The designer has applied ethics to the design process and in the selection of the final proposed design. And that, the designer has held the safety of the public to be paramount and has addressed this in the presented design wherever may be applicable.

Supervisors

**Head Supervisor:**
Dr. Khallaayoun Ahmed

**Assistant Supervisor:**
Mr. Rachid Lghoul

**Capstone Student:**
Kaiiss Bouali
Acknowledgments

I would like to thank all those who contributed to the success of my capstone project and the ones who supported me in technical contributions.

First of all, I would like to thank my supervisors, Dr. Khallaayoun Ahmed and Mr. Rachid Lghoul, who have helped me throughout my capstone semester and have allowed me to go further in the field of Mechatronics alongside Computer Science. Also, a special thanks to Mr. Iraqi Omar for welcoming me into his office and advising me in time of need. Finally, I would like to thank all the people who advised me during the writing of this report, as well as the implementation of the project.
# Table of Content

Acknowledgments .................................................................................................................. 2  
Table of Content .................................................................................................................. 3  
Abstract .................................................................................................................................. 5  
Introduction ............................................................................................................................. 6  
Project Specifications* .......................................................................................................... 7  
  Advantages of MQTT over HTTP/HTTPS ................................................................................. 11  
  Cellular Data (4G) .................................................................................................................... 11  
  Wi-Fi ........................................................................................................................................ 12  
Feasibility Study* ................................................................................................................... 14  
  i. ANALYSIS ............................................................................................................................ 14  
  ii. TECHNICAL FEASIBILITY ................................................................................................. 15  
      a. POWER CONSUMPTION ............................................................................................... 15  
      b. COMMUNICATION CAPACITIES ................................................................................... 15  
      c. ECONOMIC FEASIBILITY ............................................................................................. 17  
      d. LEGAL FEASIBILITY ...................................................................................................... 17  
      e. OPERATIONAL FEASIBILITY ....................................................................................... 17  
      f. SCHEDULE FEASIBILITY ............................................................................................. 18  
Architecture ............................................................................................................................. 18  
Proof of Concept ..................................................................................................................... 19  
  Wiring Diagram ..................................................................................................................... 20  
  Schematics ............................................................................................................................. 20  
Software Implementation ........................................................................................................ 21  
  Server-side ............................................................................................................................. 21  
  Client-side ............................................................................................................................... 22  
STEEPLE Analysis .................................................................................................................. 26  
  Definition ............................................................................................................................... 26  
  Factors ..................................................................................................................................... 27  
SWOT Analysis ....................................................................................................................... 28
Abstract

The most important benefit to smart homes is accessibility, as more connected devices can handle more daily tasks, especially the ones related to security. Some smart devices can also help in energy efficiency and power consumption saving. The only problem to be solved is the cost of this trending technology. Most of the already existing smart home devices are quite expensive, and this is where BrainyHAB comes along. BrainyHAB is a smart home system that is composed of different separated connected devices that do not require any drilling or wiring or pre-installation. Those devices, such as gas detectors, fire detectors, and burglary detectors, are all bought separately and installed simply by sticking them onto the wall with an adhesive, and they can be paired with a smartphone or any other device. In this project, as a solution to the problems stated above, my goal was to implement an affordable wireless smart home system, which doesn’t include any device that will not be useful like a gateway. The goal of this project is to find the cheapest and simplest alternative to a conventional smart home system. The conducted research led to finding an alternative of communication since the gateway was omitted, so MQTT was discovered, and replaced HTTP requests. In addition, the server side was to be hosted in the cloud for more than the reason of no gateway; it was also hosted in the cloud to give the advantage of centralized information and their accessibility from any device and any network.
Introduction

Smart devices are here for a while. First, Smart PDAs, then smartphones, then smart car devices, and now, smart homes. Making our home smarter is getting easier and easier; however, smart home systems are still expensive to acquire, and difficult to install. The goal of this project is to build a smart home system that works out of the box, does not require drilling or wiring, no pre-installations, and no power lines. This system will have different devices that are connected wirelessly, and operating on long lasting Li-Po batteries. In this report, the reader will find different phases that I’ve gone through before starting the implementation. The first phase is research; as you probably know, there are always different methods of implementation, especially in the field of computer science; however, a successful project is the one realized by the most efficient, and easiest way. The aim here is to simplify the work by taking advantage of software reuse, and this is done by using open source platforms and frameworks, instead of building from scratch. In this report, I will start by talking about designing an architecture, talking about its different approaches, and issues encountered in each one, then go through the analysis phase, where I will be benchmarking different outcomes and comparing them with the aimed results. After choosing the right approach, I will perform a feasibility study, proof of concept, STEEPLE analysis, and work left to be done.
Project Specifications*

