Amrita School of Engineering, Coimbatore

Amritanagar – P. O., Ettimadai, Coimbatore – 641112

Department of Electrical and Electronics Engineering

21ES614 – Internet of Things
Lab Manual

M.Tech Embedded Systems, Second Semester

Name: Cheppalli Naga Sai Varun Kumar Reddy

Roll no: CB.EN. P2EBS24006

Regulation: 2021

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:

Expt. No:	Experiment	Page Number
1	Familiarization of various communication networks in NetSim and Wireshark	
2	IoT end nodes with Ubidots, Adafruit, ThingSpeak	
3	Simulation study on IEEE 802.3/802.11 networks using NetSim	
4	Simulation study on ZigBee/Wireless Sensor Networks using NetSim	
5	IoT networks simulation in NetSim & Wireshark packet data extraction	
6	Familiarization of socket connection using microcontroller board and PC/Laptop	
7	IoT edge node – Data aggregation and communication	
8	IoT edge node – Edge computing and communication	
9	Demonstration of IoT edge device – aggregation, edge computing & communication	
10	Implementation of UI for data visualization & remote control	
11	Implementation of database for edge/end node data storage	
12	Implementation of a server with database and UI	
13	Project Implementation	

Experiment No: 1

Date:12-12-2024

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].

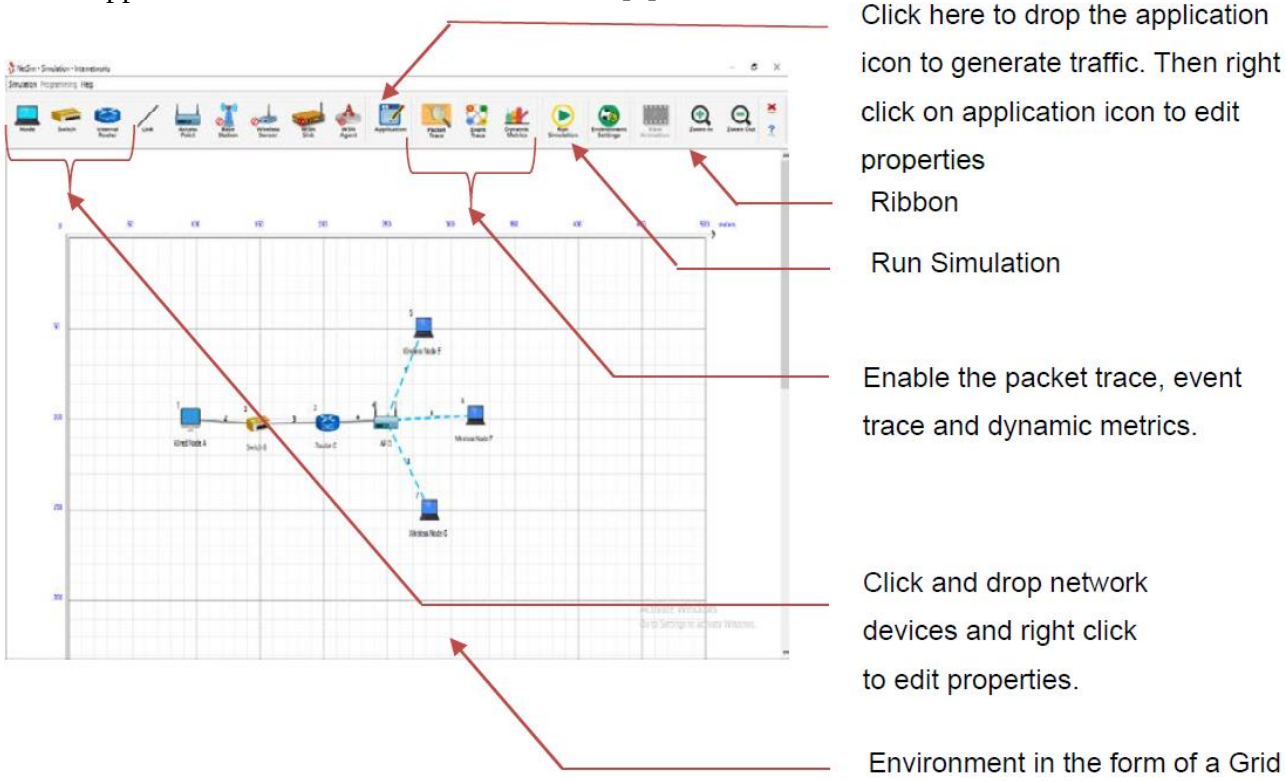


Fig 1: Network Design Window – NetSim [1]

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].

