Department of
Electrical and Electronics Engineering

21ES614 – Internet of Things
Lab Manual
Submitted by

CB.EN.P2EBS24024 ANAND P S

M.Tech Embedded Systems, Second Semester

Amrita School of Engineering, Coimbatore
Amritanagar – P. O., Ettimadai, Coimbatore – 641112

Regulation : 2021

April 2025

Vision of the Institute

To be a global leader in the delivery of engineering education, transforming individuals to become creative, innovative, and socially responsible contributors in their professions.

Mission of the Institute

- To provide best-in-class infrastructure and resources to achieve excellence in technical
- education,
- To promote knowledge development in thematic research areas that have a positive impact on society, both nationally and globally,
- To design and maintain the highest quality education through active engagement with all
- stakeholders – students, faculty, industry, alumni and reputed academic institutions,
- To contribute to the quality enhancement of the local and global education ecosystem,
- To promote a culture of collaboration that allows creativity, innovation, and entrepreneurship to flourish, and
- To practice and promote high standards of professional ethics, transparency, and accountability

Vision of the Department

Mould generations of electrical and electronics engineers on global standards with multi-disciplinary perspective to meet evolving societal needs.

Mission of the Department

M1 Empower students with knowledge in electrical, electronics and allied engineering facilitated in innovative class rooms and state-of-the art laboratories.

M2 Inculcate technical competence and promote research through industry interactions, field exposures and global collaborations.

M3 Promote professional ethics and selfless service.

Program Educational Objective - PEOs

The educational objectives of the MTech Embedded Systems program include:

PEO1: Graduates will acquire the ability to migrate to all domains of embedded solutions

PEO2: Graduates will practice ethics in their professional domain and imbibe the professional attitude developed

PEO3: Graduates will learn to work well in team environment considering the multidisciplinary aspects of embedded systems

Program Outcomes – POs

On completion of the MTech (Embedded Systems) program, the graduate will develop:

PO1: An ability to independently carryout research/investigation and development work to solve practical problems

PO2: An ability to write and present a substantial technical report/ document

PO3: Students should be able to demonstrate a degree of mastery over the area as per the specialization of the program. The mastery should be at a level higher than the requirements in the appropriate bachelor program

Program Specific Outcomes – PSOs

PSO1: Acquire state - of - the – art technologies for development of embedded solutions

PSO2: Ability to work in a multidisciplinary environment employing ethical values and social responsibility

Course Objective

This course aims to provide a good understanding of both fundamental concepts and advanced topics in IoT and discuss the role and features of a IoT system in an application development.

Syllabus

Introduction to IoT - Definitions, frameworks and key technologies. Functional blocks of IoT systems: hardware and software elements- devices, communications, services, management, security, and application. Challenges to solve in IoT.

Basics of Networking & Sensor Networks - Applications, challenges - ISO/OSI Model, TCP/IP Model. Sensor network architecture and design principles. IoT technology stack - overview of protocols in each layer. Communication Protocols. Communication models, Application protocols for the transfer of sensor data. Infrastructure for IoT: LoRa-Wan, 6LoWPAN, 5G and Sigfox. Operating systems and programming environments for embedded units (Contiki).

Introduction to Cloud, Fog and Edge Computing- Modern trends in IoT – Industrial IoT, Wearable. Applications of IoT - Smart Homes/Buildings, Smart Cities, Smart Industry, and Smart Medical care, Smart Automation etc.

Course Outcomes and its Mapping with POs

CO	Course Outcomes	PO1	PO2	PO3	PSO1	PSO2
21ES614.1	Understand the concepts and principles of IoT	1		1	3	
21ES614.2	Implement communication protocols related to IoT and machine to machine communication	3	1	2	3	
21ES614.3	Familiarize key technologies in an IoT framework.	1		1	3	2
21ES614.4	Develop IoT based solution for real world applications.	3	1	3	1	2
Mode	ES, Project					

Lab Experiments:

Experiment No: 1 **Date: 14.12.2024**

Familiarization of NetSim and simulation of communication networks

AIM

1. To familiarize NetSim – a communication network simulation software.
2. To perform of simulation of various networks using NetSim.

THEORY

NetSim - NetSim is a network simulation tool that allows you to create network scenarios, model traffic, design protocols and analyze network performance. Users can study the behavior of a network by test combinations of network parameters [1].
NetSim enables users to simulate protocols that function in various networks, and is organized as: Internetworks, Legacy Networks, Advanced Routing, Advanced Wireless Networks, Cellular Networks, Wireless Sensor Networks, Personal Area Networks, LTE/LTE-A Networks, Cognitive Radio Networks, Internet of Things and VANETs [1].
Network Design Window: NetSim's network design window enables users to model a network comprising of network devices switches, routers, nodes, etc, connect them through links and model application traffic to flow in the network [1].

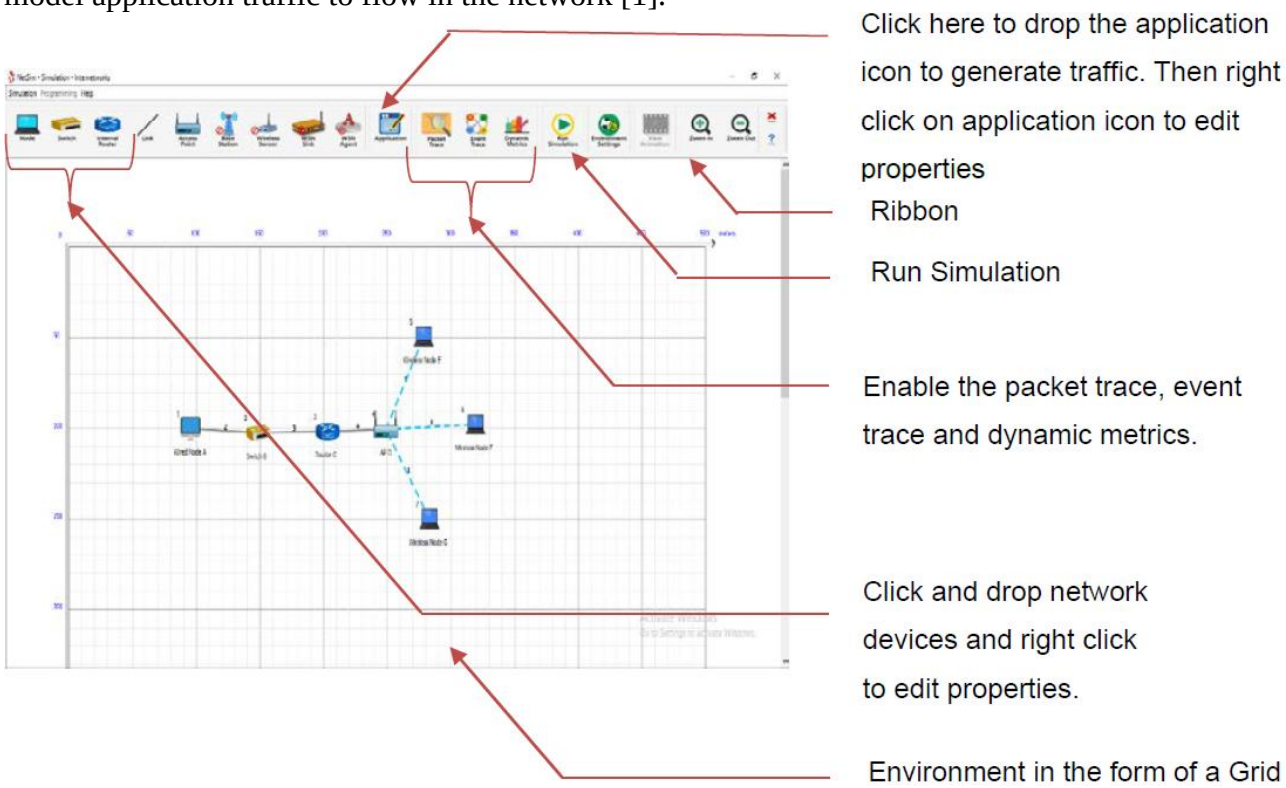


Fig 1: Network Design Window – NetSim [1]

Results Window: Upon completion of simulation, Network statistics or network performance metrics reported in the form of graphs and tables. The report includes metrics like throughput, simulation time, packets generated, packets dropped, collision counts etc. [1].

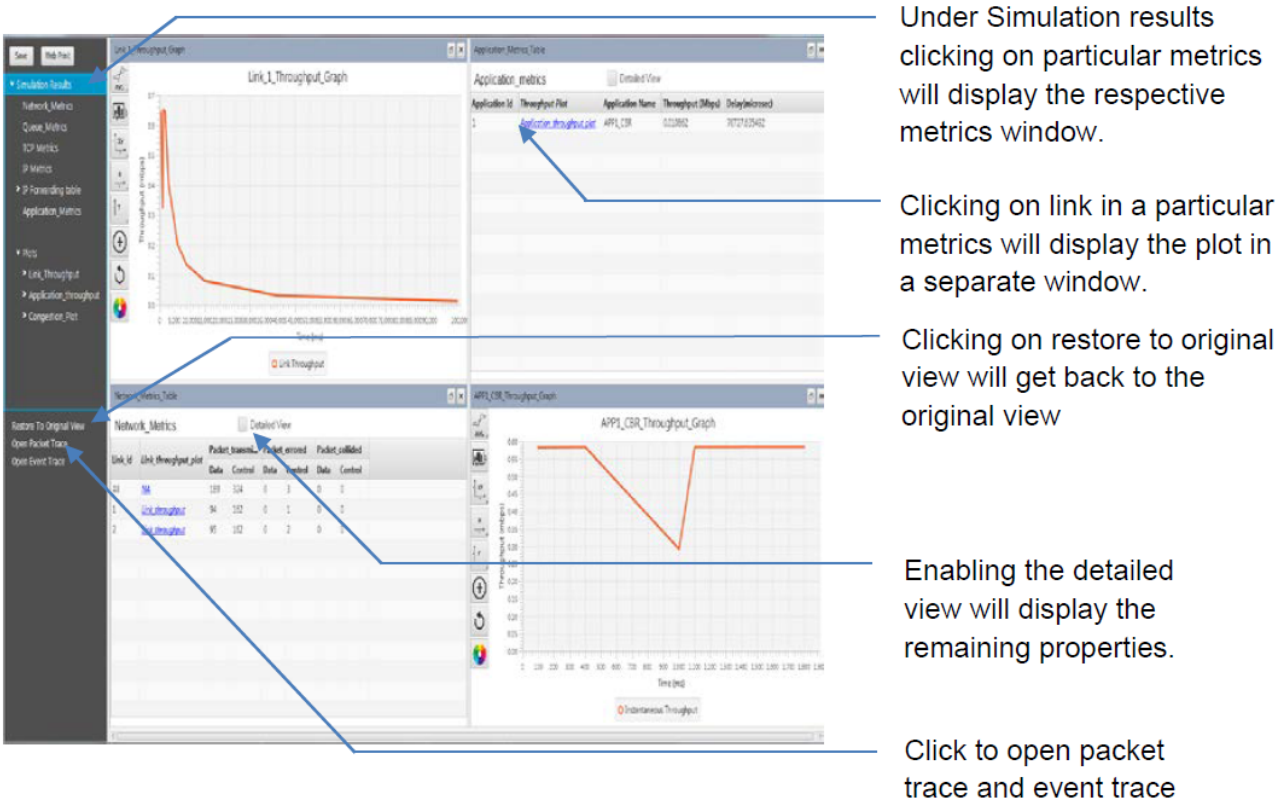


Fig 2: Results Window – NetSim [1]

Packet Animation Window: When running a simulation, an option is available to play or record animations. If this is enabled, upon completion of the simulation users can see the flow of packet through the network, along with 20+ fields of packet information available as a table at the bottom. This table contains all the fields recorded in the packet trace. In addition, animation options are available for viewing different graphs, IP Addresses, Node movement etc. [1].

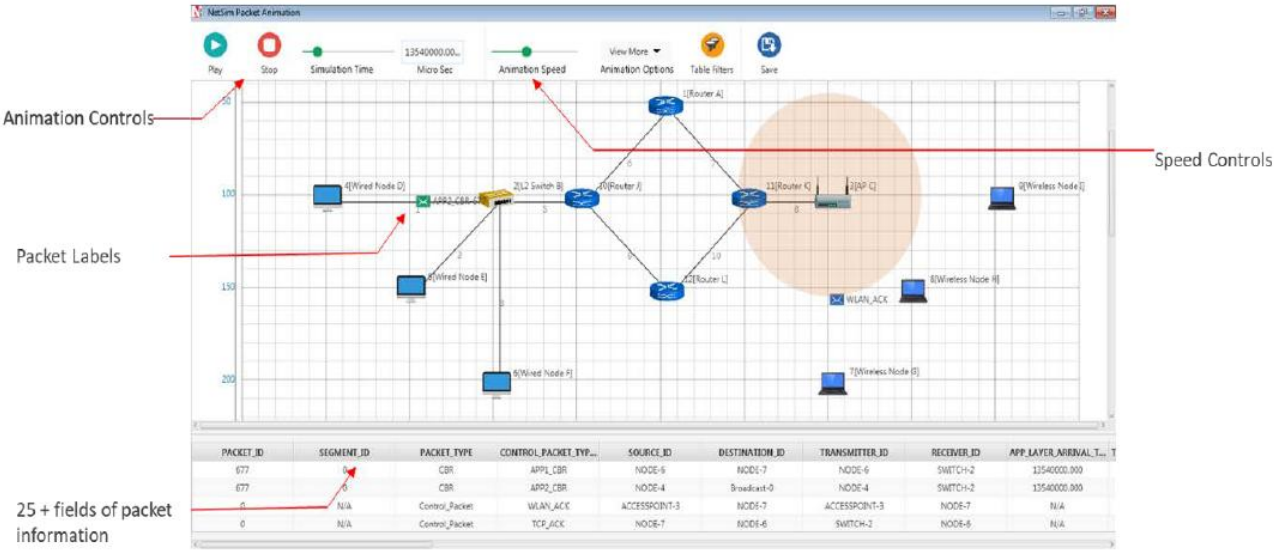


Fig 2: Packet Animation Window – NetSim [1]

Packet Trace: NetSim allows users to generate trace files which provide detailed packet information useful for performance validation, statistical analysis & custom code de-bugging.

Packet Trace logs a set of chosen parameters for every packet as it flows through the network such as arrival, queuing, and departure times, payload, overhead, errors, collisions etc. [4].
By providing a host of information and parameters of every packet that flows through the network, packet trace provides necessary forensics for users to catch logical errors without setting a lot of breakpoints or restarting the program often. Window size variation in TCP, Route Table Formation in OSPF, Medium Access in Wi-fi, etc., are examples of protocol functionalities that can be easily understood from the trace [4].

The typical steps involved in doing experiments in NetSim are [1]

- Network Set up: Drag and drop devices, and connect them using wired or wireless links
- Configure Properties: Configure device, protocol or link properties by right clicking on the device or link and modifying parameters in the properties window.
- Model Traffic: Drag and drop the application icon, right click to enter and set traffic properties
- Enable trace/dynamic metrics (optional): Click on packet trace, event trace and dynamic metrics to enable. Packet trace logs packet flow, event trace logs each event (NetSim is a discrete event simulation) and dynamic metrics plots various throughputs over time.
- Save/Open/Edit Experiment: Users can save the experiments using CTRL+S or clicking on the save icon. Saved experiments can then opened, parameters modified and simulation run.
- Visualize through the animator to understand working (or) Analyze results and draw inferences

REFERENCE

1. Tetcos, "NetSim Experiments Manual," Rev 10.2 (V), 11(V), 2018.
2. Kalpalatha S, Venkatesh Ramaiyan, Ashwini Chinta Girish, Surabhi Vyas, “Learning WiFi using NetSim”, Indian Institute of Technology Madras, 2018 [Online]. Available: https://www.tetcos.com/downloads/user_perspectives/IIT-Madras-learn-wifi-using-netsim.pdf
3. Krishna Bharadwaj, Pavithra Krishnan, Venkatesh Ramaiyan, “Introduction to TCP using NetSim” Indian Institute of Technology Madras, 2018 [Online]. Available: https://www.tetcos.com/downloads/user_perspectives/IIT-Madras-learn-tcp-using-netsim.pdf
4. Tetcos, "NetSim User Manual," Rev 10.2 (V), 11(V), 2019.

PROCEDURE

- Install proper NetSim version in the laptop.
- With the help of the user manual, understand the software environment.
- Set up various networks with different topologies.
- Simulate the networks with various configuration options and observe the changes.
- Study the performance of the network in all simulations from data available in the results window and the packet and event trace files. Bring clarity in your understanding by reviewing literature on the standard.
- Bring out the understanding from the experiment in the inference section.

Experiment No: 2 **Date:24.02.25**

Development of IoT end nodes with Ubidots/Adafruit/ThingSpeak

AIM

To develop IoT end nodes based on Arduino IDE and demonstrate data transfer to the Ubidots / Adafruit / ThingSpeak platform.

TOOLS & SYSTEMS

Software Used

Arduino IDE
Thing speak IOT
Platform Adafruit
IOT platform
Ubidots IOT platform

Hardware Components

Arduino Micro
Controller DHT11
Temperature Sensor
MQ7 Gas Sensor
DC Motor
L293D
Ultrasonic Sensor

IMPLEMENTATION

ThingSpeak is used to control a DC motor connected via an L293D motor driver by fetching data from a specific field in a ThingSpeak channel. When the field value is 1, the motor runs forward; a value of 2 makes it run in reverse; and any other value stops the motor. This setup enables remote motor control through a simple data-driven approach.

Ubidots handles data from the MQ7 gas sensor, which detects carbon monoxide (CO) levels. Sensor readings are transmitted to the Ubidots platform under a device labeled “co” with a variable named “co-levels,” using the MQTT protocol. This allows continuous monitoring of CO levels and can trigger alerts or actions based on the readings.

Adafruit IO manages the data from the DHT11 sensor and an ultrasonic sensor. The DHT11 captures temperature and humidity data, which are sent to individual feeds named “temperature” and “humidity,” respectively. The ultrasonic sensor measures distance and transmits the data to a feed called “sensor1.” Like Ubidots, Adafruit IO also uses MQTT for efficient, real-time data communication, enabling responsive IoT applications.