The analysis phase of this project comprises the feasibility of implementing each device by itself, being independently connected with the server on the cloud, and autonomously maintained without compromising the rest of the system. This part has been already done in a 2-year research. The design of the system will be as follows: all devices are powered with a battery and connected using WiFi, to a server on the cloud, to be able to control it over the internet and not having to be physically present. The devices are the following: Gas sensor (including CO gas sensor, Methane/Butane sensor, and smoke sensor), a fire sensor, a temperature sensor, a shock sensor (for windows’ security), a water leakage sensor, a water flow sensor (for water consumption), an IR motion detector, an IP camera, a keypad lock for the main door, and a magnetic sensor for intrusion sensing. Because of lack of time and resources, the devices implemented in the project will be: a smart plug, and a gas sensor. There have been other devices omitted from the project (like heating and isolation devices, solar panel, backup battery, current sensor-for power consumption- and others) but will be included in future use. The smart plugs will be composed of an ESP8266 controller, and a relay. The relay will replace standard outlets to make them controlled remotely. The user can shut down any device, appliance or light bulb at home from anywhere, as s/he can also open and close the door remotely, all this from a dynamic dashboard on the cloud, that can be accessed from any laptop or mobile phone. The doors will still keep the traditional key system in case of an emergency (power cut, dead batteries). The server (dashboard) will allow the user to read and store all data, as it will serve as a black box for chaotic events. The project will consist of 2 parts: the electronic design, and the
software implementation. Each device will be hooked to an ESP8266 (as a replacement for Arduino with WiFi shield), that will be exchanging data with the server using the MQTT protocol (instead of HTTP) because it is faster and less energy consuming. The web server will host a web platform that controls all the devices that are connected to the system, as well as monitoring statistics about the house (temperature, humidity, water consumption, etc.).


MQTT, or Message Queuing Telemetry Transport, is a publish-subscribe mail protocol based on the TCP/IP protocol, built by IBM in 1999. This protocol was originally created for M2M, or Machine To Machine communication, but it can be used in other purposes (i.e. Facebook Messenger uses MQTT).

To establish a connection between two or more devices, one can subscribe so a specific topic, and shall receive any message that was sent to that topic; and this can work in either directions. A sensor can be subscribed to a specific topic called /sensor/#, and a device can send any data to that sensor using that topic. But the sensor can also send data to the device or the server by sending information to /server/# or /phone. This was we can either have a user control a lamp by sending data to a smart plug, or a temperature sensor sending temperature and humidity information to the server. To perform this kind of communication, we need something that could coordinate the data flow for all devices. This thing is called a Broker. “Depending on the concrete implementation, a broker can handle up to thousands of concurrently connected MQTT clients.

The broker is primarily responsible for receiving all messages, filtering them, decide who is interested
in it and then sending the message to all subscribed clients. It also holds the session of all persisted clients including subscriptions and missed messages. Another responsibility of the broker is the authentication and authorization of clients. And at most of the times a broker is also extensible, which allows to easily integrate custom authentication, authorization and integration into backend systems. Especially the integration is an important aspect, because often the broker is the component, which is directly exposed on the internet and handles a lot of clients and then passes messages along to downstream analyzing and processing systems. [...] All in all the broker is the central hub, which every message needs to pass. Therefore it is important, that it is highly scalable, integratable into backend systems, easy to monitor and of course failure-resistant.” - MQTT Essentials Part 3: Client, Broker and Connection Establishment. (2015, December 20). Retrieved April 22, 2017, from http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment

There are many MQTT brokers out there, for the purpose of this project, we are using as much open source software as we can get. In this project, we are using Mosquitto, an open source MQTT broker, hosted on a cloud server in Frankfurt, Germany, sitting on a Ubuntu 16 64 bit operating system, with a support of an SSH key. To subscribe to a topic, a simple command line
is executed: mosquito_sub -t “some/random/topic”. There are many variations to this command line, you can find them in the appendix at the end of this report.

To send data to a specific topic, we use the following command line: mosquito_pub -t “some/random/topic” -m message_content. We can use the double quotes in the message if it contains spaces.

![MQTT OSI Layers Diagram](http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment)
Advantages of MQTT over HTTP/HTTPS

MQTT is used for its speed, lower power consumption, and simplicity. MQTT is a binary-based protocol, whereas HTTP is text-based, which required more parsing, and has more overhead which goes against our goal. Here is a simple comparison between the two:

![HTTP vs MQTT](image)

With the help of some benchmarking tools from github and other sources, here is a detailed and true look at the real difference between the two protocols and why MQTT is the right choice for us. Here are some preliminary results:

**Cellular Data (4G)**

<table>
<thead>
<tr>
<th></th>
<th>Sending</th>
<th>Receiving</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MQTT</td>
<td>HTTPS</td>
</tr>
<tr>
<td>% of battery / hour</td>
<td>16%</td>
<td>18%</td>
</tr>
<tr>
<td>Messages / hour</td>
<td>14500</td>
<td>1802</td>
</tr>
<tr>
<td>% of battery / message</td>
<td>0.00018</td>
<td>0.024</td>
</tr>
</tbody>
</table>

*Figure 3: HTTP vs MQTT*
Wi-Fi

<table>
<thead>
<tr>
<th></th>
<th>Sending</th>
<th>Receiving</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MQTT</td>
<td>HTTPS</td>
</tr>
<tr>
<td>% of battery / hour</td>
<td>4%</td>
<td>3.46%</td>
</tr>
<tr>
<td></td>
<td>3.8%</td>
<td>18%</td>
</tr>
<tr>
<td>Messages / hour</td>
<td>28490</td>
<td>3647</td>
</tr>
<tr>
<td></td>
<td>40739</td>
<td>6258</td>
</tr>
<tr>
<td>% of battery / message</td>
<td>0.00008</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>0.00019</td>
<td>0.0018</td>
</tr>
</tbody>
</table>

