Internet of Things

Course Project – Proof of Concept

Submitted by

Name : Meda Ekta Sai

Register Number : 220200019

E-mail ID : Meda Ekta Sai

School of Study : SCDS

Year of Study : 3rd

Problem Statement: Real-Time Bus Tracking and Proximity Notification System

The students face difficulties in deciding when to leave for the bus stop, either reaching there in a rush or waiting too long. What is needed is an intelligent notification system that integrates with real-time bus tracking and lets users know when the bus is arriving, making the ride smoother and hassle-free.

Hardware components :

- ESP32
- I2C-Compatible LCD Display
- GPS Module (NEO-6M)

Process flow :

1. Setting Up and Initializing ESP32
Wi-Fi Connection: ESP32 connects to a WiFi network through the ssid and password.MQTT Client Setup: ESP32 connects to the MQTT broker, which is at broker.hivemq.com, using port 1883 and subscribes to the topic bus/tracking.

2. Bus Movement Simulation:

The position of the bus is manually simulated from a point 5 km south of the pickup location: Each iteration means the increment of value by 0.009 latitude, equivalent to almost 1 km toward the pickup point north.

Note: If a GPS-6M module were available :

Latitude and longitude values would be acquired in real time from the GPS module.The manual increment simulation would no longer be required.

3. Calculating Distance to Pickup Point

- Using the **Haversine formula**, the ESP32 calculates the distance between the bus's simulated location and the fixed pickup point.

The Haversine formula calculates the shortest distance between two points on a sphere using their latitudes and longitudes measured along the surface.
The haversine can be expressed in trigonometric function as:
$haversine(\theta) = sin^2(\frac{\theta}{2})$
The haversine of the central angle (which is d/r) is calculated by the following formula:

$$\frac{d}{r} = haversine(\Phi_2 - \Phi_1) + cos(\Phi_1)cos(\Phi_2)haversine(\lambda_2 - \lambda_1)$$

where r is the radius of the earth(6371 km), d is the distance between two points, (Φ_1, Φ_2) is the latitude of the two points, and (λ_1, λ_2) is the longitude of the two points respectively.

Solving d by applying the inverse haversine or by using the inverse sine function, we get:

$$d = rhav^{-1}(h) = 2rsin^{-1}(\sqrt{h})$$

Or

$$d = 2rsin^{-1}(\sqrt{sin^2(\frac{\Phi_2-\Phi_1}{2}) + cos(\Phi_1)cos(\Phi_2)sin^2(\frac{(\lambda_2-\lambda_1}{2}}))$$

4. Publishing Distance Updates

The ESP32 publishes the calculated distance as a message to the MQTT topic bus/tracking in the format:

Distance to pickup: X.X km

5. Updating the LCD Display

The LCD displays:Arriving soon!" when the bus is within 1 km.The current distance is in kilometers otherwise.

6. Node-RED Flow Processing
MQTT Subscriber Node
Input Topic: The Node-RED flow subscribes to the topic bus/tracking.
Payload Received: A message like "Distance to pickup: X.X km" is sent to the Function Node.
Function Node (Processing Logic):Payload Parsing: The distance value is retrieved from the MQTT message.
Conditional Check:
If distance ≤ 1 km, then the node:
Send a message to the Email Node to alert the user.
If distance > 1 km, then the node:
Logs the distance in the debug console for observation

Email Node
Trigger: The bus is triggered at a distance of 1 km from the pickup point.
Action: Send an email to the address provided (ektasai2005@gmail.com) with the message:
The bus will arrive in 5 minutes!

Debug Node

Purpose: Logs the current distance to the debug sidebar in Node-RED for verification and troubleshooting.

4. Output:

bus/tracking

Inbox ×

E

ektasai2005@gmail.com

to me

The bus will arrive in 5 minutes!

