MQTT Based Remote Gas level Monitoring

Project Done By: Archisman Majumdar (179),Abanindra Kumar Mandal (180) ,Vanshika Agarwal (181)

MINI-PROJECT REPORT

OBJECTIVE : Remotely monitoring gas levels over the internet

INTRODUCTION:

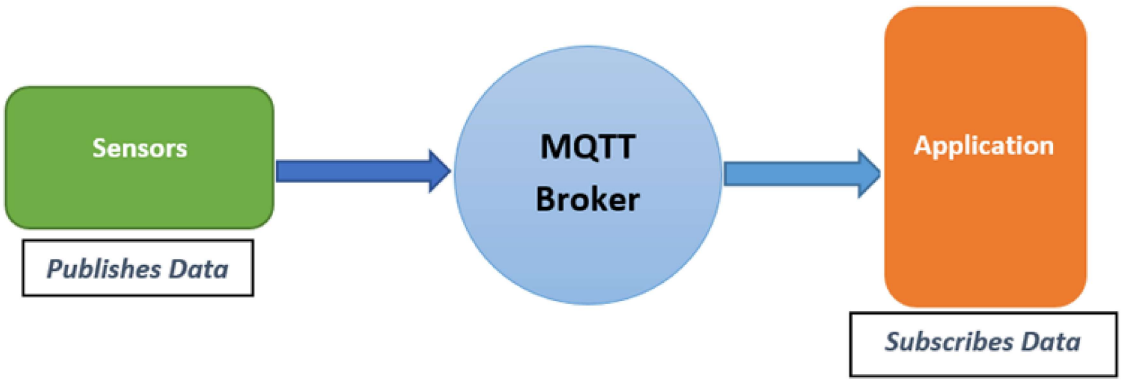
Industrial environments often require live monitoring of gas levels. However, they also provide a unique challenge of unreliable infrastructure in terms of wireless communication due to poor network layout as well as interference from industrial tools. We aim to utilize the MQTT protocol which is not bandwidth sensitive to live monitor the gas levels using a MQ135 sensor.
The data is sent to a cloud server (hiveMQ cloud MQTT broker) which then forwards the data to client sides. The applications of such a system provides live monitoring from consolidated sources as well as greater safety with less resources.

REQUIRED COMPONENTS:

- MQ135 sensor
- Arduino
- ESP8266 wifi module
- Breadboard
- Connecting Wires

What is MQTT protocol?

MQTT (Message Queuing Telemetry Transport) protocol. It is a lightweight messaging protocol that uses the publish/subscribe method and translates messages between multiple devices. Using MQTT protocol, we can also send/receive data and control various output devices, like read sensor data, etc. It's developed on top of TCP, which is why it's faster than similar protocols like HTTP. Other than that, it has many other advantages over other protocols like its very lightweight, so it doesn't consume excess memory, it can work with very less network bandwidth, on top of that, it has a robust security protocol inbuilt. These features make it suitable for many applications.



How MQTT Works?

In order to understand the working of the MQTT protocol, we just need to understand three basic things; the above diagram shows that.

MQTT Client:

An MQTT client is any device (it can be a microcontroller or a server) that runs MQTT functions and communicates with a central server, which is known as the "broker."

MQTT Publisher:

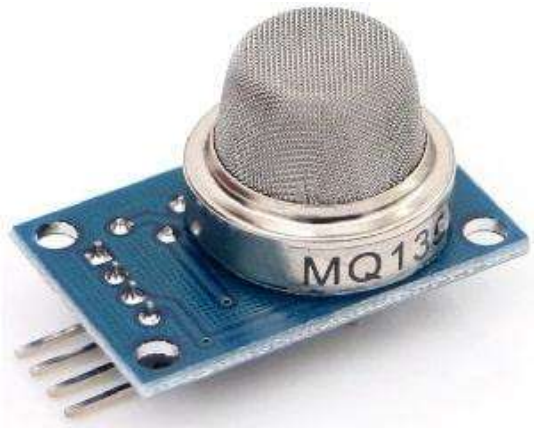
When a client wants to send any information, the client is known as a "Publisher".

MQTT Subscriber:

The MQTT Subscriber subscribes to topics on an MQTT broker to read the messages sent by the broker.