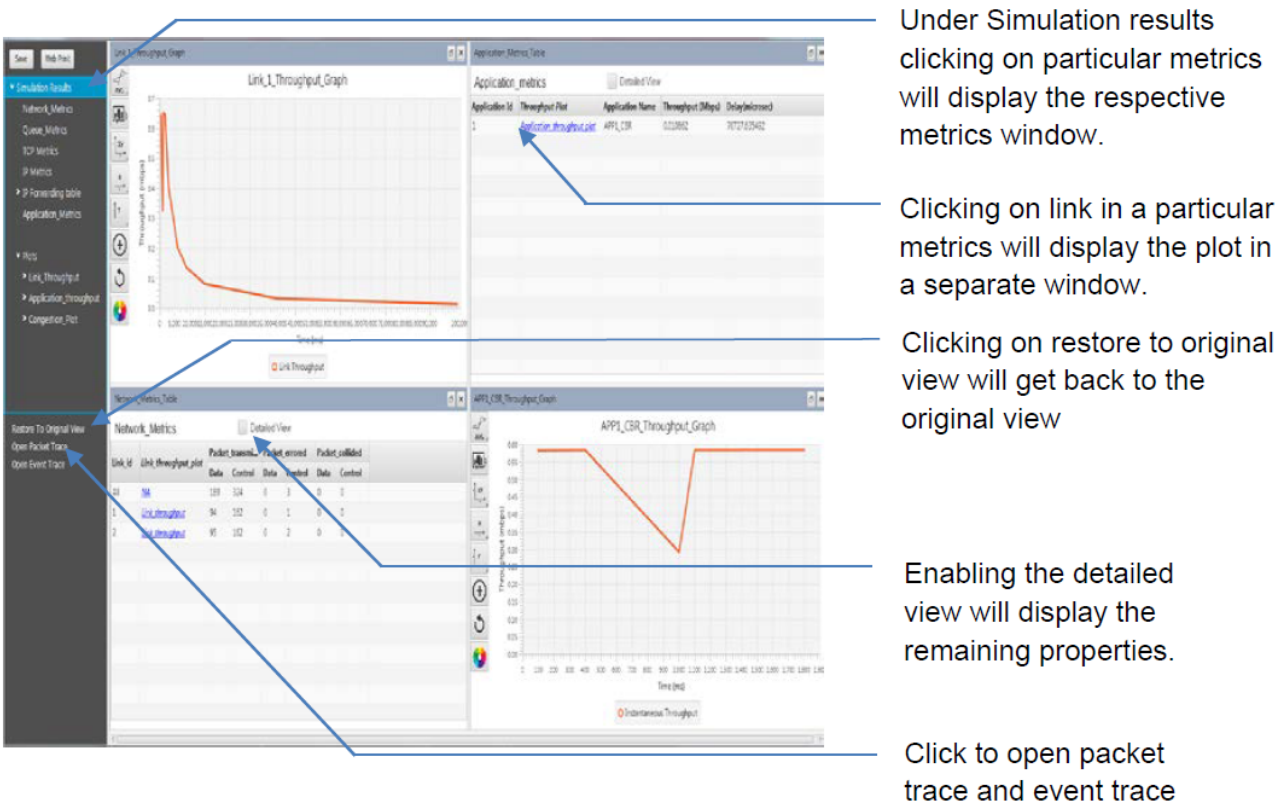


Fig 2: Results Window – NetSim [1]

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].