Code:

In sketch.ino:

```
#include <Wire.h>

#include <LiquidCrystal_I2C.h>

#include <WiFi.h>

#include <PubSubClient.h>

#include <math.h>  // Wi-Fi and MQTT settings


const char* ssid = "Wokwi-GUEST";

const char* password = "";

const char* mqtt_server = "broker.hivemq.com";

const int mqttPort = 1883;

const char* mqtt_topic = "bus/tracking";


// Fixed pickup point (latitude and longitude)

float pickup_lat = 12.971598;  // Latitude of the pickup point

float pickup_lon = 77.594566;  // Longitude of the pickup point


// Bus starting point (5 km away from pickup point)

float bus_lat = 12.927000;  // Starting latitude (5 km south of pickup)

float bus_lon = 77.594566;  // Starting longitude


// Movement simulation (simulating bus movement by changing coordinates)

float increment = 0.009; // Approx. 1 km of movement per update (latitude change)
```

```
// LCD setup
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Wi-Fi and MQTT setup
WiFiClient espClient;
PubSubClient client(espClient);

// Function to set up MQTT connection
void setupMQTT() {
  client.setServer(mqtt_server, mqttPort);
  while (!client.connected()) {
    Serial.print("Connecting to MQTT...");
    if (client.connect("ESP32Client")) {
      Serial.println("Connected to MQTT!");
    } else {
      Serial.print("Failed: ");
      Serial.println(client.state());
      delay(2000);
    }
  }
}

// Function to publish MQTT message
```

```
void publishMQTT(float distance) {
  String message = "Distance to pickup: " + String(distance, 1) + " km";
  client.publish(mqtt_topic, message.c_str());
  Serial.println("Published: " + message);
}

// Function to update the LCD
void updateDisplay(float distance) {
  lcd.clear();
  if (distance <= 1.0) {
    lcd.print("Arriving soon!");
  } else {
    lcd.print("Distance: ");
    lcd.setCursor(0, 1);
    lcd.print(String(distance, 1) + " km");
  }
}

// Function to calculate distance between two coordinates
float calculateDistance(float lat1, float lon1, float lat2, float lon2) {
  const float R = 6371.0; // Radius of Earth in km
  float dLat = radians(lat2 - lat1);
  float dLon = radians(lon2 - lon1);
  float a = sin(dLat / 2) * sin(dLat / 2) +
```

```
          cos(radians(lat1)) * cos(radians(lat2)) *
          sin(dLon / 2) * sin(dLon / 2);
  float c = 2 * atan2(sqrt(a), sqrt(1 - a));
  return R * c;
}

void setup() {
  Serial.begin(115200);

  lcd.init();
  lcd.backlight();

  Serial.println("Connecting to WiFi...");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected!");
  setupMQTT();

  lcd.print("System Ready");
  delay(2000);
}
```

```
void loop() {
  if (!client.connected()) {
    setupMQTT();
  }
  client.loop();

  // Calculate distance to the pickup point
  float distance = calculateDistance(bus_lat, bus_lon, pickup_lat, pickup_lon);

  // Update display and log distance to debug
  updateDisplay(distance);
  Serial.println("Current distance: " + String(distance, 1) + " km");

  // Send MQTT message for debug
  publishMQTT(distance);

  // Simulate movement towards the pickup point only if the bus is farther than 1
km
  if (distance > 1.0) {
    // Moving towards the pickup point by decreasing latitude (moving north)
    bus_lat += increment; // Move bus closer to pickup point (north)
  } else {
    Serial.println("Bus is near the pickup point!");
  }
```

```
    delay(5000); // Simulate delay between updates

}
```

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include <math.h>  // Wi-Fi and MQTT settings

const char* ssid = "Wokwi-GUEST";
const char* password = "";
const char* mqtt_server = "broker.hivemq.com";
const int mqttPort = 1883;
const char* mqtt_topic = "bus/tracking";

// Fixed pickup point (latitude and longitude)
float pickup_lat = 12.971598;  // Latitude of the pickup point
float pickup_lon = 77.594566;  // Longitude of the pickup point

// Bus starting point (5 km away from pickup point)
float bus_lat = 12.927000;  // Starting latitude (5 km south of pickup)
float bus_lon = 77.594566;  // Starting longitude

// Movement simulation (simulating bus movement by changing coordinates)
float increment = 0.009; // Approx. 1 km of movement per update (latitude change)

// LCD setup
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Wi-Fi and MQTT setup
WiFiClient espClient;
PubSubClient client(espClient);
```

```
// Function to set up MQTT connection
void setupMQTT() {
    client.setServer(mqtt_server, mqttPort);
    while (!client.connected()) {
        Serial.print("Connecting to MQTT...");
        if (client.connect("ESP32Client")) {
            Serial.println("Connected to MQTT!");
        } else {
            Serial.print("Failed: ");
            Serial.println(client.state());
            delay(2000);
        }
    }
}

