The final page looks as follows:

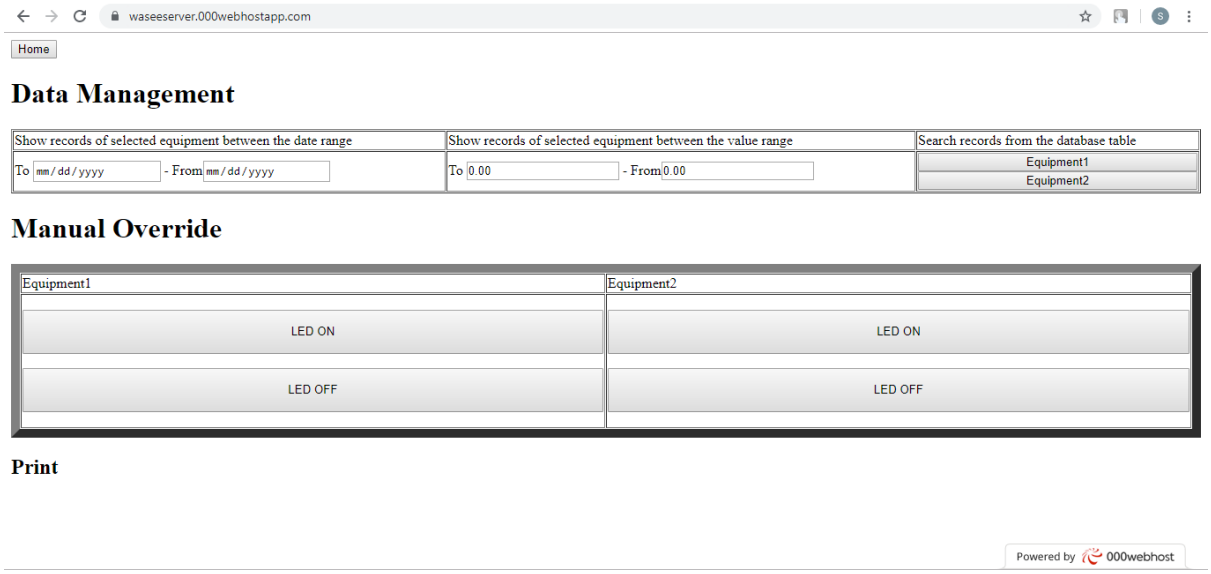



*Figure 5.3: Interface of the website*

## 5.4 Server side code for HTTP protocol (in Appendix H)

This section will cover the coding and the concepts behind the server. The entire system was written in PHP language and the files stored on 000webhost as mentioned earlier. It begins with the index page, and the relevant functions, or other PHP files explained later.

1) Index:

In the index page, the first portion is HTML coding which was used to create the interface, with title, refresh button and "<input ….. />" holds the information of the button, with the writing on top of the button given through "value" and the information it passes given by "name". Such arrangement is created for all next buttons. The picture below portrays all the buttons used.

```
<input type="submit" style="width: 100%" name="Equipment1" value="Equipment1"/>
<input type="submit" style="width: 100%" name="Equipment2" value="Equipment2"/>
</td>
<td>
            <tr>
<td>
<p><input type="submit" style="height: 50px; width: 100%" name="LED1_ON" value="Equipment 1 ON" /></p>
<p><input type="submit" style="height: 50px; width: 100%" name="LED1_OFF" value="Equipment 1 OFF"
    /></p>
</td>
<td>
<p><input type="submit" style="height: 50px; width: 100%" name="LED2_ON" value="Equipment 2 ON" /></p>
<p><input type="submit" style="height: 50px; width: 100%" name="LED2_OFF" value="Equipment 2 OFF"
    /></p>
```

*Figure 5.4: Code of buttons in the website*

2) Data Viewing:

The "Equipment1" and "Equipment2" button's job is to show the values of current, with the time and date of entry. So when the button is pressed, for example "Equipment1" information is passed on. This is then detected, and the function "showAllData ($tableName, $dataBaseTableName, $curValtoStr,$curValfromStr) " is called.

3) The function showAllData:

The function takes input variable of the name of the table (the heading that will appear above the table), the database table name (source of the data), cur value to and cur value from represents the ranges of data to be shown. The rest of it just shows the data as we please with the table lines, the column names, etc. which basically web is designing. The output is as follows:

| *All Equipment 1 Data* | | | |
|---|---|---|---|
| **SL No** | **Current(Amps)** | **Received Time** | **Received Date** |
| 2680 | 0.42 | 01:00:38:pm | 15-Jul-2019 |
| 2679 | 0.43 | 01:00:33:pm | 15-Jul-2019 |
| 2678 | 0.91 | 01:00:16:pm | 15-Jul-2019 |
| 2677 | 0.9 | 01:00:11:pm | 15-Jul-2019 |
| 2676 | 0.9 | 01:00:06:pm | 15-Jul-2019 |
| 2675 | 0.9 | 01:00:00:pm | 15-Jul-2019 |
| 2674 | 0.89 | 12:59:55:pm | 15-Jul-2019 |
| 2673 | 0.9 | 12:59:49:pm | 15-Jul-2019 |
| 2672 | 0.91 | 12:59:44:pm | 15-Jul-2019 |
| 2671 | 0.9 | 12:59:39:pm | 15-Jul-2019 |
| 2670 | 0.9 | 12:59:33:pm | 15-Jul-2019 |
| 2669 | 0.89 | 12:59:28:pm | 15-Jul-2019 |
| 2668 | 0.9 | 12:59:23:pm | 15-Jul-2019 |
| 2667 | 0.9 | 12:59:17:pm | 15-Jul-2019 |
| 2666 | 0.9 | 12:59:12:pm | 15-Jul-2019 |

*Figure 5.5: Data is shown on our webpage*

4) Server-side Control:

With the part regarding observation of data done, the second task was to control the bulbs. Four buttons have been programed to perform this operation. For example, when “LED ON” of equipment 1 is pressed, “LED1_ON” data is passed which is picked up by “isset($_GET['LED_ON'])”, So whenever this is called, it updates the “command” database. From here data is sent to the microcontroller following the same link that is used when data enters (data sent by using echo from the message coming in). We have done this through a database to ensure faster and smoother communication without any data loss.

Similar technique has been used for the other buttons, with the variables “LED1_ON”, and “eqp1ON” changing as per requirement.

5) Data Entry:

As like others, there is no button for data entry as data will come from other sources. In our case, data will be sent from the microcontroller by using the API key and data extracted from it. The API key that MCU sends looks something as follows:

```
"GET /add1.php\?cur=00.00 HTTP/1.1\r\nHost: waseeserver.000webhostapp.com\r\n\r\n";
```

*Figure 5.6: API key coming from the MCU*

As we can see there is a variable “cur”, which triggers the if segment shown above. Using the “$query” and “INSERT INTO”, data is made to enter the database.

6) Database connection:

The “require_once('mysql_connect.php')” command is needed to be used every time database interaction is needed. Its job is to connect to the MySQL database to extract or provide inputs. It has been made in a separate PHP file so that it is accessible by several files which we required. To connect, we simply used “@mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME)” which is a built in function with the required parameters that is host, user name, password, database name respectively which was defined initially .

## 5.5 Server side code for MQTT protocol (in Appendix I)

The server side for the MQTT protocol mostly the same as the server side for HTTP as explained earlier like with the database creation, database linkup, the interface of the webpage etc. Only a few adjustments were needed to establish MQTT protocol here and in this section we shall only the parts that were changed in the communication system. That is, sending and receiving data. We here used the library file “phpMQTT.php” to publish (send information from the server) and subscribe (to receive data).

1) Publishing data:

Since the server has to be able to control the bulbs, it needs to be able to send data, to the MCU. Thus sending, that is publishing data to the microcontroller. If we carry on with the example form the HTTP side, upon hitting the “LED ON” button of equipment 1, LED1_ON is detected, hence the if statement is activated. Line number 170 creates an object of $MQTT, with host, port number and a client name. Afterwards we connect to CloudMQTT by providing user name and password, in the 3rd and 4th parameter. After a successful connection, it publishes data using the “publish ("SERVER01", "ON1", 0)” where, “SERVER01” is the topic and “ON1” is the message. With publishing done, we close the connection afterwards to complete transmission. This sends data to CloudMQTT, and the subscribed device (MCU in our case) will receive the message with the topic (we have already seen how the microcontroller uses the message to control the bulbs). The other buttons use the same technique to send data.

2) Data Entry:

Data reception is made to work through subscription to topics, thus, the subscribed device will always receive data from when the broker gets new data. The connection establishment is identical to when we were publishing data that is an object of $mqtt is created with host, port number and a client name. A subscription connect message is sent using user name and password. After that, we need to subscribe to a topic. For this to happen, we used the $mqtt object and used the “subscribe (“array”, 0)” function of the “phpMQTT.php” library. For the array part, it is quite a special array, as shown in line 29. Here the topic we will be subscribing to is “device”, that is the topic name that we are subscribing to for both the devices, with the other parameters “qos” being 0 which is constant, and a function “procmsg” (explained later).

With our server now being subscribed, it is now possible for data to enter the server. The broker will automatically send data to the server, we will however need to detect it and to do this, a while loop is run, which runs to detect data. We will run the loop for 58 seconds by using "!hasTimedout()" function (explained later). Upon entering the loop, the $mqtt object is run through the "proc()" function of the library, which detects new data, so for 58 seconds, we will be continuously searching for new data. If new data is found, the "procmsg" function is called which changes the value of the flag to 1 and so enters the "if segment", where data is stored to MySQL. In the "procmsg" function, values for "$eqp" and "$cur" will be set. "$eqp" represents which equipment has sent the data, and "$cur" being the value of the current. So, by setting up a "if" statement with "$eqp", we distributed the value of current in the correct database. Thus, topic for data entry became a single entity. So as long as the page is running and the program is running, we will get new data stored in our database, which will essentially become one of the biggest problems, the page needs to be running on some web browser, which continually be needed to get refreshed after getting timed out.

3) Other functions:

As mentioned earlier, the "procmsg" function is called as soon as data is sent by the broker. This function is declared in the array when topic was declared. The when "$mqtt->proc()" runs, and new data is detected, this function is called and the topic and the message is passed through. Upon entering, the flag bit is turned to 1 to indicate that data has entered, and hence get data into the database. The message that arrives here is from the MCU, and the message is constructed to our benefit. For example if equipment 1 has the value of current 12.09A, the published message would be "E112.09". In the server end, the message is stored in an array and we have programmed it so that from the first two positions of the array will indicate the equipment and the others the current. Since now we know what equipment it is from and we keep it in the variable "$eqp", and the last 5 characters were stored it in variable "$cur".

using an “if” statement upon “$eqp”, and comparing whether the value of “$eqp” is “E1” or “E2”, we make the value of current enter their respective tables.

The “hasTimedout()” function returns 1 when it is done, otherwise 0.  when the time is above 58, and thus the while loop as mentioned earlier exits. Since in the while loop (in the previous code) its “!hasTimeout()” so when it gets 1 which is false for the loop, it will break.

## 5.6 Cronjob

As we have already seen we needed “main.php” to be running for data to enter the database. However, this is not feasible for a remote server as we cannot keep running a webpage all the time.  Cronjob helped in solving our problem, because what we needed was running the page in the background. It is essentially a software (however, in our case it is a website) that can run some web page or a website at scheduled time. To set it up we first had to open an account there, and then simply add the URL of the page that needed to be run automatically and set up the time interval after which it shall be called. Since what we needed was continuous communication, we set it as low as possible which was 1 min. The reason for running the loop for 58 seconds was, since cron job ran the program every minute, if we run the program itself of 58 seconds, that is collecting data for the mentioned time, we would get data all the time (the 2 sec gap was left intentionally considering loading time etc.).

# Chapter 6

# Mobile Application

## 6.1 Creation of pahara app

The “pahara” app is created for constant monitoring of data from 000webhost (server) on our smartphones. Due to the popularity of smartphones, many people uses it to keep pace with the modern technology. As smartphones are available in almost every household, we have introduced a mobile app so that we can easily avail the opportunity of tracking our appliances from anywhere. For building the app, we have used android studio.

Android Studio is an integrated development environment (IDE) for Google's Android operating system which is built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is accessible for download in different operating systems such as Windows, Mac OS and Linux.

In addition, Android is a mobile operating system developed by Google, based on Linux kernel. It is used for smart phones, tablet, computer and TV. Our primary focus is on smartphones. There are different version of android along with code name, version number and API level. In our case, our minSdkVersion is 23 which means it can be installed on any version of android starting from Marshmallow. Moreover, the targetSdkVersion for our application is 28.

To begin with, a project needs to be created and the name of Application is given “pahara”. The following procedure needs to be followed for creation of the project.