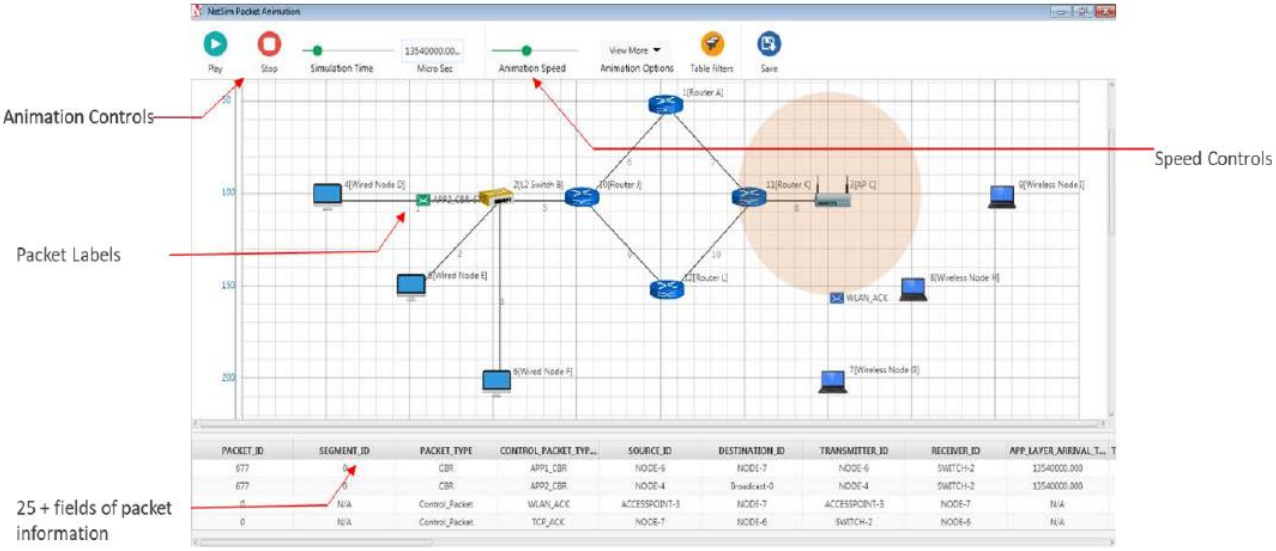


Fig 2: Packet Animation Window – NetSim [1]

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:30-12-2024**

Development of IoT end nodes with Ubidots/Adafruit/ThingSpeak

AIM

To develop IoT end nodes based on <microcontroller platform> and demonstrate data transfer to the < Ubidots/Adafruit/ThingSpeak> platform.

TOOLS & SYSTEMS:

Software Components:

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:

Demonstrate integration of:

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)

Code Implementation:

This code has already been structured very well. For a lab submission, keep these sections:

- WiFi credentials
- Sensor reads and prints
- Simple logic to read motor command from ThingSpeak
- Basic publishing to Ubidots and Adafruit IO

If you want to trim down the output for lab, you can:

1. Keep only essential debug prints
2. Reduce retry attempts (WiFi, MQTT) to simplify behavior
3. Add comments indicating what each section is for

Suggested Lab Demo Steps:

1. Power the ESP32 via USB.
2. Verify Serial Monitor at 115200 baud rate.
3. Ensure Wi-Fi is connected.
4. Open ThingSpeak and change the field value (1, 2, or 0).
    - 1 → Motor Forward
    - 2 → Motor Reverse
    - 0 → Motor Stop
5. Observe Serial Monitor:
    - Sensor readings printed.
    - Data sent to Ubidots and Adafruit.
    - Motor state updated.
6. Show all 3 platform dashboards for real-time updates.

RESULTS:

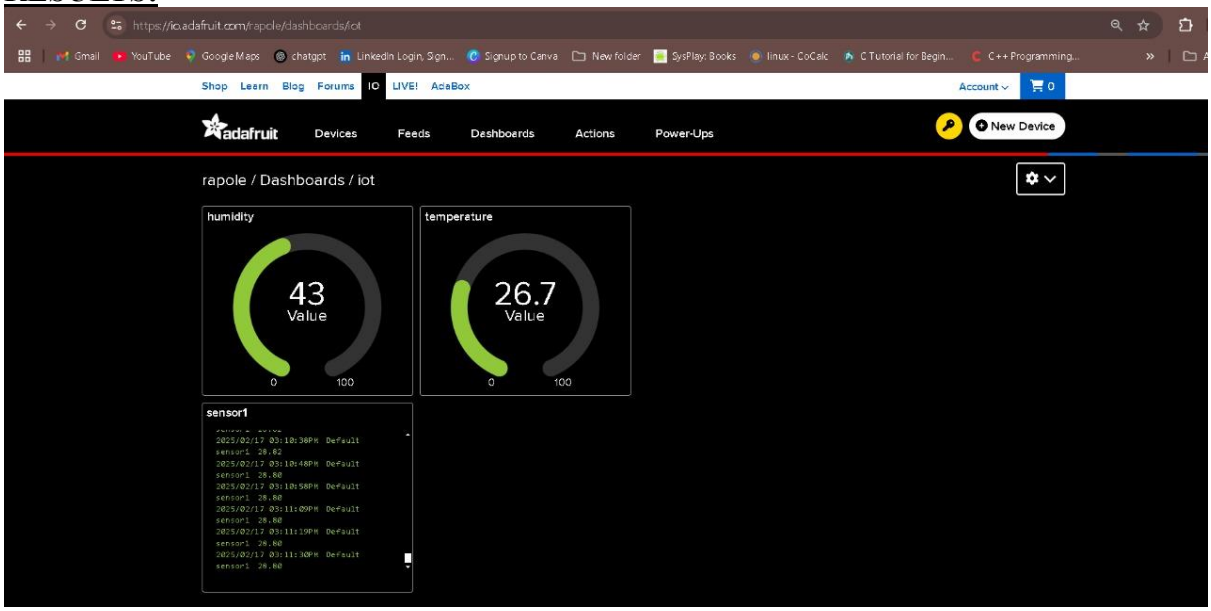


Fig: Shows the result of Adafruit Platform

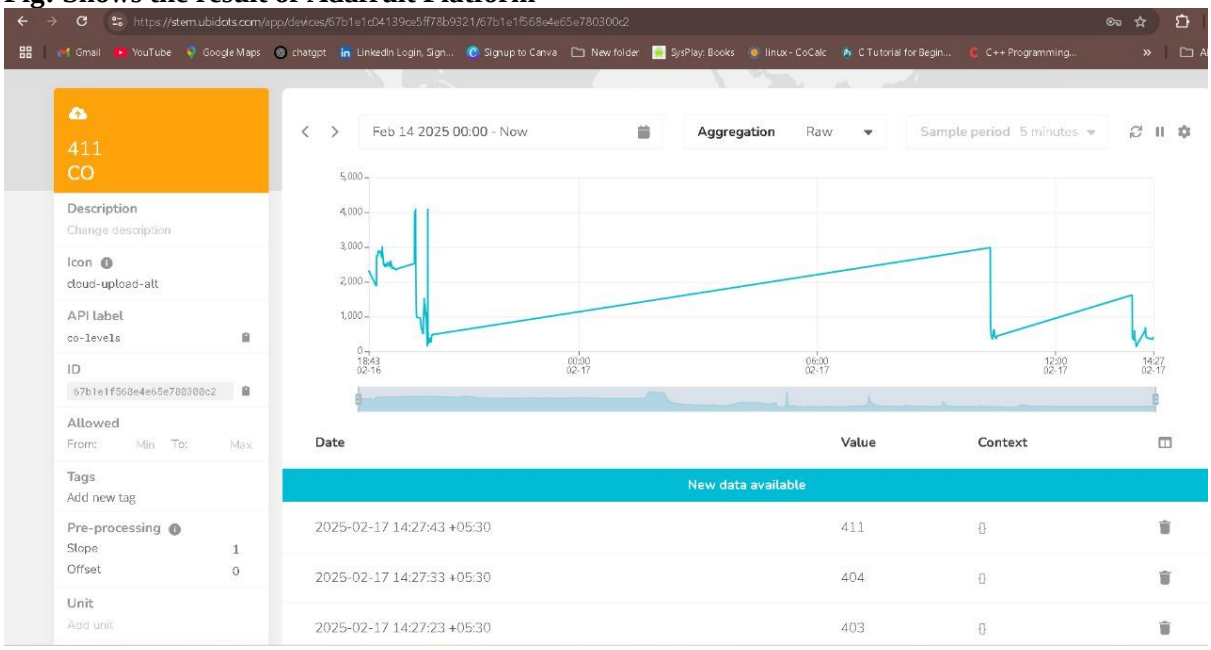


Fig: Shows the result of ubidots

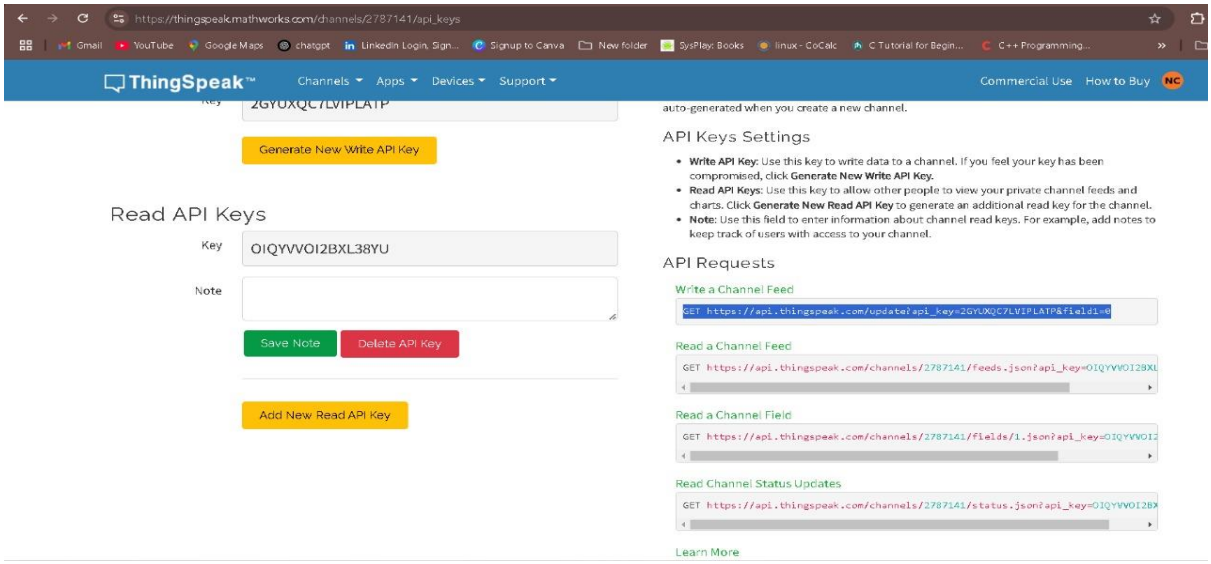
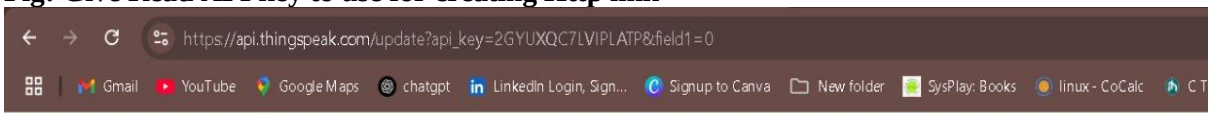


