University of Moratuwa, Sri Lanka

Faculty of Engineering

Department of Electronics and Telecommunication Engineering
Semester 5 (Intake 2021)

EN3251 - Internet of Things

Course Project Proposal

Gunawardana W.N.M.	210199D
Dilshan N.L.	210129P
Sehara G.M.M.	210583B

This report is submitted as a partial fulfillment for the module EN3251 - Internet of Things, Department of Electronic and Telecommunication Engineering, University of Moratuwa.

1 Project Overview

The Smart Fire Detection System is an innovative IoT application aimed at improving fire safety across various environments, particularly residential and commercial spaces. The system continuously monitors essential environmental parameters such as smoke levels, temperature, and ambient light, allowing for early fire hazard detection. By utilizing a comprehensive sensor network, real-time data is captured and transmitted to a cloud-based dashboard, where users can visualize the data and respond to alerts in a timely manner. In addition to providing real-time alerts through push notifications or SMS, the system incorporates a management interface built using Node-RED, which enables users to interact with the system, view historical data, and configure alert thresholds, ensuring the solution aligns with modern IoT standards.

2 Objectives

The primary objectives of this project are as follows:

- Develop a robust and reliable fire detection system that can monitor and detect early signs of fire hazards using IoT sensors.
- Provide real-time notifications and alerts to users in case of fire-related events, ensuring timely action can be taken.
- Create a user-friendly dashboard for monitoring live sensor data, with additional features for accessing historical records and reports.
- Increase user interaction and engagement by offering customizable settings such as sensor thresholds and alert preferences.

These objectives are designed to provide users with an enhanced fire safety mechanism, allowing them to actively manage risks while receiving timely information to prevent potential damage.

3 System Architecture

The Smart Fire Detection System is divided into three primary components, each playing a critical role in ensuring system reliability and functionality.

3.1 End-Devices

End-devices are a combination of hardware components and simulated devices that collect environmental data related to fire detection. The system leverages multiple sensors for accurate fire detection, including a smoke sensor, temperature sensor, and light sensor, integrated with the ESP32 microcontroller. These devices are strategically positioned in environments prone to fire hazards, providing the first line of defense through continuous monitoring.

3.2 Central Controller

The central controller is implemented using a Raspberry Pi. It serves as the main hub for receiving data from multiple ESP32 end-devices over the MQTT protocol. The Raspberry Pi processes incoming data and manages communication with the cloud-based dashboard, facilitating real-time monitoring and alerting.

3.3 Management Interface

The management interface is a web-based dashboard developed using Node-RED, hosted on the Raspberry Pi. This interface provides real-time data visualization and serves as a control hub where users can interact with the system. The dashboard is accessible through any modern web browser, allowing users to monitor device status, view sensor readings, set thresholds, and respond to alerts from both desktop and mobile devices.

3.4 End-Devices Details

The following hardware components are used as end-devices, integrated with the ESP32 microcontroller:

- **ESP32 Microcontroller:** The ESP32 serves as the core microcontroller for the end-devices. It interfaces with the sensors, collects data, and communicates with the Raspberry Pi central controller using the MQTT protocol. Its built-in Wi-Fi capability makes it ideal for IoT applications.

- **MQ-2 Gas Sensor:** This sensor is responsible for detecting smoke and various gases, such as methane and propane. Its high sensitivity to smoke makes it ideal for fire detection in enclosed spaces.

- **DS18B20 Temperature Sensor:** This sensor measures ambient temperature in real-time. A significant rise in temperature, along with smoke detection, serves as a strong indicator of fire, helping to prevent false alarms.

- **BH1750 Light Sensor:** The BH1750 measures ambient light levels, which can assist in differentiating between smoke and other environmental conditions, such as dust or steam, reducing the likelihood of false positives.

4 Features

The system offers several key features to ensure comprehensive fire detection:

- **Real-Time Alerts:** Users receive instant notifications when smoke or high temperature is detected. These alerts are sent via push notifications or SMS, ensuring that users are informed regardless of their location.

- **Dashboard Visualization:** A web-based dashboard displays live sensor data, including smoke levels, temperature, and ambient light. Users can also track historical data and trends for detailed analysis.

- **User Interaction:** Users can configure alert thresholds for each sensor, view device statuses, and adjust system settings through the dashboard. The system is customizable to meet different environmental needs.
- **Historical Data Reports:** Users have access to past data and incident reports. This feature is useful for reviewing fire incidents, analyzing patterns, and taking preventive measures.