MQTT also has other criteria. QoS is one of them. QoS, stands for Quality of Service, is a variable (-q used in command lines as you can see in the appendix), which allows us to specify the sending mode that we desire. QoS can be set to either 0, 1 or 2. When QoS is set to 0, it means that the message should be sent at most once, which is the value by default, and the least secure. Thus, if the message was not successfully sent, it will be ignored and discarded. Even if TCP is a reliable protocol, this doesn’t mean that the message will be sent successfully. When QoS is set to 1, it means the message should be sent at least once, which means that if the message was not sent successfully, it will be sent again. When QoS is set to 1, the publisher is expecting an acknowledgement after having sent the message. The acknowledgment is in a form of PUBACK. When QoS is set to 2, it means that the message should be sent exactly once. This is the safest way of sending a message and ensuring that the message will be sent once, not more, and not less. When a publisher sends a message, it will expect a PUBREC from the receiver, then the sender will send a PUBREL back to the receiver, and finally the receiver will send a final PUBCOMP to the sender, terminating the flow. See appendix for more information.
Another point in favor of MQTT, which is retained messages. A retained message is just like any message, only is it stored with the sender so that even if a receiver was subscribed to the topic after the message was sent, it will still receive the message.

Thanks to open source software, we saved time and effort by using an open source home automation software, hosting on the cloud, on the same server where the broker is, which is called home-assistant.io. This means that there is almost no delay when a command is sent to the broker from the software since they are both on the same server, thus same IP address, thus the commands are sent to simply, localhost, or 127.0.0.1.

There are other alternatives that we tried as well like KaaProject, and openHAB, but HA (HomeAssistant) was the best choice in terms of simplicity, UI, and documentation. HA is an open source software that runs on Python 3, and available on github with more than 9221 commits, 116 releases, and 484 contributors. These numbers make the software very reliable.
Feasibility Study

i. ANALYSIS

BrainyHAB is a smart home system composed of multiple devices that are hereby referred to as “nodes” that are independently connected to a cloud based server, hereby referred to as “dashboard”. Each node is connected via Wi-Fi and powered with a battery. For the software part, the node is connected to the server and exchanges data using the MQTT protocol (Message Queuing Telemetry Transport) which is a protocol that is based on the TCP/IP protocol stack and uses the subscribe/publish method. The reason why we chose this protocol is that it is faster than HTTP or JSON because it is a binary-based protocol, not text-base; therefore, not only is it faster but also less energy consuming. The dashboard will be implemented using Python (Home-Assistant.io) and will include a simple login form to a simple member area where we will have a pairing function that pairs a particular device with a particular user. This will allow the user to monitor and control his or her house from any computer or mobile phone remotely. Each node will either have a sensor, relay/switch, or a simple smart plug. There will be one particular node that is special in design and architecture which is the PIN code keypad for the door lock. Nodes use a sensor or a switch connected with a microcontroller that uses Wi-Fi. The chosen microcontroller for the nodes is the ESP8266 model 1. The ESP has 8 pins, TX/RX for information exchange (serial), VCC and ground, 2 GPIO pins, and 2 extra pins which are RESET and CH_PD. The last two pins are optional, though useful in some cases. Depending on the sensor used, sometimes only TX and RX pins are used, and sometimes only the GPIO pins are used. For prototyping purposes, the ESP8266-12E NodeMCU will be used.
ii. TECHNICAL FEASIBILITY
   a. POWER CONSUMPTION
      The ESP8266 uses between 2.5V and 3.6V. But the recommended maximum for the
input voltage is 3.3V. Most of the sensors run on 3.3V as well but in case we use 5V sensors, we
will then need to add a voltage regulator. The board draws 80 mA, with a Li-Po battery of 3000
mAh, it could last for ~26.25 hours without charging. When implementing a sleep-mode, it could
last for days, or even months without charging.