Hardware Setup

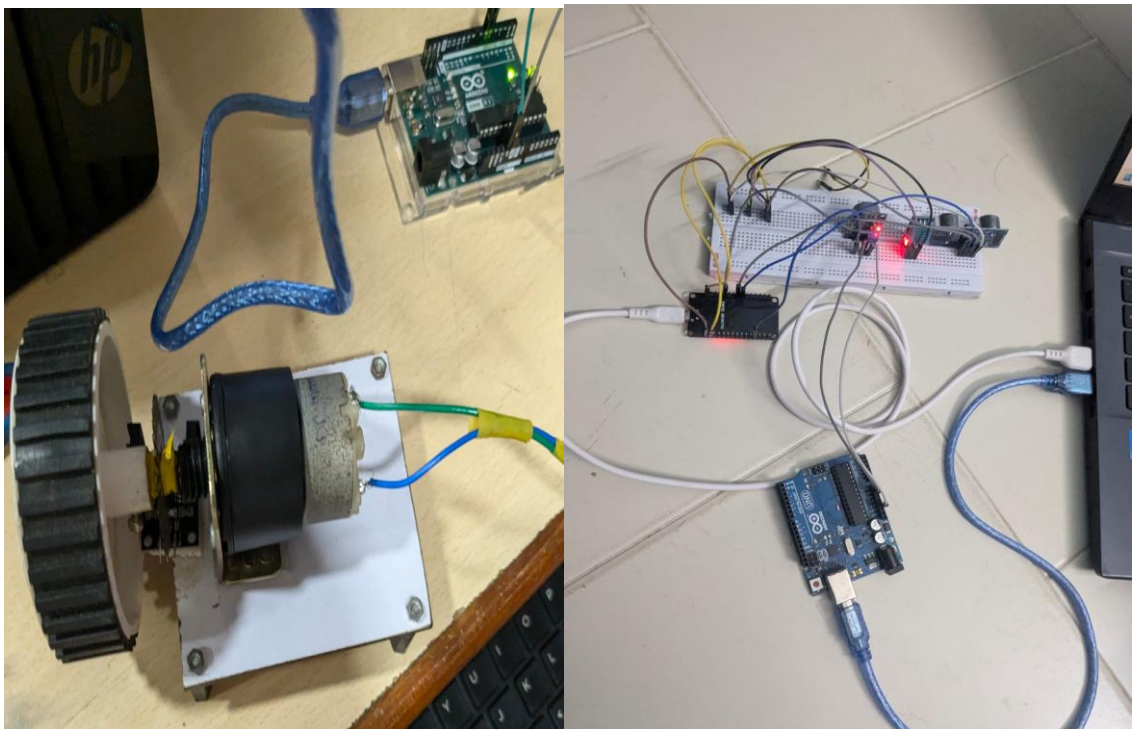


Fig: Hardware setup for end nodes

IOT Platforms

1. ThingSpeak : Controls a DC motor (via L293D)
2. Ubidots : Sends CO level from MQ7 sensor
3. Adafruit IO : Sends temperature, humidity (DHT11) and distance (Ultrasonic).

RESULTS

Adafruit:

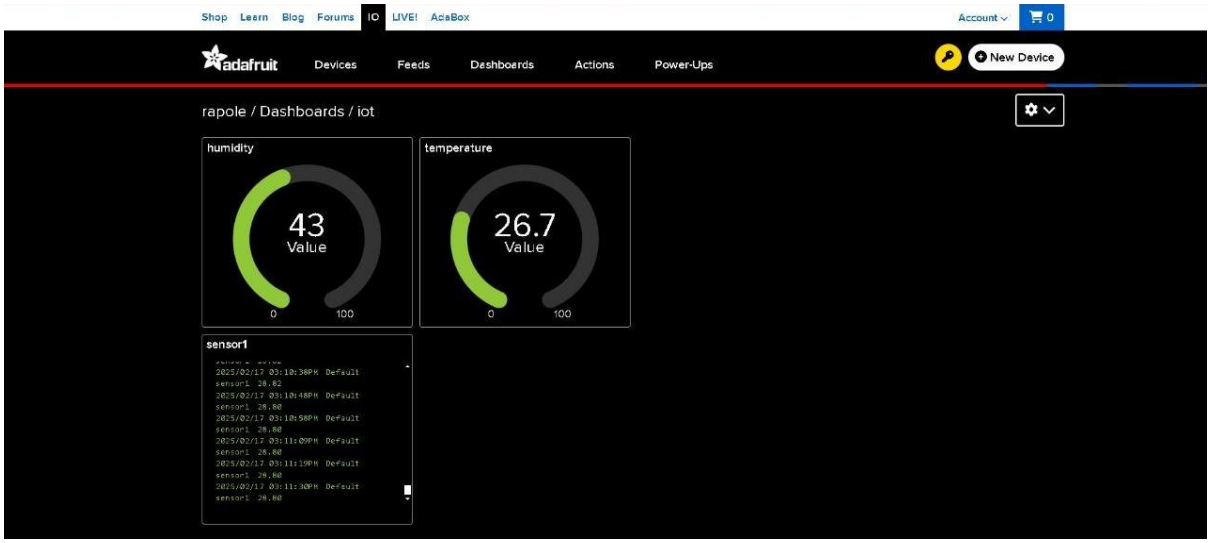


Fig: Adafruit Platform Results

Ubidots:

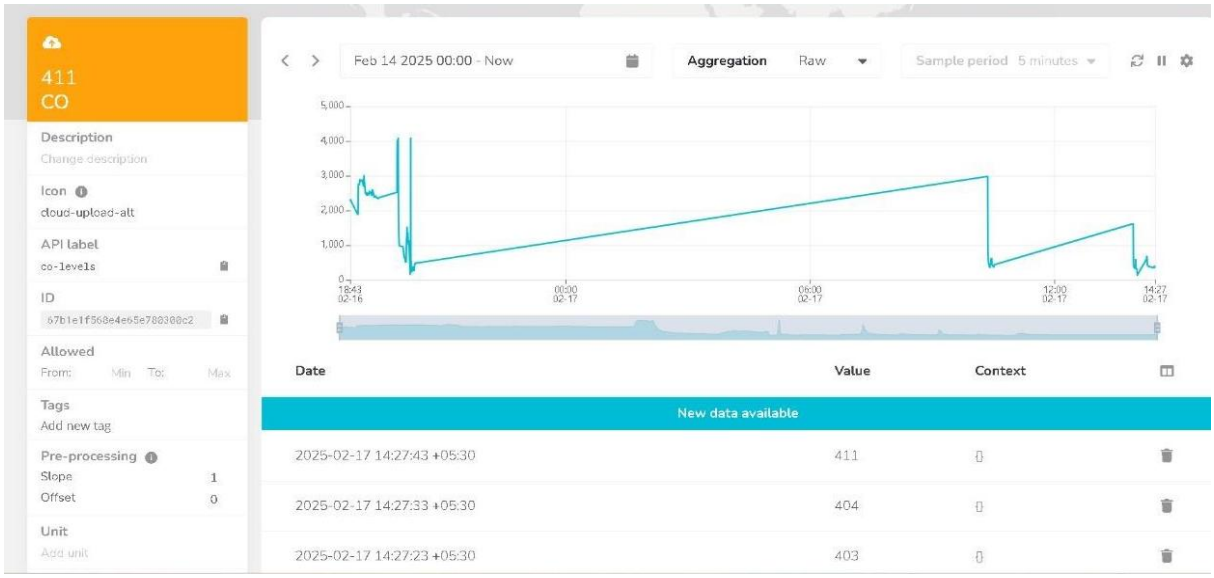


Fig: Ubidots Results

ThingsSpeak:

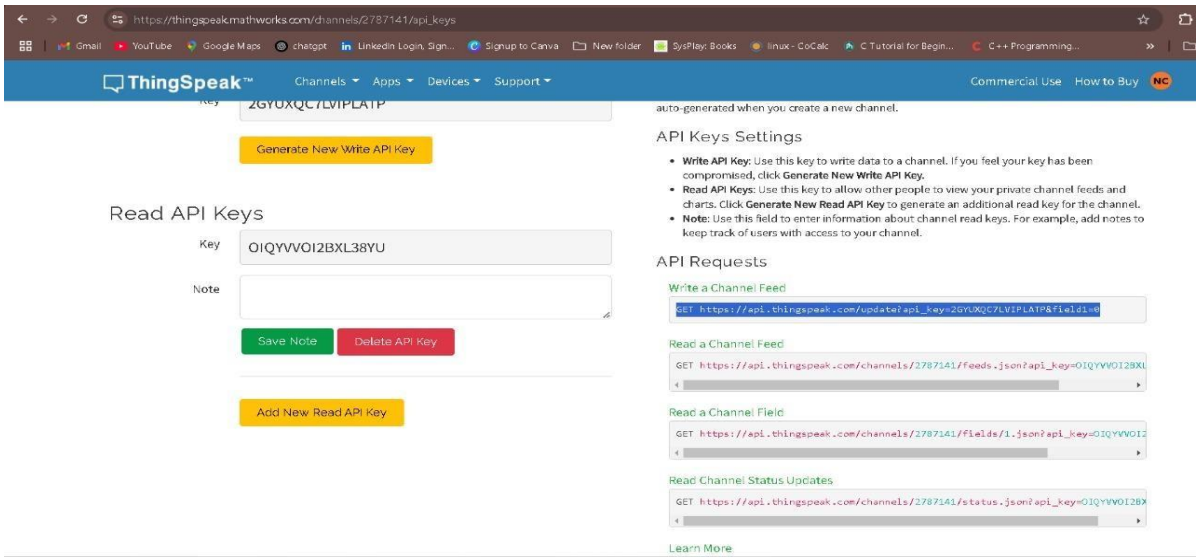


Fig: Give Read API key to create Http link

127

Fig: hyper link used (Thingspeak)

Results:

Implemented End device and demonstrated the working in various IoT Platforms like Thingspeak, Adafruit, Ubidots.

Experiment No: 3 **Date: 06.01.2025**

Simulation Study on IEEE 802.3/802.11 networks using NetSim

AIM

To simulate IEEE 802.3 and IEEE 802.11 networks in NetSim and study the operation of the networks under various configurations and scenarios.

THEORY - IEEE 802.3 & 802.11 STANDARDS

In 1985, the Computer Society of the IEEE started a project, called Project 802, to set standards to enable intercommunication among equipment from a variety of manufacturers. Project 802 does not seek to replace any part of the OSI or the Internet model. Instead, it is a way of specifying functions of the physical layer and the data link layer of major LAN protocols. The standard was adopted by the American National Standards Institute (ANSI). In 1987, the International Organization for Standardization (ISO) also approved it as an international standard under the designation ISO 8802.

IEEE 802.3 is a working group and a collection of Institute of Electrical and Electronics Engineers (IEEE) standards produced by the working group defining the physical layer and data link layer's media access control (MAC) of wired Ethernet. This is generally a local area network (LAN) technology with some wide area network (WAN) applications. Physical connections are made between nodes and/or infrastructure devices (hubs, switches, routers) by various types of copper or fiber cable. 802.3 is a technology that supports the IEEE 802.1 network architecture. 802.3 also defines LAN access method using CSMA/CD.

IEEE 802.11 is part of the IEEE 802 set of local area network (LAN) technical standards, and specifies the set of media access control (MAC) and physical layer (PHY) protocols for implementing wireless local area network (WLAN) computer communication. The standard and amendments provide the basis for wireless network products using the Wi-Fi brand and are the world's most widely used wireless computer networking standards. IEEE 802.11 is used in most home and office networks to allow laptops, printers, smartphones, and other devices to communicate with each other and access the Internet without connecting wires.

REFERENCE

5. Behrouz Forouzan, “Data Communications and Networking”, 4th edition, McGraw-Hill, New York, 2007.
6. Wikipedia, “IEEE 802.3”, 2022 [Online]. Available: https://en.wikipedia.org/wiki/IEEE_802.3
7. Wikipedia, “IEEE 802.11”, 2022 [Online]. Available: https://en.wikipedia.org/wiki/IEEE_802.11

PROCEDURE

- Set up IEEE 802.3 & 802.11 networks, in NetSim, with different topologies.
- Simulate the networks with various configuration options and observe the changes.

- Study the performance of the network in all simulations from data available in the results window and the packet and event trace files. Bring clarity in your understanding by reviewing literature on the standard.

Bring out the understanding from the experiment in the inference section.

IMPLEMENTATION

IEEE 802.3 (Ethernet)

- In NetSim, the 802.3 protocol is implemented by creating a wired LAN with Ethernet-enabled nodes.
- Devices are connected using cables through switches or directly for point-to-point.
- Traffic is generated using applications like FTP or CBR to evaluate performance.
- Simulation provides insights into delay, throughput, and packet flow in wired environments.

IEEE 802.11 (Wi-Fi)

- 802.11 simulation in NetSim involves deploying wireless nodes and an access point.
- Nodes communicate over shared wireless channels using CSMA/CA access control.
- Parameters like range, data rate, and mobility are adjusted to observe real-world behavior.
- Results include signal strength, collisions, and overall wireless network efficiency.

RESULTS

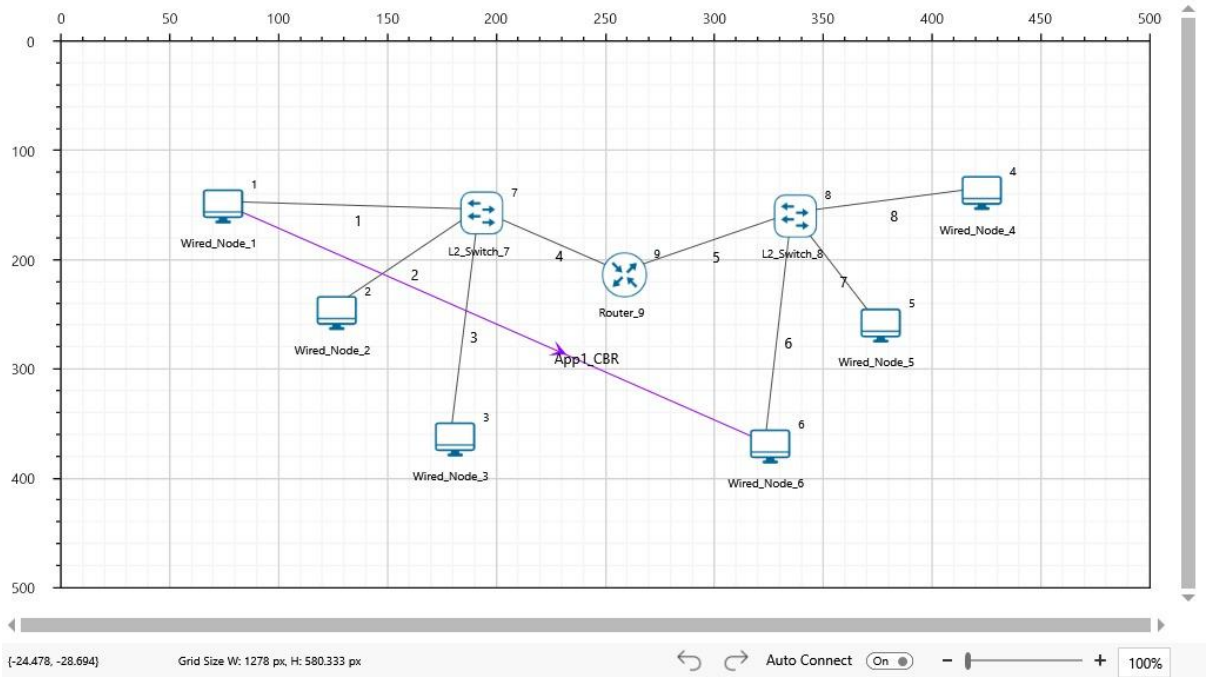


FIG:802.3-Ethernet

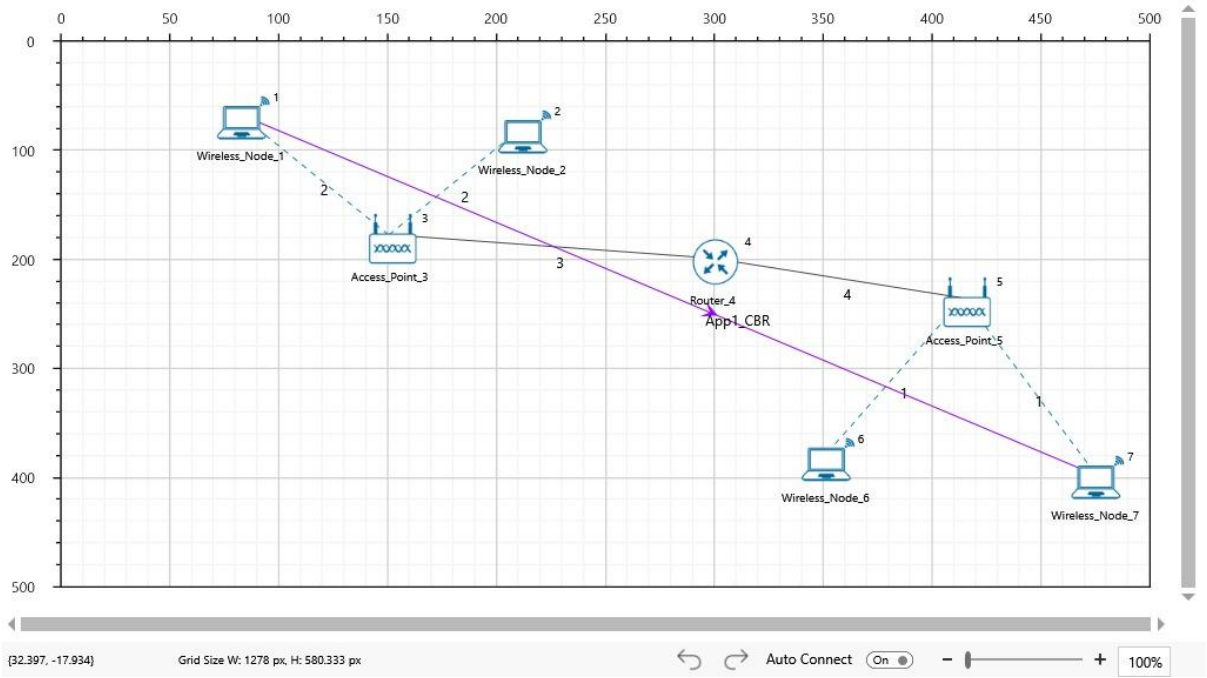


FIG:802.11-Wifi

INFERENCE

In this experiment, IEEE 802.3 (wired) and IEEE 802.11 (wireless) networks were simulated using NetSim. The wired setup showed stable, low-latency communication, while the wireless network enabled flexible connectivity with moderate performance due to signal interference.