MQ135 GAS SENSOR

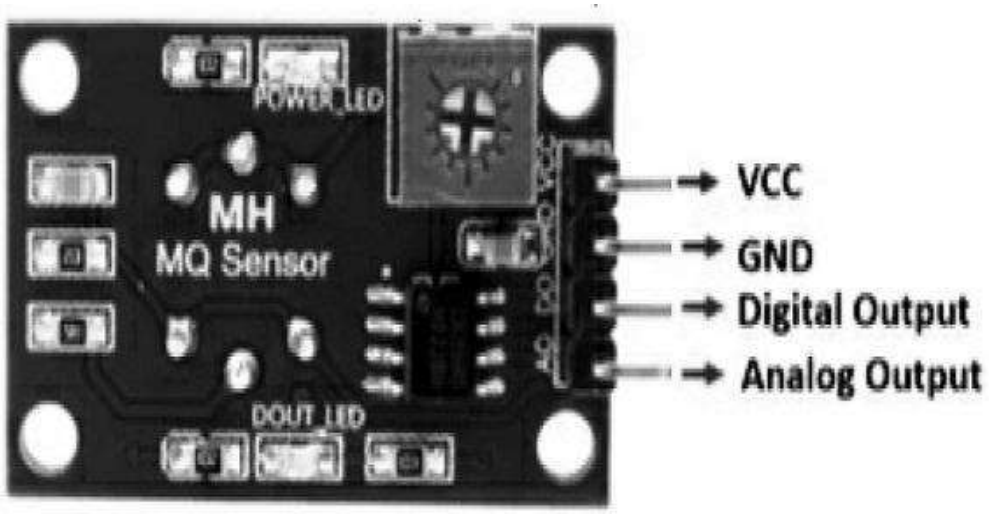
The MQ-135 gas sensor senses gases like ammonia nitrogen, oxygen, alcohols, aromatic compounds, sulfide, and smoke. The MQ-3 gas sensor has a lower conductivity to clean the air as a gas sensing material. In the atmosphere, we can find polluting gases, but the conductivity of the gas sensor increases as the concentration of polluting gas increases. MQ-135 gas sensor can be implemented to detect the smoke, benzene, steam, and other harmful gases. It has the potential to detect different harmful gases. It is low cost and is particularly suitable for Air quality monitoring applications.



Pin Configuration:

The MQ135 air quality sensor is a 4-pin sensor module that features both analog and digital output from the corresponding pins. The MQ135 air quality sensor pin configuration is shown below.

For MQ135 Air Quality Sensor Module:



The MQ135 air quality sensor module is shown below.

Pin 1: VCC: This pin refers to a positive power supply of 5V that power up the MQ135 sensor module.

Pin 2: GND (Ground): This is a reference potential pin, which connects the MQ135 sensor module to the ground.

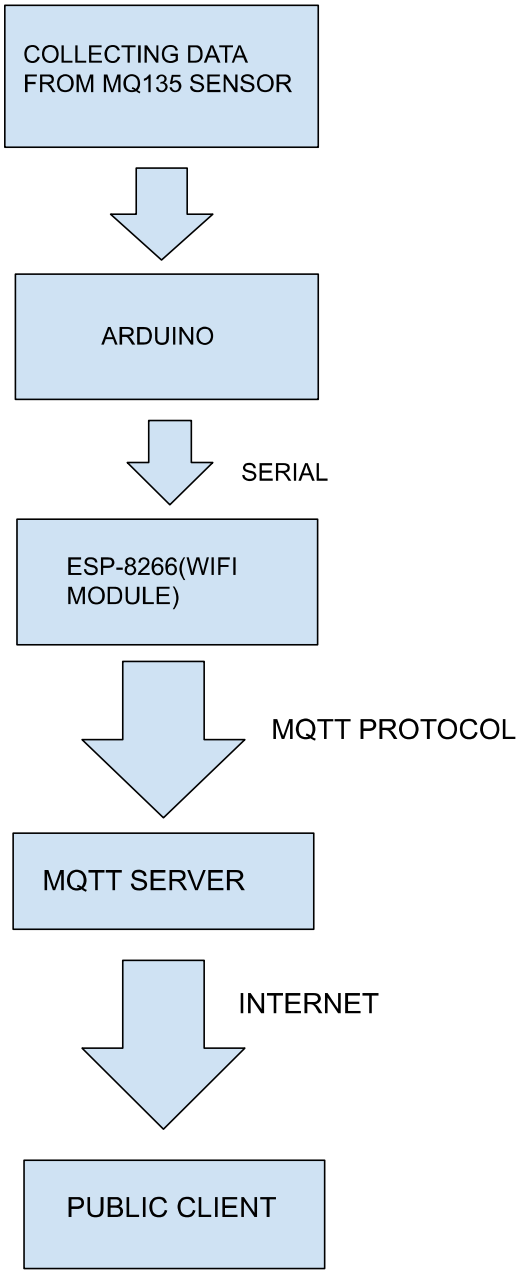
Pin 3: Digital Out (Do): This pin refers to the digital output pin that gives the digital output by adjusting the threshold value with the help of a potentiometer. This pin is used to detect and measure any one particular gas and makes the MQ135 sensor work without a microcontroller.