   b. COMMUNICATION CAPACITIES
      The ESP8266 uses standard Wi-Fi capabilities (802.11 b/g/n/e/i) with security
capabilities of WPA/WPA2 and encryption capabilities of WEP/TKIP/AES. The ESP uses AT
commands in order to be communicated to or from using serial (TX/RX). Here are the AT
commands:
<table>
<thead>
<tr>
<th>Function</th>
<th>AT Command</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working</td>
<td>AT</td>
<td>OK</td>
</tr>
<tr>
<td>Restart</td>
<td>AT+RST</td>
<td>OK</td>
</tr>
<tr>
<td>Firmware version</td>
<td>AT+GMR</td>
<td>AT+GMR 0018000902 OK</td>
</tr>
<tr>
<td>Join Access Point</td>
<td>AT+CWLAP?</td>
<td>Query AT+CWLAP? +CWLAP=&quot;RochefortSurLac&quot; OK</td>
</tr>
<tr>
<td>Quit Access Point</td>
<td>AT+CWLAP?=&quot;SSID&quot;,&quot;Password&quot;</td>
<td>Query OK</td>
</tr>
<tr>
<td>Get IP Address</td>
<td>AT+CIFSR</td>
<td>AT+CIFSR 192.168.0.105 OK</td>
</tr>
<tr>
<td>Set Parameters of Access Point</td>
<td>AT+CWSAP?</td>
<td>Query ssid, pwd, ch, ecn = channel, enc = encryption</td>
</tr>
<tr>
<td>WiFi Mode</td>
<td>AT+CWMODE?</td>
<td>Query STA, AP, BOTH</td>
</tr>
<tr>
<td>Set up TCP or UDP connection</td>
<td>AT+CIPSTART=</td>
<td>Query id = 0-4, type = TCP/UDP, addr = IP address, port = port</td>
</tr>
<tr>
<td>TCP/UDP Connections</td>
<td>AT+CIPMUX?</td>
<td>Query Single, Multiple</td>
</tr>
<tr>
<td>Check join devices' IP Status</td>
<td>AT+CWLIF</td>
<td>AT+CWLIF Status</td>
</tr>
<tr>
<td>Send TCP/IP Data</td>
<td>AT+CIPSEND=&lt;length&gt;</td>
<td>AT+CIPSEND=&lt;id&gt;&lt;length&gt;</td>
</tr>
<tr>
<td>Close TCP/UDP connection</td>
<td>AT+CIPCLOSE=&lt;id&gt; or AT+CIPCLOSE</td>
<td>mode 0 to close server mode, mode 1 to open, port = port</td>
</tr>
<tr>
<td>Set as server</td>
<td>AT+CIPSERVER=&lt;mode&gt;&lt;port&gt;</td>
<td>Query &lt;time&gt;=28800 in seconds</td>
</tr>
<tr>
<td>Set the server timeout</td>
<td>AT+CIPSTO?</td>
<td>Query &lt;time&gt;=&quot;28800 in seconds</td>
</tr>
<tr>
<td>Baud Rate*</td>
<td>AT+CIOBAUD?</td>
<td>AT+CIOBAUD? +CIOBAUD=9600 OK</td>
</tr>
<tr>
<td>Check IP address</td>
<td>AT+CIFSR</td>
<td>AT+CIFSR 192.168.0.106 OK</td>
</tr>
<tr>
<td>Firmware Upgrade (from Cloud)</td>
<td>AT+CIUPDATE</td>
<td>1. +CIUPDATE:1 found server 2. +CIUPDATE:2 connect server 3. +CIUPDATE:3 got edition 4. +CIUPDATE:4 start update</td>
</tr>
<tr>
<td>Received data</td>
<td>+IPD</td>
<td>(CIPMUX=0): +IPD, &lt;len&gt;: (CIPMUX=1): +IPD, &lt;id&gt;, &lt;len&gt;: &lt;data&gt;</td>
</tr>
<tr>
<td>Watchdog Enable*</td>
<td>AT+CSYSWDTENABLE</td>
<td>Watchdog, auto restart when program errors occur: enable</td>
</tr>
<tr>
<td>Watchdog Disable*</td>
<td>AT+CSYSWDTDISABLE</td>
<td>Watchdog, auto restart when program errors occur: disable</td>
</tr>
</tbody>
</table>

*Figure 4: ESP8266 AT instruction set (Expressif Systems ©2016)*
c. **ECONOMIC FEASIBILITY**

The ESP8266 boards cost 2.46 USD at minimum, and 8 USD in Morocco. The cheapest device or node that could be manufactured will cost no more than 20 USD (all included). The expected budget spent for this project is around 247 USD. Total budget spent is 1970 MAD (+ 500 MAD (risk)) \(\rightarrow\) 2470 MAD.

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Cost (MAD)</th>
<th>Amount</th>
<th>Total Cost (MAD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP8266</td>
<td>80</td>
<td>7</td>
<td>560</td>
</tr>
<tr>
<td>Temperature Sensor</td>
<td>60</td>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td>Humidity Sensor</td>
<td>55</td>
<td>1</td>
<td>55</td>
</tr>
<tr>
<td>Gas Sensor</td>
<td>50</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>Magnetic Sensor</td>
<td>50</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>IR Sensor</td>
<td>40</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>Keypad</td>
<td>45</td>
<td>1</td>
<td>45</td>
</tr>
<tr>
<td>Stepper Motor</td>
<td>195</td>
<td>1</td>
<td>195</td>
</tr>
<tr>
<td>Arduino UNO (for prototyping)</td>
<td>140</td>
<td>1</td>
<td>140</td>
</tr>
<tr>
<td>PCB prototyping board</td>
<td>10</td>
<td>7</td>
<td>70</td>
</tr>
<tr>
<td>Relay</td>
<td>20</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>Jumper cables 70 (M-M)</td>
<td>15</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>Jumper cables 40 (M-F)</td>
<td>15</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>Jumper cables 40 (F-F)</td>
<td>15</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>LEDs 10</td>
<td>5</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1370</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Service</th>
<th>Cost (MAD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Hosting (1 year)</td>
<td>600</td>
</tr>
</tbody>
</table>

d. **LEGAL FEASIBILITY**

All software and hardware used in this project are Open Source, some are under GPL License. Codes used are all taken from public github repositories and open source solutions.

e. **OPERATIONAL FEASIBILITY**

Some nodes may not be implemented within the final prototype manufacturing process for operational issues. For example, the keypad door locking system needs an actual door and a mechanical lock. The keypad is connected to an ESP8266 board that controls a stepper motor that turns the lock when told to.
f. SCHEDULE FEASIBILITY

The first sensor will be operational 9 days after the arrival of all components. This will be our proof of concept (gas sensor), then the rest of the project will be based on that. By March 31st 2017, all sensors should be implemented and connected to each other.