Fig: Give Read API key to use for creating Http link



127

Fig: shows the hyper linked used

Inference:

Successfully Demonstrated End device using IOT platforms like Thingspeak, Adafruit and Ubidots

Experiment No: 3 **Date:6-1-2025**

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 [1].

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 [2].

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 [3].

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:

Observed Topology Overview

- Router (Router_5) connected centrally
- Switches:
    - L3Switch_4 (left side)
    - L2Switch_7 (right side)
- Nodes:
    - Left side (under L3Switch_4):
        - Wired_Node_1 (IP: 111.1.1)
        - Wired_Node_2_CBR (IP: 111.3.3)
        - Wired_Node_3 (IP: 111.1.4)
    - Right side (under L2Switch_7):
        - Wired_Node_6 (IP: 112.3.2)
        - Wired_Node_7 (IP: 112.1.3)
        - Wired_Node_8 (IP: 112.1.4)
- Application Flow: CBR (Constant Bit Rate) from Wired_Node_2_CBR to Wired_Node_6 via Router_5

NetSim Implementation

Step 1: Create the Nodes

1. Add one Router
2. Add one L3 Switch (Switch_4) and one L2 Switch (Switch_7)
3. Add six Wired Nodes
    - Assign appropriate IP addresses as per the image

Step 2: Connect Nodes

- Connect:
    - Wired_Node_1, Wired_Node_2_CBR, Wired_Node_3 → Switch_4 (L3)
    - Wired_Node_6, 7, 8 → Switch_7 (L2)
    - Both Switches → Router_5

Use wired Ethernet links between nodes and switches.

Step 3: Configure IP Addresses

Assign IPs as visible:

- Wired_Node_2_CBR → 111.3.3
- Wired_Node_6 → 112.3.2
- Router_5 → 111.1.1 (left interface), 112.1.1 (right interface)

Assign other nodes unique IPs in their respective subnets (already labeled).

Step 4: Create Application Layer Traffic

1. Go to Application tab in NetSim
2. Create:
    - Source: Wired_Node_2_CBR
    - Destination: Wired_Node_6
    - Application Type: CBR (UDP)
    - Configure parameters:
        - Packet size (e.g., 512 bytes)
        - Inter-packet interval (e.g., 1 ms)
        - Duration or number of packets

Step 5: Routing Configuration

- Enable static or dynamic routing on Router_5 to ensure packets from 111.3.3 reach 112.3.2.

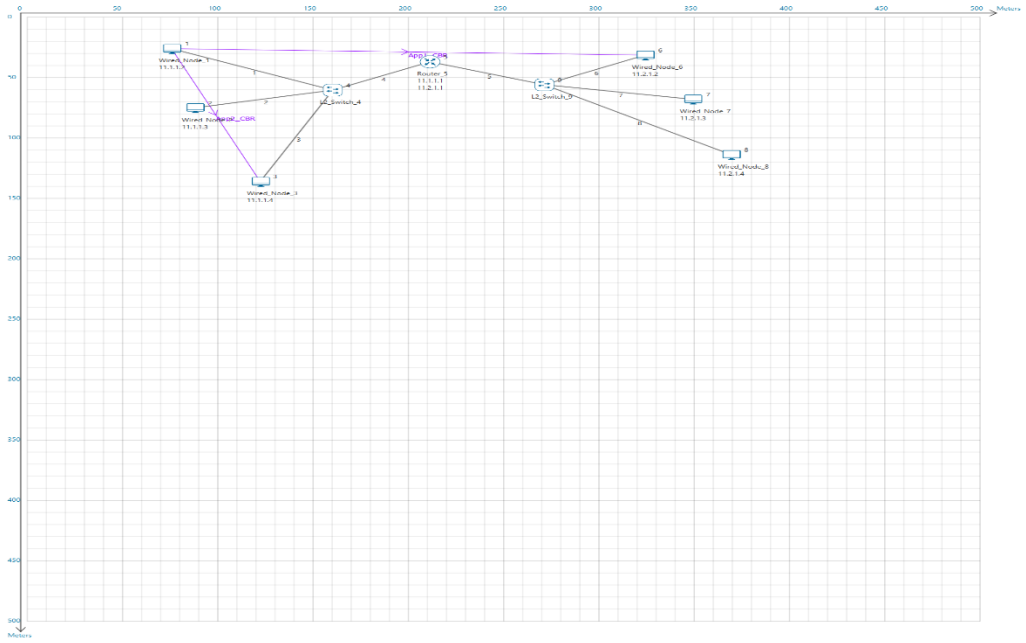
Step 6: Run Simulation

- Set simulation time (e.g., 10 seconds)
- Start the simulation
- Analyze:
    - Throughput
    - Packet delivery ratio
    - Delay
    - Jitter