Pin 4: Analog Out (Ao): This pin generates the analog output signal of 0V to 5V and it depends on the gas intensity. This analog output signal is proportional to the gas vapor concentration, which is measured by the MQ135 sensor module. This pin is used to measure the gases in PPM. It is driven by TTL logic, operates with 5V, and is mostly interfaced with microcontrollers.

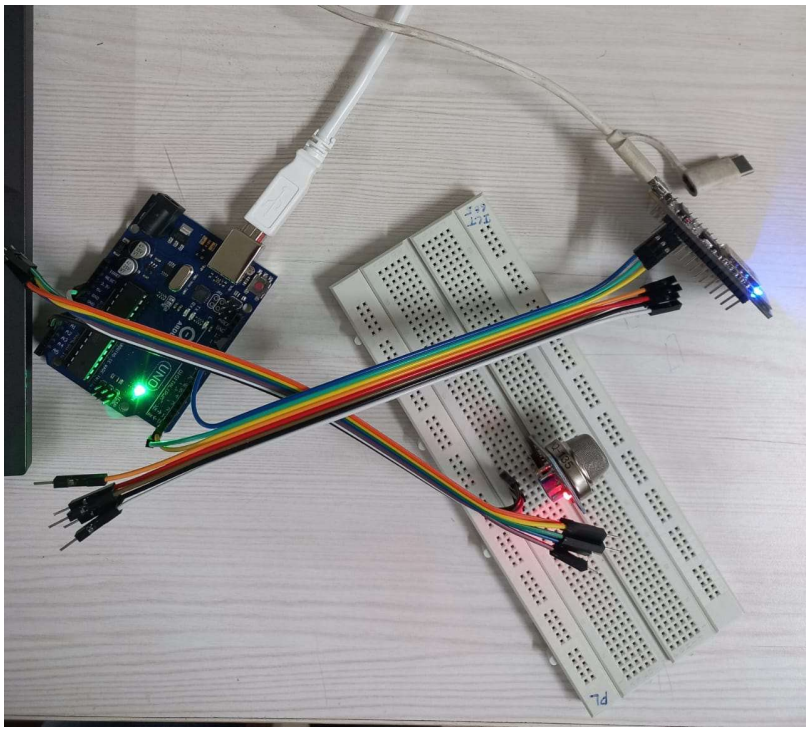
Wi-Fi ModuleESP8266:

The ESP8266 Wi-Fi Module contained SOC with integrated IP protocol stack that can give any microcontroller access to a particular Wi-Fi network. It is capable of sending the data and it is capable of offloading. We can easily connect to Arduino device. The ESP8266 module is a less priced chip.

FLOWCHART:



WORKING CIRCUIT:



Interfacing MQ-135 Gas Sensor with Arduino:

The table below shows the connections you need to make between the MQ3 sensor module and Arduino using both the analog output and the digital output pins of the sensor.

MQ-135 Module	Arduino
VCC	5V
GND	GND
A0	A0
D0	Pin 2

Connect MQ-135 sensor's VCC pin with 5V terminal of Arduino UNO. This will power up the sensor.
Additionally, we will connect the analog pin AO with A0 and DO with Pin 2 of Arduino UNO. Both the devices will be commonly grounded.
MQ 135 Sensor takes air as input and analyzes the gas particles. We have used the MQ 135 sensor for detecting the harmful gases in air and the ESP8266 wifi module to send the data to the webpage. When the power supply is given to Arduino, the sensor will turn on and it will send the data to Arduino. ESP 8266 module will send the data to the cloud through the internet when it is connected to wifi.