// Function to publish MQTT message
void publishMQTT(float distance) {
    String message = "Distance to pickup: " + String(distance, 1) + " km";
    client.publish(mqtt_topic, message.c_str());
    Serial.println("Published: " + message);
}
```

```
// Function to update the LCD
void updateDisplay(float distance) {
    lcd.clear();
    if (distance <= 1.0) {
        lcd.print("Arriving soon!");
    } else {
        lcd.print("Distance: ");
        lcd.setCursor(0, 1);
        lcd.print(String(distance, 1) + " km");
    }
}

// Function to calculate distance between two coordinates
float calculateDistance(float lat1, float lon1, float lat2, float lon2) {
    const float R = 6371.0; // Radius of Earth in km
    float dLat = radians(lat2 - lat1);
    float dLon = radians(lon2 - lon1);
    float a = sin(dLat / 2) * sin(dLat / 2) +
              cos(radians(lat1)) * cos(radians(lat2)) *
              sin(dLon / 2) * sin(dLon / 2);
    float c = 2 * atan2(sqrt(a), sqrt(1 - a));
    return R * c;
}
```

```
void setup() {
    Serial.begin(115200);

    lcd.init();
    lcd.backlight();

    Serial.println("Connecting to WiFi...");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
    }
    Serial.println("\nWiFi connected!");
    setupMQTT();

    lcd.print("System Ready");
    delay(2000);
}

void loop() {
    if (!client.connected()) {
        setupMQTT();
    }
    client.loop();

    // Calculate distance to the pickup point
    float distance = calculateDistance(bus_lat, bus_lon, pickup_lat, pickup_lon);

```

```
    // Update display and log distance to debug
    updateDisplay(distance);
    Serial.println("Current distance: " + String(distance, 1) + " km");

    // Send MQTT message for debug
    publishMQTT(distance);

    // Simulate movement towards the pickup point only if the bus is farther than 1 km
    if (distance > 1.0) {
        // Moving towards the pickup point by decreasing latitude (moving north)
        bus_lat += increment; // Move bus closer to pickup point (north)
    } else {
        Serial.println("Bus is near the pickup point!");
    }

    delay(5000); // Simulate delay between updates
}
```

Diagram.json:

{

"version": 1,

"author": "Meda Ekta Sai",

"editor": "wokwi",

"parts": [

{ "type": "board-esp32-devkit-c-v4", "id": "esp", "top": 0, "left": -14.36, "attrs": {} },

{ "type": "chip-gps-neo6m", "id": "chip1", "top": -37.38, "left": 168, "attrs": {} },

{

"type": "wokwi-lcd1602",

"id": "lcd1",

"top": 150.4,

"left": 149.6,

"attrs": { "pins": "i2c" }

}

],

"connections": [

["esp:TX", "$serialMonitor:RX", "", []],

["esp:RX", "$serialMonitor:TX", "", []],

["chip1:GND", "esp:GND.2", "black", ["h-67.2", "v48"]],

["chip1:VCC", "esp:3V3", "red", ["h-172.8", "v0", "h0", "v0"]],

["chip1:RX", "esp:TX", "green", ["v76.8", "h-17.39"]],

["chip1:TX", "esp:RX", "green", ["h21.01", "v0", "h0", "v96", "h0"]],

["lcd1:GND", "esp:GND.1", "black", ["h-48", "v48", "h-134.4", "v-76.8", "h0"]],

```
  [ "lcd1:VCC", "esp:5V", "red", [ "h-38.4", "v48.1", "h-124.8" ] ],
  [ "lcd1:SDA", "esp:21", "green", [ "h-28.8", "v0.2" ] ],
  [ "lcd1:SCL", "esp:22", "green", [ "h-19.2", "v0.3" ] ]
 ],
 "dependencies": {}
}
```