Architecture

This is a simple architecture of the project. The nodes will be independently connected to the server on the cloud, and each one of them will be using an ESP microcontroller. Each ESP will be connected with a WiFi access point, and will be assigned a unique IP address, and will be using the TCP/IP protocol stack to communicate with the cloud server using its floating IP address. In this project, there is the hardware part (sensors and microcontrollers), and the software part (web application and mobile application).
Proof of Concept

To be able to make sure that the laid architecture is working, a proof of concept was needed. The proof of concept will be one node, which is representing a smart plug. This node will be composed by a 5VDC/220VAC relay, an ESP8266-12E microcontroller, and an appliance of choice. The ESP will be communicating with a web application that is hosted in a cloud server, on DigitalOcean™. The web application will use the MQTT broker to communicate with the relay. When we send the command “mosquito_pub –t “ESP8266/smart” –m on”, the relay will close the circuit and the appliance will turn on. The ESP8266 was programmed with a special firmware called NodeMCU. This firmware was chosen because it allows the developer to implement the code using a programming language called LUA. With the help of a LUA IDE, we can send functions and not only AT commands, directly to the ESP. The following schematics as well as the wiring diagram represent the proof of concept.
Wiring Diagram

Figure 6: Wiring diagram of the smart plug

Schematics

Figure 7: Schematics of the smart plug
Software Implementation

Server-side

To illustrate the working process of the Home-Assistant.io software in the server side, let’s start by viewing a general diagram:

![Diagram of HA architecture](image)

In HA, there are components that we can add directly without the need of coding. Unfortunately, in our case, we should implement them manually. There is a file called configuration.yaml that we can use to add components and also configure the core of the software. To add an MQTT component, we added the following code:
To connect to our broker we use:

```
mqtt:
  broker: 46.101.132.231
```

For using a switch like we did in the prototype, we add the following lines:

```
switch:
  - platform: mqtt
    command_topic: "ESP8266/smart"
```

The command_topic is the one we should use in the hardware code (the client-side software) which is hardcoded in the ESP8266 microcontrollers.

**Client-side**

The client-side of the software, is embedded in the hardware, hardcoded in the ESP8266 microcontrollers. Here is the full code of a single ESP8266 node, hooked to a relay, which then constitutes a smart plug:

```cpp
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = "auiGuest"; // WiFi SSID
const char* password = ""; // WiFi PASS
const char* mqtt_server = "46.101.132.231"; // MQTT BROKER
int i = 0;

WiFiClient espClient;
PubSubClient client(espClient);
```
void setup() {
    pinMode(12, INPUT); // D6
    pinMode(4, OUTPUT); // D2
    Serial.begin(115200);
    setup_wifi();
    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);
    reconnect();
}

void setup_wifi(){
delay(10);

    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("\n");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
void callb(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message arrived [");
    Serial.print(topic);
    Serial.print("]");
    for (int i = 0; i < length; i++) {
        Serial.print((char)payload[i]);
    }

    if((char)payload[0] == 'o' && (char)payload[1] == 'n') //on
        digitalWrite(4,HIGH);
    else if((char)payload[0] == 'o' && (char)payload[1] == 'f' && (char)payload[2] == 'f') //off
        digitalWrite(4,LOW);

    Serial.println();
}

void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("ESP8266Client")) {
            Serial.println("connected");
            // Once connected, publish an announcement...
            client.publish("ESP8266/connection status", "Connected!");
        }
    }
}
First, we include the two important libraries: **ESP8266WiFi.h** and **PubSubClient.h**. The first library contains functions that are used to connect our microcontroller to the WiFi network. We define variables of SSID and password and we use the function WiFi.begin with the variables used to connect the ESP8266 to WiFi. The second library is responsible for functions used for MQTT purposes. We need some functions that are able to either publish messages or receive them. And to be able to receive messages, we need to be subscribed to a specific topic. Of course we need the broker’s IP address for these operations.
STEEPLE Analysis

Definition

“STEEPLE is one of the concepts that are used to analyze the macro-environmental factors that are used in business analysis. It is an extension of PEST model. STEEPLE is very often used with the SWOT analysis, which acts as an analysis of internal factors.”

**Factors**

<table>
<thead>
<tr>
<th>Social</th>
<th>Technological</th>
<th>Environmental</th>
<th>Economic</th>
<th>Political</th>
<th>Ethical</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The new way of creating smart technologies is IoT. Internet of Things is connecting physical objects and building interactions with the real world. This concept allows us to communicate with day-to-day objects like coffee makers, doors, cars, and so on.</td>
<td>Especially with the water-leakage sensors and power consumption sensors, this system will allow us to save a lot of money and save the environment. Also as part of the future works that may or may not be included in the capstone, a ‘smart’ solar panel will be included. A smart solar panel is a dynamic panel that follows the sun from sunrise to sunset, to optimize and maximize the intake of energy</td>
<td>By having an efficient home, power consumption and water consumption will also have an impact on the economic growth of the users by reducing their bills every month, hence optimizing their revenues and savings. Also, a new technology which will be used in every home, and that is the cheapest in the market, will create a whole new job market and also an impressive positive cash flow.</td>
<td>Advancing Morocco to the top ranks of Africa of being a country that actually manufactures smart systems, instead of importing foreign products.</td>
<td></td>
</tr>
</tbody>
</table>
SWOT Analysis

**Strengths**: Cheapest Smart Home System in the market, does not require wiring or drilling, easy to install

**Weaknesses**: A WiFi Router has to be available in the house

**Opportunities**: Having the possibility to access other markets like security devices, smart cars, smart cities, and so on

**Threats**: The possibility of having a competitor with better resources because the project is not patented

*Figure 9: SWOT analysis*

Future Works

Besides the various sensors that we mentioned previously, other components like the dynamic solar panel could be added to enhance our product. Also, devices could be featured and embedded in newly built residences with the cooperation with real estate companies, who will offer their clients smart homes on the go. This capstone project will allow me to continue working on Internet Of Things projects, similar to this one, and extend them to Smart Cars and Smart Cities. Another project in mind will be about Smart Transportation, which will include sending packages between cities, in a medium other than trucks. This medium will be a small tunnel going all the way from a city to another. In this tunnel, there will be a capsule, outside the capsule and inside
the tunnel, will be vacuum, with the help of Newtonian mechanics (pneumatic), the capsule will travel in the medium at a very high speed allowing the package to arrive to its destination on the same day.