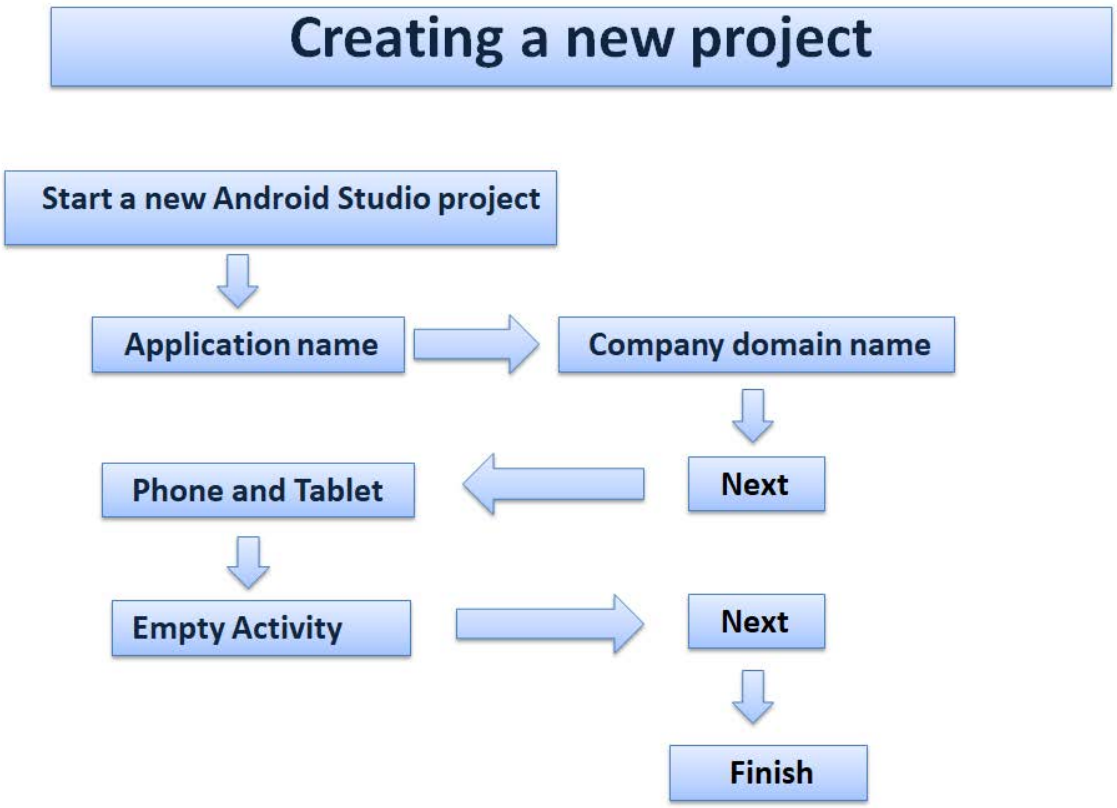



*Figure 6.1: Creation of project*

To run an Android application, a hardware device or virtual device is needed. Virtual device is a configuration that defines the features of an Android phone, tablet, Wear OS, Android TV or Automotive OS device we want to replicate in the Android Emulator. The following approach needs to be followed for constructing an Android virtual device.

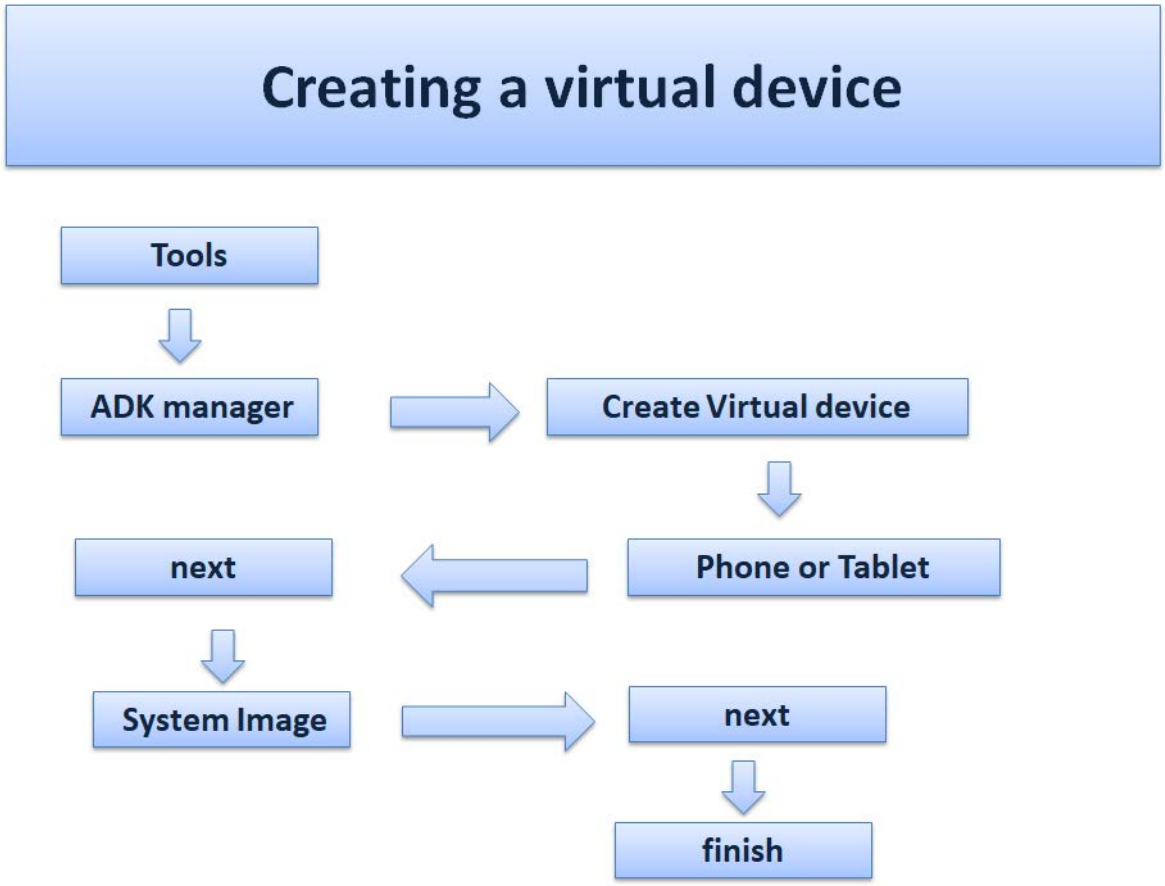



*Figure 6.2: Constructing of an Android virtual device*

A smartphone can also be used to view and observe the properties of an Android application. For this, we have to connect our smartphone with the laptop containing Android studio software with the help of a USB cable.

The procedure for connection is given below

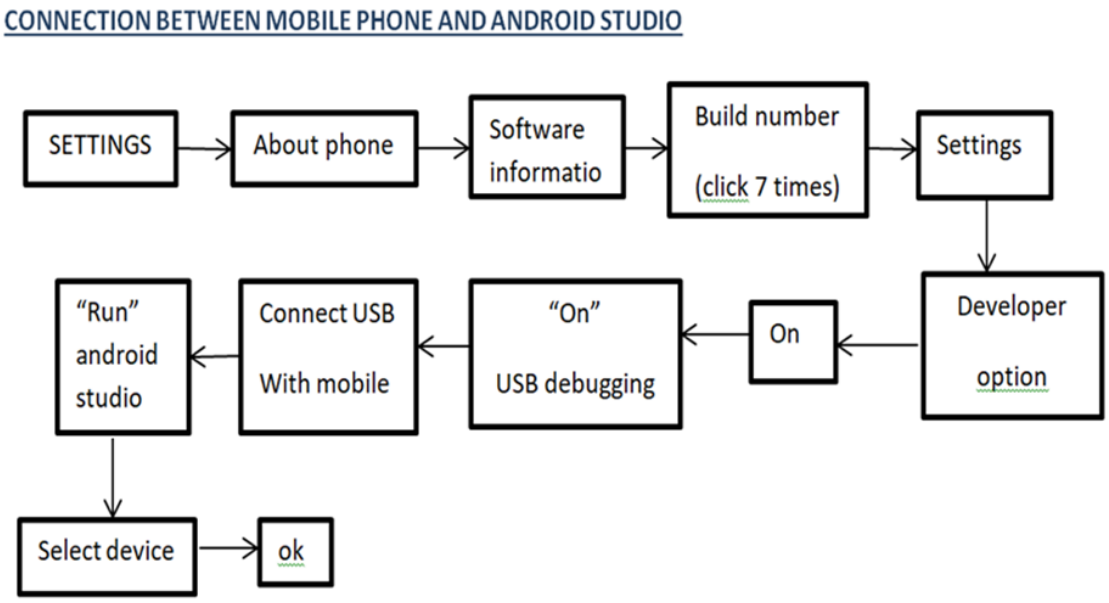



*Figure 6.3: Configuring smartphone for android application*

## 6.2 Android file structure and different widgets:

**Types of files:**

**1) Manifest**

- All the activity will be there in manifest file
- When an application runs, it starts from manifest file
- There is a launcher activity from where the activity begins
- Program starts with that activity that has intent filter on it

**2) Java file:**

Android Studio makes a Java document with skeleton code that can be altered.

**3) Res**

We get the following files in res:

- **Drawable**: this is the directory where all the drawable resource and images are stored.
- **Layout**: xml file will be there in layout, we can design the UI here.
- **Mipmap**: images of different size will be present here. Depending on the size of the device, images are set.
- **Values**: there are colors, strings and styles present here
- **Colors**: different colors are set using key value pair.
- **String**: strings are stored in this file using key value pair. We can declare different string.xml file for localization purpose depending on the country code.
- **Styles**: this is the file where different style for different views are declared. we can use style for the Views in XML that specified in the layout.

**Program execution:**

Manifest ➡ java ➡ xml

**View group:**

A ViewGroup is a special view that can contain other views (called children.) The view group is the base class for layouts and views containers. For example, Linear Layout,Relative Layout, Table Layout, Frame Layout, Web View, List View, Grid View. We have used LinearLayout in our project.

The LinearLayout determine whether their children will appear horizontally or vertically. The root LinearLayout is vertical and its child LinearLayout is horizontal. The order in which children are defined determines the order in which they appear on screen. In a vertical

LinearLayout, the first child defined will appear topmost. In horizontal LinearLayout, the first child defined will be leftmost.

**View:**

- It does not contain other views.
- Widget Attributes
- What is displayed in a mobile screen is widget

The android: layout_width and android: layout_height attributes are required for almost every type of widgets. They are typically set to either wrap_content or match_parent:

- wrap_content: view will be as big as its content requires.
- match_parent: view will be as big as its parent

For the root LinearLayout, the value of both height and width attributes is match_parent .The other widgets have their widths and heights set to wrap_content.

- sp stands for scale-independent pixels. Sp is used for text size because but it is scaled by the user's font size preference.
- dp stands for density-independent pixels. This attribute tells the widget to add specific amount of space to its contents when determining its size.
- Padding: using this attribute the text goes away from the left corner of the screen.
- Margin: The rectangle pushes its surrounding contents from itself by the dimension specified in the margin attribute. Here, the surrounding content will be the screen of the mobile

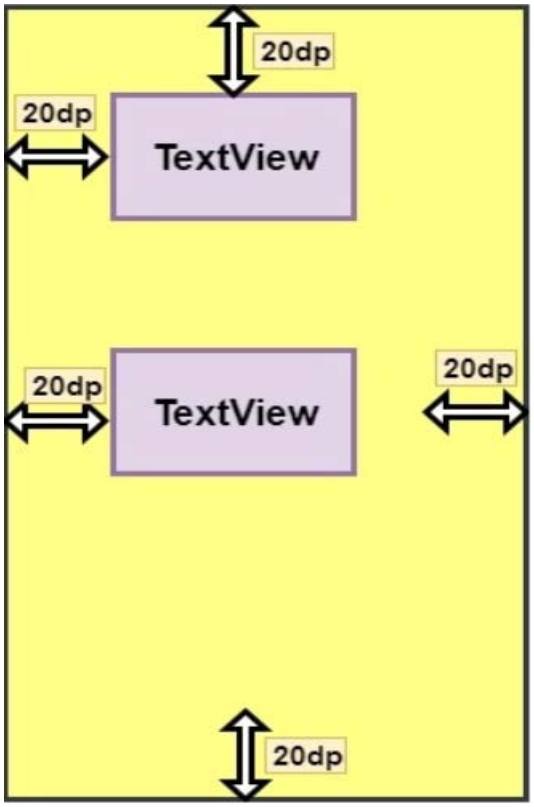



*Figure 6.4: Padding*

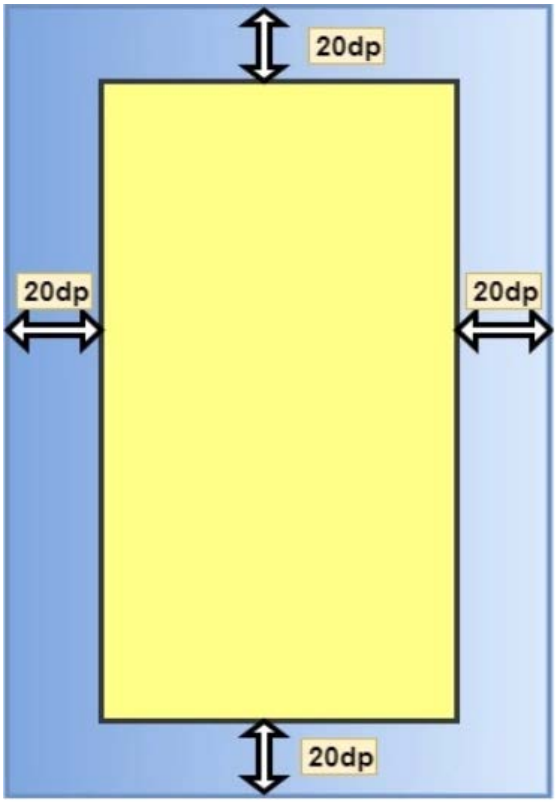



*Figure 6.5: Margin*

## 6.3 Code Explanation: (Appendix J)

**1) Manifest File in our project:**

Manifest file handles all the activity and has a description of each activity. There are about three activities in our project. The <intent-filter> is only present at SplashScreenActivity determines which activity will be launched first. That is, when our application will be opened, we will see this activity at first followed by ApplianceListActivity and ApplianceDetailsActivity.

**2) Creation of Splash Screen**

When the app is going to be starting a Splash screen will appear for 500ms.

A java class SplashScreenActivity is created which is the child class of AppCompatActivity. AppCompatActivity is a subclass of Android's Activity class that provides compatibility

support for older version of android. Moreover, it contains important methods including the commonly used methods named onCreate(Bundle savedInstanceState) and findViewById . The onCreate(Bundle) method is called when an instance of the activity subclass is created. When an activity is created, it needs a user interface to manage. To get the activity its user interface the Activity method setContentView is used. This method inflates a layout and puts it on screen. When a layout is inflated, each widget in the layout is instantiated as defined by its attributes.