```
{
  "version": 1,
  "author": "Meda Ekta Sai",
  "editor": "wokwi",
  "parts": [
    { "type": "board-esp32-devkit-c-v4", "id": "esp", "top": 0, "left": -14.36, "attrs"
    { "type": "chip-gps-neo6m", "id": "chip1", "top": -37.38, "left": 168, "attrs": {}
    {
      "type": "wokwi-lcd1602",
      "id": "lcd1",
      "top": 150.4,
      "left": 149.6,
      "attrs": { "pins": "i2c" }
    }
  ],
  "connections": [
    [ "esp:TX", "$serialMonitor:RX", "", [] ],
    [ "esp:RX", "$serialMonitor:TX", "", [] ],
    [ "chip1:GND", "esp:GND.2", "black", [ "h-67.2", "v48" ] ],
    [ "chip1:VCC", "esp:3V3", "red", [ "h-172.8", "v0", "h0", "v0" ] ],
    [ "chip1:RX", "esp:TX", "green", [ "v76.8", "h-17.39" ] ],
    [ "chip1:TX", "esp:RX", "green", [ "h21.01", "v0", "h0", "v96", "h0" ] ],
    [ "lcd1:GND", "esp:GND.1", "black", [ "h-48", "v48", "h-134.4", "v-76.8", "h0" ] ],
    [ "lcd1:VCC", "esp:5V", "red", [ "h-38.4", "v48.1", "h-124.8" ] ],
    [ "lcd1:SDA", "esp:21", "green", [ "h-28.8", "v0.2" ] ],
    [ "lcd1:SCL", "esp:22", "green", [ "h-19.2", "v0.3" ] ]
  ],
  "dependencies": {}
}
```

Gps-neo6m.json:

```
{
  "name": "gps-neo6m",
  "author": "Meda Ekta Sai",
  "pins": [
    "VCC",
    "GND",
    "RX",
    "TX"
  ],
  "controls": []
}
```


```
{
  "name": "gps-neo6m",
  "author": "Meda Ekta Sai",
  "pins": [
    "VCC",
    "GND",
    "RX",
    "TX"
  ],
  "controls": []
}
```

Gps-neo6m.chip.c:

```c
#include <stdio.h>

#include <math.h>

#include <unistd.h>


// GPS coordinates (starting location 5 km away from pickup point)

double bus_lat = 12.927000;

double bus_lon = 77.594566;

double pickup_lat = 12.971598;

double pickup_lon = 77.594566;

double increment = 0.009; // Approx. 1 km of movement per update (latitude
change)


// Function to calculate distance between two coordinates

double calculate_distance(double lat1, double lon1, double lat2, double lon2) {

   const double R = 6371.0; // Earth's radius in km

   double dLat = (lat2 - lat1) * M_PI / 180.0;

   double dLon = (lon2 - lon1) * M_PI / 180.0;

   double a = sin(dLat / 2) * sin(dLat / 2) +

            cos(lat1 * M_PI / 180.0) * cos(lat2 * M_PI / 180.0) *

            sin(dLon / 2) * sin(dLon / 2);

   double c = 2 * atan2(sqrt(a), sqrt(1 - a));

   return R * c;

}
```

```
void simulateGpsData() {
    char nmea_sentence[100];

    // Calculate distance to pickup point
    double distance = calculate_distance(bus_lat, bus_lon, pickup_lat, pickup_lon);

    // Format a GPGGA NMEA sentence
    snprintf(nmea_sentence, sizeof(nmea_sentence),
             "$GPGGA,123519,%.6f,N,%.6f,E,1,08,0.9,545.4,M,46.9,M,,*47\n",
             bus_lat, bus_lon);

    printf("%s", nmea_sentence);
    fflush(stdout); // Ensure data is sent immediately

    // Simulate movement towards pickup point
    if (distance > 1.0) {
        bus_lat += increment; // Move bus closer to the pickup point (north)
    } else {
        printf("Bus is within 1 km of the pickup point!\n");
    }
}

int main() {
    while (1) {
```

```
        simulateGpsData();

        sleep(5); // Simulate real-time updates every 5 seconds

    }

}
```

```c
#include <stdio.h>
#include <math.h>
#include <unistd.h>

// GPS coordinates (starting location 5 km away from pickup point)
double bus_lat = 12.927000;
double bus_lon = 77.594566;
double pickup_lat = 12.971598;
double pickup_lon = 77.594566;
double increment = 0.009; // Approx. 1 km of movement per update (latitude change)

// Function to calculate distance between two coordinates
double calculate_distance(double lat1, double lon1, double lat2, double lon2) {
    const double R = 6371.0; // Earth's radius in km
    double dLat = (lat2 - lat1) * M_PI / 180.0;
    double dLon = (lon2 - lon1) * M_PI / 180.0;
    double a = sin(dLat / 2) * sin(dLat / 2) +
               cos(lat1 * M_PI / 180.0) * cos(lat2 * M_PI / 180.0) *
               sin(dLon / 2) * sin(dLon / 2);
    double c = 2 * atan2(sqrt(a), sqrt(1 - a));
    return R * c;
}

void simulateGpsData() {
    char nmea_sentence[100];