Experiment No: 4 **Date: 20.01.2025**

Simulation Study on ZigBee/Wireless Sensor Networks using NetSim

AIM

To simulate ZigBee and Wireless Sensor networks in NetSim and study the operation of the networks under various configurations and scenarios.

THEORY

ZigBee and Wireless Sensor Networks (WSNs) are integral components of modern wireless communication and IoT systems. ZigBee is a wireless communication protocol based on the IEEE 802.15.4 standard, designed for low-power, low-data-rate, and short-range communication. It supports various network topologies such as star, tree, and mesh, allowing for flexible and scalable network design. ZigBee operates in the 2.4 GHz, 868 MHz, and 915 MHz frequency bands and is widely used in home automation, industrial control, and smart metering due to its low cost and energy efficiency. In ZigBee networks, devices play specific roles such as coordinator, router, and end device. The coordinator manages the network and establishes communication, routers help in forwarding data and expanding coverage, and end devices handle sensing or control tasks by communicating through the router or coordinator. These roles collectively contribute to efficient network formation, data routing, and sensing.

Wireless Sensor Networks (WSNs), on the other hand, consist of spatially distributed autonomous sensor nodes that monitor physical or environmental conditions like temperature, humidity, or motion. Each sensor node typically includes a sensing unit, a microcontroller, a wireless transceiver, and a power source. These nodes collaborate to collect and transmit data to a central location, often using communication protocols such as ZigBee. WSNs are extensively used in applications including environmental monitoring, smart agriculture, disaster detection, military surveillance, and smart cities. By integrating the ZigBee protocol with the WSN architecture, efficient, reliable, and low-power wireless communication systems can be developed to support a wide range of real-time monitoring and automation applications.

PROCEDURE

- Set up IEEE ZigBee & Wireless Sensor networks, in NetSim, with different topologies.
- Simulate the networks with various configuration options and observe the changes.
- Study the performance of the network in all simulations from data available in the results window and the packet and event trace files. Bring clarity in your understanding by reviewing literature on the standard.
- Bring out the understanding from the experiment in the inference section.

IMPLEMENTATION

ZigBee

- ZigBee simulation in NetSim uses low-power wireless nodes set up in a star or mesh topology.
- A PAN Coordinator manages communication between ZigBee-enabled end devices.
- The simulation shows how data is routed in energy-efficient networks using AODV or ZigBee routing.
- It's useful for evaluating power consumption, delay, and packet loss in IoT applications.

Wireless Sensor Network (WSN)

- WSN in NetSim is built using sensor nodes that collect and forward data to a sink node.
- Routing protocols like LEACH or AODV help in forming clusters or finding shortest paths.
- Environmental factors like battery, mobility, and transmission range are considered.
- Results show how efficiently the network handles sensing and communication over time.

RESULTS

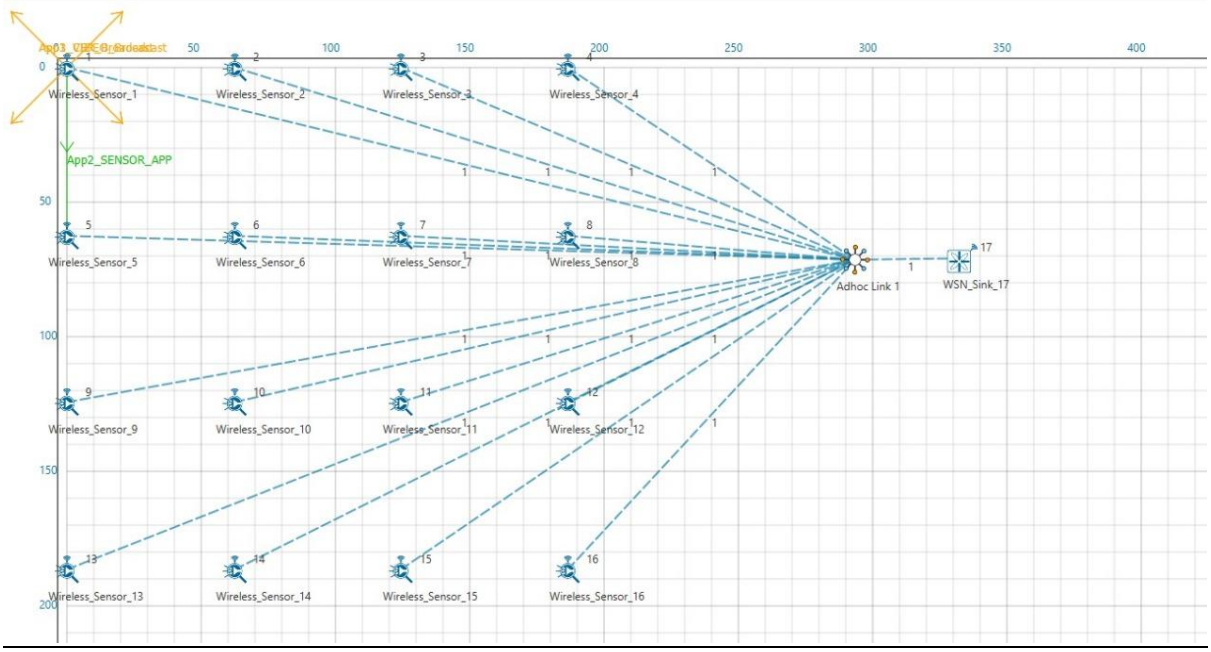


FIG : Wireless Sensor Network

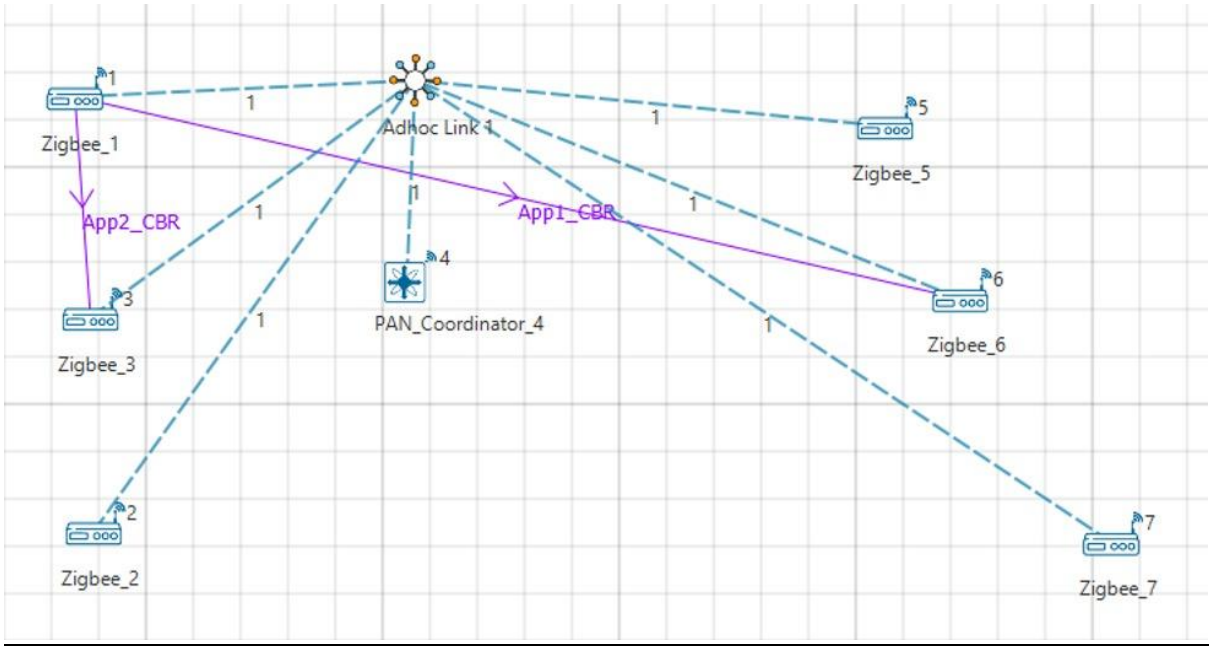


FIG : Zigbee

INFERENCE

In this experiment, ZigBee and Wireless Sensor Networks were simulated using NetSim to observe their behaviour under various configurations. The simulation highlighted ZigBee's energy-efficient communication and WSN's data collection capability with routing adaptability, essential for IoT and remote monitoring applications.

Experiment No: 5 **Date: 25.01.2025**

IoT network simulation in NetSim & Wireshark packet data extraction

AIM

1. To simulate IoT networks in NetSim and study the operation of the networks under various configurations and scenarios.
2. To familiarize Wireshark – a network protocol analyzer software.
3. To study the packet format of various IEEE 802.3 and 802.11 packets using Wireshark.

THEORY

Wireshark is a network packet analyzer that presents captured packet data in as much detail as possible. It helps a user understand what's happening inside a network cable, at a higher level. Wireshark is available for free, is open source, and is one of the best packet analyzers available today. Wireshark can be used by,

- Network administrators use it to troubleshoot network problems
- Network security engineers use it to examine security problems
- QA engineers use it to verify network applications
- Developers use it to debug protocol implementations
- People use it to learn network protocol internals

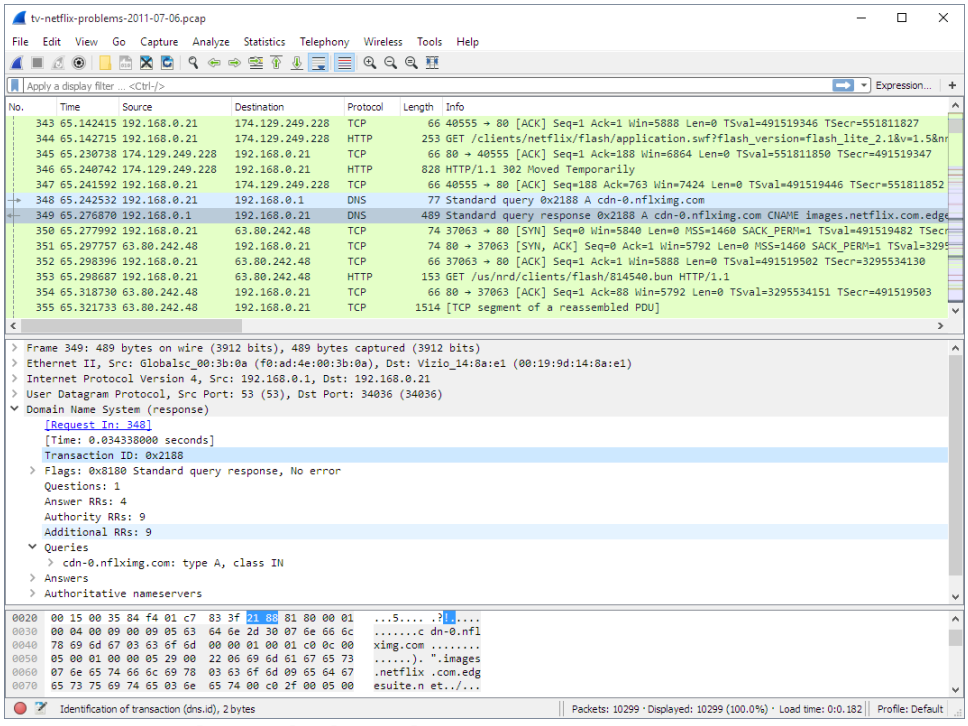


Fig 1: A typical Wireshark window

PROCEDURE

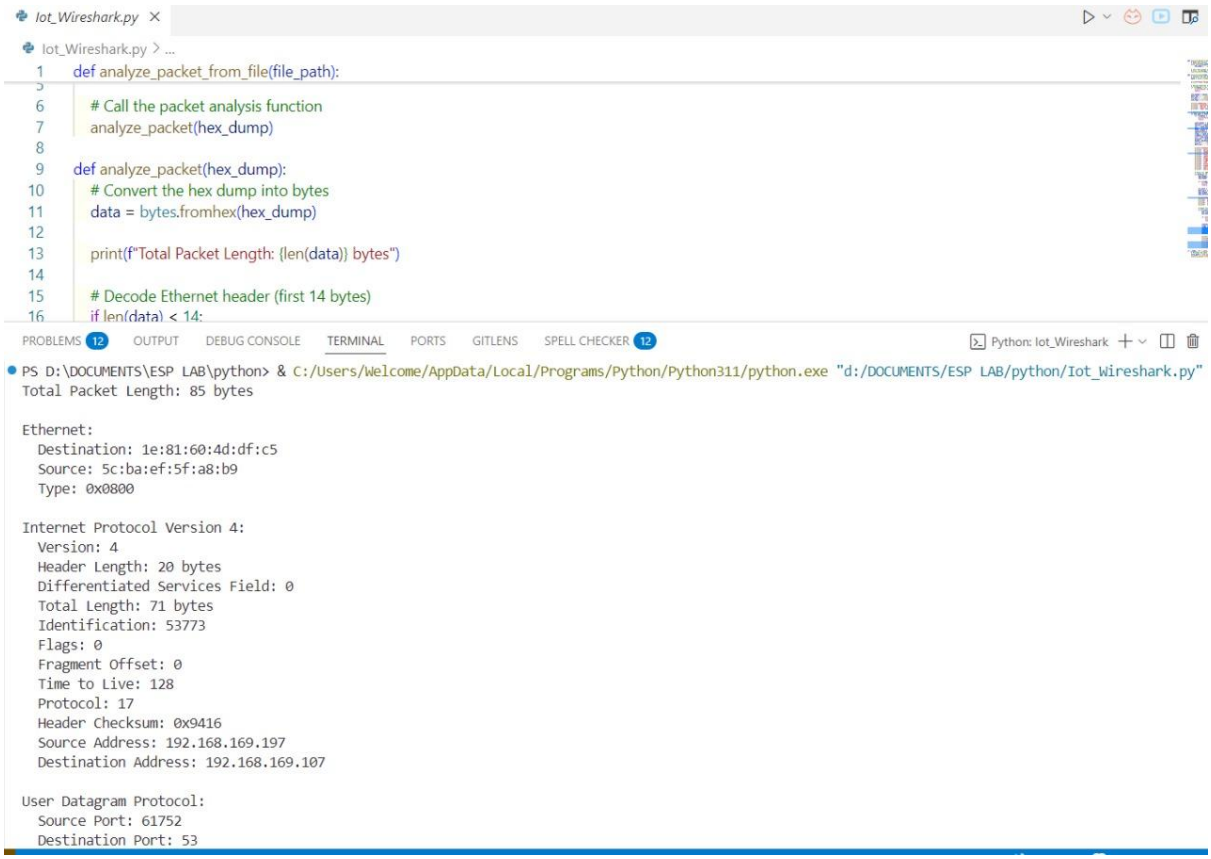
- Set up IEEE 802.3 & 802.11 networks, in NetSim, with different topologies.
- Simulate the networks with various configuration options and observe the changes.
- Study the performance of the network in all simulations from data available in the results window and the packet and event trace files. Bring clarity in your understanding by reviewing literature on the standard.
- Bring out the understanding from the experiment in the inference section.
- Install Wireshark in the laptop.
- With the help of the user manual, understand the software environment.
- Turn the capture ON after selecting the proper interface used for data exchange.
- Select a packet in the packet list pane and study the details in the packet details pane and observe the corresponding information from the packet bytes frame.
- Also, using Edit-Preferences-Layout select packet diagram pane too and understand the protocol frame format for various protocols.
- Export the packet information captured into a .txt file.

IMPLEMENTATION

This implementation represents a simplified version of how packet analyzers like Wireshark operate. The process begins with reading network packet data, typically represented in hexadecimal format. This data is first transformed into a byte sequence to enable interpretation according to networking protocols. The analysis starts at the data link layer with the Ethernet header, which contains essential information such as the destination and source MAC addresses and the EtherType field, which indicates the type of protocol encapsulated in the payload.

If the EtherType denotes an IPv4 packet, the analysis moves to the network layer. Here, details such as the IP version, header length, total packet length, source and destination IP addresses, Time To Live (TTL), and protocol type are extracted. If the protocol specified is UDP, the transport layer is examined next. The UDP header includes fields like source and destination ports, packet length, and checksum. A further inspection is carried out if either port corresponds to 53, which typically indicates DNS traffic. In such cases, the DNS header is decoded to retrieve fields such as the transaction ID, flags, and the count of questions and resource records in the DNS message. This structured, hierarchical decoding of packet data mirrors the way network analyzers dissect packets across the OSI model layers to provide detailed insights into network communication.

RESULTS



Total Packet Length: 81 bytes

Ethernet:

Destination: 00:00:5e:00:01:fe

Source: d4:25:8b:8b:56:c8

Type: 0x0800

Internet Protocol Version 4:

Version: 4

Header Length: 20 bytes

Differentiated Services Field: 0

Total Length: 67 bytes

Identification: 41878

Flags: 0

Fragment Offset: 0

Time to Live: 128

Protocol: 17

Header Checksum: 0x4ac4

Source Address: 10.11.132.47

Destination Address: 172.17.18.4

User Datagram Protocol:

Source Port: 55412

Destination Port: 53

Length: 47

Checksum: 0x5146

Domain Name System:

Transaction ID: 0x0d9b

Flags: 0x0100

Questions: 1

Answer RRs: 0

Authority RRs: 0

Additional RRs: 0

INFERENCE

The experiment provided insight into IoT network configurations using NetSim and detailed packet structures using Wireshark. Protocols like Ethernet, IP, UDP, and DNS were analysed to understand data flow across network layers.