Arduino code:

```
#include <ArduinoJson.h>

/************************MQ135sensor*************************************/
/************************Hardware Related
Macros************************************/
#define MQ135PIN (A0)                     // define which analog input
channel you are going to use
#define RL_VALUE_MQ135 (1)                // define the load resistance on
the board, in kilo ohms
#define RO_CLEAN_AIR_FACTOR_MQ135 (3.59) // RO_CLEAR_AIR_FACTOR=(Sensor
resistance in clean air)/RO,
                                          // which is derived from the
chart in datasheet

/************************Software Related
Macros************************************/
#define CALIBARAION_SAMPLE_TIMES (50)     // define how many samples you
are going to take in the calibration phase
#define CALIBRATION_SAMPLE_INTERVAL (500) // define the time interal(in
milisecond) between each samples in the
                                           // cablibration phase
#define READ_SAMPLE_INTERVAL (50) // define how many samples you are going
to take in normal operation
#define READ_SAMPLE_TIMES (5)     // define the time interal(in
milisecond) between each samples in
                             // normal operation

/************************Application Related
Macros************************************/
#define GAS_CARBON_DIOXIDE (9)
#define GAS_CARBON_MONOXIDE (3)
#define GAS_ALCOHOL (4)
#define GAS_AMMONIUM (10)
#define GAS_TOLUENE (11)
#define GAS_ACETONE (12)
// #define         accuracy                    (0)   //for linearcurves
#define accuracy (1) // for nonlinearcurves, un comment this line and
comment the above line if calculations
                     // are to be done using non linear curve equations
/************************Globals************************************
**********/
float Ro = 10; // Ro is initialized to 10 kilo ohms

void setup()
```

```
{
  Serial.begin(9600); // UART setup, baudrate = 9600bps
  Serial.print("Calibrating...\n");
  Ro = MQCalibration(MQ135PIN); // Calibrating the sensor. Please make
sure the sensor is in clean air
                                // when you perform the calibration
  Serial.print("Calibration is done...\n");
  Serial.print("Ro=");
  Serial.print(Ro);
  Serial.print("kohm");
  Serial.print("\n");
}

void loop()
{

  DynamicJsonDocument doc(200);
  JsonArray array = doc.to<JsonArray>();
  JsonObject cdoxide = array.createNestedObject();
  cdoxide["gas"] = "Co2";
  cdoxide["ppm"] = MQGetGasPercentage(MQRead(MQ135PIN) / Ro,
GAS_CARBON_DIOXIDE);
  JsonObject cmoxide = array.createNestedObject();
  cmoxide["gas"] = "Co";
  cmoxide["ppm"] = MQGetGasPercentage(MQRead(MQ135PIN) / Ro,
GAS_CARBON_MONOXIDE);
  JsonObject alco = array.createNestedObject();
  alco["gas"] = "Alcohol";
  alco["ppm"] = MQGetGasPercentage(MQRead(MQ135PIN) / Ro, GAS_ALCOHOL);
  JsonObject amm = array.createNestedObject();
  amm["gas"] = "Ammonium";
  amm["ppm"] = MQGetGasPercentage(MQRead(MQ135PIN) / Ro, GAS_AMMONIUM);
  JsonObject tol = array.createNestedObject();
  tol["gas"] = "Toluene";
  tol["ppm"] = MQGetGasPercentage(MQRead(MQ135PIN) / Ro, GAS_TOLUENE);
  JsonObject ace = array.createNestedObject();
  ace["gas"] = "Acetone";
  ace["ppm"] = MQGetGasPercentage(MQRead(MQ135PIN) / Ro, GAS_ACETONE);
  serializeJson(array, Serial);
  Serial.print("\n");

  delay(200);
}

/****************** MQResistanceCalculation
*************************************
Input:   raw_adc - raw value read from adc, which represents the voltage
```

```
Output:  the calculated sensor resistance
Remarks: The sensor and the load resistor forms a voltage divider. Given
the voltage
         across the load resistor and its resistance, the resistance of
the sensor
         could be derived.
************************************************************************
**********/
float MQResistanceCalculation(int raw_adc)
{
  return (((float)RL_VALUE_MQ135 * (1023 - raw_adc) / raw_adc));
}

/***************************** MQCalibration
****************************************
Input:   mq_pin - analog channel
Output:  Ro of the sensor
Remarks: This function assumes that the sensor is in clean air. It use
         MQResistanceCalculation to calculates the sensor resistance in
clean air
         and then divides it with RO_CLEAN_AIR_FACTOR. RO_CLEAN_AIR_FACTOR
is about
         10, which differs slightly between different sensors.
************************************************************************
**********/
float MQCalibration(int mq_pin)
{
  int i;
  float RS_AIR_val = 0, r0;

  for (i = 0; i < CALIBARAION_SAMPLE_TIMES; i++)
  { // take multiple samples
    RS_AIR_val += MQResistanceCalculation(analogRead(mq_pin));
    delay(CALIBRATION_SAMPLE_INTERVAL);
  }
  RS_AIR_val = RS_AIR_val / CALIBARAION_SAMPLE_TIMES; // calculate the
average value

  r0 = RS_AIR_val / RO_CLEAN_AIR_FACTOR_MQ135; // RS_AIR_val divided by
RO_CLEAN_AIR_FACTOR yields the Ro
                                               // according to the chart
in the datasheet

  return r0;
}
```

```
/****************************  MQRead
*****************************************
Input:   mq_pin - analog channel
Output:  Rs of the sensor
Remarks: This function use MQResistanceCalculation to caculate the sensor
resistenc (Rs).
         The Rs changes as the sensor is in the different consentration of
the target
         gas. The sample times and the time interval between samples could
be configured
         by changing the definition of the macros.
************************************************************************
**********/
float MQRead(int mq_pin)
{
  int i;
  float rs = 0;

  for (i = 0; i < READ_SAMPLE_TIMES; i++)
  {
    rs += MQResistanceCalculation(analogRead(mq_pin));
    delay(READ_SAMPLE_INTERVAL);
  }

  rs = rs / READ_SAMPLE_TIMES;

  return rs;
}

/****************************  MQGetGasPercentage
*********************************
Input:   rs_ro_ratio - Rs divided by Ro
         gas_id      - target gas type
Output:  ppm of the target gas
Remarks: This function uses different equations representing curves of
each gas to
         calculate the ppm (parts per million) of the target gas.
************************************************************************
**********/
int MQGetGasPercentage(float rs_ro_ratio, int gas_id)
{
  if (accuracy == 0)
  {
    if (gas_id == GAS_CARBON_DIOXIDE)
    {
      return (pow(10, ((-2.890 * (log10(rs_ro_ratio))) + 2.055)));
    }
```

```
    else if (gas_id == GAS_CARBON_MONOXIDE)
    {
      return (pow(10, ((-3.891 * (log10(rs_ro_ratio))) + 2.750)));
    }
    else if (gas_id == GAS_ALCOHOL)
    {
      return (pow(10, ((-3.181 * (log10(rs_ro_ratio))) + 1.895)));
    }
    else if (gas_id == GAS_AMMONIUM)
    {
      return (pow(10, ((-2.469 * (log10(rs_ro_ratio))) + 2.005)));
    }
    else if (gas_id == GAS_TOLUENE)
    {
      return (pow(10, ((-3.479 * (log10(rs_ro_ratio))) + 1.658)));
    }
    else if (gas_id == GAS_ACETONE)
    {
      return (pow(10, ((-3.452 * (log10(rs_ro_ratio))) + 1.542)));
    }
  }

  else if (accuracy == 1)
  {
    if (gas_id == GAS_CARBON_DIOXIDE)
    {
      return (pow(10, ((-2.890 * (log10(rs_ro_ratio))) + 2.055))) * 1000;
    }
    else if (gas_id == GAS_CARBON_MONOXIDE)
    {
      return (pow(10, (1.457 * pow((log10(rs_ro_ratio)), 2) - 4.725 *
(log10(rs_ro_ratio)) + 2.855))) * 100;
    }
    else if (gas_id == GAS_ALCOHOL)
    {
      return (pow(10, ((-3.181 * (log10(rs_ro_ratio))) + 1.895))) * 1000;
    }
    else if (gas_id == GAS_AMMONIUM)
    {
      return (pow(10, ((-2.469 * (log10(rs_ro_ratio))) + 2.005))) * 1000;
    }
    else if (gas_id == GAS_TOLUENE)
    {
      return (pow(10, ((-3.479 * (log10(rs_ro_ratio))) + 1.658))) * 1000;
    }
    else if (gas_id == GAS_ACETONE)
    {
```

```
      return (pow(10, (-1.004 * pow((log10(rs_ro_ratio)), 2) - 3.525 *
(log10(rs_ro_ratio)) + 1.553))) * 1000;
    }
  }
  return 0;
}
```