An integer type variable named ***SPLASH_DISPLAY_LENGTH*** is used to initiate the time of display of splash screen. The activity starts with splash screen. This is because the main intent start activity is included here. An object of Handler class is created. A handler enables other background thread to communicate with the UI thread. This is helpful for android as it does not allow other threads to interact directly with the thread of the UI. For stopping the backward navigation of splash screen from ApplianceListActivity class, finish() method is used.

**activity_splash_screen.xml file:**

In this file, the UI of splash screen is created. The background color is set to magenta. A text “Welcome to Pahara App” is added in <TextView/>. Along with that, the size is made to 26sp and the color of the text is white. The text style is made bold.

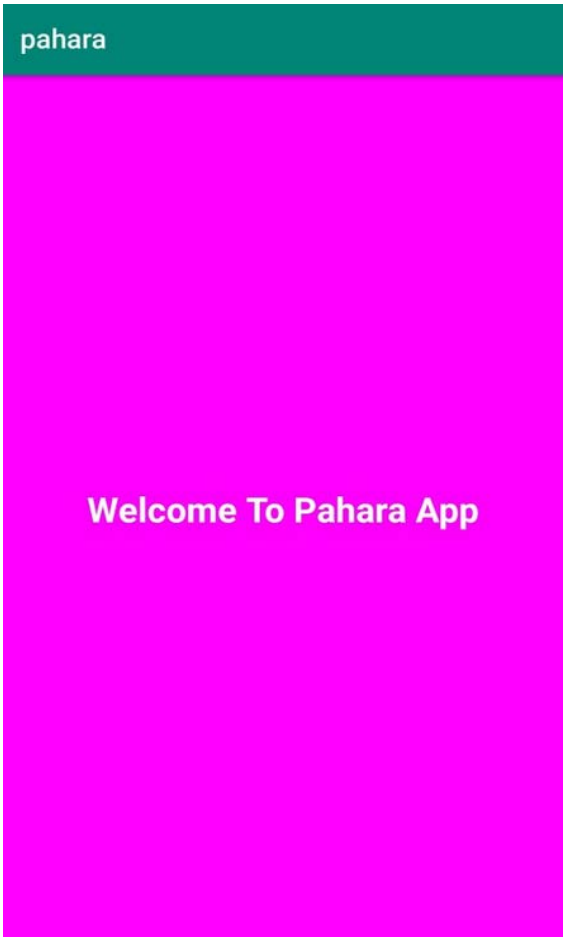



*Figure 6.6: Splash Screen*

## 3) Creation of activity_appliances_list

**Xml file:**

The UI of the activity represents that two buttons are created respectively by using the button class in android studio. To execute an action, this user interface component can be taped or clicked. In the button, text “APPLIANCE 1” is inserted. Similarly, for second button “APPLIANCE 2” is added. To retrieve information later with View.findViewById android:id is used to provide an identifier name for this perspective. This must be a resource reference which is usually set to create a new ID resource using the @+ syntax. For example: @+id/appliance_one_button which enables us to findViewById(R.id.my I d) to retrieve the perspective later.

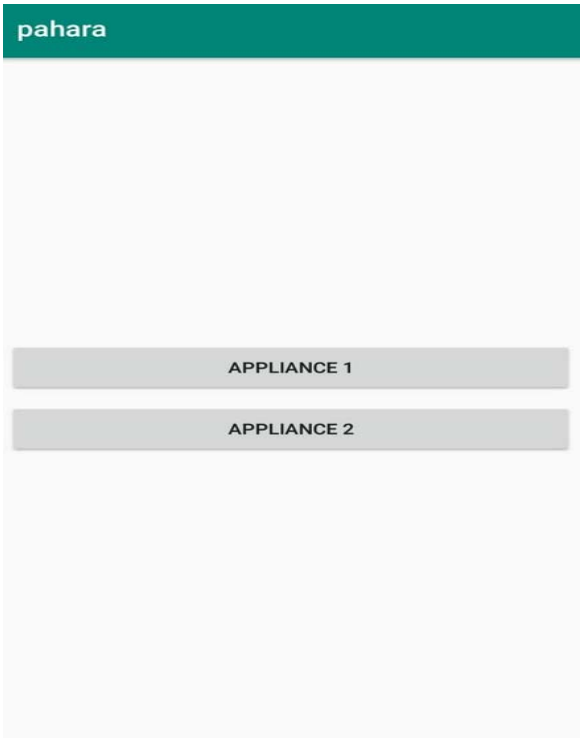



*Figure 6.7: Appliance list*

**Java file of Appliance list Activity:**

The android SDK comes with a listener interface named View. OnClickListener that provides an event to be listened whenever button is pressed.

Changing from one activity to another:

Intent class helps to navigate from one activity to another activity. An object of intent class is created and, in its constructor, the current activity should be added followed by the activity desired activity it has to progress. In this case the current activity is ApplianceListActivity and the activity it will go is ApplianceDetailsActivity. Data is transferred when moving from ApplianceListActivity to ApplianceDetailsActivity by using putExtra method. The putExtra method inputs two parameters which are a key and a value. The ApplianceDetailsActivity receives it by creating a variable of Bundle class. The startActivity method takes object of intent class as parameter.

**4) Construction of Appliance Information class:**

This is a model class. As we are going to show date, time and value of current we have declared this class with the following properties date, time and current with a constructor and corresponding setter and getter methods. When we will get the response from the server (000webhost) we will map the response with is ApplicanceInformation class. The date, time and value of current are initialized as String and setter getter methods are used to successfully place the values of date, time and current and retrieve that information after connecting to 000webhost.

**5) Designing App class:**

For network call using Volley, we need to prepare a layer. Volley is an HTTP library that facilitates and, most importantly, quicker networking for Android apps. App class makes network call by preparing itself first. RequestQueue is there in the volley library which enables the entire request to come in FIFO format and get executed. The method getInstance() return objects of class App. In addition, addToRequestQueue method can send String request, JSONArray request or JSON object request due to the usage of generics.

**6) Building ApplianceDetailsActivity class:**

Android offers a widget that implements the swipe-to-refresh design pattern, enabling the user to activate a vertical swipe update. This is done with the SwipeRefreshLayout widget that detects the vertical swipe, shows a unique progress bar and triggers callback methods in the app. The method onRefresh() is used when a gesture of swipe triggers refreshment.

Several datas will be received from 000Webhost. For this reason, we have created ArrayList named items. An object of adapter class is created as adapter converts data so that it can be shown in recyclerView.  An object of class RecyclerView sets layoutManager for showing

data. The method named setAdapter is used for making connection with recyclerView and adapter.

A method named fetchArray is used to extract the data from 000Webhost. It contains two variables of String type named tag and URL. To execute the request, a progress dialog is used. As long as the request is executing, the loader will not be cancelled. A String request of post type is created. If the request is successful, it will receive response of String type. From the String, an object of JSONArray class is created every time the loop runs. The JSON object contains current, time and date which is extracted and put in object of type ApplianceInformation. The object is then passed to the adapter. Furthermore, after processing of data, refreshing of datas gets cancelled. When new data are added, the adapter gets its information by calling method notifyDataSetChanged().

On the other hand, when there is an error, the progress dialog is hidden. Map is of String type and request is going as parameter. The valueTo and valueFrom are set to 0.0 as set in the web. A method named getBodyContentType() is used to know the type of the available content and is form-urlencoded type same as that of web. Consequently, the request is added in the getInstance() method of app class.

**7) Designing ApplianceAdapter class:**

In Android, Adapter is a bridge between the element of the UI and the source of information that helps us fill in the element of the UI. It holds the information and sends the information to a perspective of the adapter, and then view can take the information from the perspective of the adapter and display the information on various perspectives such as ListView, GridView, and Spinner. We can use the base adapter or custom adapters for further customization in Views.

We have created an adapter named ApplianceAdapter which is extending from RecyclerView.Adapter<RecyclerView.ViewHolder> which takes a RecyclerView.ViewHolder as it is of generic type. As we extended RecyclerView.Adapter we need to provide implementation of few methods.

The method onCreateViewHolder(ViewGroup parent, int viewType) is used to provide the layout that is going to be inflated from an XML layout file and will be shown in the UI. Furthermore, from the implementation of the onCreateViewHolder(@NonNull ViewGroup viewGroup, int i) we are returning return new ApplianceAdapterViewHolder(v) which is then passed to the onBindViewHolder(@NonNull RecyclerView.ViewHolder viewHolder, int position)

The method onBindViewHolder(VH holder, int position) is called for every single item in the RecyclerView. This method binds the data with the RecyclerView.ViewHolder.

In addition, the method getItemCount() returns the number of items to be shown in the RecyclerView. Moreover, the method addItem(ApplianceInformation appliance) can not be overridden. We have used this method to add items in the recycler view whenever we want.

Whenever we are extending RecyclerView.Adapter<RecyclerView.ViewHolder> we need to provide an implementation of RecyclerView.ViewHolder. So we have created an inner class like this.

**8) Conversion of 000Webhost List File to JSON format:**

JSON is brief for JavaScript Object Notation and is an easy-to-access way to store data. To retrieve complex data from the server end to android application JSON is essential. The information in the server end is converted to JSON format and in the client side we have to convert it into java. We have used json_encode converts PHP value to json value.


*Figure 6.8: Pahara App for ease of monitoring data.*

# Chapter 7

# Results

## 7.1 ACS 712

**Proteus simulation:**

The Proteus simulation is shown for three light bulbs where they are connected in parallel, which allowed us to change the value of current as we would in the practical scenario. This helps us to test out the code before practical implementation. Readings of ammeter are compared to microcontroller in Proteus to observe if our calculation method is correct. We checked output voltage of ACS and of the current using oscilloscope. All of this helped us to give a rough idea before hardware implementation. The figure below shows the Proteus diagram of the setup.

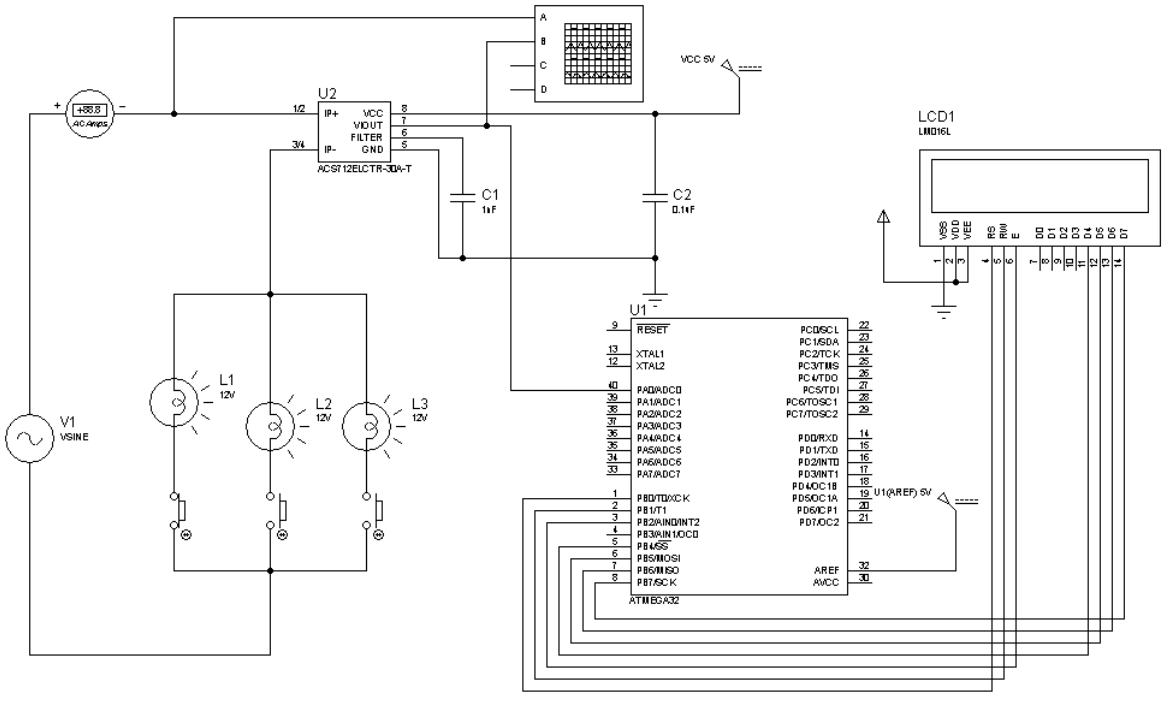



*Figure 7.1: Proteus diagram of ACS 712 with Atmega32*

The simulation shows us that the output from the ACS should be a proportional. However, a varying DC voltage that matches with the frequency of the current flows through the ACS712, with the Voltage value significantly smaller than the input. For now we have the idea of how the sensor behaves, we went on to test the device practically to compare.

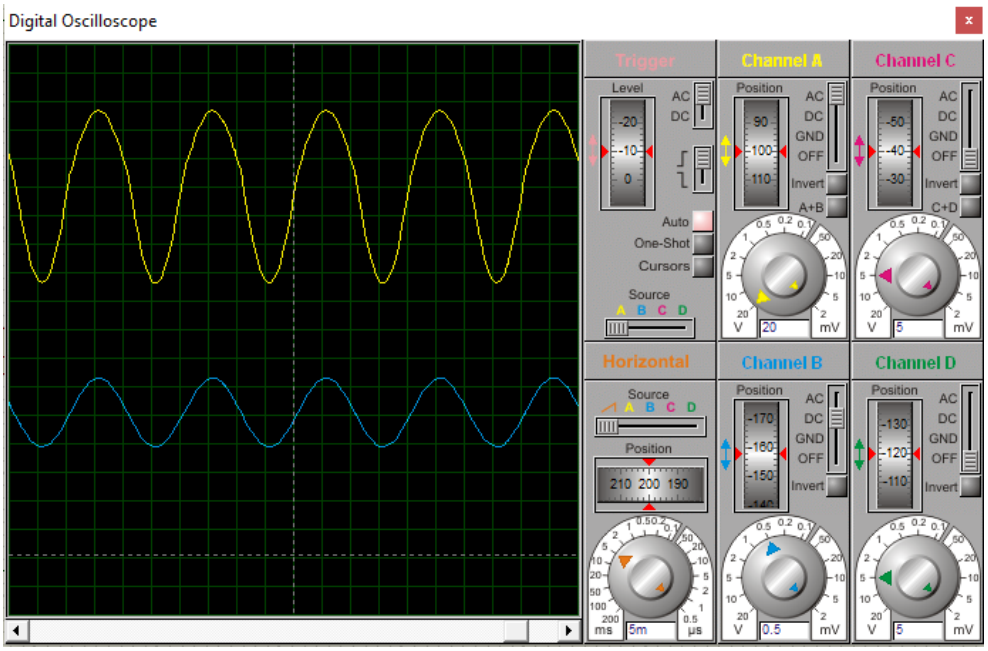



*Figure 7.2: Oscilloscope graph in Proteus*

**Hardware implementation**

To test the sensor, we used several loads to vary current through the sensor, and using the oscilloscope, we tried to observe the output of the ACS712 sensor. This was done to check whether sensor was giving consistent readings with regards change of current flow. Although this was provided in the datasheet, we retested it to ensure if it works well in our conditions or not.