Experiment No: 6 **Date: 27.01.2025**

Implementation of Socket connection using microcontroller board and PC/Laptop

AIM

1. To familiarize socket programming for TCP/IP client-server communication.
2. To develop programs to demonstrate client-server programming using any microcontroller and PC.

THEORY

Sockets are endpoints built for sending and receiving data. A single network will have two sockets, one for each communicating device or program. These sockets are a combination of an IP address and a Port. The socket application programming interface (API) is used to send messages across a network. A network socket is bound to the combination of a type of network protocol to be used for transmissions, a network address of the host, and a port number.

An application can communicate with a remote process by exchanging data with TCP/IP by knowing the combination of protocol type, IP address, and port number. This combination is often known as a socket address. It is the network-facing access handle to the network socket. The API that programs use to communicate with the protocol stack, using network sockets, is called a socket API. The development of application programs that utilize this API is called socket programming or network programming.

In computing, a server is a piece of computer hardware or software (computer program) that provides functionality for other programs or devices, called "clients". Servers can provide various functionalities, often called "services", such as sharing data or resources among multiple clients or performing computation for a client. In computing, a client is a piece of computer hardware or software that accesses a service made available by a server as part of the client-server model of computer networks. In a client-server module, clients request services from servers.

The commonly used API functions [1] and methods for socket program are:

- socket()
- .bind()
- .listen()
- .accept()
- .connect()
- .sendall()
- .revc()
- .close()

PROCEDURE

- Create a network of two computers using Wi-Fi or Ethernet.
- In the first computer run the sample server code.
- In the second computer run the sample client code.

- Study the functions and methods used for the implementation of server and client.
- Create your own server and client code to demonstrate client-server programming using any microcontroller and PC

FLOW CHART

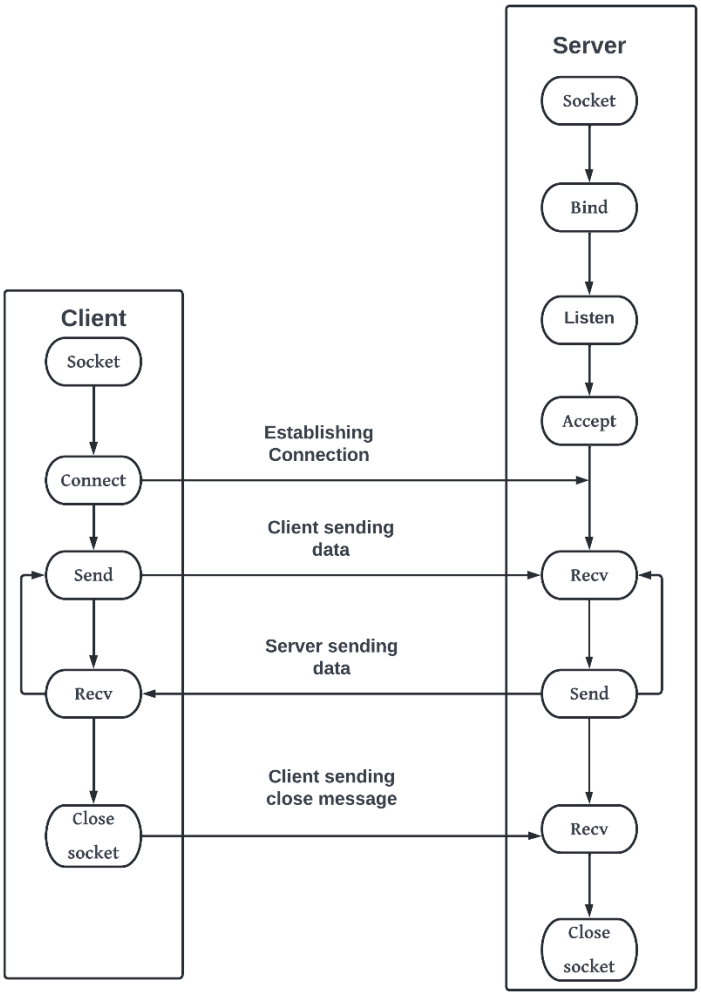


Figure 1. TCP socket flow

REFERENCE

1. Socket — Low-level networking interface, 2022[Online] Available: https://docs.python.org/3/library/socket.html

Experiment No: 7, 8, 9 **Date: 22.04.2025**

Implementation of an IoT Edge Node

AIM

1. To implement an edge node using microcontrollers and communication modules.
2. To implement and demonstrate edge computing capability in the edge node developed.
3. To demonstrate the operation of a full-fledged IoT edge node.

TOOLS & SYSTEMS

Hardware:

1. ESP32 (Edge Node)
2. Arduino UNO
3. HM-10 Bluetooth Module
4. DHT11 Sensor
5. MQ2 Gas Sensor
6. MQ135 Gas Sensor
7. HX711 Load Cell
8. 12V DC Fan and Humidifier
9. Flame Detector
10. Relay Modules
11. Motor Driver

Software:

1. Arduino IDE

IMPLEMENTATION

The **Edge Node** in your Smart Onion Preservation System, implemented using an **ESP32 microcontroller**, serves as a key component that integrates **sensor data collection, local processing**, and **cloud communication**. The role of this node is to manage real-time data from multiple **BLE-based End Nodes**, process it locally, and send meaningful data to a cloud server for remote monitoring.

1**. Wi-Fi Connectivity**:

- The Edge Node begins by establishing a **Wi-Fi connection** using the credentials provided for the local network (SSID and password). The ESP32 connects to Wi-Fi and waits for a successful connection before proceeding with data collection and cloud communication.
- If Wi-Fi connectivity is established successfully, the Edge Node is ready to interact with other devices and transmit data.

2. **BLE Communication Setup**:

- The Edge Node communicates with two **BLE-enabled End Nodes** that collect environmental data (temperature, humidity, methane gas levels, flame detection, and weight).
- These nodes are identified by their **MAC addresses**, and the Edge Node connects to them using the **NimBLE** library for efficient BLE communication.
- The **service** and **characteristic UUIDs** for the BLE communication protocol are defined for each End Node, which ensures the proper exchange of data between devices.

3. **Sensor Data Collection**:

- The Edge Node listens for **notifications** from both End Nodes via BLE, receiving data that includes **temperature**, **humidity**, **methane levels**, **weight**, and **flame sensor status**.
- This data is stored temporarily in **JSON format** for easy parsing and analysis. Once the data is received, it is parsed into structured values such as temperature, humidity, and methane levels.

4. **Data Validation and Processing**:

- The sensor data, especially from **humidity sensors**, is processed by calculating the **average humidity** over a rolling buffer. This ensures smoother data handling and prevents sudden spikes or drops in the sensor readings from affecting the system's decisions.
- Additionally, **flame detection** is monitored in real-time, and if a value exceeds a threshold (indicating a potential fire), an immediate warning is triggered.

5. **Local Decision Making (Edge Computing)**:

- The Edge Node can perform **local processing** based on the received data. For example:
    - If the **humidity** is below 60%, the system triggers the **humidifier** to turn on and adjusts the **fan** speed to **half**.
    - If the **humidity** is within the optimal range (60%-69%), both the **humidifier** and **fan** are turned off.
    - If the **humidity** exceeds 70%, the system turns off the **humidifier** and operates the **fan at full speed** to reduce moisture.
- This **local decision-making** ensures quick and responsive actions without waiting for cloud updates, minimizing latency and improving system reliability.

6. **Data Queuing for Transmission**:

- The data received from the sensors is temporarily stored in a **FIFO (First In, First Out) queue**. This queue helps manage the data flow by ensuring that only the most recent and relevant data is sent to the cloud.
- Each sensor data set is processed and added to the queue. If the queue is full, the system will indicate this and stop adding new data, preventing potential data loss or overflow.

7. **Data Transmission to Cloud**:

- After data validation and processing, the Edge Node transmits the sensor data to a cloud server using the **HTTP POST** method. The data is sent as a **JSON payload**, which includes:
    - Warehouse ID
    - Temperature
    - Humidity
    - Methane level
    - Weight
    - Flame status
    - Average humidity (computed locally)
- The data is sent to the server URL (e.g., https://onion-server-g77l.onrender.com/api/update-sensors), allowing the cloud-based monitoring system to update the dashboard in real-time.

8. **Remote Monitoring and Control**:

- The Edge Node also enables **remote monitoring** and **control** through the web dashboard. Once the data is sent to the cloud, it can be displayed on a web interface using platforms like **ReactJS** and **Firebase**.
- Additionally, the system allows for **remote control** of actuators (e.g., fan and humidifier) based on the user's commands from the dashboard. This two-way communication is facilitated by the Edge Node.

RESULTS

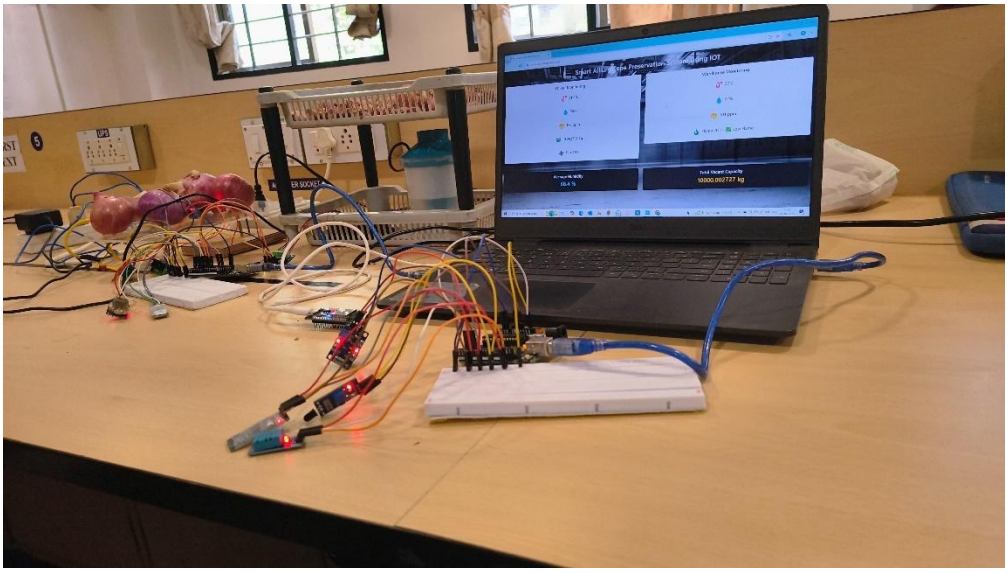


Fig. Entire Setup

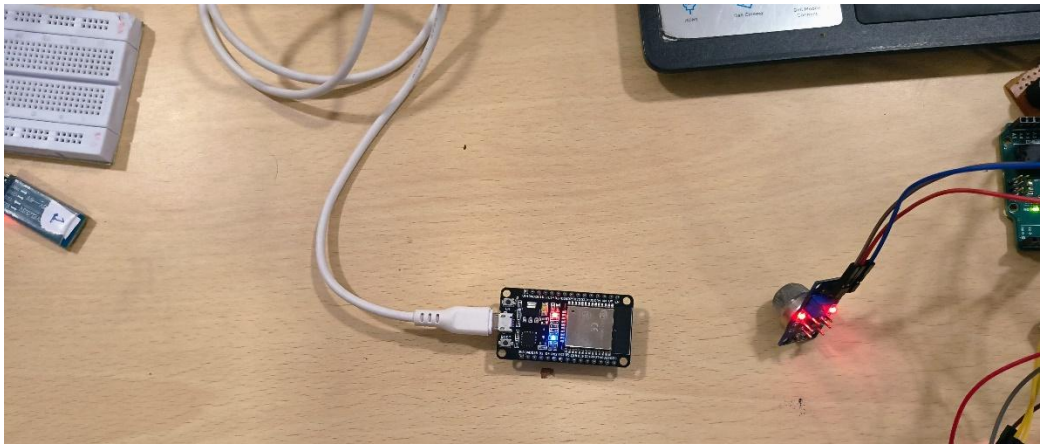


Fig. Edge Node

INFERENCE:

The experiment successfully demonstrated the implementation of an IoT Edge Node using ESP32, enabling efficient local data collection, edge-level processing, and cloud integration. The system performed real-time decision-making based on sensor inputs and enabled remote monitoring and actuator control, validating the core principles of edge computing in a practical setup.

Experiment No: 10,11,12 **Date: 22.04.2025**

Implementation of a Local Server with UI & Database

AIM

1. To implement a local server that can receive data from multiple client nodes.
2. To develop a data storage mechanism in server machine to store client data.
3. To develop a UI for the server to visualize the data of client nodes.

TOOLS & SYSTEMS

Hardware Used:

1. Arduino UNO
2. ESP32
3. HM-10 Bluetooth Module
4. DHT11 Sensor
5. MQ2 Gas Sensor
6. MQ135 Gas Sensor
7. HX711 Load Cell
8. 12V DC Fan and Humidifier
9. Flame Detector
10. Relay Modules
11. Motor Driver

Software Used:

1. Arduino IDE
2. Firebase
3. ReactJS
4. Node.js
5. React-Toastify
6. Cloud Server (Render)

IMPLEMENTATION

1. Local Server with Database

- **Purpose**: The local server is responsible for processing sensor data and updating the database with real-time information. It communicates with the Edge Node (ESP32), which gathers sensor data from the End Nodes (Arduino-based).
- **Database**: The **Firebase Realtime Database** is used to store data such as temperature, humidity, gas levels, weight, and flame status. It ensures real-time synchronization across all connected devices.
- **Data Flow**:

- The **ESP32** sends sensor data from the End Nodes (which includes various environmental parameters) to the local server.
- The server processes this data and stores it in the Firebase Realtime Database.
- The database updates are instant, providing real-time access to the system's status.

2. GUI (Web Dashboard)

- **Purpose**: The web dashboard, built using **ReactJS**, allows users to monitor warehouse conditions and control system components (such as fans and humidifiers) remotely.
- **Features**:
  - **Real-time Monitoring**: Displays temperature, humidity, weight, gas levels, and flame status from both warehouses.
  - **Control Mechanism**: Allows users to control the fan and humidifier based on real-time data.
  - **Notifications**: Alerts users in case of abnormal conditions, such as high gas levels or temperature changes, through the **react-toastify** library.
- **User Interaction**:
  - The dashboard fetches data from Firebase and updates the display in real-time.
  - Users can manually control actuators (fan and humidifier) by sending commands to the backend server, which then updates the Edge Node.
- **Notifications**: The system alerts users to potential risks like fire hazards or imbalanced environmental conditions, ensuring timely intervention.

Key Benefits:

1. **Real-Time Data**: Continuous monitoring of environmental parameters ensures effective storage management.
2. **Cloud Integration**: Firebase enables centralized, scalable data storage accessible from any location.
3. **User-Friendly Interface**: The ReactJS dashboard provides an intuitive way to manage storage conditions and respond to alerts.
4. **Automated Alerts**: Notifications help users respond to issues like fire hazards or humidity imbalances before they cause damage.