esp8266(pub client code):

```
/*void setup() {
  // put your setup code here, to run once:

  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:

  if(Serial.available()>0){
    String data = Serial.readStringUntil('\n');
    Serial.println(data);
  }


}*/

#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <time.h>
#include <TZ.h>
#include <FS.h>
#include <LittleFS.h>
#include <CertStoreBearSSL.h>

// Update these with values suitable for your network.
const char *ssid = "POCOM2Pro";
const char *password = "indrajit00";
const char *mqtt_server =
"44207d5c2b4140f0b94a465b307c62a8.s2.eu.hivemq.cloud";

// A single, global CertStore which can be used by all connections.
// Needs to stay live the entire time any of the WiFiClientBearSSLs
// are present.
BearSSL::CertStore certStore;

WiFiClientSecure espClient;
```

```
PubSubClient *client;
unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE (500)
char msg[MSG_BUFFER_SIZE];
int value = 0;

void setup_wifi()
{
  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

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

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

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

void setDateTime()
{
  // You can use your own timezone, but the exact time is not used at all.
  // Only the date is needed for validating the certificates.
  configTime(TZ_Europe_Berlin, "pool.ntp.org", "time.nist.gov");

  Serial.print("Waiting for NTP time sync: ");
  time_t now = time(nullptr);
  while (now < 8 * 3600 * 2)
  {
    delay(100);
    Serial.print(".");
    now = time(nullptr);
  }
  Serial.println();
```

```
  struct tm timeinfo;
  gmtime_r(&now, &timeinfo);
  Serial.printf("%s %s", tzname[0], asctime(&timeinfo));
}

void callback(char *topic, byte *payload, unsigned int length)
{
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++)
  {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  // Switch on the LED if the first character is present
  if ((char)payload[0] != NULL)
  {
    digitalWrite(LED_BUILTIN, LOW); // Turn the LED on (Note that LOW is
the voltage level
    // but actually the LED is on; this is because
    // it is active low on the ESP-01)
    delay(500);
    digitalWrite(LED_BUILTIN, HIGH); // Turn the LED off by making the
voltage HIGH
  }
  else
  {
    digitalWrite(LED_BUILTIN, HIGH); // Turn the LED off by making the
voltage HIGH
  }
}

void reconnect()
{
  // Loop until we're reconnected
  while (!client->connected())
  {
    Serial.print("Attempting MQTT connection…");
    String clientId = "ESP8266Client - MyClient";
    // Attempt to connect
    // Insert your password
    if (client->connect(clientId.c_str(), "Gas_Level_179",
"Miniproject179"))
    {
      Serial.println("connected");
```

```
      // Once connected, publish an announcement…
      client->publish("testTopic", "hello world");
      // … and resubscribe
      client->subscribe("testTopic");
    }
    else
    {
      Serial.print("failed, rc = ");
      Serial.print(client->state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup()
{
  delay(500);
  // When opening the Serial Monitor, select 9600 Baud
  Serial.begin(9600);
  delay(500);

  LittleFS.begin();
  setup_wifi();
  setDateTime();

  pinMode(LED_BUILTIN, OUTPUT); // Initialize the LED_BUILTIN pin as an
output
  // you can use the insecure mode, when you want to avoid the
certificates
  // espclient->setInsecure();

  int numCerts = certStore.initCertStore(LittleFS, PSTR("/certs.idx"),
PSTR("/certs.ar"));
  Serial.printf("Number of CA certs read: %d\n", numCerts);
  if (numCerts == 0)
  {
    Serial.printf("No certs found. Did you run certs-from-mozilla.py and
upload the LittleFS directory before running?\n");
    return; // Can't connect to anything w/o certs!
  }

  BearSSL::WiFiClientSecure *bear = new BearSSL::WiFiClientSecure();
  // Integrate the cert store with this connection
  bear->setCertStore(&certStore);
```

```
  client = new PubSubClient(*bear);

  client->setServer(mqtt_server, 8883);
  client->setCallback(callback);
}

void loop()
{
  if (!client->connected())
  {
    reconnect();
  }
  client->loop();

  unsigned long now = millis();

  String data = Serial.readStringUntil('\n');

    sprintf(msg, "%s", data.c_str());
    if(*msg != '\n'){
    Serial.println(msg);

    client->publish("mytopic", msg);
  }
}
```