Figure 7.2, shows an example of the output wave shape from the ACS712 where we have measured the peak to peak voltage and found it to be 0.88V. As per division voltage is 0.5V, so real peak to peak voltage is 1.76V (0.88x2). This was for a load of 1000W.

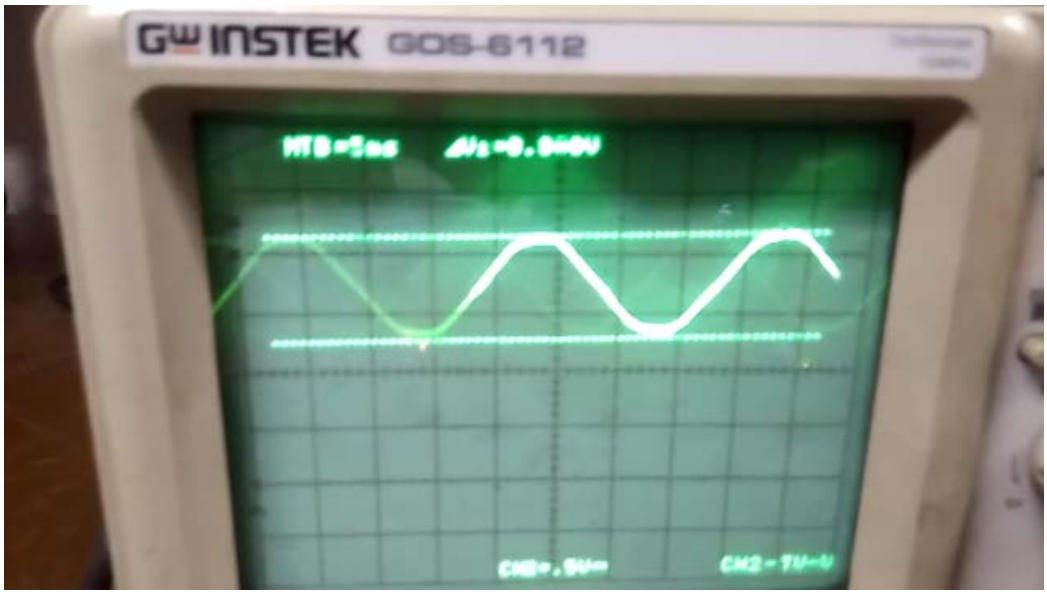



*Figure 7.3: Output waveshape from ACS sensor*

Following this method, we calculated peak to peak for various loads. Afterwards the same testing was done with MCU to compare the output value of the current from the MCU to the value of the ammeter. This helped us to check the reliability of our system.

Readings for different loads

| Loads/W | 0 | 150 | 500 | 1000 | 1500 |
|---|---|---|---|---|---|
| Peak to peak Voltage/mv | 0 | 0.20 | 0.95 | 1.76 | 2.33 |
| Ammeter I/A | 0 | 0.69 | 2.09 | 4.49 | 6.06 |
| MCU I/A | 0 | 0.69 | 2.12 | 4.46 | 6.12 |

*Table 7.1: Output current and voltage for various loads*

The curve below shows the nominal sensitivity and transfer characteristics of the ACS712-30A sensor powered with a 5.0V supply.

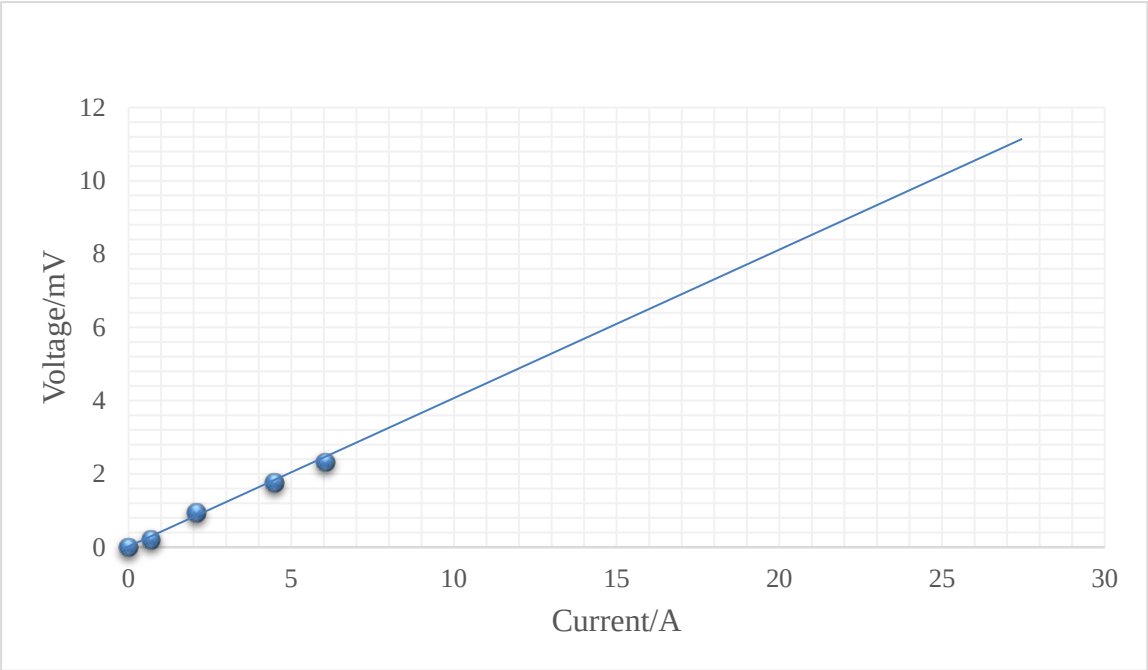



*Figure 7.4: Voltage vs ammeter current curve*

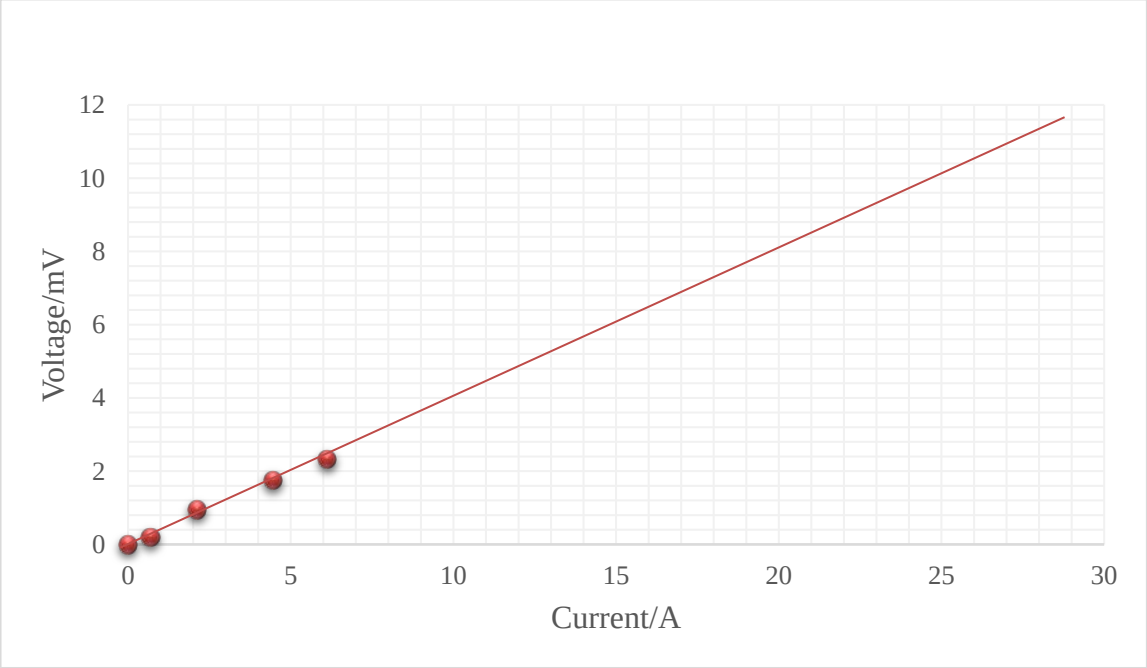



*Figure 7.5: Voltage vs MCU current curve*

From the graphs we can see that both the curves are almost identical meaning that the MCU has now been properly integrated with the MCU and provides us with reliable value of current.

## 7.2 MQTT

Before integrating ESP8266 with microcontroller, we tested the AT commands in serial terminal. This enabled us to check whether we are able to communicate with broker. For testing purpose, we are sending the message “hellowasee” with topic “buetwork” from serial terminal. At the same time we can see CloudMQTT broker receiving this message. The figure below illustrates the communication of ESP8266 with CloudMQTT broker.

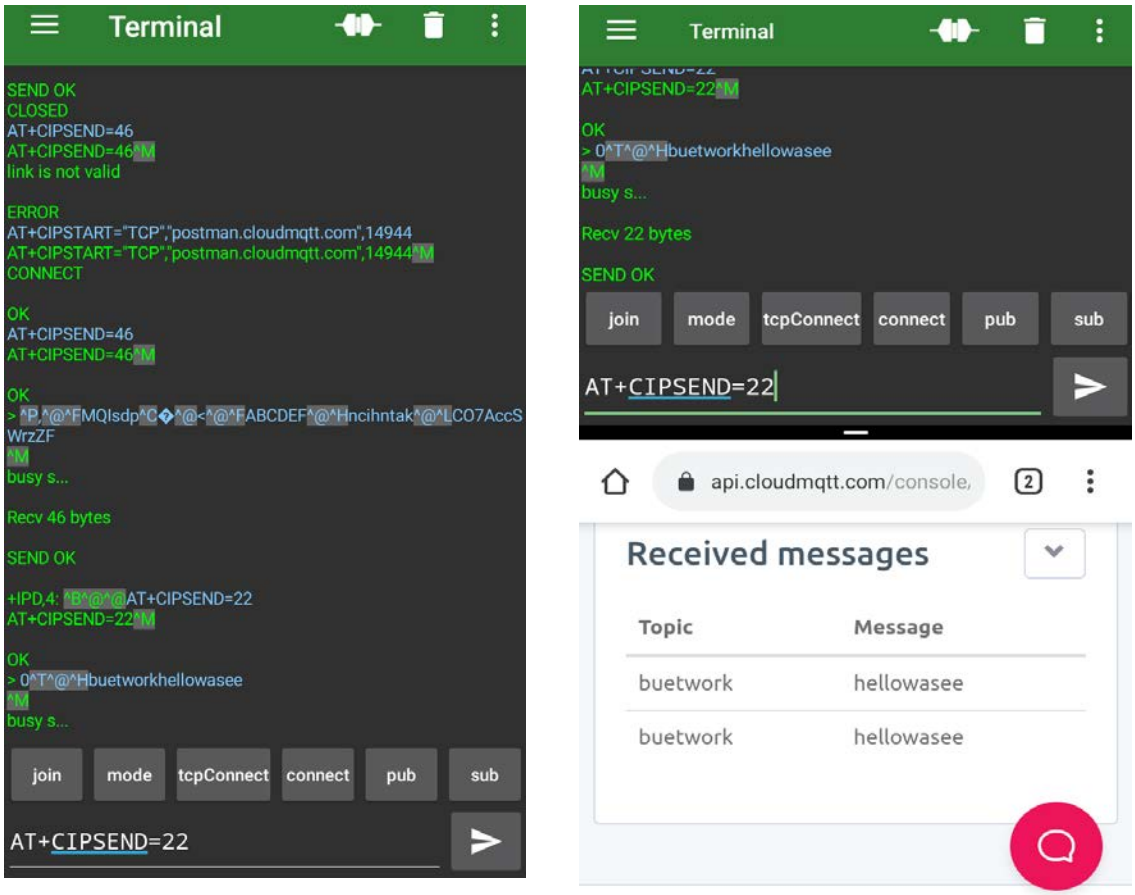



*Figure 7.6: ESP8266 communication with CloudMQTT broker*

**Data transfer rate protocols:**

In communication, the two most important factors by which a system is judged is speed and reliability of data transfer. Having worked with two protocols, it allowed us to observe and deduce advantages and disadvantages which helped us to compare between them. The tables below show the rate of data entry to the database for HTTP and MQTT.