RESULTS

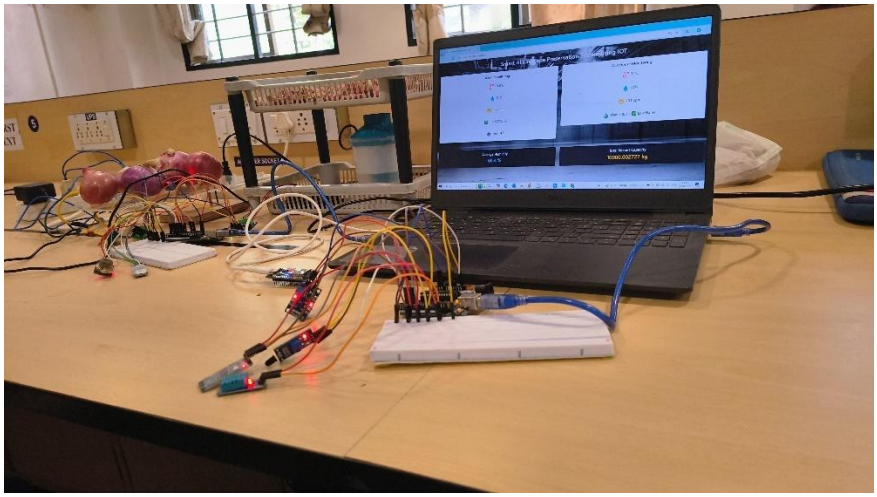


Fig. Entire Setup

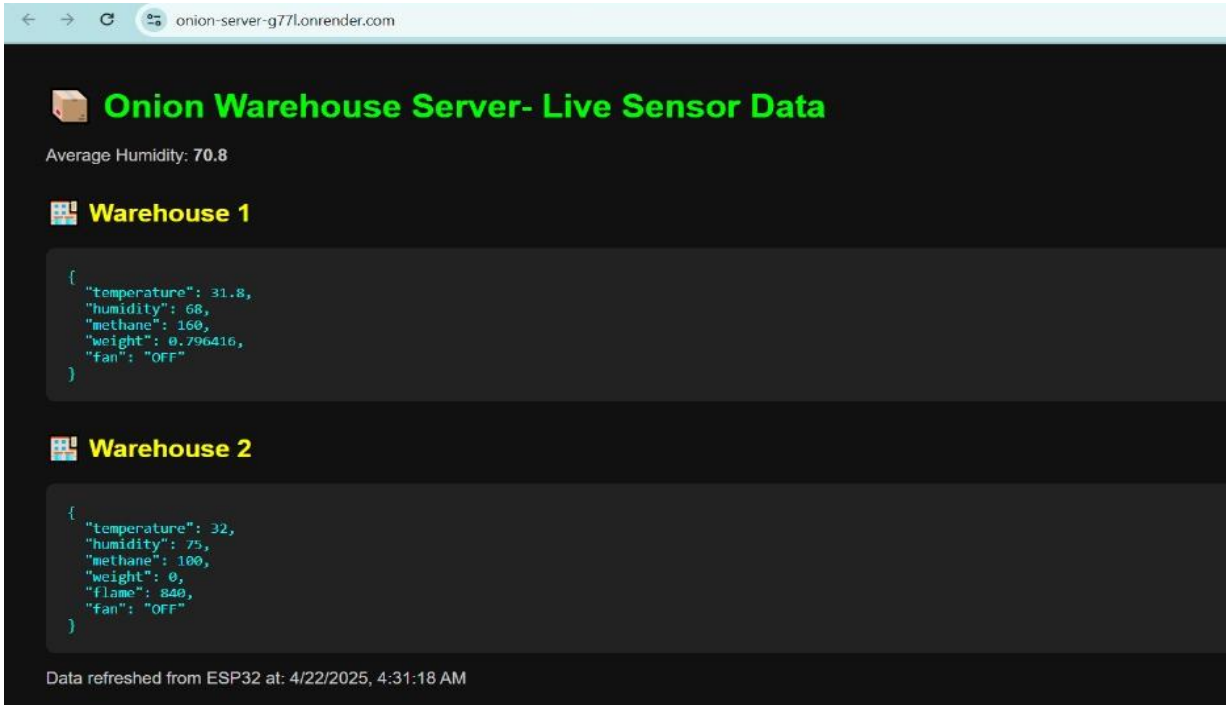


Fig. Server

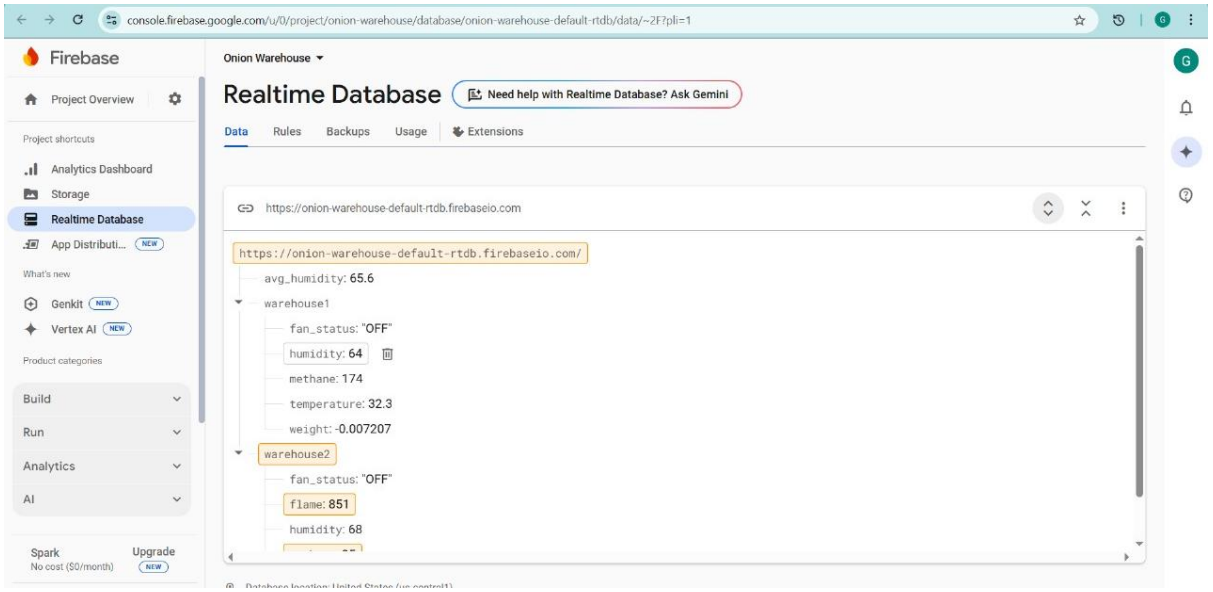


Fig. Realtime Database

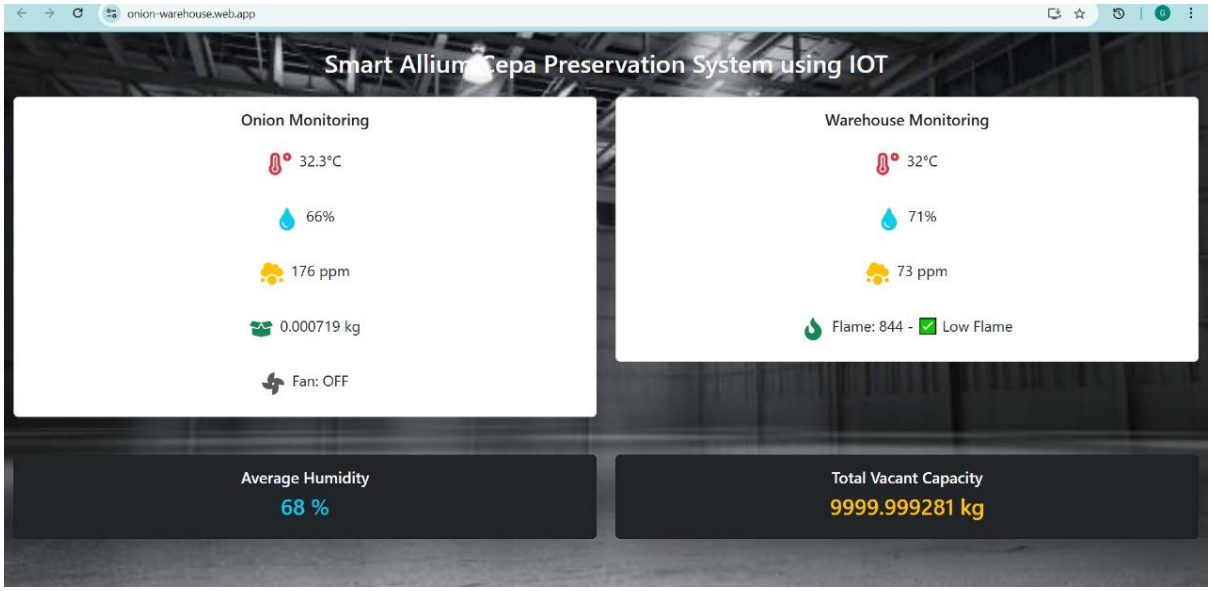


Fig. Graphical User Interface

INFERENCE:

The experiment successfully established a local server integrated with Firebase, enabling real-time data collection, storage, and visualization from multiple IoT nodes. The ReactJS-based UI allowed effective monitoring and control of actuators, enhancing environmental management and providing timely alerts for critical conditions.

SMART ALLIUM CEPA PRESERVATION SYSTEM USING IOT

21ES614 - INTERNET OF THINGS PROJECT REPORT

Submitted by

CB.EN.P2EBS24008 GIRISHANKAR P L

CB.EN.P2EBS24009 GOKUL S

CB.EN.P2EBS24024 ANAND P S

MASTER OF TECHNOLOGY
IN EMBEDDED SYSTEMS

DEPARTMENT OF ELECTRICAL
AND ELECTRONICS ENGINEERING

AMRITA SCHOOL OF ENGINEERING, COIMBATORE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112

APRIL 2025

Contents

List of Figures

Abstract

Onion storage post-harvest losses are a major concern for producers and suppliers, and are generally brought about by unfavorable environmental conditions, such as high temperatures, low humidity, and build-up of gases and other factors. The focus of this project is to design an intelligent storage system for onions that will monitor and control the storage environment using embedded systems and Internet of Things (IoT) technology. The system has embedded sensors that monitor temperature, humidity, gas levels, and fire hazards continuously within the storage environment. Depending on the data collected, the system can automatically control equipment like fans and humidifiers to create perfect conditions for onion storage.

The data collected is sent to a central processing unit, which interprets the data and then sends it to a cloud-based system. A web-based interface, which is easy to use, displays real-time environmental data and offers remote access to the system. In case of any unusual fluctuations, warnings and alerts are sent, allowing timely interventions to prevent spoilage.

This project minimizes the requirement for manual checks and guarantees that onions are kept in a safe and efficient setting, thereby minimizing financial losses. In future applications, the system can be able to incorporate machine learning technologies to forecast spoilage and recommend improvements. Generally, it is an intelligent and automated solution for the optimization of post-harvest onion management practices.

Chapter 1

Introduction

Onions (*Allium cepa*) are highly perishable and sensitive to environmental conditions, particularly humidity, temperature, and air quality. Poor post-harvest storage practices frequently result in spoilage, leading to significant financial losses. This project presents a smart and cost-effective IoT-based system for onion preservation in warehouse environments through real-time monitoring and control.

The proposed system comprises two Arduino UNO-based End Nodes and an ESP32-based Edge Node. End Node 1 represents the onion preservation unit. It communicates with the Edge Node via HM-10 Bluetooth modules, ensuring low-power, short-range wireless data transfer. This node integrates a DHT11 temperature and humidity sensor, an MQ2 gas sensor, an HX711 load cell sensor for weight measurement, and a 12V DC fan and humidifier.

Based on humidity levels, the system operates as follows:

- **Below 60% humidity**: The fan operates at half speed, and the humidifier is turned on.
- **Between 60%–69% humidity**: Both fan and humidifier remain off.
- **Above 70% humidity**: The fan runs at full speed to prevent excessive moisture accumulation.

End Node 2 monitors the warehouse environment using a DHT11 sensor for temperature, an MQ135 sensor for air quality, and a fan for ventilation. Additionally, it incorporates a flame detector that activates during fire hazards.

Sensor data from both nodes are transmitted periodically to the ESP32 Wi-Fi development board, which acts as the central Bluetooth Low Energy (BLE) coordinator. The ESP32 reads, parses, and sends the data in JSON format via HTTP to a cloud server hosted on Render. This server updates a live Firebase Realtime Database.

A web dashboard, developed using ReactJS and hosted via Firebase Hosting, accesses this database. It provides real-time monitoring of warehouse parameters including temperature, humidity, weight, gas level, and flame status. The dashboard supports remote fan control and delivers notifications using the `react-toastify` library.

By integrating Bluetooth, Wi-Fi, Firebase, and modern web technologies, the system offers an efficient, scalable, and user-friendly solution for smart onion storage management. It facilitates early risk detection, ensures optimal storage conditions, and enables users to respond promptly through a cloud-based interface.

Chapter 2

Literature Survey

2.1 IoT-Based Onion Storage Monitoring System

This article describes the construction of an IoT-based system for monitoring and controlling environmental conditions in onion storage rooms. It uses sensors to detect temperature, humidity, and gas concentrations—crucial parameters in ensuring onion quality during storage. A microcontroller such as an Arduino or ESP8266 handles the data and uploads it to the cloud, where it's stored and displayed on a web dashboard for real-time remote monitoring. Actuators such as fans and humidifiers are automatically regulated through sensor data, maximizing conditions. Cloud-based platforms increase real-time decision-making, which suits your project's goal of efficient onion storage through automation and continuous monitoring.

2.2 Smart Energy Management System for Onion Preservation

This research suggests a smart energy management system that keeps onions fresh while maximizing energy usage. IoT sensors, such as temperature and humidity sensors, track storage conditions. The microcontroller handles this information to govern fans and humidifiers, functioning only as needed to keep humidity at optimal levels. This energy-specific application optimizes storage while keeping energy usage to a minimum and costs of operation low. The emphasis on energy efficiency makes this system directly applicable to your project, providing a method that optimizes preservation with an eye towards sustainability.

2.3 IoT-Based Smart Onion Storage System

This work presents a remotely controlled, real-time IoT system specifically designed for onion storage. With the aid of sensors such as DHT11 for temperature and humidity, and gas sensors, it transmits data over Wi-Fi to a cloud dashboard. Environmental parameters exceeding the preferred range trigger alerts. Remote control of actuators to regulate conditions to prevent

spoilage and prolong onion shelf life is also supported by the system. This system caters to your project's objectives of real-time monitoring, remote access, and automatic environmental controls.

2.4 IoT-Enabled Onion Growth Monitoring System Using Cloud

While this research is directed to onion growth as compared to storage, it offers useful insights into cloud-based monitoring by using IoT sensors for soil moisture, temperature, and light intensity parameters. Data is displayed in the form of real-time graphs that can be accessed via cloud platforms. It can be extended to post-harvest storage as well, where the same sensor networks assist in keeping conditions optimal for maintaining onion quality through data analysis and control in real time.

2.5 IoT-Based Onion Storage System with Data Analytics

This paper highlights how IoT and data analytics can be combined for efficient onion storage management. Sensors like DHT11 track temperature and humidity, with data transmitted to a cloud platform via ESP8266. Real-time dashboards display storage conditions, while predictive analytics uses historical data to anticipate and adjust for environmental changes proactively. Integrating data-driven decision-making adds a powerful layer to your project, enabling smarter, preventive measures that ensure storage remains consistently optimal.

2.6 Impact of Temperature and Humidity on Storage of Onions

This research investigates the role of temperature and humidity in onion storage. Stable conditions are essential to prevent spoilage from sprouting, mold, or dehydration. Using IoT-based sensors, the system continuously monitors these parameters and issues alerts when thresholds are breached. The emphasis on environmental stability directly supports your project's core objectives, showing that consistent real-time monitoring is critical to extending shelf life and maintaining onion quality.

2.7 Detection of Onion Leaf Disease by Feature Extraction and Selection

While centered on disease detection in onion plants, this article presents applicable techniques such as GLCM and LBP for feature extraction in image analysis, which would be useful for feature extraction in image analysis. These methods would allow for early detection of abnormalities. While your project is centered around post-harvest storage, the early detection and monitoring concepts could be extended to identify marks of deterioration or contamination storage for enhanced quality control and safety.

2.8 Radiation-Based Preservation for Nutritional Quality

This research investigates the application of gamma and electron radiation to store onions by increasing shelf life without loss of nutritional value. Although not IoT-related, radiation may supplement your project's sensor-based monitoring with an added preservation layer. A system that integrates environmental monitoring and radiation treatment could provide a complete solution for long-term storage and less spoilage.

2.9 Preservation of Onion Pollen for Seed Production

With a focus on onion pollen preservation, this article highlights controlled humidity and temperature for seed viability. Methods like dehydration and cold storage are covered. While it aims at seed storage, these learnings can be applied to onion bulb storage, especially in preventing sprouting and overall quality maintenance. Adding the same environmental controls to your IoT system may improve onion preservation further.

2.10 Easy and Effective Onion Seed Storage Method

This study provides a simple technique for onion seed preservation using controlled humidity and temperature, focusing on dehydration and cold storage. These are basic storage principles that are in direct correspondence with your project goals. With the use of IoT-based sensors to mimic such conditions, your system will be able to create ideal environments, lower spoilage rates, and make onions available for longer periods.

Chapter 3

Objectives

The project aims to implement and integrate an intelligent onion storage system in warehouses using embedded systems and Internet of Things (IoT) technologies. The focus is on real-time monitoring, automatic control, and remote accessibility to guarantee favorable storage conditions, hence minimizing spoilage and economic loss due to undesirable environmental conditions.

3.1 Primary Objectives

3.1.1 To develop an intelligent onion storage system with embedded hardware and IoT platforms

The system is intended to automate monitoring and control of environmental conditions affecting the shelf life of onions. Through the integration of sensors, microcontrollers, and wireless communication, it minimizes the need for human intervention and maximizes the reliability of storage facilities.

3.1.2 To monitor environmental parameters in real-time

Critical storage parameters such as temperature, humidity, gas level, and weight are monitored in real time by sensors like DHT11, MQ2, MQ135, HX711, and a flame sensor. Real-time monitoring detects suboptimal conditions early, thus allowing proactive action.

3.1.3 To enable remote monitoring and control over a web dashboard

The data collected by the sensor nodes is made accessible through a web interface. This dashboard, developed using ReactJS, displays real-time values and offers the possibility of remote control of devices such as fans and humidifiers, thereby enhancing user convenience and operating efficiency.

3.2 Secondary Objectives

3.2.1 To implement condition-based control of actuators

The system will automatically regulate the operation of fans and humidifiers based on the readings of humidity levels detected:

- **Under 60%:** Fan runs at half speed, and humidifier is on to introduce moisture.
- **60% to 69%:** Fan and humidifier are turned off because the environment is perfect.
- **Over 70%:** Fan runs at full speed to counteract excessive humidity.

This regulated rule keeps onions stored under stable conditions.

3.2.2 To improve security by identifying risks and alarm mechanisms

A fire sensor and gas sensors are integrated in order to detect fire risks or toxic gas levels. In case of detected risks, they issue alerts instantly through the web dashboard so that users can take immediate action and prevent excessive damage.

3.2.3 To store historic data for trends

Sensor data is uploaded to a Firebase Realtime Database and saved with timestamps. This allows users to review over time, make smart decisions, and improve warehouse conditions based on previous patterns.

3.3 Technical Objectives

3.3.1 To design and implement sensor nodes using Arduino UNO

Two Arduino UNO boards serve as End Nodes. These nodes collect environmental information with various sensors and send it wirelessly to the central node. They form the foundation of the sensing layer of the system.

3.3.2 To establish wireless communication based on HM-10 Bluetooth modules

BLE modules are used to connect End Nodes to the ESP32-based Edge Node. It ensures efficient data transfer with less power consumption, which is suitable for indoor warehouse environments.

3.3.3 To utilize ESP32 for data aggregation and cloud communication

The ESP32 microcontroller serves as the main coordinator. It gathers information from both End Nodes, processes, and sends to the cloud server via Wi-Fi. It also translates the information into JSON format for simpler web platform integration.

3.3.4 For storing and synchronizing data through Firebase Realtime Database

Firebase is used as the back-end platform to maintain and store incoming data. It is made possible for real-time data syncing with the dashboard such that users receive the most recent information in real-time without any latency.

3.3.5 To create a responsive dashboard with ReactJS

An intuitive dashboard is designed to display real-time sensor data and notifications. Remote hardware control and the use of ReactJS components like `useState` and `useEffect` are utilized to handle data fetching and UI updates in a smooth manner. The notifications are handled using the `react-toastify` library.

3.3.6 To format data using JSON for efficient communication

All sensor data is in JSON format before it gets sent over to the server. This is to ensure consistency and ease parsing and displaying the data on the web dashboard.