**Challenges**

The issues I found in this project were the following: lack of online resources, lack of open source software, lack of material, and lack of tools in AUI laboratories. These bumps participated in slowing down the performance of the implementation of the project, as well as been the de facto reason of not being able to include all different sensors in the prototype as it was intended to be.
Conclusion

BrainyHAB is nothing but an enhancement on the technology we are already experiencing and discovering its growth. Most of the population who would like to use this type of products, would certainly hate the process of calling the technician to come drill holes and wire devices all over the walls, crushing the esthetics and ruining the purpose of a ‘Home’. With this particular technology, and with the help of cloud computing, BrainyHAB could be, any time soon, a successful business.
References


https://home-assistant.io/developers/architecture/
Figures

FIGURE 1 (HTTPS://TNEWSTACK.IO/MQTT-PROTOCOL-IOT/) 9
FIGURE 2 (HTTP://WWW.HIVEMQ.COM/BLOG/MQTT-ESSENTIALS-PART-3-CLIENT-BROKER-CONNECTION-ESTABLISHMENT) 10
FIGURE 3: HTTP VS MQTT 11
FIGURE 4: ESP8266 AT INSTRUCTION SET (EXPRESSIF SYSTEMS ©2016) 16
FIGURE 5: ARCHITECTURE 18
FIGURE 6: WIRING DIAGRAM OF THE SMART PLUG 20
FIGURE 7: SCHEMATICS OF THE SMART PLUG 20
FIGURE 8: OVERVIEW OF THE HA ARCHITECTURE (BY PAULUS SCHOUTSEN ©2014) 21
FIGURE 9: SWOT ANALYSIS 28Z
Appendix

Mosquitto command lines (source: Mosquitto.org)

Subscription

- `A`

Bind the outgoing connection to a local ip address/hostname. Use this argument if you need to restrict network communication to a particular interface.

- `-c, --disable-clean-session`

Disable the 'clean session' flag. This means that all of the subscriptions for the client will be maintained after it disconnects, along with subsequent QoS 1 and QoS 2 messages that arrive. When the client reconnects, it will receive all of the queued messages.

If using this option, it is recommended that the client id is set manually with `--id`

- `--cafile`

Define the path to a file containing PEM encoded CA certificates that are trusted. Used to enable SSL communication.

See also `--capath`

- `--capath`

Define the path to a directory containing PEM encoded CA certificates that are trusted. Used to enable SSL communication.

For `--capath` to work correctly, the certificate files must have ".crt" as the file ending and you must run "c_rehash <path to capath>" each time you add/remove a certificate.

See also `--cafile`

- `--cert`

Define the path to a file containing a PEM encoded certificate for this client, if required by the server.

See also `--key`.

- `--ciphers`

An openssl compatible list of TLS ciphers to support in the client. See ciphers(1) for more information.
-C

Disconnect and exit the program immediately after the given count of messages have been received. This may be useful in shell scripts where on a single status value is required, for example.

Combine with -R to print only the first set of fresh messages (i.e. that does not have the retained flag set), or with -T to filter which topics are processed.

-d, --debug

Enable debug messages.

--help

Display usage information.

-h, --host

Specify the host to connect to. Defaults to localhost.

-i, --id

The id to use for this client. If not given, defaults to mosquitto_sub_ appended with the process id of the client. Cannot be used at the same time as the --id-prefix argument.

-I, --id-prefix

Provide a prefix that the client id will be built from by appending the process id of the client. This is useful where the broker is using the clientid_prefixes option. Cannot be used at the same time as the --id argument.

--insecure

When using certificate based encryption, this option disables verification of the server hostname in the server certificate. This can be useful when testing initial server configurations but makes it possible for a malicious third party to impersonate your server through DNS spoofing, for example. Use this option in testing only. If you need to resort to using this option in a production environment, your setup is at fault and there is no point using encryption.

-k, --keepalive

The number of seconds between sending PING commands to the broker for the purposes of informing it we are still connected and functioning. Defaults to 60 seconds.
--key

Define the path to a file containing a PEM encoded private key for this client, if required by the server.

See also --cert.

-N

Do not append an end of line character to the payload when printing. This allows streaming of payload data from multiple messages directly to another application unmodified. Only really makes sense when not using -v.

-p, --port

Connect to the port specified instead of the default 1883.

-P, --pw

Provide a password to be used for authenticating with the broker. Using this argument without also specifying a username is invalid. This requires a broker that supports MQTT v3.1. See also the --username option.

--proxy

Specify a SOCKS5 proxy to connect through. "None" and "username" authentication types are supported. The socks-url must be of the form socks5h://[username[:password]@]host[:port]. The protocol prefix socks5h means that hostnames are resolved by the proxy. The symbols %25, %3A and %40 are URL decoded into %, : and @ respectively, if present in the username or password.