INFERENCE;

1. The simulation models a wired network where a CBR application sends data from Wired_Node_2_CBR to Wired_Node_6 through a router.
2. Layer 3 and Layer 2 switches segment the network into two logical parts for efficient data routing and switching.
3. The central Router_5 handles inter-network communication between two subnets: 111.x.x.x and 112.x.x.x.
4. Traffic flow is one-directional from source to destination with minimal hops, optimizing latency.
5. This setup effectively demonstrates basic LAN-WAN integration and routing behavior in a mixed-switch environment.

RESULTS:



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

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 Network Implementation Steps:

1. Network Topology Setup

- Coordinator: Place 1 PAN coordinator node (e.g., BAN_Coordinator_4) as the central controller.
- End Devices: Place 7 Zigbee End Devices (e.g., Zigbee_1 to Zigbee_7) around the coordinator.
- Application Nodes: Add App2_CBR and App2_DST for traffic generation and reception.

2. Configure Node Parameters

- Zigbee Coordinator:
    - PAN ID: 0x1A2B
    - Channel: 15
    - Beacon-enabled: No (optional)
- Zigbee End Devices:
    - Associated with coordinator PAN ID
    - Routing mode: End Device
    - Power mode: Sleep (optional for energy-saving simulations)

3. Application Configuration

- CBR (Constant Bit Rate) Traffic:
    - Source: App2_CBR
    - Destination: App2_DST
    - Packet size: 512 bytes
    - Interval: 0.2 sec
    - Start time: 10s
    - Stop time: End of simulation

4. Protocol and MAC Layer Configuration

- Zigbee MAC/PHY:
    - IEEE 802.15.4 standard
    - Transmission power: 0 dBm (default)
    - Data Rate: 250 kbps
- Routing: Enable AODV or Zigbee Tree routing

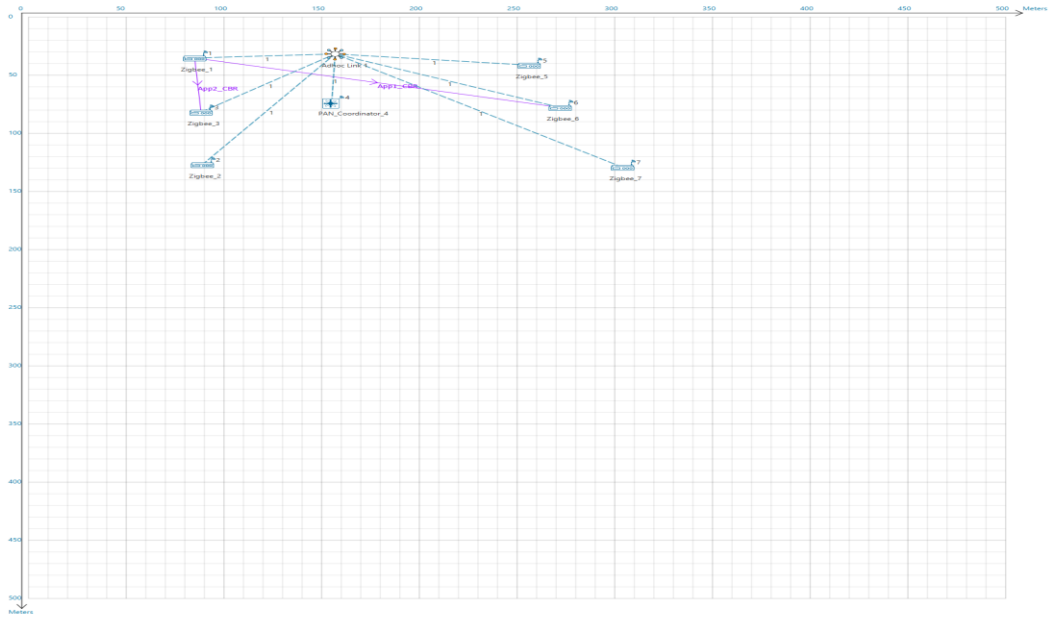
5. Simulation Configuration

- Duration: 300 seconds
- Metrics to Collect:
    - End-to-end delay
    - Packet delivery ratio
    - Throughput

INFERENCE:

1. The network uses a Zigbee PAN Coordinator to manage and route data among multiple Zigbee end devices.
2. A CBR (Constant Bit Rate) application sends data from App2_CBR to App2_DST, simulating real-time sensor traffic.
3. Nodes communicate over short-range, low-power IEEE 802.15.4 links ideal for IoT and sensor networks.
4. The centralized star topology reduces routing complexity but may introduce load on the coordinator.
5. This setup effectively demonstrates a basic Zigbee WSN model for smart environments or body area networks.