Chapter 4

Methodology

The process of designing this intelligent onion preservation system was taken in a step-by-step iterative approach by integrating embedded systems, wireless technology, and cloud technology to tackle real-world storage problems in agriculture.

4.1 Problem Identification and Parameter Selection

The project started with a thorough review of the key factors leading to post-harvest onion spoilage. Environment parameters such as humidity, temperature, and air quality were considered vital. Fire safety and weight monitoring in real-time were also key in a full-fledged monitoring system. This step made it easier to define the hardware and software specifications for the system.

4.2 Modular System Architecture

A modular architecture was followed to guarantee scalability and readability of operations. The system architecture was segregated into:

- **End Node 1:** Dedicated to the task of monitoring and maintaining the best storage conditions within the onion chamber.
- **End Node 2:** Directed towards more general warehouse environment monitoring.
- **Edge Node (ESP32):** Used as the central communication gateway and cloud interface.

4.3 Sensor and Actuator Integration

Each node had suitable sensors and actuators:

- **End Node 1** had a DHT11 (temperature/humidity), MQ2 gas sensor, HX711 with load cell (for weight), 12V DC fan, and humidifier.

- Control Logic:
  - $< 60\%$ humidity: Fan is half speed; humidifier is on.
  - $60\% - 69\%$ humidity: Fan and humidifier are off.
  - $> 70\%$ humidity: Fan at full speed; humidifier off.
- **End Node 2** contained another DHT11, an MQ135 air quality sensor, a flame sensor, and a fan for overall warehouse ventilation.

4.4 Wireless Communication Implementation

For low-power and reliable communication, Bluetooth Low Energy (BLE) was employed. Both Arduino-based End Nodes sent data to the ESP32 Edge Node via HM-10 BLE modules. The ESP32 was a BLE master, scanning and reading sensor data from both End Nodes continuously.

4.5 Cloud Connectivity and Data Management

After data was gathered by the ESP32, it was parsed and formatted into JSON format. The ESP32 utilized HTTP to send this data to a cloud server hosted on Render, which updated a Firebase Realtime Database. This provided smooth and continuous data storage available to users at any time.

4.6 Web Dashboard Development

A personalized ReactJS dashboard was created to show real-time environmental data from both nodes. Main characteristics are:

- Automatic real-time updates of temperature, humidity, gas level, flame, and onion weight.
- Manual control switches for fan on/off.
- Implemented notification system using react-toastify for timely notifications.

The dashboard was hosted using Firebase Hosting, so it could be accessed through mobile and desktop browsers, making it easy and straightforward to monitor remotely.

4.7 System Testing and Optimization

Thorough testing was done in a simulated warehouse scenario. Readings from each sensor were validated for accuracy, and actuation timings were optimized. BLE reliability was tested for repeatability across multiple cycles. Cloud connectivity and dashboard refresh was tested for real-time behavior.

Chapter 5

System Overview

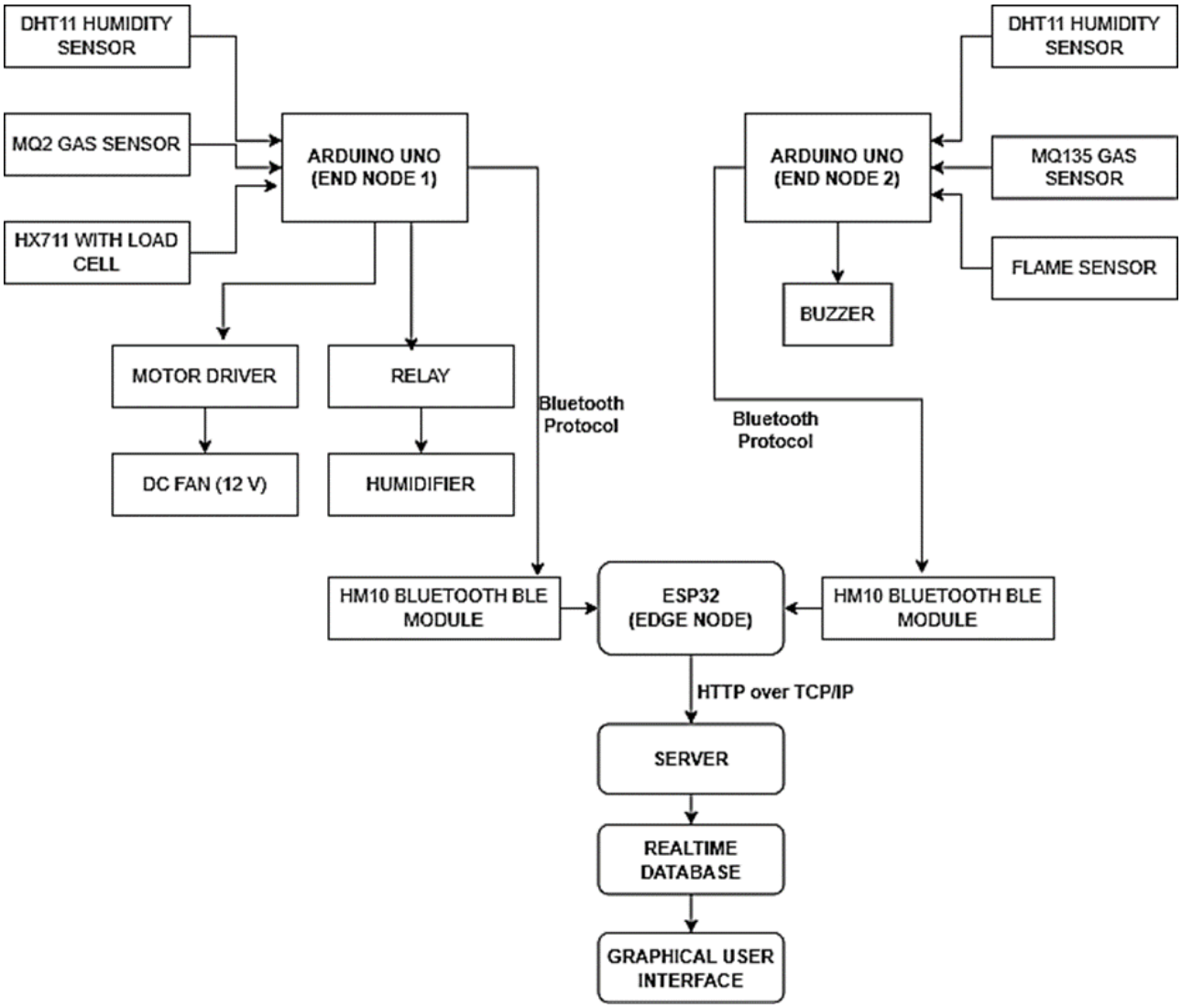


Figure 5.1: Block Diagram

5.1 End Nodes

The End Nodes are tasked with sensing and gathering environment data in the warehouse used to store onions. There are two End Nodes within the system, and they both have their unique tasks undertaken with different sensors to track the status of the warehouse environment.

- **End Node 1** takes care of the onion preservation unit. It is assigned the following sensors:
  - DHT11: Temperature and humidity sensing.
  - MQ2: Gas level sensing (e.g., methane, CO2).
  - HX711: Weighs the onions through a load cell.

  All this data is sent to the Edge Node for further processing. Actuators such as:
  - 12V DC Fan (to control airflow depending upon humidity) are also controlled by End Node 1.
  - Humidifier (to keep the humidity at optimal levels for onion storage).

  The system controls the actuators according to certain humidity levels:
  - Less than 60% humidity: The fan is at half speed and the humidifier is activated.
  - 60%-69% humidity: Both the fan and humidifier are deactivated.
  - More than 70% humidity: The fan is at full speed to avoid too much moisture buildup.
- **End Node 2** observes the general warehouse setting:
  - DHT11: For temperature and humidity sensing.
  - MQ135: Senses air quality.
  - Flame Sensor: For sensing fire hazards.
  - Fan: Turned on for ventilation depending on air quality or emergency situations.

  The End Nodes communicate with the Edge Node via Bluetooth Low Energy (BLE). The data is transmitted at regular intervals, ensuring that the **Edge Node** always has up-to-date environmental data to make decisions.

5.2 Edge Node

The Edge Node functions as a central processing unit of the system. It is the interface between the End Nodes and the cloud server. The **Edge Node** uses an ESP32 microcontroller, which has Wi-Fi and Bluetooth capabilities.

- **Data Processing:**
  - The Edge Node summarizes information from the End Nodes (temperature, humidity, gas concentration, flame detection) and carries out elementary data processing. As an illustration, it computes the **average humidity of both End Nodes** in order to supply a better overall environmental reading for the warehouse.
  - The Edge Node can also take actions such as protocol conversion, converting data received through Bluetooth Low Energy (BLE) into a format that can be transmitted over the Internet (HTTP over TCP/IP).
- **Protocol Conversion:** The Edge Node is tasked with protocol conversion. While **End Nodes** are communicating to the Edge Node in Bluetooth, the **Edge Node** sends the data to the cloud server using HTTP through TCP/IP. Such a conversion makes the system possible to send information to the cloud and allows it to be remotely accessed via the internet.
- **Communication with Server:** Once the data is aggregated and processed, the Edge Node transmits the data to the cloud server via HTTP over TCP/IP. It is transmitted in the **JSON format**, which renders the data readily comprehensible and accessible to the cloud services.

5.3 Server (Cloud Server)

The Cloud Server contains the system's centralized processing and data storage system. The **server** is where all of the information gathered from the End Nodes through the Edge Node is stored, processed, and available for use.

- **Firebase Realtime Database:** The cloud server supports a Firebase Realtime Database, where data from all the sensors (temperature, humidity, gas levels, flame status, and weight readings) is stored. The database is updated in real time as fresh data is streamed from the **Edge Node**.
- **Real-Time Data Updates:** Any environmental change (e.g., a humidity spike or fire alarm) is instantly updated in the **Firebase Database**, keeping the data available in real-time.
- **Data Storage and Processing:** The server takes care of having a secure and trustworthy storage mechanism for sensor data. The data is then available to be analyzed, patterns recognized, and even fed into machine learning models in the future.
- The server makes sure that data is archived and accessible via various interfaces, such as the Web Dashboard.

5.4 Database (Firebase Realtime Database)

The Firebase Realtime Database is the core of the data storage for your system. It holds live sensor data, updated in real time, that is essential to monitor the warehouse environment.

- **Data Storage:** The Firebase Database holds a range of data types such as temperature, humidity, gas levels, and flame detection status of both End Nodes. It also holds the control signals for actuators such as the fan and humidifier, which are activated according to pre-defined conditions (e.g., humidity levels).
- **Real-Time Synchronization:** The Firebase Realtime Database guarantees that data is synchronized between all devices logged into the system. This means that any alteration to the warehouse environment (for example, a sudden shift in humidity or an unexpected gas leak) is automatically reflected in the system, and the Web Dashboard is able to report it without latency.
- **Notifications and Alerts:** The Firebase Database is also able to send out warnings when a specified condition occurs. For instance, if the flame sensor identifies an outbreak of flames, the user can immediately receive a warning in the form of the Web Dashboard.

5.5 GUI (Web Dashboard)

The Web Dashboard is the user interface of the overall system. The Web Dashboard gives real-time visibility to the warehouse setup, such as temperature, humidity, gas readings, flame signals, and statuses of actuators. The Web Dashboard is built upon ReactJS and is hosted on Firebase Hosting.

- **Real-Time Monitoring:** The dashboard constantly retrieves information from the Firebase Realtime Database and updates the values shown in real-time. The information is shown in a simple, easy-to-understand manner (e.g., graphs for humidity and temperature, gauges for gas levels).
- **Control of Actuators:** The dashboard provides remote control of the fan and humidifier. According to the real-time data, users can manually change the actuators' settings via the dashboard if they wish to override or take control of the automatic operation.
- **Alerts and Notifications:** The dashboard is also embedded with a notification system (e.g., React Toastify) that sends real-time notifications to users. For example, if there is a flame sensor pick up of fire, the dashboard will display a fire alert and potentially send alerts to the user.
- **User-Friendly Interface:** The dashboard is user-friendly, providing simple access to vital metrics like humidity, temperature, gas levels, etc. This way, warehouse operators are able to keep an eye on the environment without extensive technical knowledge.

Chapter 6

Tools and Systems

6.1 Hardware

- **DHT11 Humidity Sensor:**
  DHT11 sensor is used to measure humidity and temperature levels. It is a digital sensor that provides precise measurements and is appropriate for the measurement of environmental conditions.

- **MQ2 Gas Sensor:**
  This sensor is utilized to detect various gases like LPG, smoke, alcohol, and methane. It's interfaced with Arduino to detect air quality and toxic gas leaks.

- **HX711 with Load Cell:**
  HX711 is an amplifier module, and it is used to interface the load cell with Arduino. It is used in real-time weight measurement.

- **Arduino Uno (End Node 1 and 2):**
  Two Arduino Uno boards are used as end nodes for actuator control and sensor interfacing. They receive the sensor data and send it through Bluetooth.

- **Motor Driver:**
  A 12V DC fan is regulated by a motor driver. It is addressed by the Arduino and regulates power to the motor.

- **DC Fan (12V):**
  It is utilized for ventilation and is used for cooling and is run based on sensor readings such as gas detection.

- **Relay:**
  The relay module is a switch that controls the humidifier. It enables the low-voltage Arduino to power high-voltage devices.

- **Humidifier:**
  Humidifier is regulated by a relay and utilized for the regulation of the humidity in the environment at its best.

- **MQ135 Gas Sensor:**
  It is utilized for the detection of poisonous gases such as CO2, ammonia, and benzene. It's utilized in indoor air quality monitoring in the system.

- **Flame Sensor:**
  Flame sensor is used to detect fire or flame. It is a key part of fire protection and early warning systems.

- **Buzzer:**
  The buzzer is employed as an alarm system. It beeps when it senses critical levels, i.e., gas leakages or fire detection.

- **HM10 Bluetooth BLE Module:**
  The HM10 module provides Bluetooth Low Energy communication. It wirelessly links the Arduino end nodes to the ESP32 edge node.

- **ESP32 (Edge Node):**
  ESP32 is an edge node that serves as a centralized hub, collecting data using Bluetooth and transmitting it to the server using HTTP over TCP/IP.

6.2 Software

- **Server (Render):**
  Online-based, the server receives data from the ESP32. It processes and saves data in a real-time database and can be accessed and analyzed. The server performs parsing data, computation of averages, and JSON response formatting for the frontend interface.

- **Realtime Database (Firebase):**
  Firebase holds sensor data such as humidity, temperature, gas level, weight, flame, and fan status. Firebase is a secure, scalable real-time data monitoring and synchronization solution across devices and platforms.

- **Graphical User Interface (GUI) (React App):**
  A web dashboard based on React presents sensor data in real-time. Temperature, humidity, methane, flame level, fan status, and weight of warehouse conditions are presented by the GUI. Usability is improved through the availability of an easy-to-use interface for administrators and warehouse staff.

Chapter 7

Nodes

The system is organized into three key categories of nodes:

- End Node 1 (Onion Preservation Unit)
- End Node 2 (Warehouse Environment Monitoring Unit)
- ESP32 Edge Node (Central Coordinator)

Each of these nodes has its own unique role and collaborates to ensure the best storage conditions for onions. Let's take a closer look at what each node does and how it operates.

7.1 End Node 1 – Onion Preservation Unit

Platform: Arduino UNO
Communication: Bluetooth (HM-10 Bluetooth Module to ESP32 Edge Node)

Role and Functionality

End Node 1 manages the ideal conditions within the onion preservation unit, which could be a specific area like a room, container, or storage bin where the onions are kept. The sensors and actuators linked to this node work to control temperature, humidity, and detect spoilage—key factors in keeping onions fresh.

Components

1. **DHT11 Sensor (Temperature and Humidity Sensor)**:
    - **Function:** Monitors the temperature and humidity levels inside the onion preservation unit.
    - **Importance:** Keeping the right temperature and humidity is vital to avoid moisture loss or excess moisture, both of which can lead to spoilage. The system responds based on these readings:

- Below 60% humidity: The humidifier kicks in, and the fan runs at half speed to add moisture.
- Between 60%-69% humidity: Both the fan and humidifier are turned off to keep conditions just right.
- Above 70% humidity: The fan ramps up to full speed to prevent too much moisture and ensure good airflow.

2. **MQ2 Gas Sensor (Gas Detection)**:

   - **Function:** Detects gases like methane, carbon dioxide, propane, and other harmful gases that could signal spoilage or dangerous conditions (like fire).
   - **Importance:** This gas sensor plays a crucial role in spotting spoilage or the onset of anaerobic conditions that could be detrimental to the onions.

3. **HX711 Load Cell and Weight Measurement**:

   - **Function:** This device measures the weight of onions stored in the unit.
   - **Importance:** Keeping an eye on the weight is crucial for tracking any losses due to spoilage or poor storage conditions. If you notice a sudden drop in weight, it could signal issues with the quality or safety of the onions, like them drying out.

4. **12V DC Fan**:

   - **Function:** This fan serves as a ventilation system for the storage unit.
   - **Importance:** The fan adjusts its speed based on humidity and temperature levels. It plays a key role in controlling moisture and ensuring proper airflow. Here's how it operates:
     - Half speed (when humidity ¡ 60%): Provides gentle airflow while adding some moisture.
     - Full speed (when humidity ¿ 70%): Acts as an exhaust to eliminate excess moisture and keep the air circulating.

5. **Humidifier**:

   - **Function:** This device introduces moisture into the air inside the storage unit.
   - **Importance:** When humidity levels dip below 60%, the humidifier kicks in to boost moisture levels, helping to create the best conditions for preserving onions.

Communication

End Node 1 connects wirelessly to the ESP32 Edge Node through the HM-10 Bluetooth module. The sensors gather data and send it to the ESP32 at regular intervals, which then processes the information and uploads it to the cloud.

7.2 End Node 2 – Warehouse Environment Monitoring Unit

Platform: Arduino UNO
Communication: Bluetooth (HM-10 Bluetooth Module to ESP32 Edge Node)

Role and Functionality

End Node 2 is responsible for monitoring the overall warehouse environment where the onions are stored. While it doesn't directly manage the preservation unit, it plays a crucial role in monitoring external conditions like temperature, air quality, and potential fire hazards.