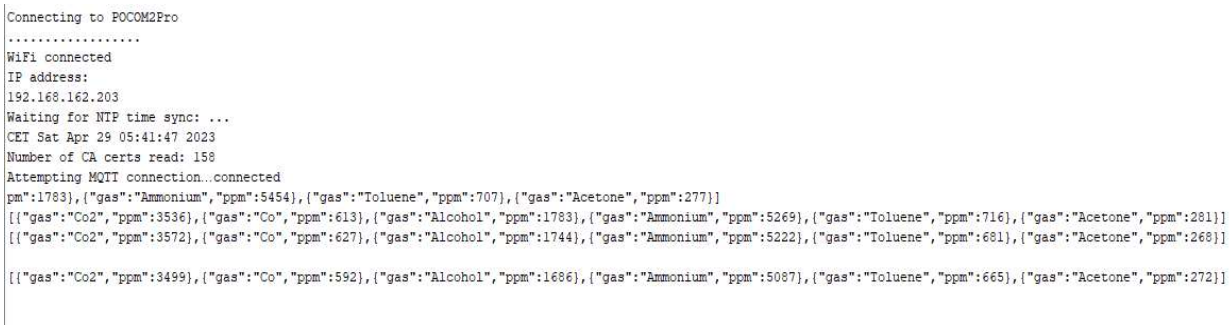
Javascript client (sub) code:

```
var mqtt = require('mqtt');
var fs = require('fs');

var options = {
    host: '44207d5c2b4140f0b94a465b307c62a8.s2.eu.hivemq.cloud',
    port: 8883,
    protocol: 'mqtts',
    username: 'Gas_Level_179_subscribe',
    password: 'Miniproject179'
}

// initialize the MQTT client
var client = mqtt.connect(options);
```