    // Calculate distance to pickup point
    double distance = calculate_distance(bus_lat, bus_lon, pickup_lat, pickup_lon);

```

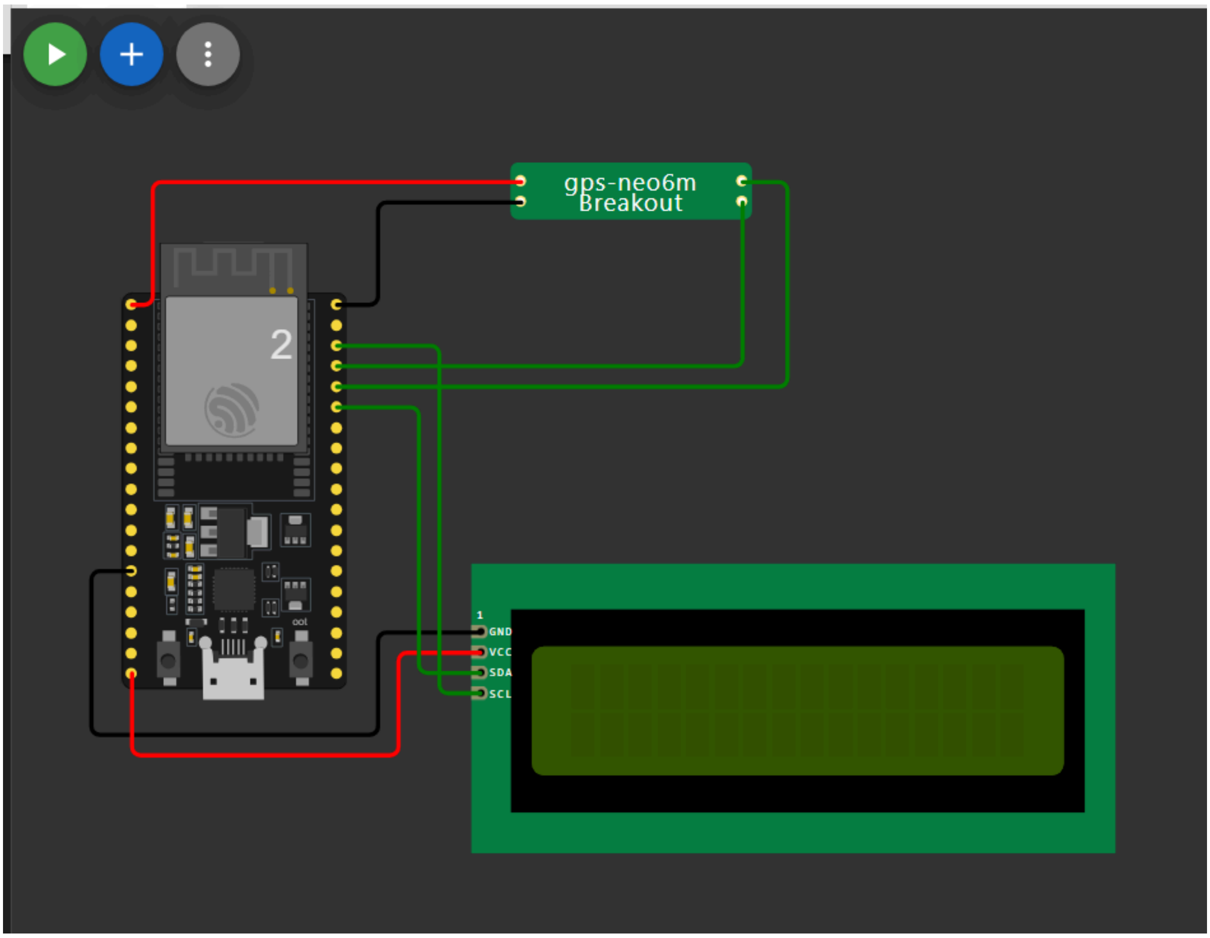
```c