Components

1. **DHT11 Sensor (Temperature and Humidity Sensor)**:
    - **Function:** This sensor measures the temperature and humidity levels in the warehouse.
    - **Importance:** Keeping tabs on these factors is vital for ensuring the onions stay fresh and well-preserved.
2. **MQ135 Air Quality Sensor**:
    - **Function:** Detects indoor air pollutants such as ammonia, carbon dioxide, benzene, and smoke.
    - **Importance:** If the air quality dips, it can speed up the spoilage of onions. The system can alert users if harmful gas levels rise, helping to prevent spoilage or other hazardous situations.
3. **Flame Sensor**:
    - **Function:** This sensor is designed to spot flames or fire hazards.
    - **Importance:** Detecting fire is critical for safety. A fire in the warehouse could not only ruin the onion stock but also create serious safety risks for everyone. When it senses a fire, it can trigger an alarm or send a notification to alert the user.

Communication

End Node 2 communicates with the ESP32 Edge Node through the HM-10 Bluetooth module, sending all that important sensor data to the ESP32 for processing and cloud transmission.

7.3 ESP32 Edge Node – Central Coordinator and Cloud Interface

Platform: ESP32 Wi-Fi Development Board
Communication:

- Bluetooth Low Energy (BLE) for connecting with both End Nodes.
- Wi-Fi for cloud communication (using HTTP with JSON format).

Role and Functionality

The ESP32 Edge Node serves as the main hub that facilitates communication between the two Arduino-based End Nodes and the cloud system. Its primary job is to gather sensor data from both End Node 1 and End Node 2, then send it off to the cloud for storage, analysis, and real-time monitoring.

Components and Functions

1. **Bluetooth Low Energy (BLE) Communication:**
    - The ESP32 wirelessly connects to both Arduino End Nodes using Bluetooth, specifically through HM-10 Bluetooth modules. This setup offers low-power, short-range communication, making it perfect for indoor environments like warehouses.
2. **Wi-Fi Communication:**
    - The ESP32 links up to a Wi-Fi network to send sensor data over the internet. This information is transmitted in JSON format via HTTP to a cloud server hosted on Render, which then updates a Firebase Realtime Database.
3. **Data Parsing and Uploading:**
    - After the ESP32 collects data from the End Nodes, it parses that information and uploads it to the cloud for further processing. This allows for real-time monitoring and alerts for the onion storage system.
4. **Cloud Integration:**
    - The data sent by the ESP32 updates the Firebase Realtime Database, which can be accessed through a web dashboard for live monitoring of warehouse conditions (like temperature, humidity, weight, gas levels, etc.).
    - The system employs ReactJS to build an interactive web dashboard that enables users to monitor data in real-time and remotely control systems such as the fan and humidifier.

7.4 Cloud and Web Layer (Firebase and ReactJS)

The cloud layer and web interface serve as the user-facing component of the system, enabling operators to keep an eye on and manage the onion preservation process from afar.

Firebase Realtime Database

- The ESP32 Edge Node sends sensor data straight to Firebase, where it gets stored in real-time.
- This data is constantly updated and can be accessed through the ReactJS Web Dashboard.

ReactJS Dashboard

- The ReactJS application fetches data from the Firebase database and displays it in real-time.
- Users can monitor current readings for temperature, humidity, gas levels, and flame status.
- It also features notifications (using react-toastify) to alert users about potential issues, such as fire hazards or unusual temperature fluctuations.
- The dashboard allows for remote control of the fan and humidifier, giving users the ability to manually adjust the warehouse conditions if needed.

Chapter 8

Networks

8.1 Bluetooth Low Energy (BLE) Network

In this setup, Bluetooth Low Energy (BLE) is utilized to communicate between the End Nodes (Arduino UNO-based nodes) and the ESP32 Edge Node. BLE is a low-range and low-power wireless communication protocol, making it perfect since the End Nodes are nearby the ESP32 in the warehouse setup.

How BLE Works

- Both End Node 1 (onion preservation unit) and End Node 2 (environment monitoring unit) are fitted with HM-10 Bluetooth modules.
- The nodes transmit sensor data like temperature, humidity, gas concentrations, and flame detection signals to the ESP32 Edge Node.
- The ESP32 is the central BLE coordinator, collecting data from the End Nodes and passing it on to the cloud for storage and visualization.

Why BLE is Used

- **Low Power Consumption**: BLE reduces the energy consumption of the End Nodes, assisting in keeping the system efficient.
- **Short-range Communication**: BLE provides communication in proximity, perfectly suitable for devices within the same warehouse.

8.2 Wi-Fi Network

The ESP32 Edge Node has Wi-Fi functionality, which is utilized to link the system to the cloud server for uploading data and real-time monitoring. Remote access to sensor data, actuator control, and status information is made available through a web interface.

How Wi-Fi Works

- The ESP32 is connected to the Wi-Fi network and serves as a bridge between the End Nodes (BLE) and the cloud server.
- The ESP32 transfers aggregated sensor information from the End Nodes to the Render cloud server using HTTP requests.
- The cloud server refreshes the Firebase Realtime Database with the current data.

Why Wi-Fi is Used

- **High-speed Data Transfer**: Wi-Fi provides quick data transfer, which is required for real-time updates and cloud communication.
- **Long-range Connectivity**: Wi-Fi facilitates long-range communication, providing internet connectivity for cloud-based interactions.
- **Cloud Integration**: Wi-Fi provides effortless integration with cloud solutions for remote control and monitoring.

8.3 Cloud Network (Render, Firebase, and ReactJS Dashboard)

The cloud network is organized around the Render server, Firebase, and a ReactJS web dashboard. Sensor data is saved in the cloud infrastructure, while data visualization and remote control are supported.

How Cloud Network Works

- **Render Server**: Hosts the backend server, enabling communication between the ESP32 Edge Node and the Firebase Realtime Database.
- **Firebase Realtime Database**: Stores sensor data (temperature, humidity, gas levels, etc.) in real-time, keeping data up-to-date and available for remote monitoring.
- **ReactJS Web Dashboard**: Showcases real-time data from the Firebase Realtime Database. Users are able to see the status of the warehouse environment and interact with the system, such as controlling the fan and humidifier.

Why Cloud Network is Used

- **Real-time Data Updates**: Firebase provides real-time data updates, offering users the latest information on environmental conditions.

- **Remote Monitoring and Control**: An interface to monitor and control the system remotely is made available in the cloud-based system.
- **Data Storage**: Cloud services provide secure, expandable storage for environmental data to store and analyze over the long term.

8.4 Communication Protocols Used

Bluetooth Low Energy (BLE)

- Utilized for End Node-to-ESP32 Edge Node communication.
- For the transmission of sensor data such as temperature, humidity, gas concentration, and flame detection.

HTTP (Hypertext Transfer Protocol)

- The ESP32 Edge Node employs HTTP for posting sensor values to the Render cloud server.
- Data is posted in JSON format, which is web service compliant.

JSON (JavaScript Object Notation)

- The data transferred from the ESP32 to the cloud server is in JSON format.
- JSON is lightweight, human-readable, and easy to process, making it perfect for web-based applications such as the ReactJS dashboard.

Firebase Realtime Database

- Firebase allows real-time data storage and access. It employs WebSockets to establish an open connection between the cloud database and the ReactJS web dashboard.
- The ReactJS dashboard fetches data from Firebase to show sensor readings and manipulate the fan and humidifier.

8.5 Communication Flow

- **End Nodes**:
  - Read sensor data (temperature, humidity, gas levels, flame detection).
  - Transmit this data through BLE to the ESP32 Edge Node.
- **ESP32 Edge Node**:

  - Serves as the BLE central coordinator, gathering data from the End Nodes.
  - Transmits data to the cloud server through HTTP.
  - Transmits sensor data in JSON format to Firebase Realtime Database.

- **Firebase Realtime Database**:
  - Serves as storage for data acquired from the ESP32 Edge Node.
  - Offers real-time data access for the ReactJS web dashboard.

- **ReactJS Web Dashboard**:
  - Synchronizes with the Firebase Realtime Database to read and show real-time data (temperature, humidity, gas levels, flame status).
  - Enables users to remotely control the fan and humidifier.

Chapter 9

Results and Analysis

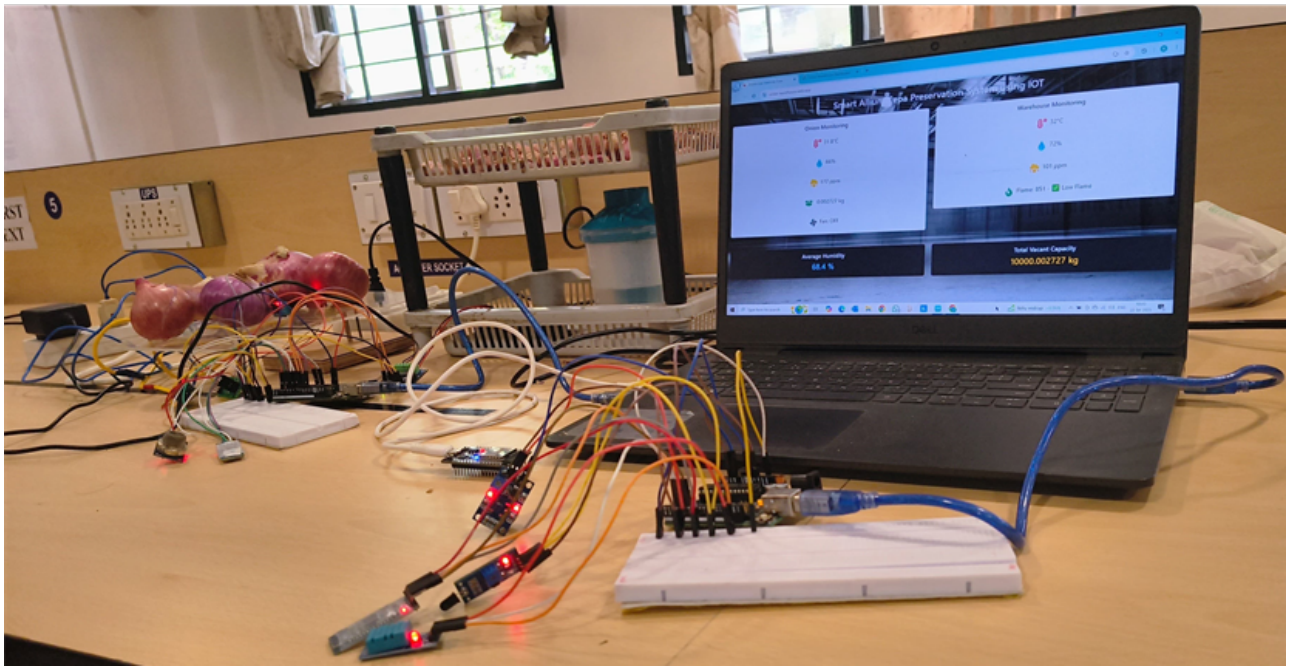


Figure 9.1: Smart Allium Cepa Preservation System using IOT (Entire Setup)

This project aims to create a smart onion storage and preservation system using embedded systems and IoT technologies. It's designed to track environmental factors such as temperature, humidity, and gas levels in real-time, keeping the onions in the best possible conditions. With the integration of sensors, microcontrollers, and a cloud-based dashboard, this system not only preserves the onions but also reduces spoilage and economic losses. It also provides the luxury of remote monitoring, making it easy for anyone operating the storage facility.