RESULTS:


Meters
Zigbee_1
Zigbee_2
Zigbee_3
RAN_Coordinator_4
Zigbee_5
Zigbee_6
Zigbee_7
App2_CBR
App1_CBR

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

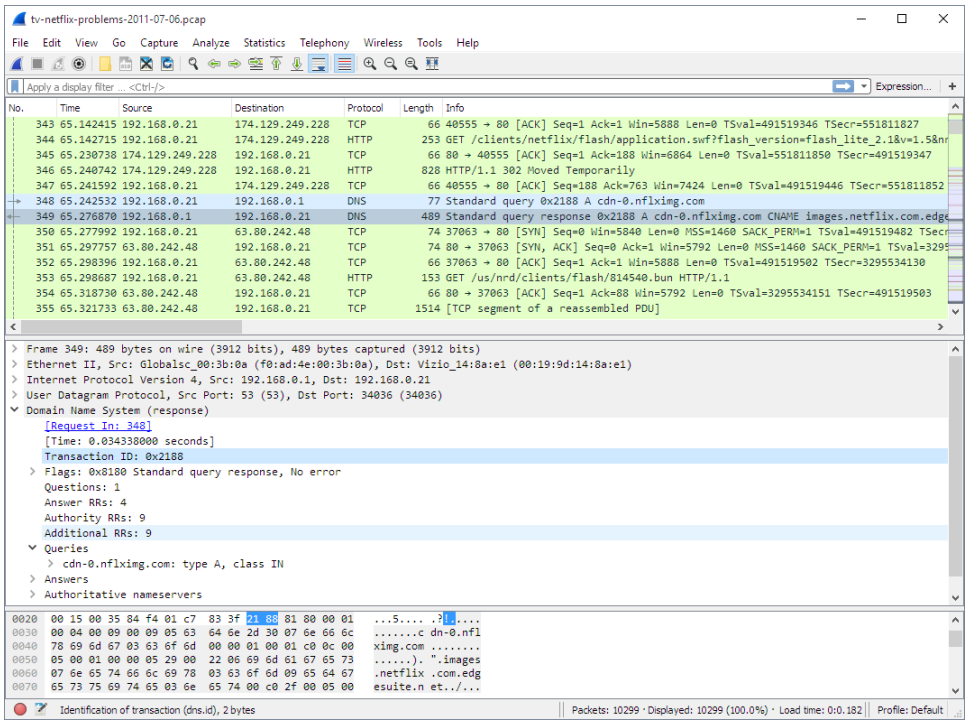


Fig 1: A typical Wireshark window

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:

To further process and understand packet data, a Python script (decode_tcp.py) was used. The script performs:

• Conversion of raw hex dump to byte data

• Parsing Ethernet header (first 14 bytes)

• Extracting IPv4 and TCP header fields

• Printing structured information such as source/destination IPs, ports, sequence numbers, window size, etc.

This manual decoding provides better control and learning opportunity for students to understand each field in a TCP/IP packet and how data is structured in network transmission

INFERENCE:

1. The code extracts and decodes Ethernet, IP, TCP, and DNS headers from a raw packet in hexadecimal format.
2. The Ethernet frame shows a packet going from source e8:fb:1c:47:3c:bd to destination 00:00:5e:00:01:fe, indicating multicast or a virtual address.
3. The IP packet is using IPv4, has a total length of 74 bytes, TTL 128, and uses TCP (protocol 6) with source 10.11.130.250 to destination 172.17.18.4.
4. The TCP segment carries a DNS query from source port 52462 to destination port 53, showing a query to www.cricbuzz.com.
5. The DNS header indicates 1 query, with no answers or authority/additional records, showing a standard name resolution request in progress.