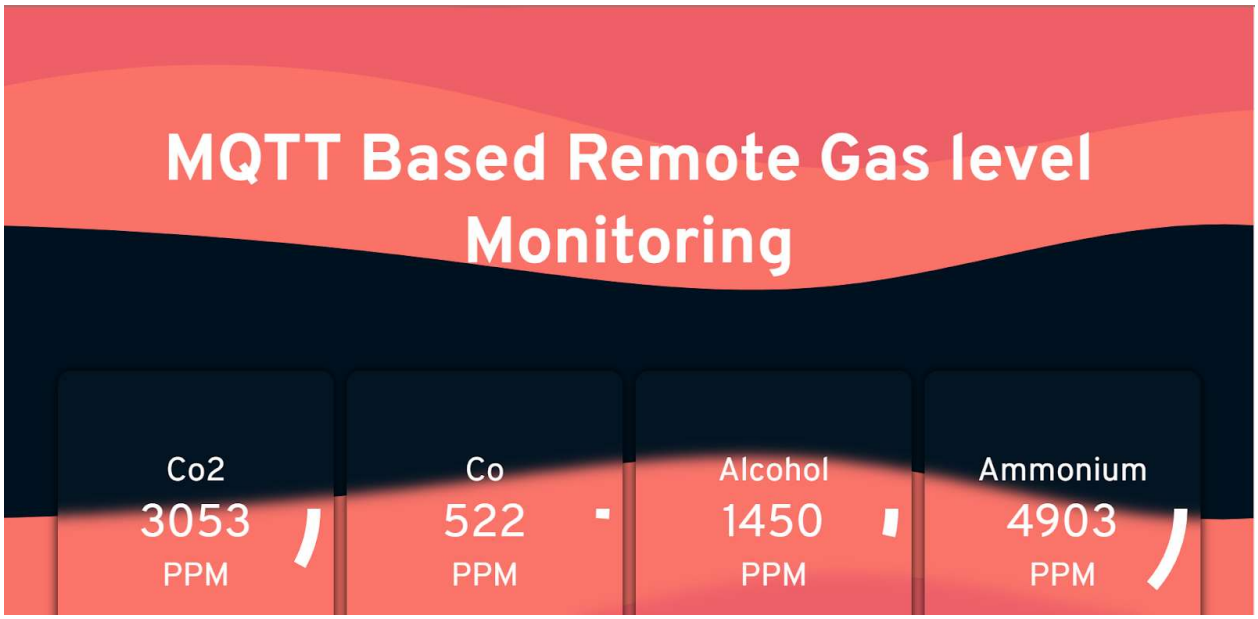
```
// setup the callbacks
client.on('connect', function () {
    console.log('Connected');
});

client.on('error', function (error) {
    console.log(error);
});

client.on('message', function (topic, message) {
    // called each time a message is received
    console.log('Received message:', topic, message.toString());
    createtor(message.toString());
});

// subscribe to topic 'my/test/topic'
client.subscribe('mytopic');




function createtor(message){


    if(message.includes("gas")){
        fs.writeFile("dt.json", message, function(err,result){
            if(err) console.log('error',err);
        })
    }

}
```