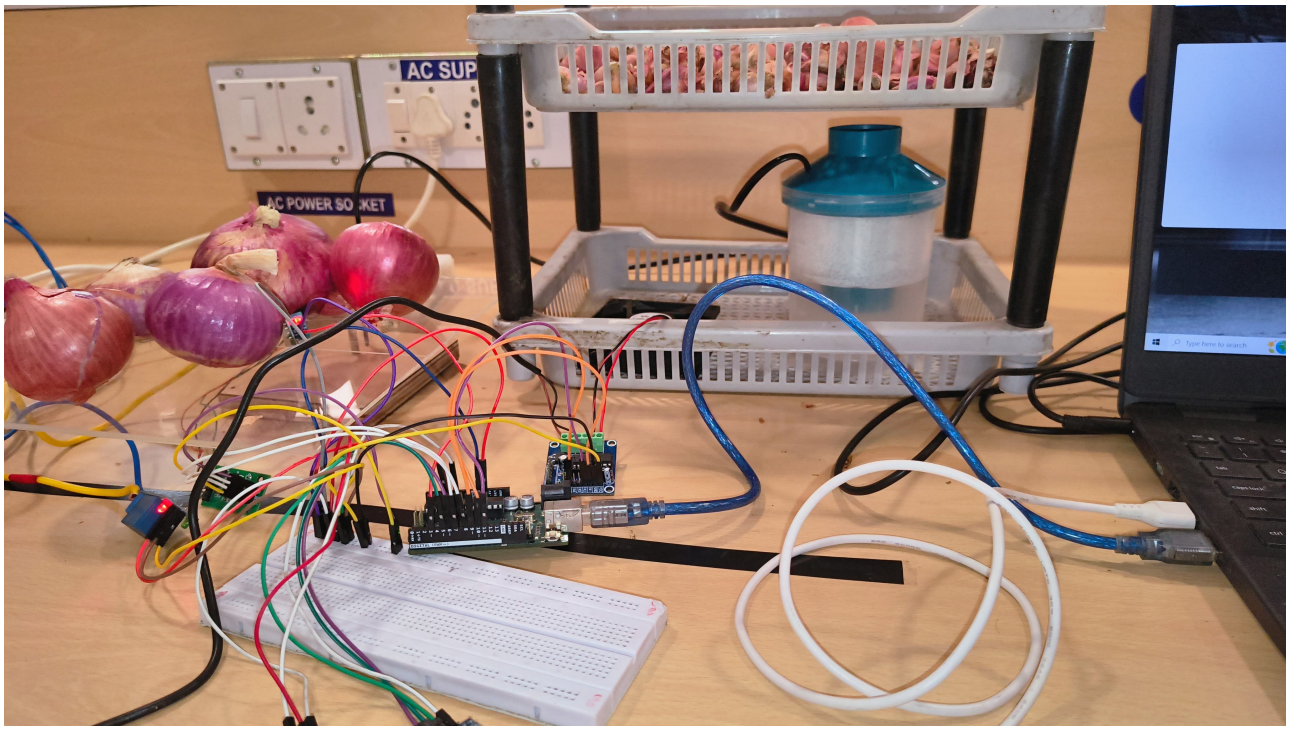


Figure 9.2: End Node 1

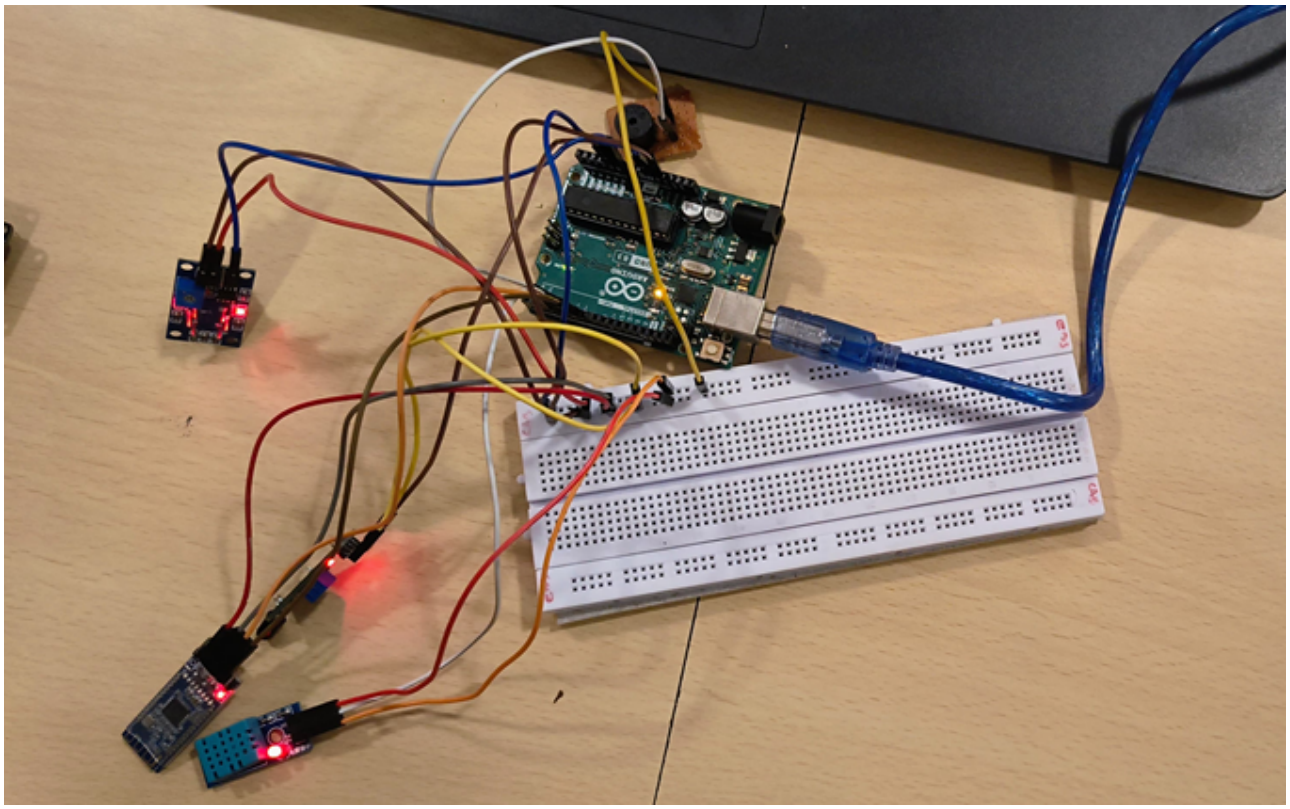


Figure 9.3: End Node 2

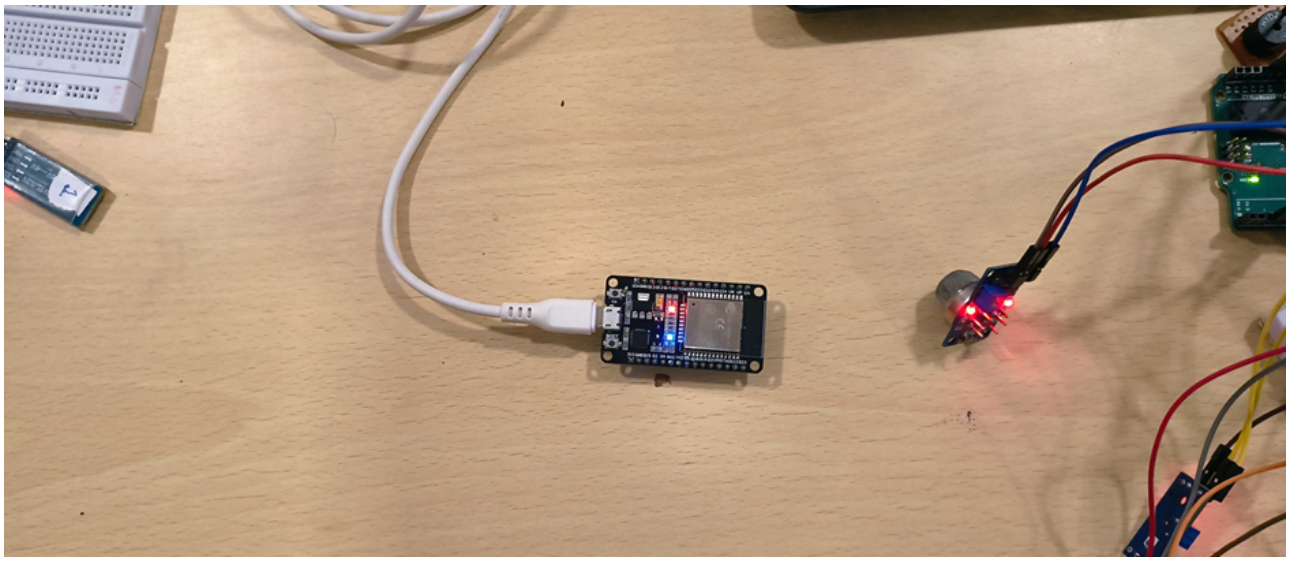


Figure 9.4: Edge Node

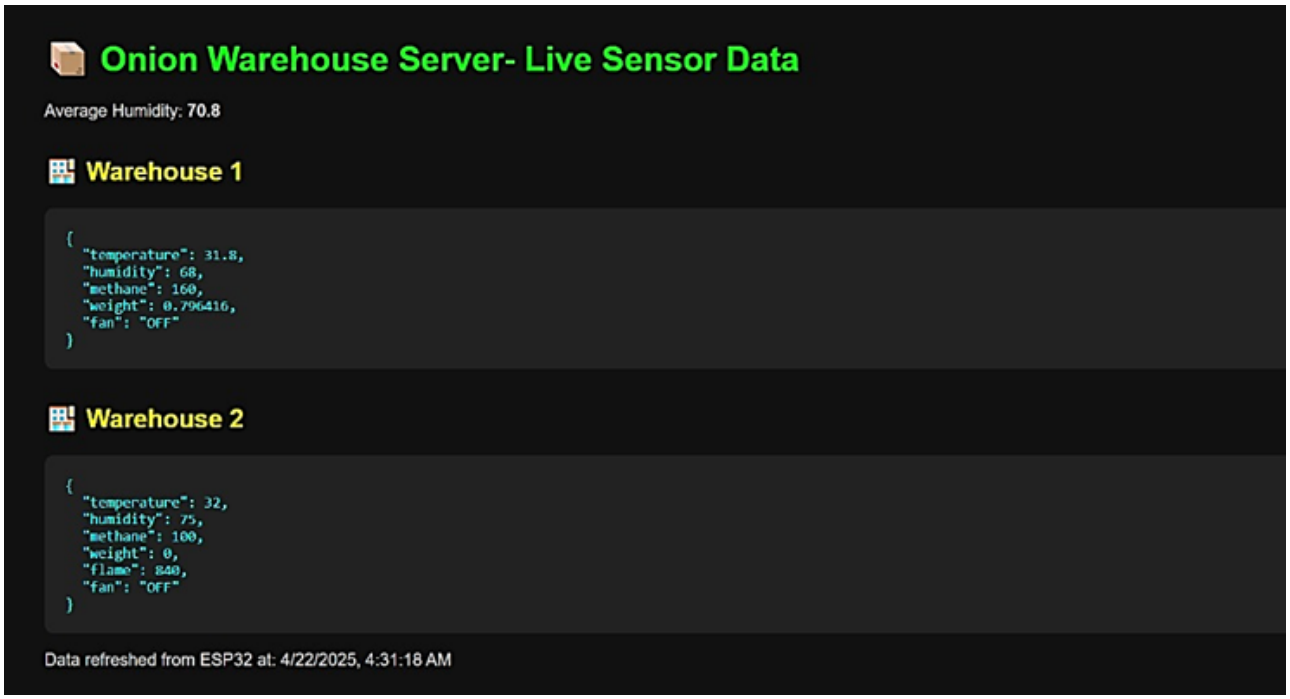


Figure 9.5: Server

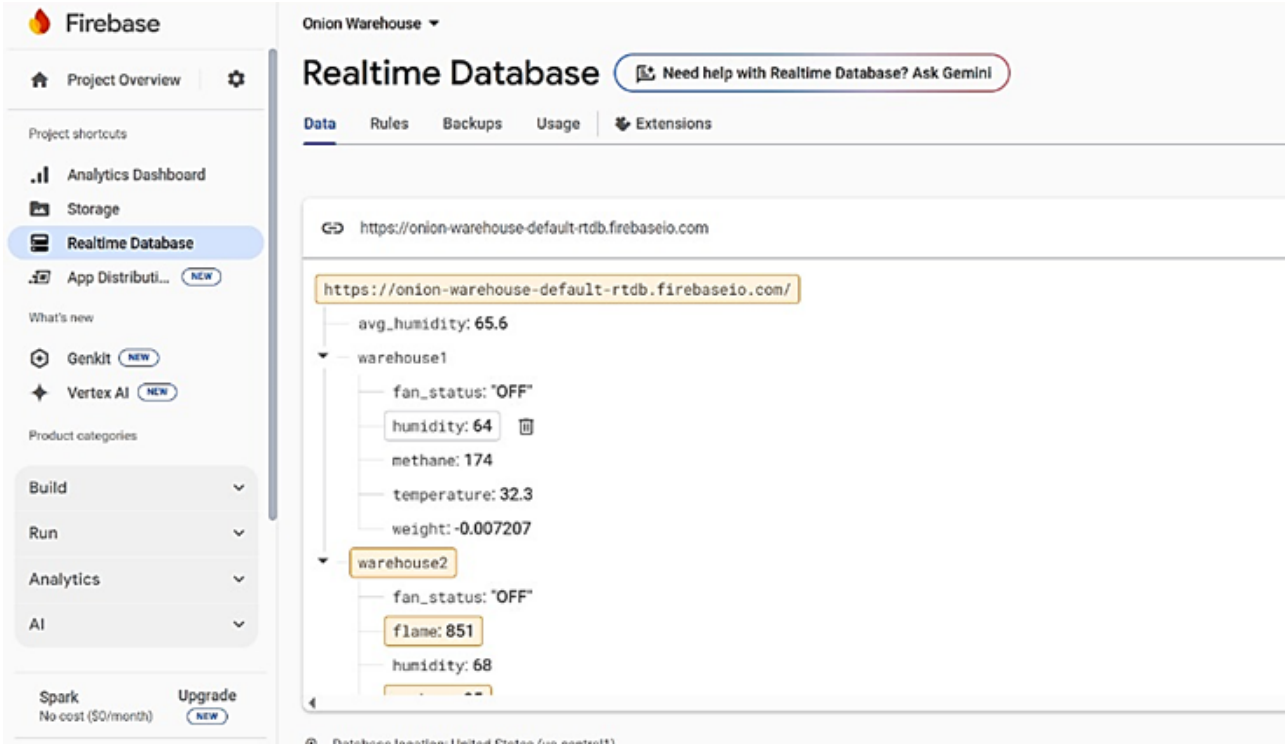


Figure 9.6: Realtime Database

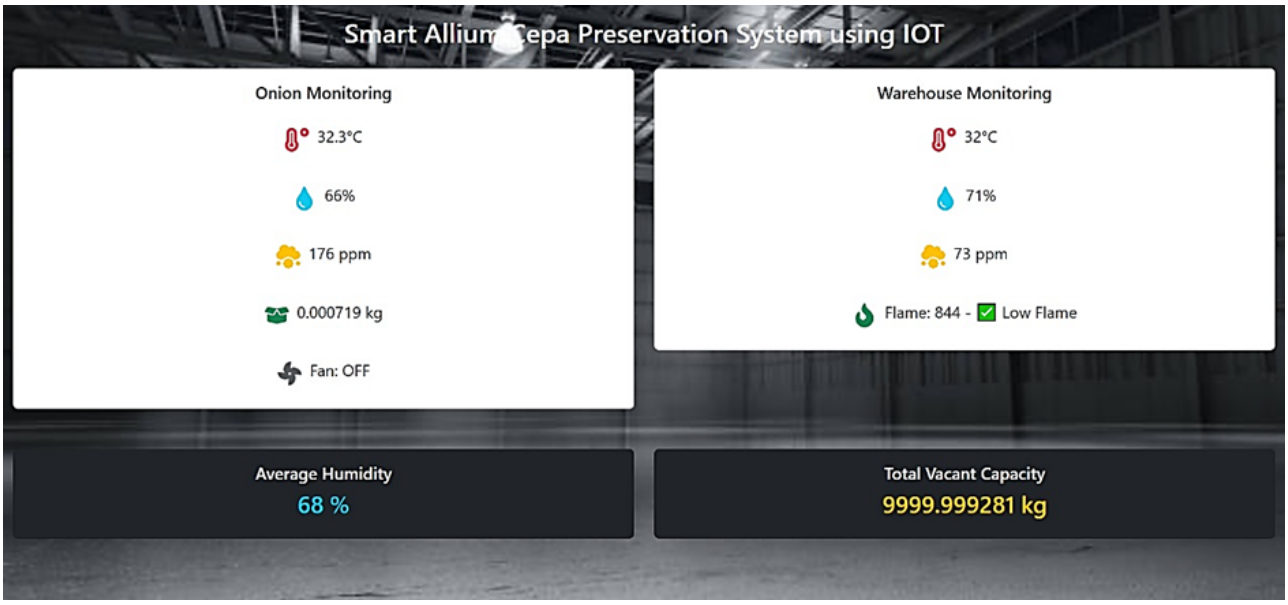


Figure 9.7: Graphical User Interface

Chapter 10

Conclusion and Scope for further Research

10.1 Conclusion

This **Smart Allium Cepa Preservation System using IOT** project successfully combines embedded systems, Internet of Things (IoT), and cloud computing to address the challenges of onion storage.By using sensors such as DHT11 for temperature and humidity monitoring, MQ2 and MQ135 for monitoring gas concentration, and HX711 for weight monitoring of the onions, the system ensures optimal environmental conditions to effectively store the onions. Furthermore, the use of actuators such as fans and humidifiers allows for automated control based on real-time feedback from sensors, thus ensuring the storage environment supports the quality preservation of the onions and reducing spoilage.

One of the system's strongest features is its real-time monitoring, which is accomplished through the cloud-based dashboard. The dashboard offers a user-friendly interface through which remote monitoring of the conditions in the warehouse and manual actuator operation when necessary can be done. The system also includes automatic alarms for dangerous situations like fire detection or high levels of gases, enabling warehouse operators to act quickly in the event of potential problems.

The system's architecture, based on **Edge Nodes** for processing and **Firebase Realtime Database** for data storage and synchronization, ensures high reliability and minimal latency in communication. The use of **Bluetooth Low Energy (BLE)** for local communication and **HTTP over TCP/IP** for cloud communication enables efficient data transfer while maintaining low power consumption. The cloud infrastructure further supports scalability, allowing for easy expansion of the system to monitor larger storage facilities or additional storage units.

In Conclusion, the system not only enhances the storage capacity of onions but also aids in waste minimization, a serious issue in the agricultural sector. It is a rock-solid solution for storage facilities that want to automate and streamline their operations, thus enhancing both operational and economic efficiency.

10.2 Future Scope

While the **Smart Allium Cepa Preservation System using IOT** has shown significant promise in terms of functionality, scalability, and reliability, there are several avenues for future development and enhancement:

10.2.1 Integration of Advanced Sensors

- **Improved Sensors**: Future versions of the system could incorporate more accurate and reliable sensors, such as advanced **temperature and humidity sensors** with lower error margins or **gas sensors** with a wider range of gas detection capabilities.
- **Smart Cameras or Imaging Systems**: Applying machine vision or smart cameras to observe the physical condition of onions (such as mold or other forms of spoilage) can enhance the value of the preservation process by allowing the system to respond to what it observes.

10.2.2 Machine Learning for Predictive Analytics

- **Predictive Modeling**: By using machine learning algorithms, the system can identify patterns in historical data and forecast future environmental changes. This proactive approach allows for adjustments to storage conditions before any issues come up, leading to better management of the storage environment.
- **Anomaly Detection**: Machine learning models can be used to spot unusual patterns in sensor data that might signal equipment failure or a potential environmental threat, like a gas leak, before it becomes a serious issue.

10.2.3 Energy Efficiency Improvements

- **Power Optimization**: While the current system relies on Bluetooth Low Energy (BLE) to keep power usage low, there's definitely room for improvement. By incorporating energy harvesting technologies, like solar power, we could actually power the sensors and actuators directly. This would lessen our reliance on external power sources and boost the overall sustainability of the system.
- **Advanced Actuator Control**: Finding smarter ways to optimize how actuators use power by implementing advanced control algorithms could significantly cut down on energy consumption, all while maintaining the best storage conditions.

10.2.4 Integration with Other IoT Platforms

- **Cross-Platform Integration**: The system could be integrated with other IoT platforms such as **ThingSpeak**, **Ubidots**, or **Microsoft Azure IoT Hub**, which would

allow the data to be accessed and processed in different cloud environments. This would provide enhanced data visualization, additional analytics capabilities, and possibly even integration with other automation systems.

- **Data Sharing and Analysis**: Sharing data with other storage facilities can really enhance our ability to analyze information across various warehouses. This collaboration helps us spot larger trends and ultimately leads to better management of onion storage on an industry-wide scale.

10.2.5 Enhanced User Interface and User Experience

- **Mobile Application**: Alongside the web dashboard, we could create a mobile app to enhance accessibility and make things more convenient. This app would enable warehouse operators to keep an eye on conditions and manage actuators right from their mobile devices.

- **User Customization**: In the future, the system might let users customize alerts, set their own thresholds, and tweak control algorithms right from the dashboard. This would provide a lot more flexibility in how the system can be utilized across various storage facilities.

10.2.6 Automation and Expansion

- **Automated Decision Making**: With some additional development, the system has the potential to achieve a greater level of autonomy by integrating automated decision-making processes. This would lessen the need for human involvement, further enhancing the storage environment and helping to minimize spoilage.

- **Scalability**: The system can be expanded to keep an eye on several storage units at once, making it perfect for large warehouses. This might mean adding more End Nodes and Edge Nodes, all linked to the main cloud server.

10.2.7 Blockchain for Data Integrity and Security

- **Blockchain Integration**: As the system gathers sensitive information—like environmental conditions and fire detection—it's vital to ensure that this data remains secure and intact. In future iterations, the system might leverage blockchain technology to establish a tamper-proof ledger for sensor data. This would not only guarantee its authenticity but also provide a clear and transparent record of every action taken within the storage facility.

In summary, it is clear that the system being developed has come a long way from the conventional onion storage practices. However, there are several areas of improvement that

can make it even more efficient, scalable, and overall impactful to the agricultural sector. By embracing new technologies and enhancing the aspects of the system, the project has the potential to become a comprehensive solution towards wiser and more efficient food storage and preservation.

References

[1] V. Karthik, P. L. Girishankar, A. Jerry Samraj and S. Dhanush, "Smart Energy Management System for Onion Preservation," 2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS), Pudukkottai, India, 2023.

[2] S. S. Bachal, S. M. Kolekar and R. P. More, "Smart System for Protecting Onion from Different Attack," 2018 International Conference on Information, Communication, Engineering and Technology (ICICET), Pune, India, 2018.

[3] "Onion Storage Monitoring System Based on IoT and Data Analytics," by Y. Xie, Y. Zhang, X. Fang, and Y. Xie, published in the IEEE Transactions on Instrumentation and Measurement in 2019.

[4] "IoT-Based Onion Storage Monitoring System with Data Analytics," by R. G. Singh, S. K. Patel, and A. K. Singh, published in the IEEE International Conference on Computing, Communication and Automation (ICCCA) in 2019.

[5] "Development of an IoT-Based Smart Onion Storage System," by B. K. Samanta ray and D. K. Parhi, published in the IEEE International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)in 2020.

[6] M. Mythili and P. V. Kumari, "Internet of Things enabled Onion Growth Monitoring System using Cloud," 2021 2nd International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 2021.

[7] R. Achary, R. R, R. K and P. V, "Effect of Temperature and Relative Humidity on Onion farms and its Monitoring by using IoT Based Smart Farming System," 2022 International Conference on Communication, Computing and Internet of Things (IC3IoT), Chennai, India, 2022.

[8] Firouzi S, Khorshidi A, Soltani-Nabipour J, Barzi SM, Amani M, Ay MR. Evaluation of gamma and electron radiations impact on vitamins for onion preservation. Applied Radiation and Isotopes. 2021 Jan 1; 167:109442.

[9] Fayos O, Ech´avarri B, Vall´es MP, Mallor C, Garc´es-Claver A, Castillo AM. A simple and efficient method for onion pollen preservation: Germination, dehydration, storage conditions, and seed production. Scientia Horticulturae. 2022 Nov 17; 305:111358.