For HTTP protocol, we could send data approximately for about every 5 seconds and if any packet drops, it would have to be a 10 second interval. Moreover, we could only send the command to control the light bulbs every 5 seconds as well since it is dependent on data entry and so, in an occurrence of any packet being dropped, it the MCU that would get the message about 10 seconds later from the server. In our experiments we have observed several packets being dropped which lead to a slow system.

| Current(Amps) | Received Time |
|---|---|
| 0.8 | 12:13:14:pm |
| 0.79 | 12:13:08:pm |
| 0.79 | 12:13:03:pm |
| 0.79 | 12:12:58:pm |
| 0.8 | 12:12:47:pm |
| 0.8 | 12:12:41:pm |
| 0.79 | 12:12:36:pm |

*Figure 7.7: Data transfer rate HTTP protocol*

In case of MQTT protocol, we could send data every 2 or 3 seconds. In our tests, there were very few instances where packets got dropped. With regards to control of the light bulbs, the process is almost instantaneous and since this is independent of data entry, we could do this anytime we wanted. Overall in our view, MQTT is far superior, for transmission and consumes less data which is ideal for using this in remote locations. The system is far simpler with the broker handling data transfer.

| Current(Amps) | Received Time |
|---|---|
| 0.76 | 11:43:35:am |
| 0.76 | 11:43:32:am |
| 0.77 | 11:43:30:am |
| 0.79 | 11:43:27:am |
| 0 | 11:43:24:am |
| 0 | 11:43:22:am |
| 0 | 11:43:19:am |

*Figure 7.8: Data transfer rate for MQTT protocol*

# CHAPTER 8

# Conclusion

## 8.1 Shortcomings

In our whole projects we faced few difficulties. Firstly, we are using a free hosting site to host our sever and it may fail if they are having maintenance. There were situations when the site was down for a few days and we could do nothing. As a result, the server did not allow access of our files during that time. Secondly, Cron job is another website which calls our PHP files which is necessary to send data to our server. However, they call the files after one minute, and sometimes it also fails. This causes loss of our data and we can get instances where data is lost for a few minutes. Thirdly, the wires used for connections in hardware side tend to get damaged for prolonged use. At that time whole system fails and microcontroller sends anomalous reading. Moreover the ESP8266 12e after soldering in PCB does not function properly.

**Solutions**

The free hosting site problem can be solved by using our own server. We can make a PC our server and access it from anywhere. This will give us a server which will be live all the time. Moreover, any problem in the server can be solved whenever we want. Next is the issue with the Cron job. As we will be using our own server no external website is needed to call the files. Lastly, regarding the wire issues, we can solder everything in a PCB board. This will save us from damaging wires and enable us to use our microcontroller for a longer period of time. ESP8266 WiFi adapter can be used to avoid soldering issue. Otherwise, we can go for Wemos D1 in which the ESP8266 chip is integrated with microcontroller.

## 8.2 Future scope

There are other protocols like CoAP, AQMP which have their own benefits. Experiments on these protocols can be done to bring a more efficient transmission model. For instance, CoAP have similar features like HTTP but it uses UDP in transport layer. They require low power and small packets used which means faster communication [15]. Similarly, AQMP is in application layer and is message oriented. This protocol consists of three components, Exchange, Message queue and Binding [16]. Firstly, Exchange part gets the message and puts them in Queue. Secondly, message Queue stores the message until client app gets it safely. Finally, Binding forms the bridge between Message Queue and Exchange. Next comes server part, here we can use Python instead of PHP as it creates maximum value in the long term. Few more advantages of Python include being beginner friendly, shorter codes and clear syntax [17]. Furthermore, experiments can be done using Zigbee, Bluetooth, GSM to compare the data transfer rate. As IoT is still a growing sector, there is still a lot of scope for development.

# References


[1] T. Yokotani and Y. Sasaki, "Transfer protocols of tiny data blocks in IoT and their performance evaluation," 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), Reston, VA, 2016, pp. 54-57.

[2] G. Mois, Z. Szilagyi, T. Sanislav and S. Folea, "An HTTP-based environmental monitoring system using power harvesting," 2017 21st International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, 2017, pp. 845-848.

[3] S. K. Vishwakarma, P. Upadhyaya, B. Kumari and A. K. Mishra, "Smart Energy Efficient Home Automation System Using IoT," 2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU), Ghaziabad, India, 2019, pp. 1-4.

[4] Rana, Jitendra, and Sunil N. Pawar. "Zigbee Based Home Automation." (2010).

[5] R. Piyare and M. Tazil, "Bluetooth based home automation system using cell phone," 2011 IEEE 15th International Symposium on Consumer Electronics (ISCE), Singapore, 2011, pp. 192-195

[6] H. Benyezza, M. Bouhedda, K. Djellout and A. Saidi, "Smart Irrigation System Based Thingspeak and Arduino," 2018 International Conference on Applied Smart Systems (ICASS), Medea, Algeria, 2018, pp. 1-4.

[7] https://mafiadoc.com/engineering-instrumentation-andsimulation_59a573b21723dd08400c3ee9.html

[8] TCP/IP model: https://www.javatpoint.com/computer-network-tcp-ip-model

[9] “WiFi Module - ESP8266 - WRL-13678 - SparkFun Electronics", Sparkfun.com. [Online]. Available: https://www.sparkfun.com/products/13678. [Accessed: 01- Dec- 2018].

[10] https://en.wikipedia.org/wiki/ESP8266

[11] Murphy, S. L., & Shankar, A. U. (1991). Connection management for the transport layer: service specification and protocol verification. IEEE Transactions on Communications, 39(12), 1762-1775.

[12] Aziz, B. (2016). A formal model and analysis of an IoT protocol. Ad Hoc Networks, 36, 49-57.

[13] Murdan, A. P., & Gunness, L. (2017). An Internet of Things based system for home automation using Web Services and Cloud Computing. Journal of Electrical Engineering, Electronics, Control and Computer Science, 3(1), 29-36.

[14] https://www.CloudMQTT.com/

[15] https://www.oreilly.com/library/view/analytics-for-the/9781787120730/ee557386-c97f-48ee-82c8-625b495fffba.xhtml

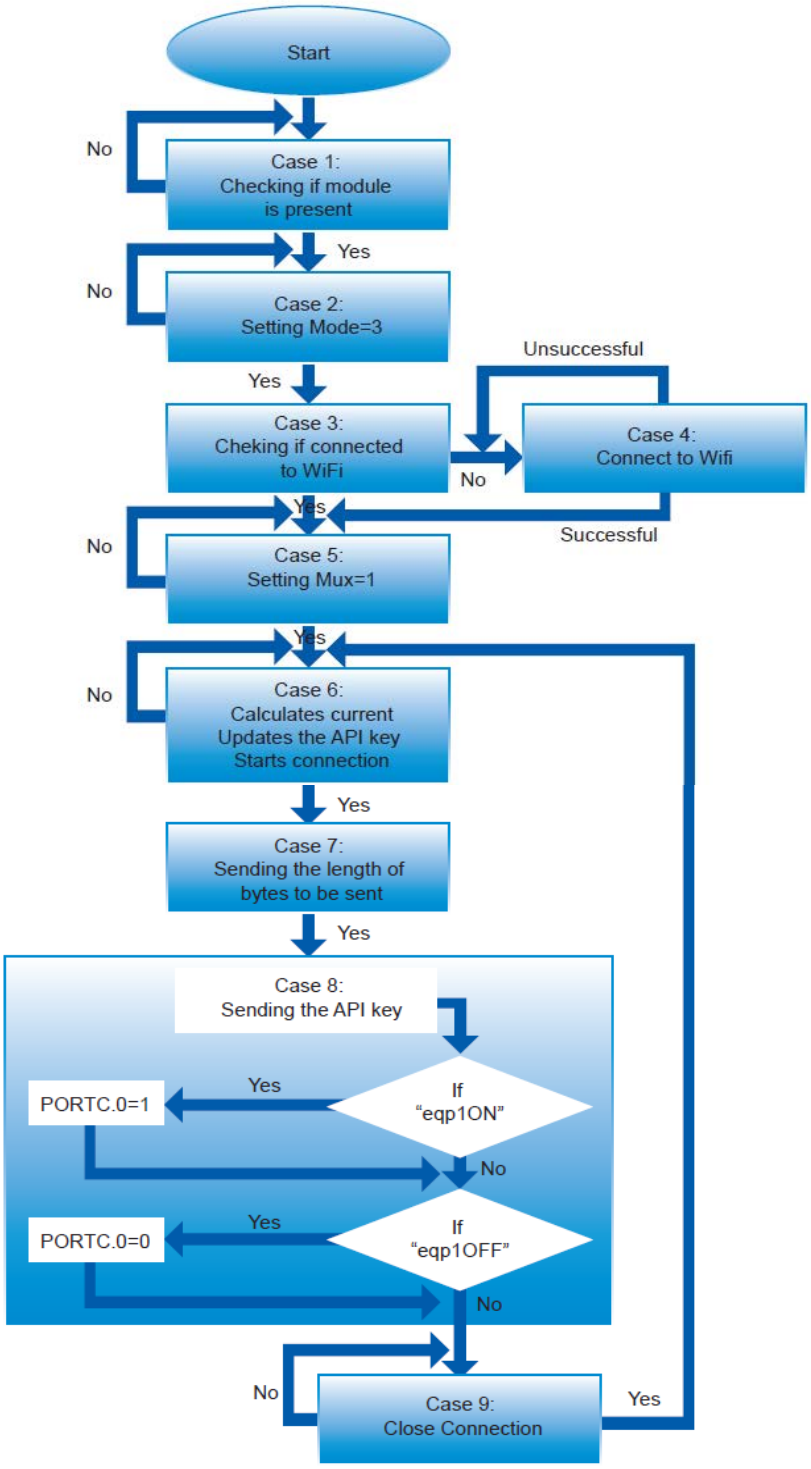
[16] https://www.ubuntupit.com/top-15-standard-IoT-protocols-that-you-must-know-about/

[17] https://www.probytes.net/blog/python-vs-php-which-is-better/

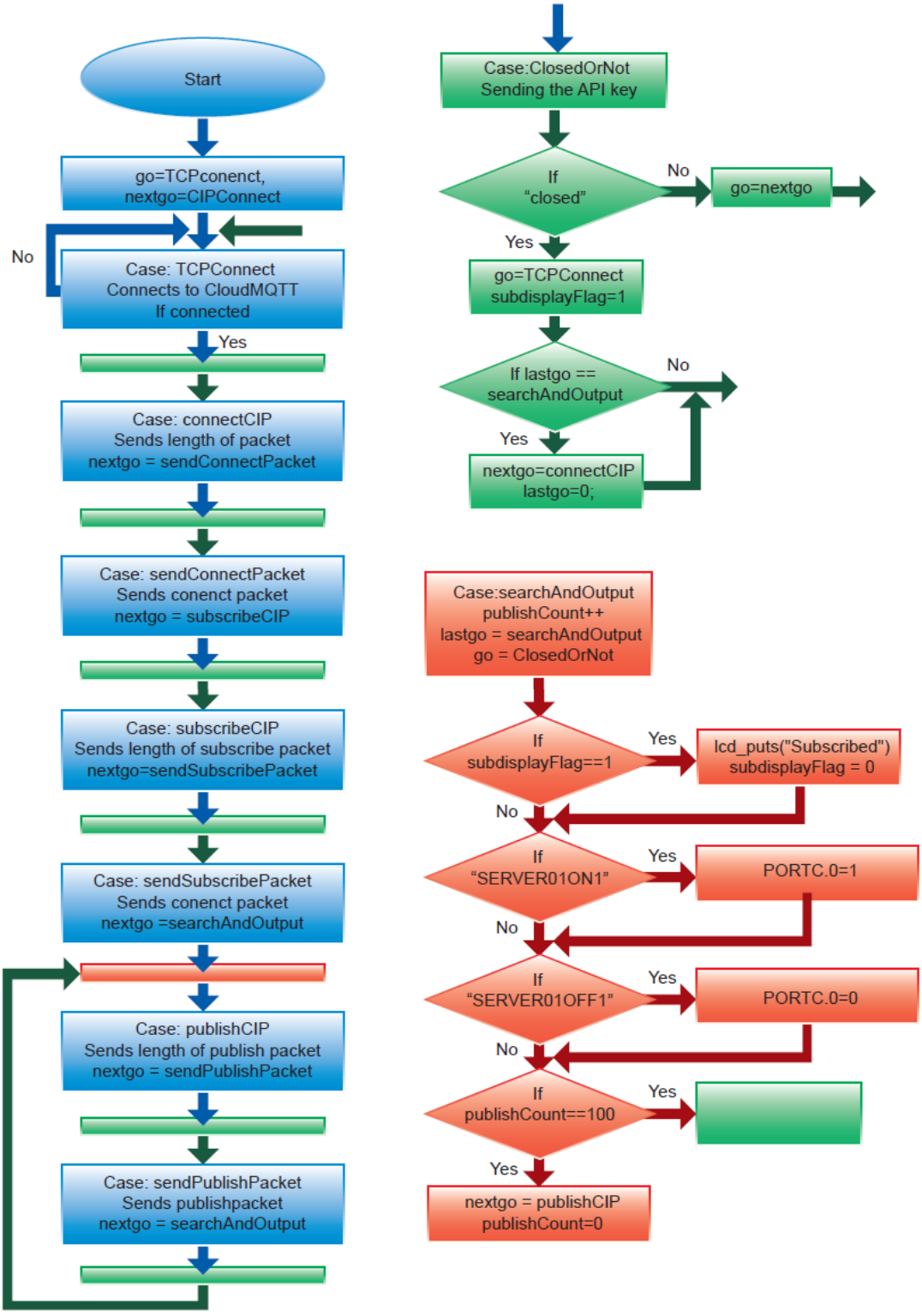
# APPENDIX:

## Appendix A: Flowchart

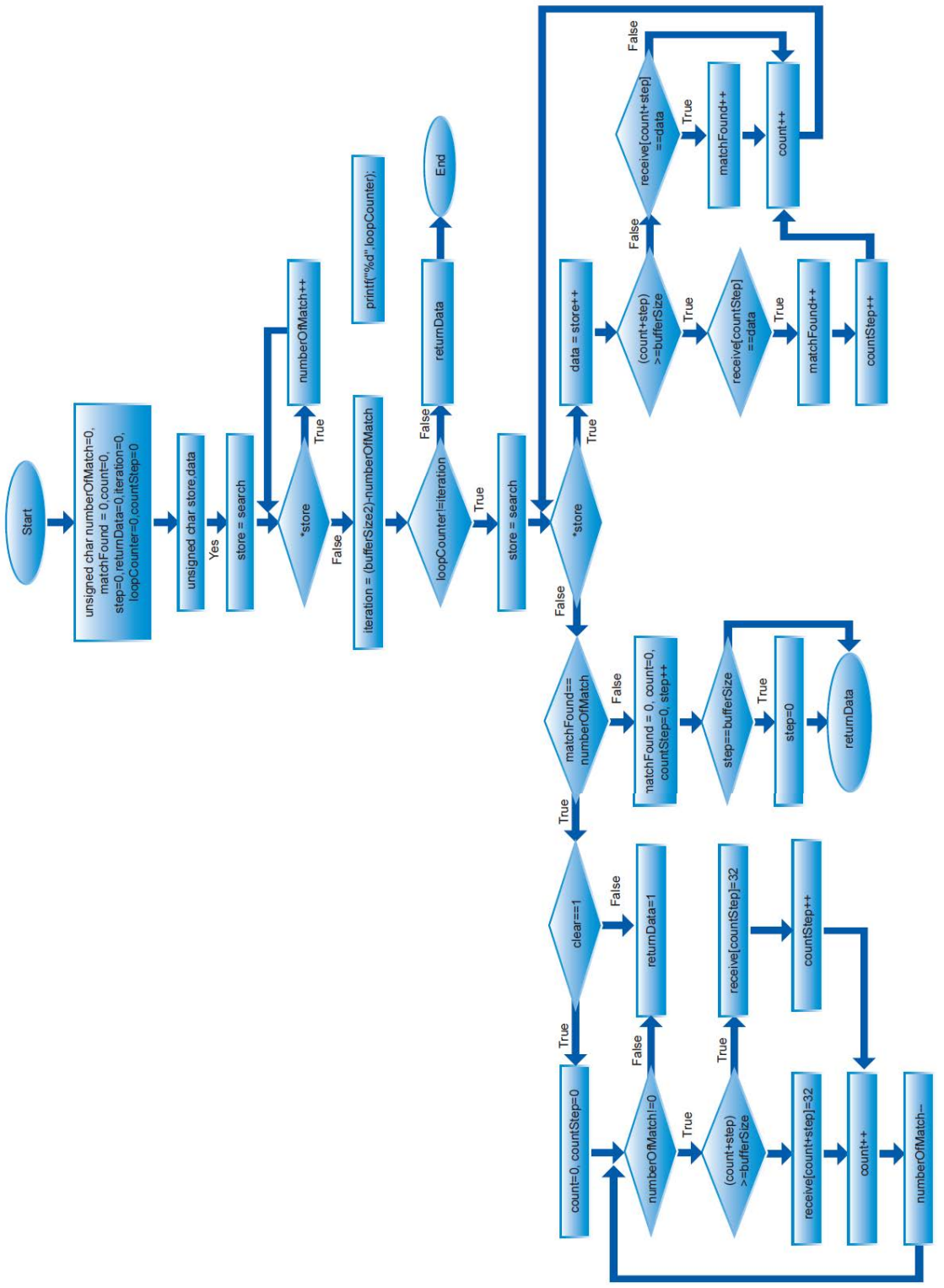
### 1) HTTP MCU main code



## 2) MQTT MCU main code



## 3) Search function of MQTT



## Appendix B:

List of Acronyms

- IoT Internet of Things
- ADC Analog to Digital Converter
- RTC Real Time Clock
- LCD Liquid Crystal Display
- IC Integrated Circuit
- RMS Root Mean Square
- MCU Microcontroller Unit
- UART Universal Asynchronous Receiver Transmitter
- ESP Espressif
- AT Attention
- USART Universal Synchronous Asynchronous Receiver Transmitter
- HTTP HyperText Transfer Protocol
- MQTT Message Queuing Telemetry Transport
- AMQP Advanced Message Queuing Protocol
- CoAP Contrained Application Protocol
- HTML Hypertext Markup Language
- QoS Quality of Service
- PHP Hypertext Preprocessor
- TCP Transmission Control Protocol
- IP Internet Protocol
- SQL Structured Query Language

# Appendix C: ACS712

1) Built in ACD function

```
//ADC
void adcInit()
{
    ADMUX=ADC_VREF_TYPE;
    ADCSRA=(1<<ADEN) | (0<<ADSC) | (0<<ADATE) | (0<<ADIF) | (0<<ADIE) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
    SFIOR=(0<<ADTS2) | (0<<ADTS1) | (0<<ADTS0);
}
unsigned int read_adc(unsigned char adc_input)
{
    ADMUX=adc_input | ADC_VREF_TYPE;
    delay_us(10);
    ADCSRA|=(1<<ADSC);
    while ((ADCSRA & (1<<ADIF))==0);
    ADCSRA|=(1<<ADIF);
    return ADCW;
}
```

2) Function getCurrent which calculates and stores the value of current in variable “cur”

```
void getCurrent(unsigned char adc_channel)
{
    char  c=0,h=0;
    sum=0;
    while (h<10)
    {
        while (c<200)
        {
            raw=read_adc(adc_channel)/204.8;
            raw=ceil(raw * 100) / 100;
            val=raw-2.5;
            sum=sum+pow(val,2);
            c++;
        }
        h++;
    }
    RMSvol=sqrt(sum)/0.02;
    cur=(RMSvol)/66;
    cur=cur*1.42-0.075;
    ftoa(cur,2,disp);
}
```

## Appendix D: HTTP MCU main code:

1) Initialization of the main code

```
#include <thesis.h>

void main(void)
{
DDRC.0 = 1;
PORTC.0 = 1;
lcdInit();//Initialize LCD Display
lcd_puts("....Welcome....");//Show on LCD
delay_ms(500);//Delay
clearLCD();//Clears LCD and goes to 1st position
USART_init();//USART initialize
receiver_init();//Initialize the buffer
adcInit();
#asm("sei")//Set interrupt
while (1)
      {
      switch(stage)
      {
```

2) Case 1 to 4

```
case 1:
        clearLCD();
        lcd_puts("Initializing");
        USART_println("AT\r\n");
        delay_ms(500);
        if(Search("OK")){ lcd_puts(".");stage=2;receiver_init();  }else{}
        break;
case 2:
        USART_println("AT+CWMODE_DEF=3\r\n");
        delay_ms(500);
        if(Search("OK")){   lcd_puts(".");stage = 3;receiver_init();  }else{}
        break;
case 3:
        USART_println("AT+CWJAP_DEF?");
        delay_ms(500);
        if(Search("No AP")){   lcd_puts(".");stage=4;receiver_init();}else{stage=5;}
        break;
case 4:
        USART_println("AT+CWJAP_DEF=\"STS\",\"tkd9663sts\"\r\n");
        delay_ms(5000);
        if(Search("WIFI CONNECTED")){   lcd_puts(".");stage=5;receiver_init();}else{}
        break;
```

3) Case 5 to 7

```
case 5:
        USART_println("AT+CIPMUX=1\r\n");
        delay_ms(500);
        if(Search("OK")){   lcd_puts(".");stage = 6;receiver_init();  }else{}
        clearLCD();
        break;
case 6:
        getCurrent(0);
        lcd_gotoxy(0,0);lcd_puts("           ");
        lcd_gotoxy(0,0);lcd_puts("Sending");
        apiKeyValueUpdate(cur);
        USART_println("AT+CIPSTART=1,\"TCP\",\"waseeserver.000webhostapp.com\",80\r\n");
        delay_ms(200);
        if(Search("CONNECT")){  lcd_puts(".");stage = 7;  }else{}
        receiver_init();
        break;
case 7:
        USART_println("AT+CIPSEND=1,73\r\n");
        delay_ms(100);
        stage = 9;
        if(Search(">")){  lcd_puts(".");stage = 8;  }else{}
        receiver_init();
        break;
```

4) Case 8 and 9

```
case 8:
        USART_println(API_key);
        delay_ms(2000);
        if(Search("eqp1ON")){PORTC.0=1;}else{}
        if(Search("eqp1OFF")){PORTC.0=0;}else{}
        stage = 9;
        receiver_init();
        break;
case 9:
        USART_println("AT+CIPCLOSE\r\n");
        delay_ms(100);
        if(Search("OK")){  lcd_puts(".");    }else{}
        stage = 6;
        receiver_init();
        break;
default:
          break;
    }
}
```

## Appendix E: HTTP MCU Library:

1) Function which sends data to ESP

```
void USART_println(char *send){
    while(*send)
    {
        /* Wait for empty transmit buffer */
        while(!( UCSRA & (1<<UDRE)));
        /* Put data into buffer, sends the data */
        UDR = *send++;
    }
}
```

2) The search function

```
int Search(char *search)
{
    int numberOfmMatch=0,matchFound = 0,count=0,step=0,returnData=0;
    char *store,data;
    store = search;
    while(*store++){numberOfmMatch++;}
    while(receive[step]!=0x00)
    {
       store = search;
       while(*store)
       {
           data = *store++;
           if(receive[count+step]==data){matchFound++;}else{}
           count++;
       }
       if(matchFound==numberOfmMatch){returnData=1;break;}
       else
       {
           matchFound=0;
           count=0;
           step++;
       }
    }
     return returnData;
}
```

3) Function that generates API Key

```
char API_key[74] = "GET /add1.php\?cur=00.00 HTTP/1.1\r\nHost: waseeserver.000webhostapp.com\r\n\r\n";
//Api key
void apiKeyValueUpdate(float value)
{
int val = 0;
val=(int)(value*100);
API_key[18]=48+(char)(val/1000);
API_key[19]=48+(char)((val%1000)/100);
API_key[21]=48+(char)(((val%1000)%100)/10);
API_key[22]=48+(char)(((val%1000)%100)%10);
}
```

4) Variables used and the built-in library files included

```
#define bufferSize 50
#include <mega32a.h>
#include <delay.h>
#include <stdio.h>
#include <stdlib.h>
#include <alcd.h>
#include <stdio.h>
#include <math.h>
#define ADC_VREF_TYPE ((0<<REFS1) | (1<<REFS0) | (0<<ADLAR))
int rcvCount=0,stage=1;
char receive[bufferSize];
char API_key[74] = "GET /add1.php\?cur=00.00 HTTP/1.1\r\nHost: waseeserver.000webhostapp.com\r\n\r\n";
float raw, RMSvol, val, sum, cur;
char disp[16];

```

5) Initialization functions

```
void receiver_init()
{
      int count = 0;
      while(count!=rcvCount)
      {
         receive[count]=0x00;
         count++;
      }
      rcvCount = 0;
}
```

```
//-----------------------------LCD----------------------
void lcdInit()
{
// Alphanumeric LCD initialization
// Connections are specified in the
// Project|Configure|C Compiler|Libraries|Alphanumeric LCD menu:
// RS - PORTA Bit 0
// RD - PORTA Bit 1
// EN - PORTA Bit 2
// D4 - PORTA Bit 4
// D5 - PORTA Bit 5
// D6 - PORTA Bit 6
// D7 - PORTA Bit 7
// Characters/line: 16
DDRB=(1<<DDB7) | (1<<DDB6) | (1<<DDB5) | (1<<DDB4) | (0<<DDB3) | (1<<DDB2) | (1<<DDB1) | (1<<DDB0);
lcd_init(16);
}
void clearLCD()
{
    lcd_clear();//Clear Display
    lcd_gotoxy(0,0);//Goto first position of display
}
```

```
//--------------------------UART-----------------------------------
void USART_init()
{
// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 115200
UCSRA=(0<<RXC) | (0<<TXC) | (0<<UDRE) | (0<<FE) | (0<<DOR) | (0<<UPE) | (0<<U2X) | (0<<MPCM);
UCSRB=(1<<RXCIE) | (0<<TXCIE) | (0<<UDRIE) | (1<<RXEN) | (1<<TXEN) | (0<<UCSZ2) | (0<<RXB8) | (0<<TXB8);
UCSRC=(1<<URSEL) | (0<<UMSEL) | (0<<UPM1) | (0<<UPM0) | (0<<USBS) | (1<<UCSZ1) | (1<<UCSZ0) | (0<<UCPOL);
UBRRH=0x00;
UBRRL=0x08;//Baud 115200
}
//--------------------------Reinit-----------------------------------
void reinit()
{
    receiver_init();
    clearLCD();
}
```

6) Interrupt used for USART

```
//Usart
interrupt [USART_RXC] void usart_rx_isr(void)
{
    receive[rcvCount]=UDR;rcvCount++;
    if(rcvCount>(bufferSize-1)){rcvCount=0;}
}
```

7) Library Link

```
extern char API_key[74];
extern float cur;
extern char disp[16];
void apiKeyValueUpdate(float value);
void getCurrent(unsigned char adc_channel);
void adcInit();
void receiver_init();
void lcdInit();
void clearLCD();
void USART_init();
void USART_println(char *send);
int Search(char *search);
void reinit();
#pragma library thesis.lib
#endif
```

# Appendix F: MQTT MCU main code:

1) Initialization of the main code

```
#define TCPConnect 1
#define ClosedOrNot 2
#define connectCIP 3
#define sendConnectPacket 4
#define subscribeCIP 5
#define sendSubscribePacket 6
#define searchAndOutput 7
#define publishCIP 8
#define sendPublishPacket 9
#include <thesis.h>
unsigned char go = 1,nextgo=3,lastgo=0,subdisplayFlag=0;
int publishCount=0;
void main(void)
{
DDRC.0 = 1;
PORTC.0 = 0;
lcdInit();
lcd_puts("|    Welcome    |\n|  MQTT Device |");
delay_ms(1000);
clearLCD();
USART_init();
adcInit();
#asm("sei")
```

2) Cases TCPConnect, ClosedOrNot and conenctCIP

```
while(1)
{
    switch(go)
    {
       case TCPConnect:
       lcd_clear();lcd_gotoxy(0,0);lcd_puts("Connecting");
       USART_println("AT+CIPSTART=\"TCP\",\"postman.cloudmqtt.com\",14944\r\n");delay_ms(1000);
       if((search("CONNECTOK",0)==1)||(search("ALREADY CONNECTED",0)==1)){go = ClosedOrNot;}
       break;

       case ClosedOrNot:
       if(search("CLOSED",0)==1)
       {
            go=TCPConnect;
            if(lastgo == searchAndOutput){nextgo=connectCIP;lastgo=0;}
            subdisplayFlag=1;
       }
       else{go=nextgo;}
       break;

       case connectCIP:
       nextgo = sendConnectPacket;
       USART_println("AT+CIPSEND=46\r\n");delay_ms(1000);
       go = ClosedOrNot;
       break;
```