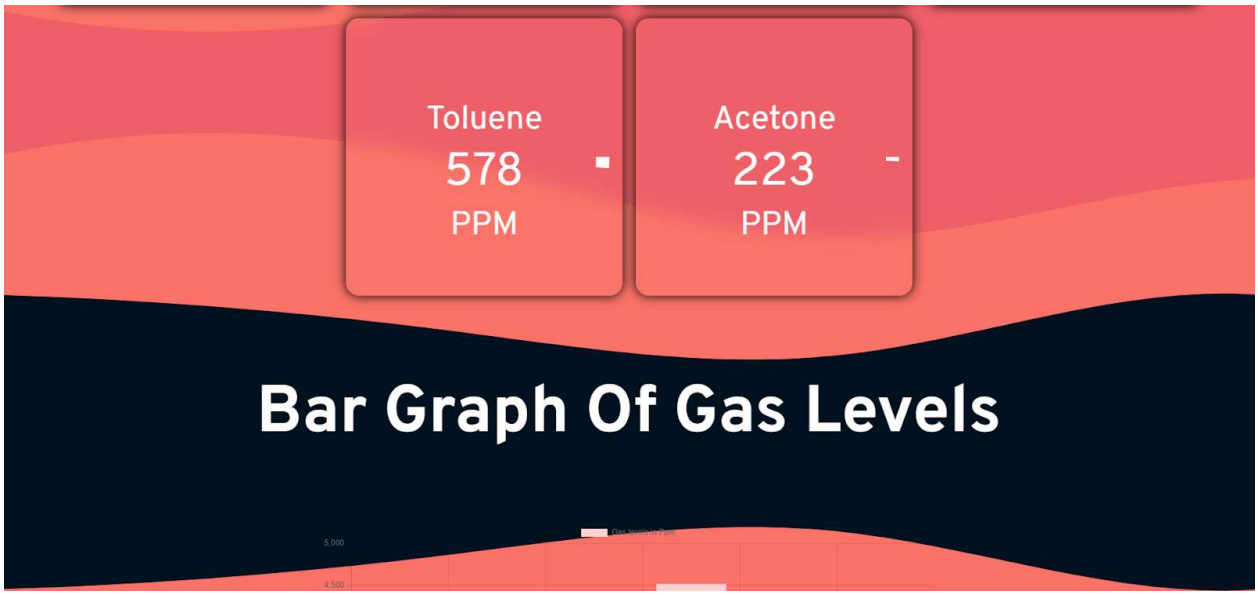
Arduino output:

Esp8266 output:



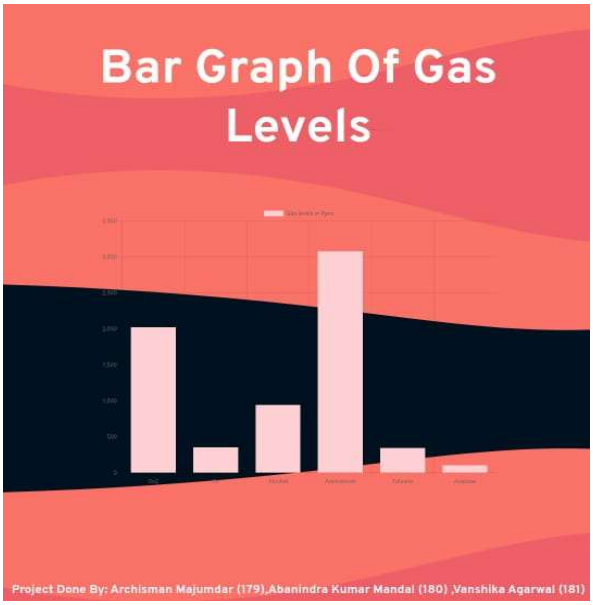
Website:


MQTT Based Remote Gas level Monitoring
Co2
3053
PPM
Co
522
PPM
Alcohol
1450
PPM
Ammonium
4903
PPM


Toluene
578
PPM
Acetone
223
PPM
Bar Graph Of Gas Levels


Project Done By: Archisman Majumdar (179),Abanindra Kumar Mandal (180) ,Vanshika Agarwal (181)

Mobile view:



Conclusion:

From our experiment, we were successful in achieving our aim of creating a robust and reliable system to remotely monitor sensor data over strained bandwidth environments.