5 Technologies Used

The system is built using the following technologies:

- **Microcontroller:** ESP32 is used to interface with the sensors and connect to the internet via Wi-Fi. Its dual-core processor allows it to handle multiple tasks simultaneously, including reading sensor data and sending updates to the dashboard.
- **Communication Protocol:** MQTT is utilized for efficient, low-bandwidth data transmission. It allows for real-time communication between the sensors and the dashboard.
- **Dashboard Interface:** Node-RED is employed for building the web-based dashboard. It provides a visual flow-based programming interface, simplifying the process of integrating sensors and creating user interactions.
- **Programming Languages:** JavaScript is used for writing Node-RED flows, while C/C++ is employed to program the ESP32 microcontroller.
- **Data Visualization:** The dashboard makes use of Node-RED's visualization nodes to display live sensor data, enabling users to monitor the environment in real-time.

6 Hardware Components

6.1 Central Controller

Raspberry Pi

- The Raspberry Pi is chosen for its ability to run a Linux-based OS and host Node-RED, providing the central management interface. It receives data from multiple ESP32 devices and manages alerts and dashboard communication.

6.2 Microcontroller

ESP32

- The ESP32 is chosen for its robust features, including built-in Wi-Fi and Bluetooth. These capabilities enable seamless integration into IoT applications. The microcontroller features a dual-core processor, which allows it to handle both sensor readings and communication tasks efficiently. Multiple GPIO pins are available, providing flexibility for connecting a variety of sensors.

6.3 Sensors

1. **MQ-2 Gas Sensor:**
   - This sensor is highly sensitive to smoke, methane, and other flammable gases. It plays a critical role in detecting fire hazards early by responding to changes in air quality.
2. **DS18B20 Temperature Sensor:**
   - The DS18B20 provides accurate digital temperature readings. A sudden spike in temperature, combined with smoke detection, triggers alerts, ensuring that even subtle changes are recognized.
3. **BH1750 Light Sensor:**
   - This sensor measures ambient light levels. By tracking light intensity, it helps filter out false alarms that might be caused by changes in light rather than actual fire hazards.

7 User Interaction

The system provides a seamless interface for user interaction:

7.1 Dashboard Features

- **Real-Time Data Visualization:** Users can view the current sensor readings and device statuses on the dashboard. The dashboard is designed to be intuitive, allowing users to monitor data without needing advanced technical knowledge.
- **Alerts and Notifications:** Users can set customizable alert thresholds for smoke and temperature sensors. These thresholds can be adjusted based on the specific requirements of the monitored environment.
- **Historical Data Access:** Users have access to historical data, which can be useful for analyzing trends and detecting patterns over time. This feature helps users take preventive action and optimize system performance.

7.2 Interaction Methods

- **Web-Based Interface:** The dashboard can be accessed from any device with a modern web browser, making it convenient for users to monitor the system from multiple locations.
- **Mobile Access:** The dashboard is designed with responsive elements, ensuring that it works well on both smartphones and tablets, providing users with flexibility in accessing the system.

8 Implementation Plan

To successfully complete the project, the following steps are outlined:

- **Research and Development:** Finalize hardware components and set up the development environment for ESP32 and Node-RED. During this phase, the most suitable sensors and technologies will be selected.
- **Hardware Setup:** Assemble the sensors with the ESP32 microcontroller, and configure the ESP32 to communicate with the Raspberry Pi central controller. This step includes testing sensor functionality, establishing proper connectivity, and ensuring seamless communication between the end-devices (ESP32 with sensors) and the dashboard hosted on the Raspberry Pi.
- **Software Development:**
  - Program the ESP32 to read data from sensors and communicate with the MQTT broker. This involves writing C/C++ code for the microcontroller to handle multiple sensors simultaneously.
  - Create Node-RED flows for processing and visualizing data on the dashboard. This step involves configuring MQTT nodes, creating alert conditions, and ensuring real-time data visualization.
- **Testing:** Perform unit and integration tests to ensure system functionality. Each component will be individually tested before the entire system is integrated and tested under real-world conditions.
- **Deployment:** Deploy the system in a controlled environment to evaluate its real-world performance. Feedback will be collected from users to refine the system as necessary.
- **Documentation:** Detailed user manuals and system documentation will be created to guide users in operating the system and customizing settings.