3) Cases sendConnectPacket, subscribeCIP and sendSubscribePacket

```
case sendConnectPacket:
nextgo = subscribeCIP;
connectSend();delay_ms(1000);
go = ClosedOrNot;
break;

case subscribeCIP:
nextgo = sendSubscribePacket;
USART_println("AT+CIPSEND=15\r\n");delay_ms(1000);
go = ClosedOrNot;
break;

case sendSubscribePacket:
nextgo = searchAndOutput;
subscribeSend();delay_ms(1000);
go = ClosedOrNot;
subdisplayFlag = 1;
break;
```

4) Case searchAndOutput

```
case searchAndOutput:
if(subdisplayFlag==1){
    lcd_clear();
    lcd_gotoxy(0,1);
    lcd_puts("Subscribed");
    subdisplayFlag = 0;
}
if(search("SERVER01ON1",1)==1){
 PORTC.0 = 1;
}
if(search("SERVER01OFF1",1)==1){
PORTC.0 = 0;
}
delay_ms(10);
publishCount++;
if(publishCount==100){
    nextgo = publishCIP;
    publishCount=0;
}
lastgo = searchAndOutput;
go = ClosedOrNot;
break;
```

5) Cases publishCIP and sendPublishPacket

```
92          case publishCIP:
93          nextgo = sendPublishPacket;
94          getCurrent(0);
95          publishValueUpdate(cur);
96          USART_println("AT+CIPSEND=17\r\n");delay_ms(1000);
97          go = ClosedOrNot;
98          break;
99
100         case sendPublishPacket:
101         nextgo = searchAndOutput;
102         publishSend();lcd_gotoxy(0,0);lcd_puts("Publishing ");delay_ms(1000);
103         go = ClosedOrNot;
104         subdisplayFlag = 1;
105         break;
106
107         default:
108         break;
109     }
110 }
```

# Appendix G

## MQTT MCU Library

1) connectSend, publishSend and subscribeSend functions

```
126 void connectSend()
127 {
128     char count =0;
129     while(count!=46)
130     {
131     /* Wait for empty transmit buffer */
132      while(!( UCSRA & (1<<UDRE)));
133      /* Put data into buffer, sends the data */
134      UDR = connect[count];
135      count++;
136      }
137 }
138 void publishSend()
139 {
140     char count =0;
141     while(count!=17)
142     {
143     /* Wait for empty transmit buffer */
144      while(!( UCSRA & (1<<UDRE)));
145      /* Put data into buffer, sends the data */
146      UDR = publish[count];
147      count++;
148      }
149 }
151 void subscribeSend()
152 {
153     char count =0;
154     while(count!=15)
155     {
156     /* Wait for empty transmit buffer */
157      while(!( UCSRA & (1<<UDRE)));
158      /* Put data into buffer, sends the data */
159      UDR = subscribe[count];
160      count++;
161      }
162 }
```

2) publishValueUpdate Function

```
void publishValueUpdate(float value)
{
     int val = 0;
     val=(int)(value*100);
     publish[12]=48+(char)(val/1000);
     publish[13]=48+(char)((val%1000)/100);
     publish[15]=48+(char)(((val%1000)%100)/10);
     publish[16]=48+(char)(((val%1000)%100)%10);
}
```

3) Variables and Built in functions

```
#define bufferSize 100
#include <mega32a.h>
#include <delay.h>
#include <stdio.h>
#include <stdlib.h>
#include <alcd.h>
#include <stdio.h>
#include <math.h>
#define ADC_VREF_TYPE ((0<<REFS1) | (1<<REFS0) | (0<<ADLAR))

//ADC
float raw, RMSvol, val, sum, cur;
char disp[5];
//USART
unsigned char rcvCount=0;
unsigned char connect[46]={16,44,0,6,'M','Q','I','s','d','p',3,194,0,60,0,6,'M','I','C','R','O',
            '1',0,8,'n','c','i','h','n','t','a','k',0,12,'C','O','7','A','c','c','S','W','r','z','Z','F'};
unsigned char publish[17]= {48,15,0,6,'d','e','v','i','c','e','E','1','0','0','.','0','0'};
unsigned char subscribe[15]={130,13,0,1,0,8,'S','E','R','V','E','R','0','1',0};
char receive[bufferSize];

```

4) Initialization Functions

```
void lcdInit()
{
// RS - PORTA Bit 0
// RD - PORTA Bit 1
// EN - PORTA Bit 2
// D4 - PORTA Bit 4
// D5 - PORTA Bit 5
// D6 - PORTA Bit 6
// D7 - PORTA Bit 7
// Characters/line: 16x2
DDRB=(1<<DDB7) | (1<<DDB6) | (1<<DDB5) | (1<<DDB4) | (0<<DDB3) | (1<<DDB2) | (1<<DDB1) | (1<<DDB0);
lcd_init(16);
}
```

```
void USART_init()
{
// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 115200
UCSRA=(0<<RXC) | (0<<TXC) | (0<<UDRE) | (0<<FE) | (0<<DOR) | (0<<UPE) | (0<<U2X) | (0<<MPCM);
UCSRB=(1<<RXCIE) | (0<<TXCIE) | (0<<UDRIE) | (1<<RXEN) | (1<<TXEN) | (0<<UCSZ2) | (0<<RXB8) | (0<<TXB8);
UCSRC=(1<<URSEL) | (0<<UMSEL) | (0<<UPM1) | (0<<UPM0) | (0<<USBS) | (1<<UCSZ1) | (1<<UCSZ0) | (0<<UCPOL);
UBRRH=0x00;
UBRRL=0x08;//Baud 115200
}
void receiver_init()
{
    rcvCount = 0;
    while(rcvCount!=bufferSize)
    {
       receive[rcvCount]=0x00;
       rcvCount++;
    }
    rcvCount = 0;
}
```

5) Search Function

```
unsigned char search(unsigned char *search,unsigned char clear){
    unsigned char numberOfMatch=0,matchFound = 0,count=0,step=0,returnData=0,iteration=0;
    unsigned char loopCounter=0,countStep=0;
    unsigned char *store,data;
    store = search;
    while(*store++){numberOfMatch++;}
    iteration = (bufferSize*2)-numberOfMatch;
    while(loopCounter!=iteration)
    {
       store = search;
       while(*store)
       {
           data = *store++;
           if((count+step)>=bufferSize)
           {
               if(receive[countStep]==data){matchFound++;}
               countStep++;
           }
           else
           {
               if(receive[count+step]==data){matchFound++;}
           }
           count++;
       }
       if(matchFound==numberOfMatch)
       {
            if(clear==1)
            {
                count = 0;
                countStep = 0;
                while(numberOfMatch!=0)
                {
                    if((count+step)>=bufferSize)
                    {
                       receive[countStep]=32;
                       countStep++;
                    }
                    else
                    {
                       receive[count+step]=32;
                    }
                    count++;
                    numberOfMatch--;
                }
            }
            else{}
            returnData=1;break;
       }
       else
       {
           matchFound=0;
           count=0;
           countStep=0;
           step++;
           if(step==bufferSize){step=0;}
       }
       loopCounter++;
    }
     //printf("%d",loopCounter);
     return returnData;
}
```

6) Interrupt used for USART and USART_println used in communicating with ESP module

```
void USART_println(char *send){
    while(*send)
    {
        /* Wait for empty transmit buffer */
        while(!( UCSRA & (1<<UDRE)));
        /* Put data into buffer, sends the data */
        UDR = *send++;
    }
}

interrupt [USART_RXC] void usart_rx_isr(void)
{
   char data=UDR;
   if(data>=32&&data<=126){receive[rcvCount]=data;rcvCount++;}else{}//space to ~
   if(rcvCount>=bufferSize){rcvCount=0;}else{}
}
```

7) “thesis.lib” Library Linker File

```
#ifndef _THESIS_INCLUDED_
#define _THESIS_INCLUDED_
#define bufferSize 100
#include <mega32a.h>
#include <delay.h>
#include <stdio.h>
#include <stdlib.h>
#include <alcd.h>
#include <stdio.h>
#include <math.h>
//ADC
extern float cur;
extern char disp[5];
//USART
extern unsigned char rcvCount;
extern char receive[bufferSize];
//LCD
void lcdInit();
void clearLCD();
//ADC
void publishValueUpdate(float value);
void adcInit();
void getCurrent(unsigned char adc_channel);
//USART
void USART_init();
void receiver_init();
void USART_println(char *send);
void connectSend();
void publishSend();
void subscribeSend();
//Others
unsigned char search(char *search,unsigned char clear);
#pragma library thesis.lib
#endif
```

# Appendix H: HTTP Server Code.

```
<!DOCTYPE html>
<html>
<head>
    <title>Thesis Buet 2018</title>
</head>
<body>
    <form action="index.php" method="post">
    <input type="submit" name="Refresh" value="Home"/>
    <h1>Data Management</h1>
    <table border="1" width = 100% cellspacing="1" cellpadding="1">
        <tr>
        <td>Show records of selected equipment between the date range</td>
        <td>Show records of selected equipment between the value range</td>
        <td>Search records from the database table</td>
        </tr>
        <tr>
        <td>
        To <input type="date" name="dateTo" value=""/>
        - From<input type="date" name="dateFrom" value=""/>
        </td>

        <td>
        To <input type="text" name="valueTo" size="20" value="0.00" />
        - From<input type="text" name="valueFrom" size="20" value="0.00" />
        </td>
```

1) HTML coding used to design the interface

```
        <td>
        <input type="submit" style = "width:100%;height:100%" name="Equipment1" value="Equipment1"/>
        <input type="submit" style = "width:100%;height:100%" name="Equipment2" value="Equipment2"/>
        </td>
        </tr>
    </table>
    <h1>Manual Override</h1>
    <table border=10 width=100% cellspacing="1" cellpadding="1">
        <tr>
        <td>Equipment1</td>
        <td>Equipment2</td>
        </tr>
        <tr>
        <td>
        <p><input type="submit" style="height:50px;width:100%" name="LED_ON1" value="LED ON"/></p>
        <p><input type="submit" style="height:50px;width:100%" name="LED_OFF1" value="LED OFF"/></p>
        </td>
        <td>
        <p><input type="submit" style="height:50px;width:100%" name="LED_ON2" value="LED ON"/></p>
        <p><input type="submit" style="height:50px;width:100%" name="LED_OFF2" value="LED OFF"/></p>
        </td>
        </tr>
    </table>
    </form>
    <a titlt="print screen" alt="print screen" onclick="window.print();"tarPOST="_blank"
```

```
     style="cursor:pointer;"><h2>Print</h2></a>
</body>
</html>
```

2) Buttons to show data

```
if(isset($_GET['Equipment1']))
{
    showAllData("All Equipment 1 Data","equipment1",$_GET['valueTo'],$_GET['valueFrom']);
}
if(isset($_GET['Equipment2']))
{
    showAllData("All Equipment 2 Data","equipment2",$_GET['valueTo'],$_GET['valueFrom']);
}
```

3) Function that shows the data in a tabular format

```php
function showAllData($tableName,$dataBaseTableName,$curValtoStr,$curValfromStr)
{
    $curValTo = (float) $curValtoStr;
    $curValFrom = (float) $curValfromStr;
    require_once('mysql_connect.php');
    $query = 'SELECT current,time,date FROM '.$dataBaseTableName.' ORDER BY serial DESC';
    $response = @mysqli_query($dbc, $query);
    if($response)
    {
        echo
        '
        <table border="5" width = 100% cellspacing="5" cellpadding="5">
            <tr><th colspan="3"> <em>'. $tableName .'</em></th></tr>
            <tr>

            <td align="center"><b>Current(Amps)</b></td>
            <td align="center"><b>Received Time</b></td>
            <td align="center"><b>Received Date</b></td>

            </tr>
        '
        ;
        function printTable($row)
        {
            echo
            '
            <tr>

            <td align="center">' . $row['current'] . '</td>
            <td align="center">' . $row['time'] . '</td>
            <td align="center">' . $row['date'] . '</td>

            </tr>
            '
            ;
        }
        while($row = mysqli_fetch_array($response))
        {
            if($_POST['dateTo']!=null && $_POST['dateFrom']!=null)
            {
                if(strtotime($row['date'])>=strtotime($_POST['dateTo']) && strtotime($row['date'])<=strtotime
                    ($_POST['dateFrom']))
                {
                    if($curValTo==0 && $curValFrom==0)
                    {
                        printTable($row);
                    }
                    else
                    {
                        if($row['current']>=$curValTo && $row['current']<=$curValFrom)
                        {
                            printTable($row);
                        }
                        else{}
                    }
                }
                else{}
            }
            else
            {
                if($curValTo==0 && $curValFrom==0)
                {
                    printTable($row);
                }
                else
                {
                    if($row['current']>=$curValTo && $row['current']<=$curValFrom)
                    {
                        printTable($row);
                    }
                    else{}
                }
            }
        }
        echo '</table>';
    }
    else
    {
        echo "Couldn't issue database query<br />";
    }
    mysqli_close($dbc);
}
```

4) If statement executed when the control of lights is used

```php
if(isset($_GET['LED1_ON']))
{
    require_once('mysql_connect.php');
    $query = "UPDATE commands SET eqp1='eqp1ON'";
    $stmt = mysqli_prepare($dbc, $query);
    mysqli_stmt_execute($stmt);
    mysqli_stmt_close($stmt);
    mysqli_close($dbc);
}
else{}
if(isset($_GET['LED1_OFF']))
{
    require_once('mysql_connect.php');
    $query = "UPDATE commands SET eqp1='eqp1OFF'";
    $stmt = mysqli_prepare($dbc, $query);
    mysqli_stmt_execute($stmt);
    mysqli_stmt_close($stmt);
    mysqli_close($dbc);
}
else{}
```