If username is not given, then no authentication is attempted. If the port is not given, then the default of 1080 is used.

More SOCKS versions may be available in the future, depending on demand, and will use different protocol prefixes as described in curl(1).

--psk

Provide the hexadecimal (no leading 0x) pre-shared-key matching the one used on the broker to use TLS-PSK encryption support. --psk-identity must also be provided to enable TLS-PSK.

--psk-identity
The client identity to use with TLS-PSK support. This may be used instead of a username if the broker is configured to do so.

-q, --qos

Specify the quality of service desired for the incoming messages, from 0, 1 and 2. Defaults to 0. See mqtt(7) for more information on QoS.

The QoS is identical for all topics subscribed to in a single instance of mosquitto_sub.

--quiet

If this argument is given, no runtime errors will be printed. This excludes any error messages given in case of invalid user input (e.g. using --port without a port).

-R

If this argument is given, messages that are received that have the retain bit set will not be printed. Messages with retain set are "stale", in that it is not known when they were originally published. When subscribing to a wildcard topic there may be a large number of retained messages. This argument suppresses their display.

-S

Use SRV lookups to determine which host to connect to. Performs lookups to _mqtt._tcp.<host> when used in conjunction with -h, otherwise uses _mqtt._tcp.<local dns domain>.

-t, --topic

The MQTT topic to subscribe to. See mqtt(7) for more information on MQTT topics.

This option may be repeated to subscribe to multiple topics.

-T, --filter-out

Suppress printing of topics that match the filter. This allows subscribing to a wildcard topic and only printing a partial set of the wildcard hierarchy.

For example, subscribe to the BBC tree, but suppress output from Radio 3:

  o  mosquitto_sub -t bbc/# -T bbc/radio3

This option may be repeated to filter out multiple topics or topic trees.

--tls-version
Choose which TLS protocol version to use when communicating with the broker. Valid options are `tlsv1.2`, `tlsv1.1` and `tlsv1`. The default value is `tlsv1.2`. If the installed version of openssl is too old, only `tlsv1` will be available. Must match the protocol version used by the broker.

```
-u, --username
```

Provide a username to be used for authenticating with the broker. This requires a broker that supports MQTT v3.1. See also the `--pw` argument.

```
-v, --verbose
```

Print received messages verbosely. With this argument, messages will be printed as "topic payload". When this argument is not given, the messages are printed as "payload".

```
-V, --protocol-version
```

Specify which version of the MQTT protocol should be used when connecting to the remote broker. Can be `mqttv31` or `mqttv311`. Defaults to `mqttv31`.

```
--will-payload
```

Specify a message that will be stored by the broker and sent out if this client disconnects unexpectedly. This must be used in conjunction with `--will-topic`.

```
--will-qos
```

The QoS to use for the Will. Defaults to 0. This must be used in conjunction with `--will-topic`.

```
--will,retain
```

If given, if the client disconnects unexpectedly the message sent out will be treated as a retained message. This must be used in conjunction with `--will-topic`.

```
--will-topic
```

The topic on which to send a Will, in the event that the client disconnects unexpectedly.

**Publication**

```
-A
```

Bind the outgoing connection to a local ip address/hostname. Use this argument if you need to restrict network communication to a particular interface.
--cafile

Define the path to a file containing PEM encoded CA certificates that are trusted. Used to enable SSL communication.

See also --capath

--capath

Define the path to a directory containing PEM encoded CA certificates that are trusted. Used to enable SSL communication.

For --capath to work correctly, the certificate files must have ".crt" as the file ending and you must run "c_rehash <path to capath>" each time you add/remove a certificate.

See also --cafile

--cert

Define the path to a file containing a PEM encoded certificate for this client, if required by the server.

See also --key.

--ciphers

An openssl compatible list of TLS ciphers to support in the client. See ciphers(1) for more information.

-d, --debug

Enable debug messages.

-f, --file

Send the contents of a file as the message.

--help

Display usage information.

-h, --host

Specify the host to connect to. Defaults to localhost.

-i, --id
The id to use for this client. If not given, defaults to mosquitto_pub_ appended with the process id of the client. Cannot be used at the same time as the --id-prefix argument.

-I, --id-prefix

Provide a prefix that the client id will be built from by appending the process id of the client. This is useful where the broker is using the clientid_prefixes option. Cannot be used at the same time as the --id argument.

--insecure

When using certificate based encryption, this option disables verification of the server hostname in the server certificate. This can be useful when testing initial server configurations but makes it possible for a malicious third party to impersonate your server through DNS spoofing, for example. Use this option in testing only. If you need to resort to using this option in a production environment, your setup is at fault and there is no point using encryption.

-k, --keepalive

The number of seconds between sending PING commands to the broker for the purposes of informing it we are still connected and functioning. Defaults to 60 seconds.

--key

Define the path to a file containing a PEM encoded private key for this client, if required by the server.

See also --cert.

-1, --stdin-line

Send messages read from stdin, splitting separate lines into separate messages. Note that blank lines won't be sent.

-m, --message

Send a single message from the command line.

-n, --null-message

Send a null (zero length) message.

-p, --port
Connect to the port specified instead of the default 1883.

-P, --pw

Provide a password to be used for authenticating with the broker. Using this argument without also specifying a username is invalid. This requires a broker that supports MQTT v3.1. See also the --username option.