    // Format a GPGGA NMEA sentence
    snprintf(nmea_sentence, sizeof(nmea_sentence),
             "$GPGGA,123519,%.6f,N,%.6f,E,1,08,0.9,545.4,M,46.9,M,,*47\n",
             bus_lat, bus_lon);

    printf("%s", nmea_sentence);
    fflush(stdout); // Ensure data is sent immediately

    // Simulate movement towards pickup point
    if (distance > 1.0) {
        bus_lat += increment;  // Move bus closer to the pickup point (north)
    } else {
        printf("Bus is within 1 km of the pickup point!\n");
    }
}

int main() {
    while (1) {
        simulateGpsData();
        sleep(5); // Simulate real-time updates every 5 seconds
    }
}

```

Libraries:

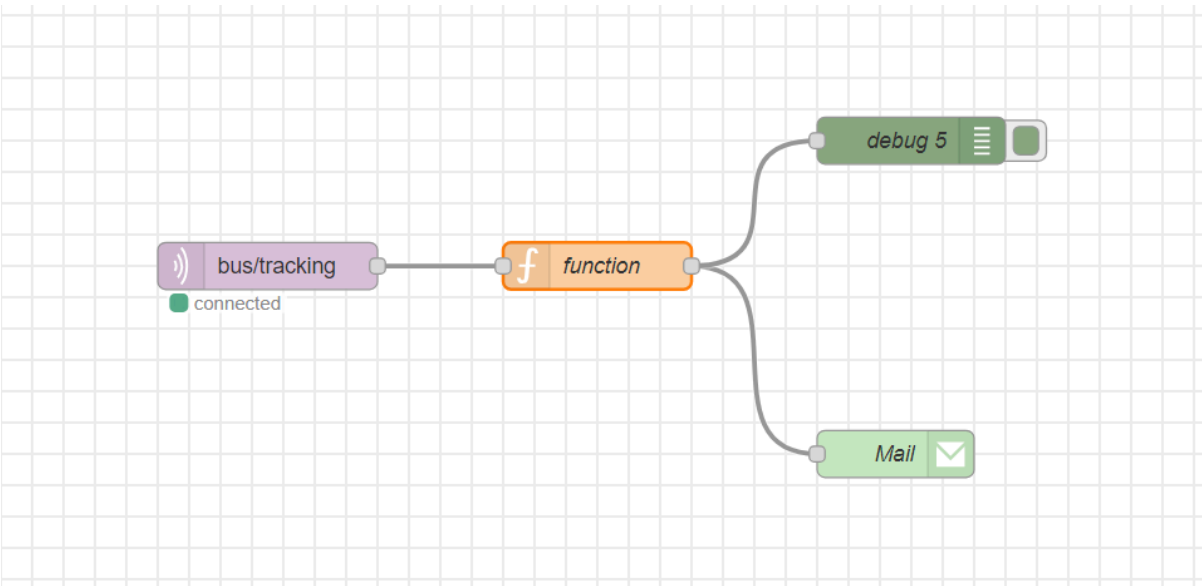
Project Libraries

Installed Libraries

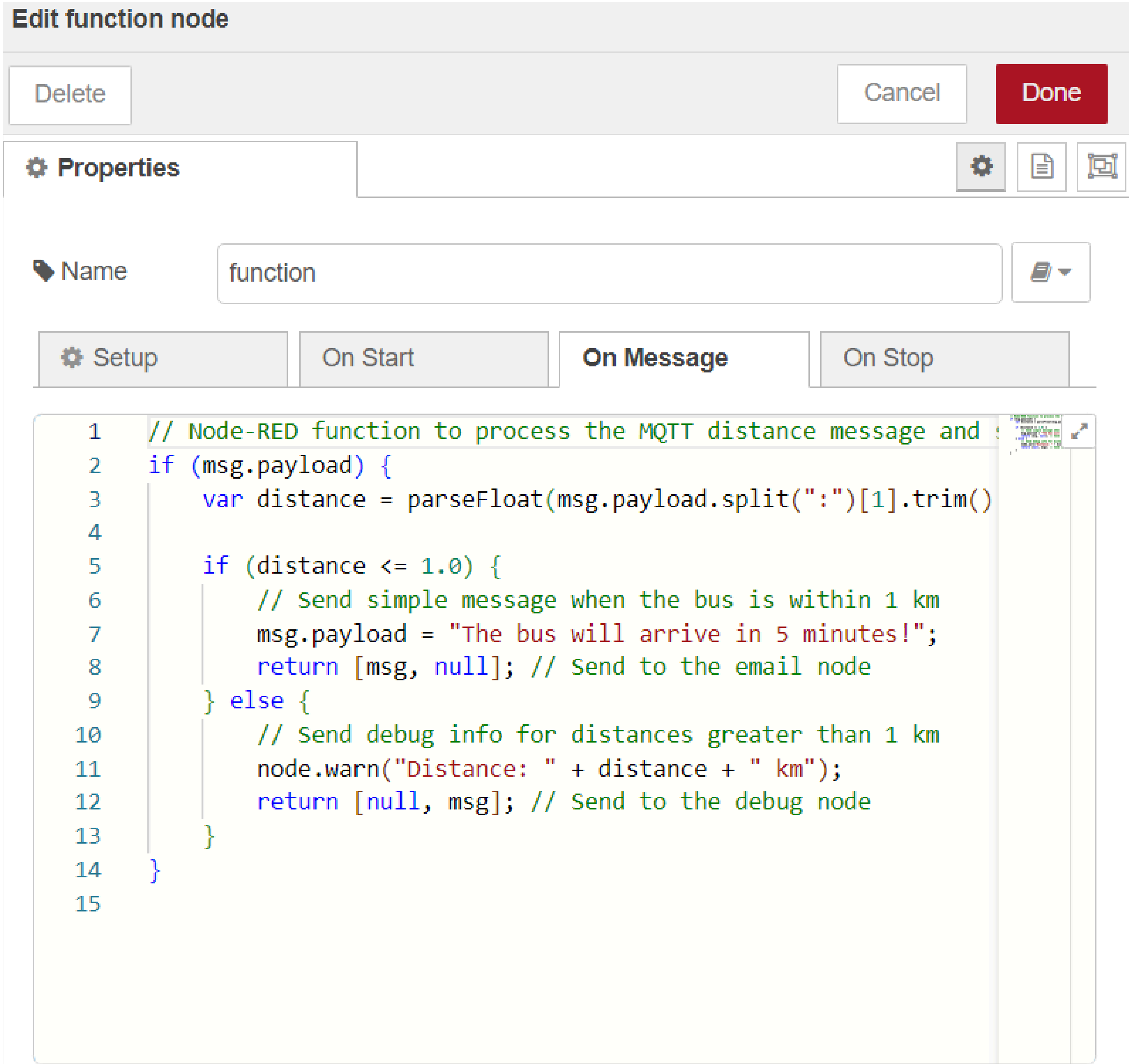
- TinyGPSPlusPlus
- TinyGPSPlus
- TinyGPSPlus-ESP32
- PubSubClient
- WiFi
- LiquidCrystal I2C


gps-neo6m
Breakout
GND
VCC
SDA
SCL

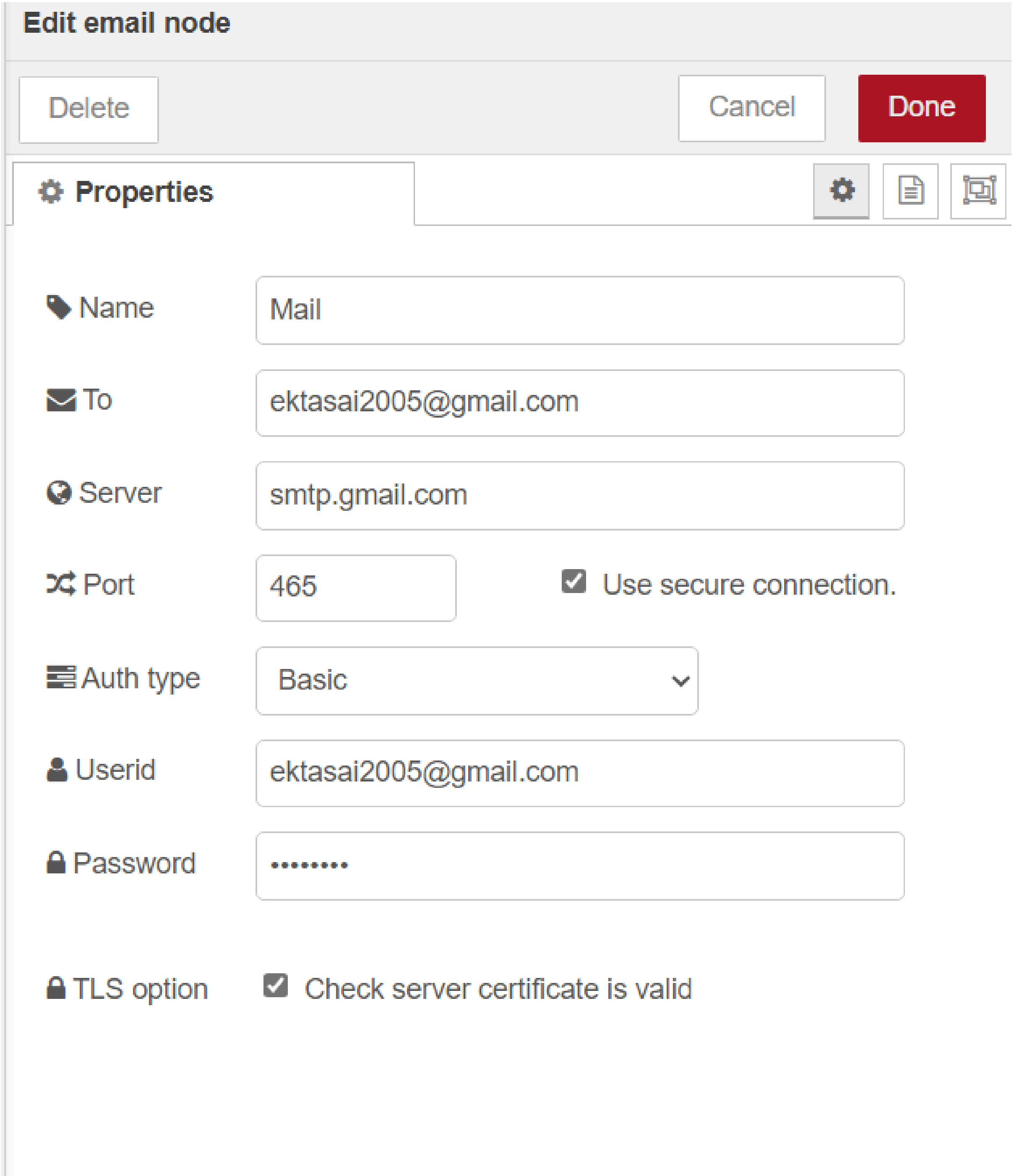
Nod-red:


debug 5
bus/tracking
connected
function
Mail

In function:


Edit function node
Delete
Cancel
Done
Properties
Name
function
Setup
On Start
On Message
On Stop
// Node-RED function to process the MQTT distance message and
if (msg.payload) {
var distance = parseFloat(msg.payload.split(":")[1].trim()
if (distance <= 1.0) {
// Send simple message when the bus is within 1 km
msg.payload = "The bus will arrive in 5 minutes!";
return [msg, null]; // Send to the email node
} else {
// Send debug info for distances greater than 1 km
node.warn("Distance: " + distance + " km");
return [null, msg]; // Send to the debug node
}
}

In mail:


Edit email node
Delete
Cancel
Done
Properties
Name
Mail
To
ektasai2005@gmail.com
Server
smtp.gmail.com
Port
465
Use secure connection.
Auth type
Basic
Userid
ektasai2005@gmail.com
Password
TLS option
Check server certificate is valid

This is the project link in wokwi:

[SmartBus - Wokwi ESP32, STM32, Arduino Simulator]

Conclusion:

This is an efficient bus tracking system designed with ESP32, simulated GPS module, and MQTT that tracks a bus in real-time relative to the fixed pickup point. This data is continuously updated, including latitude, longitude, distance from the pickup point, while sending this data over MQTT to a broker that seamlessly integrates with other platforms, like Node-RED. Notifications alert users when the bus is within 1 km of the pickup point, enhancing convenience and reducing wait-time uncertainty. This cost-effective and scalable solution is ideal for public transport, school buses, or fleet management, showcasing how IoT technology simplifies real-world transportation challenges.

Citations :

"IoT based Vehicle Tracking System using NodeMCU and Arduino IDE."
https://iotdesignpro.com/projects/iot-based-vehicle-tracking-system-using-nodemcu-and-arduino-ide

(PDF) A Cloud-Based Bus Tracking System Based on Internet-of-Things Technology (researchgate.net)