5) Statement executed when API bring new value of current

```php
<?php
if(isset($_GET['cur']))
{
    $cur = $_GET['cur'];
    require_once('mysql_connect.php');
    $date = date('d-M-Y');
    $time = date('h:i:s:a');
    $query = "INSERT INTO equipment1 (serial, current, time, date) VALUES (NULL, ?, '$time', '$date')";
    $stmt = mysqli_prepare($dbc, $query);
    mysqli_stmt_bind_param($stmt, "d", $cur);
    mysqli_stmt_execute($stmt);
    mysqli_stmt_close($stmt);
    $query = "SELECT eqp1,eqp2,eqp3 FROM commands";
    $response = @mysqli_query($dbc, $query);
    if($response) {
        while($row = mysqli_fetch_array($response))    {
            echo $row['eqp1'];
            echo $row['eqp2'];
        }
    }
    else{echo "Error DB";}
    mysqli_close($dbc);
}
?>

```

6) Database Linkup file

**/public_html/mysql_connect.php**

```php
<?php
DEFINE ('DB_USER', 'id8969656_waseezdatabase');
DEFINE ('DB_PASSWORD', 'wasee123');
DEFINE ('DB_HOST', 'localhost');
DEFINE ('DB_NAME', 'id8969656_waseezdb');
$dbc = @mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME)
OR die('Could not connect to MySQL: ' .mysqli_connect_error());
?>
```

# Appendix I: MQTT Server Side code:

1) Publish

```php
//.............Button On/Off.............//
if(isset($_POST['LED_ON1']))
{
    $mqtt = new Bluerhinos\phpMQTT("postman.cloudmqtt.com", 14944, "SERVERPUB");
    if ($mqtt->connect(true, NULL, "ncihntak", "CO7AccSWrzZF"))
    {
        $mqtt->publish("SERVER01", "ON1", 0);
        $mqtt->close();
    }
    else{echo "Could not connect";}
}
else{}

if(isset($_POST['LED_OFF1']))
{
    $mqtt = new Bluerhinos\phpMQTT("postman.cloudmqtt.com", 14944, "SERVERPUB");
    if ($mqtt->connect(true, NULL, "ncihntak", "CO7AccSWrzZF"))
    {
        $mqtt->publish("SERVER01", "OFF1", 0);
        $mqtt->close();
    }
    else{echo "Could not connect";}
}
else{}
```

2) Subscribe

**/public_html/main.php**

```php
<?php
require 'phpMQTT.php';
require_once('mysql_connect.php');
$start_time = time();
global $flag;
$flag =0;
$mqtt = new Bluerhinos\phpMQTT("postman.cloudmqtt.com", 14944, "SERVER");
if (!$mqtt->connect(true, NULL, "ncihntak", "CO7AccSWrzZF"))
{
    echo "Could not connect";
    exit(1);
}
while(!hasTimedout())
{
    $mqtt->proc();
    if($flag==1)
    {
        $date = date('d-M-Y');
        $time = date('h:i:s:a');
        if($eqp=="E1")
        {
            $query = "INSERT INTO equipment1 (serial, current, time, date) VALUES (NULL, $cur, '$time', '$date')";
            if ($dbc->query($query) === TRUE){}
        }
        if($eqp=="E2")
        {
            $query = "INSERT INTO equipment2 (serial, current, time, date) VALUES (NULL, $cur, '$time', '$date')";
            if ($dbc->query($query) === TRUE){}
        }
        $flag=0;
    }
}
$mqtt->close();
mysqli_close($dbc);

```

3) procmsg and hasTimedout Functions

```
37  function procmsg($topic, $val)
38  {
39      global $cur,$flag,$eqp;
40      $flag=1;
41      $cur = substr($val,2,8);
42      $eqp = substr($val,0,2);
43  }
44  function hasTimedout()
45  {
46      global $start_time;
47      return (time() - $start_time > 58);
48  }
49  ?>
```

## Appendix J : Mobile Application

1) Manifest file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="HTTP://schemas.android.com/apk/res/android"
package="com.example.deboky.paharaapp">
<uses-permission android:name="android.permission.INTERNET" />
<application
   android:name=".App"
   android:allowBackup="true"
   android:icon="@mipmap/ic_launcher"
   android:label="@string/app_name"
   android:usesCleartextTraffic="true"
   android:roundIcon="@mipmap/ic_launcher_round"
   android:supportsRtl="true"
   android:theme="@style/AppTheme">
   <activity android:name=".ApplianceDetailsActivity"></activity>
   <activity android:name=".SplashScreenActivity">
      <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
   </activity>
   <activity android:name=".ApplianceListActivity"></activity>
</application>
</manifest>
```

2) Splash Screen

```
package com.example.deboky.paharaapp;
import android.content.Intent;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
public class SplashScreenActivity extends AppCompatActivity {
   private static final int SPLASH_DISPLAY_LENGTH = 500;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_splash_screen);
      new Handler().postDelayed(new Runnable(){
         @Override
         public void run() {
            Intent mainIntent = new Intent(SplashScreenActivity.this,
ApplianceListActivity.class);
            startActivity(mainIntent);
            finish();
         }
      }, SPLASH_DISPLAY_LENGTH);
   }
}
```

XML file of splash screen

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="HTTP://schemas.android.com/apk/res/android"
xmlns:app="HTTP://schemas.android.com/apk/res-auto"
xmlns:tools="HTTP://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".SplashScreenActivity"
android:gravity="center"
android:background="@color/colorMagento">


<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:textColor="#ffffff"
   android:textSize="26sp"
   android:textStyle="bold"
   android:text="Welcome To Pahara App"/>

</LinearLayout>
```

3) Creation of activity_appliances_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="HTTP://schemas.android.com/apk/res/android"
   xmlns:app="HTTP://schemas.android.com/apk/res-auto"
```

```
    xmlns:tools="HTTP://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    android:padding="10dp"
    tools:context=".ApplianceListActivity">

    <Button
        android:id="@+id/appliance_one_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="10dp"
        android:text="Appliance 1"/>

    <Button
        android:id="@+id/appliance_two_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="10dp"
        android:text="Appliance 2"/>

</LinearLayout>
```

Appliance list Activity :

ApplianceListActivity

Java file

```
package com.example.deboky.paharaapp;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;

public class ApplianceListActivity extends AppCompatActivity {
    private static final String TAG = ApplianceListActivity.class.getSimpleName();

    Button applianceOneButton;
    Button applianceTwoButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_appliance_list);

        applianceOneButton = findViewById(R.id.appliance_one_button);
```

```
        applianceTwoButton = findViewById(R.id.appliance_two_button);

        applianceOneButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(ApplianceListActivity.this,
ApplianceDetailsActivity.class);
                intent.putExtra(ApplianceDetailsActivity.EXTRA_EQUIPMENT_NAME,
"Equipment1");
                startActivity(intent);
            }
        });

        applianceTwoButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(ApplianceListActivity.this,
ApplianceDetailsActivity.class);
                intent.putExtra(ApplianceDetailsActivity.EXTRA_EQUIPMENT_NAME,
"Equipment2");
                startActivity(intent);
            }
        });

    }
}
```

4) Construction of Appliance Information class

```
package com.example.deboky.paharaapp;
public class ApplianceInformation {
    private String current;
    private String time;
    private String date;
    public ApplianceInformation(String current, String time, String date) {
        this.current = current;
        this.time = time;
        this.date = date;
    }

    public String getCurrent() {
        return current;
    }

    public void setCurrent(String current) {
        this.current = current;
    }

    public String getTime() {
        return time;
```

```
    }

    public void setTime(String time) {
        this.time = time;
    }

    public String getDate() {
        return date;
    }

    public void setDate(String date) {
        this.date = date;
    }
}
```

5) Designing App class:

```
package com.example.deboky.paharaapp;
import android.app.Application;
import android.text.TextUtils;
import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.toolbox.Volley;

public class App extends Application {

    public static final String TAG = App.class.getSimpleName();

    private RequestQueue mRequestQueue;
    private static App mInstance;

    @Override
    public void onCreate() {
        super.onCreate();
        mInstance = this;
    }

    public static synchronized App getInstance() {
        return mInstance;
    }

    public RequestQueue getRequestQueue() {
        if (mRequestQueue == null) {
            mRequestQueue = Volley.newRequestQueue(getApplicationContext());
        }
        return mRequestQueue;
    }

    public <T> void addToRequestQueue(Request<T> req, String tag) {
        req.setTag(TextUtils.isEmpty(tag) ? TAG : tag);
```

```
        getRequestQueue().add(req);
    }
}
```

6) Building Appliance Details Activity class:

```
        equipmentName = getIntent().getStringExtra(EXTRA_EQUIPMENT_NAME);
        fetchArray(equipmentName);
        setUpSwipeRefreshListener();
    }

    private void setUpSwipeRefreshListener(){
        swipeRefreshLayout.setOnRefreshListener(new
SwipeRefreshLayout.OnRefreshListener() {
            @Override
            public void onRefresh() {
                if(swipeRefreshLayout.isRefreshing()){
                    fetchArray(equipmentName);
                }
            }
        });
    }

    private void fetchArray(final String equipmentName) {
        String applianceRequestTag = "json_array_req";
        String url = "HTTP://waseeserver.000webhostapp.com/list.php";

        final ProgressDialog pDialog = new ProgressDialog(this);
        pDialog.setMessage("Loading...");
        pDialog.setCancelable(false);
        pDialog.show();

        StringRequest stringRequest = new StringRequest(Request.Method.POST, url, new
Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                try {
                    JSONObject jsonResponse = new JSONObject(response);
                    String totalCount = jsonResponse.getString("TotalRecords");
                    JSONArray jsonArray = jsonResponse.getJSONArray("Results");
                    for(int i=0; i<jsonArray.length(); i++) {
                        JSONObject object = jsonArray.getJSONObject(i);
                        String current = object.getString("current");
                        String time = object.getString("time");
                        String date = object.getString("date");
                        ApplianceInformation applianceInformation = new
ApplianceInformation(current, time, date);
                        adapter.addItem(applianceInformation);
                    }
```

```
                pDialog.hide();
                swipeRefreshLayout.setRefreshing(false);
                adapter.notifyDataSetChanged();
            } catch(Exception ex){

            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            String message = error.getLocalizedMessage();
            pDialog.hide();
            swipeRefreshLayout.setRefreshing(false);
            Log.d(TAG, ""+error.getLocalizedMessage());
        }
    })

    {
        @Override
        protected Map<String, String> getParams() {
            Map<String, String> params = new HashMap<String, String>();



            if(equipmentName.equals("Equipment1")){
                params.put("Equipment1", equipmentName);
            }
            else if(equipmentName.equals("Equipment2")){
                params.put("Equipment2", equipmentName);

            }
            params.put("valueTo", "0.00");
            params.put("valueFrom", "0.00");
            return params;
        }
        @Override
        public String getBodyContentType() {
            return "application/x-www-form-urlencoded";
        }
    };
    App.getInstance().addToRequestQueue(stringRequest, applianceRequestTag);
  }
}
```

7) Designing ApplianceAdapter class:

```
package com.example.deboky.paharaapp;
import android.support.annotation.NonNull;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
```

```
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import java.util.ArrayList;

public class ApplianceAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder>
{

    private ArrayList<ApplianceInformation> items;

    public ApplianceAdapter() {
        items = new ArrayList<>();
    }

    @NonNull
    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup
viewGroup, int i) {
        View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.appliance_item,
viewGroup, false);
        return new ApplianceAdapterViewHolder(v);
    }

    @Override
    public void onBindViewHolder(@NonNull RecyclerView.ViewHolder viewHolder, int
position) {
        ApplianceInformation appliance = items.get(position);
        ApplianceAdapterViewHolder mHolder = (ApplianceAdapterViewHolder) viewHolder;
        mHolder.serialNumberTextView.setText(""+appliance.getCurrent());
        mHolder.currentTextView.setText(""+appliance.getTime());
        mHolder.timeTextView.setText(appliance.getDate());
    }

    @Override
    public int getItemCount() {
        return items.size();
    }


    public void clear() {
        items.clear();
    }

    public void addItem(ApplianceInformation appliance) {
        items.add(appliance);
    }

    class ApplianceAdapterViewHolder extends  RecyclerView.ViewHolder {
        TextView serialNumberTextView;
        TextView currentTextView;
```

```
        TextView timeTextView;

        public ApplianceAdapterViewHolder(@NonNull View itemView) {
            super(itemView);
            serialNumberTextView = itemView.findViewById(R.id.serial_number_text_view);
            currentTextView = itemView.findViewById(R.id.current_text_view);
            timeTextView = itemView.findViewById(R.id.time_text_view);
        }
    }

}
```

8) Conversion of 000Webhost List File to JSON format:

```php
<?php
function showAllData($tableName,$dataBaseTableName,$curValtoStr,$curValfromStr) {
    $curValTo = (float) $curValtoStr;
    $curValFrom = (float) $curValfromStr;
    require_once('mysql_connect.php');
    $totalCountQuery = 'SELECT COUNT(*) FROM '.$dataBaseTableName;
    $resultSet = @mysqli_query($dbc, $totalCountQuery);
    $arr = mysqli_fetch_array($resultSet);
    $totaCount = $arr["COUNT(*)"];
    $query = 'SELECT serial, current,time,date FROM '.$dataBaseTableName.' ORDER BY serial desc LIMIT 0, 50';
    $response = @mysqli_query($dbc, $query);
    $rows = array();
    while($row = mysqli_fetch_array($response)){
        $rows[] = $row;
    }
    $results = array("TotalRecords"=>$totaCount, "Results"=>$rows);
    //echo json_encode($rows);
    echo json_encode($results);
    mysqli_close($dbc);
}
if(isset($_POST['Equipment1'])){
    showAllData("All Equipment 1 Data","equipment1",$_POST['valueTo'],$_POST['valueFrom']);
}

if(isset($_POST['Equipment2'])){
    showAllData("All Equipment 2 Data","equipment2",$_POST['valueTo'],$_POST['valueFrom']);
}
?>
```