--proxy

Specify a SOCKS5 proxy to connect through. "None" and "username" authentication types are supported. The socks-url must be of the form socks5h://[username[:password]@[host[:port]]. The protocol prefix socks5h means that hostnames are resolved by the proxy. The symbols %25, %3A and %40 are URL decoded into %, : and @ respectively, if present in the username or password.

If username is not given, then no authentication is attempted. If the port is not given, then the default of 1080 is used.

More SOCKS versions may be available in the future, depending on demand, and will use different protocol prefixes as described in curl(1).

--psk

Provide the hexadecimal (no leading 0x) pre-shared-key matching the one used on the broker to use TLS-PSK encryption support. --psk-identity must also be provided to enable TLS-PSK.

--psk-identity

The client identity to use with TLS-PSK support. This may be used instead of a username if the broker is configured to do so.

-q, --qos

Specify the quality of service to use for the message, from 0, 1 and 2. Defaults to 0.

--quiet

If this argument is given, no runtime errors will be printed. This excludes any error messages given in case of invalid user input (e.g. using --port without a port).

-r, --retain

If retain is given, the message will be retained as a "last known good" value on the broker. See mqtt(7) for more information.
-s, --stdin-file

Send a message read from stdin, sending the entire content as a single message.

-S

Use SRV lookups to determine which host to connect to. Performs lookups to _mqtt._tcp.<host> when used in conjunction with -h, otherwise uses _mqtt._tcp.<local dns domain>.

-t, --topic

The MQTT topic on which to publish the message. See mqtt(7) for more information on MQTT topics.

--tls-version

Choose which TLS protocol version to use when communicating with the broker. Valid options are tlsv1.2, tlsv1.1 and tlsv1. The default value is tlsv1.2. If the installed version of openssl is too old, only tlsv1 will be available. Must match the protocol version used by the broker.

-u, --username

Provide a username to be used for authenticating with the broker. This requires a broker that supports MQTT v3.1. See also the --pw argument.

-V, --protocol-version

Specify which version of the MQTT protocol should be used when connecting to the remote broker. Can be mqttv31 or mqttv311. Defaults to mqttv31.

--will-payload

Specify a message that will be stored by the broker and sent out if this client disconnects unexpectedly. This must be used in conjunction with --will-topic.

--will-qos

The QoS to use for the Will. Defaults to 0. This must be used in conjunction with --will-topic.

--will,retain

If given, if the client disconnects unexpectedly the message sent out will be treated as a retained message. This must be used in conjunction with --will-topic.
The topic on which to send a Will, in the event that the client disconnects unexpectedly.

**PUBACK – Publish acknowledgement** (source: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html)

A PUBACK Packet is the response to a PUBLISH Packet with QoS level 1.

**Fixed header**

**PUBACK Packet fixed header**

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte 1</td>
<td>MQTT Control Packet type (4)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>byte 2</td>
<td>Remaining Length (2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Remaining Length field**

This is the length of the variable header. For the PUBACK Packet this has the value 2.

**Variable header**

This contains the Packet Identifier from the PUBLISH Packet that is being acknowledged.

**PUBACK Packet variable header**

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte 1</td>
<td>Packet Identifier MSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>byte 2</td>
<td>Packet Identifier LSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Payload**

The PUBACK Packet has no payload.

**PUBREC – Publish received (QoS 2 publish received, part 1)**

A PUBREC Packet is the response to a PUBLISH Packet with QoS 2. It is the second packet of the QoS 2 protocol exchange.
Fixed header

PUBREC Packet fixed header

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte 1</td>
<td>MQTT Control Packet type (5)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>byte 2</td>
<td>Remaining Length (2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Remaining Length field

This is the length of the variable header. For the PUBREC Packet this has the value 2.

Variable header

The variable header contains the Packet Identifier from the PUBLISH Packet that is being acknowledged.

PUBREC Packet variable header

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte 1</td>
<td>Packet Identifier MSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>byte 2</td>
<td>Packet Identifier LSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Payload

The PUBREC Packet has no payload.

PUBREL – Publish release (QoS 2 publish received, part 2)

A PUBREL Packet is the response to a PUBREC Packet. It is the third packet of the QoS 2 protocol exchange.

Fixed header

PUBREL Packet fixed header

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte 1</td>
<td>MQTT Control Packet type (6)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>byte 2</td>
<td>Remaining Length (2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Remaining Length field

This is the length of the variable header. For the PUBREL Packet this has the value 2.

Variable header

The variable header contains the same Packet Identifier as the PUBREC Packet that is being acknowledged.

PUBREL Packet variable header

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Packet Identifier MSB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>byte 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Packet Identifier LSB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Payload

The PUBREL Packet has no payload.

PUBCOMP – Publish complete (QoS 2 publish received, part 3)

The PUBCOMP Packet is the response to a PUBREL Packet. It is the fourth and final packet of the QoS 2 protocol exchange.

Fixed header

PUBCOMP Packet fixed header

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte 1</td>
<td>MQTT Control Packet type (7)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>byte 2</td>
<td>Remaining Length (2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Remaining Length field

This is the length of the variable header. For the PUBCOMP Packet this has the value 2.
Variable header

The variable header contains the same Packet Identifier as the PUBREL Packet that is being acknowledged.

**PUBCOMP Packet variable header**

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Packet Identifier MSB</td>
</tr>
<tr>
<td>byte 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Packet Identifier LSB</td>
</tr>
</tbody>
</table>

Payload

The PUBCOMP Packet has no payload.