9 Figures and Diagrams

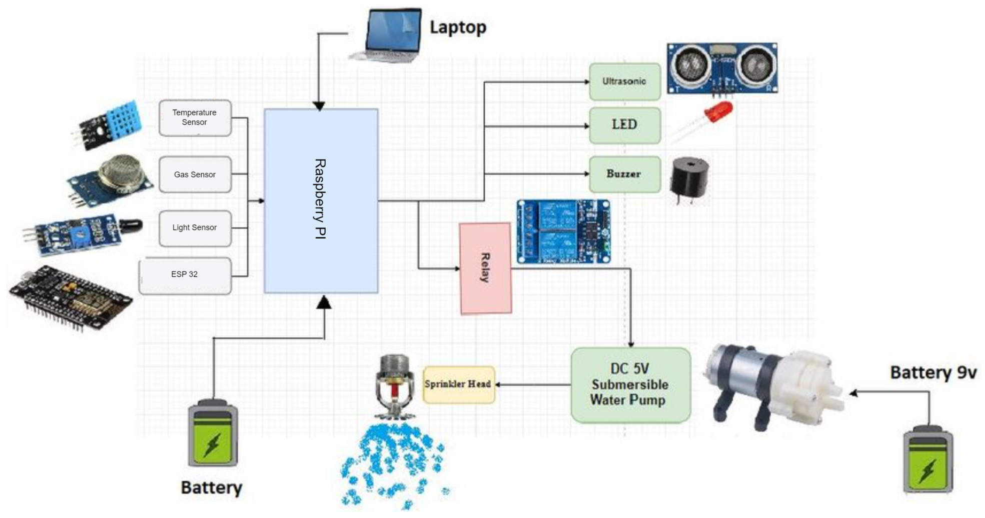

Figure 1: System Block Diagram: This diagram outlines the architecture of the Smart Fire Detection System, showing how the sensors, ESP32 microcontroller, MQTT broker, and dashboard interact.

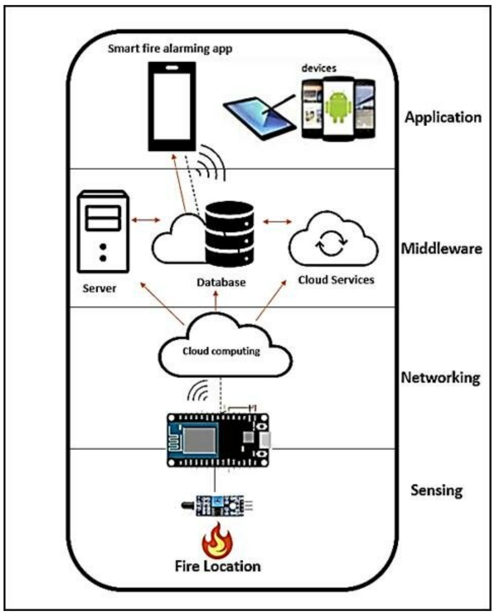

Figure 2: Data Flow Diagram: This figure illustrates how sensor data is processed and communicated between the ESP32 and the dashboard via MQTT.

10 Code Snippets

10.1 ESP32 Code Example

```
#include <WiFi.h>
#include <PubSubClient.h>

// WiFi and MQTT configuration
const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";
const char* mqttServer = "broker_address";
const int mqttPort = 1883;

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
    Serial.begin(115200);
    setupWiFi();
    client.setServer(mqttServer, mqttPort);
}

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

    // Read sensors and publish data
    // ...
}

void setupWiFi() {
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to WiFi...");
    }
    Serial.println("Connected to WiFi");
}

void reconnect() {
    // Reconnect logic
}
```

Listing 1: ESP32 Code Example

10.2 Node-RED Flow Example

```
[
    {
        "id": "mqtt-in",
        "type": "mqtt in",
        "topic": "fire/sensors",
        "qos": "2",
        "broker": "mqtt-broker",
        "x": 100,
        "y": 100,
        "wires": [["process-data"]]
    },
    {
        "id": "process-data",
        "type": "function",
        "function": "var sensorData = JSON.parse(msg.payload);\nif (
    sensorData.smoke > threshold) {\n    // Trigger alert\n}\nreturn msg
    ;",
        "x": 300,
        "y": 100,
        "wires": [["dashboard"]]
    }
]
```

Listing 2: Node-RED Flow Example

11 Conclusion

The Smart Fire Detection System offers a comprehensive solution for enhancing fire safety through IoT technology. With real-time monitoring, alerting capabilities, and a user-friendly dashboard, the system equips users with the tools to take proactive measures against fire hazards. The integration of reliable sensors, a robust communication protocol, and a customizable management interface ensures that the system is both scalable and adaptable to various environments.

12 References

- ESP32 Documentation: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html
- Node-RED Documentation: https://nodered.org/docs/
- MQTT Protocol: https://mqtt.org/