RESULTS:

```
Total Packet Length: 88 bytes

Ethernet:
  Destination: 00:00:5e:00:01:fe
  Source: e8:fb:1c:47:3c:bd
  Type: 0x0800

Internet Protocol Version 4:
  Version: 4
  Header Length: 20 bytes
  Differentiated Services Field: 0
  Total Length: 74 bytes
  Identification: 57404
  Flags: 2
  Fragment Offset: 0
  Time to Live: 128
  Protocol: 6
  Header Checksum: 0xcf56
  Source Address: 10.11.130.250
  Destination Address: 172.17.18.4


Transmission Control Protocol:
  Source Port: 52462
  Destination Port: 53
  Length: 54 bytes
  Checksum: 0x9398

Domain Name System:
  Transaction ID: 0x44268
  Flags: 0x256
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
```

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

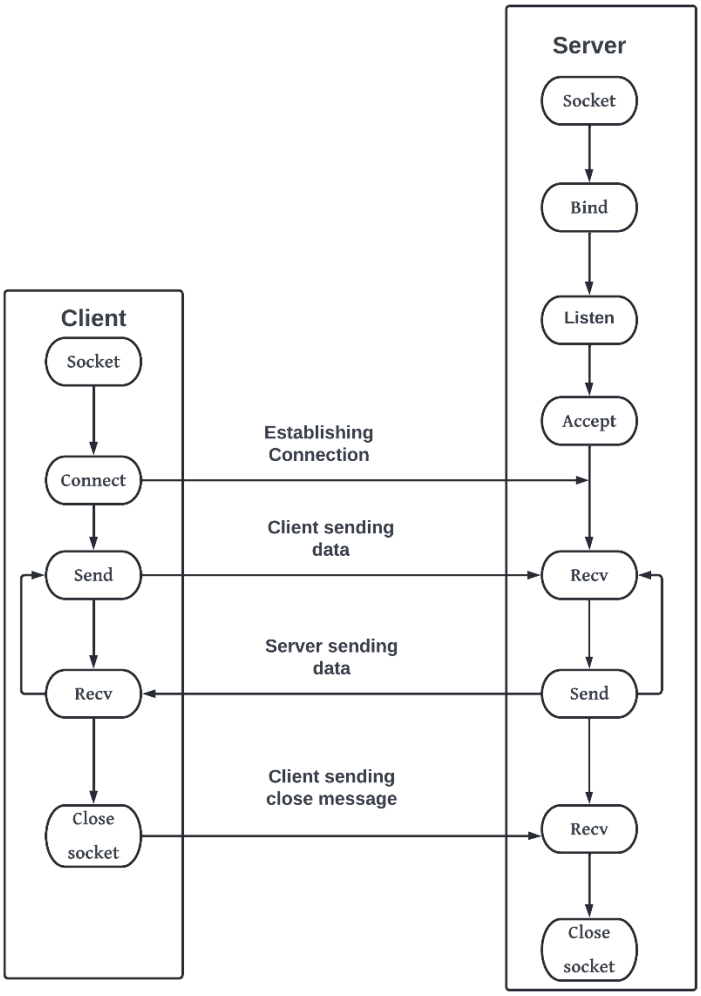


Figure 1. TCP socket flow

REFERENCE

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

Experiment No: 7, 8, 9 **Date:21-04-2025**

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 Tools: ESP32 boards, sensors (DHT11, MQ7, PIR, ultrasonic), breadboard, jumper wires, power supply.
Software Tools: Arduino IDE, ESP32 libraries, PubSubClient, Adafruit IO library.
Cloud Systems: Adafruit IO (for MQTT communication and data visualization), MQTT broker.
Communication Protocols: ESP-NOW (for local communication), MQTT (for cloud communication).
Development Tools: Arduino IDE (main tool), PlatformIO (optional), Serial Monitor (for debugging).
Integration Tools: Adafruit IO feeds

IMPLEMENTATION

The code provided implements an **edge node (ESP32_B)** that acts as an intermediary between **ESP32_A1** and **ESP32_A2** (the sensor nodes) and the **cloud** (via Adafruit IO using MQTT). Here's a step-by-step breakdown of the **theoretical implementation**:

1. Wi-Fi Setup (connectWiFi)

- **Objective:** ESP32_B needs internet connectivity to communicate with Adafruit IO.
- **Implementation:**
  - The ESP32_B connects to a **Wi-Fi network** using **SSID** and **password**.
  - The function attempts to connect and continuously checks the connection status.
  - Once connected, it allows ESP32_B to access the internet and forward data to Adafruit IO.

2. MQTT Setup (connectMQTT)

- **Objective:** Establish a connection with the **Adafruit IO MQTT server** to send sensor data.
- **Implementation:**
  - **MQTT (Message Queuing Telemetry Transport)** is a lightweight messaging protocol designed for small sensors and mobile devices.
  - ESP32_B connects to the **Adafruit IO MQTT server** using credentials (username and AIO key) provided in the code.
  - A successful connection establishes a communication channel for sending data from ESP32_B to the cloud (Adafruit IO).
  - If the connection is unsuccessful, the code keeps retrying until it succeeds.

3. Communication via ESP-NOW

- **Objective:** ESP32_B receives sensor data from **ESP32_A1** and **ESP32_A2** using **ESP-NOW**.
- **Implementation:**
  - **ESP-NOW** is a protocol that allows ESP32 devices to communicate with each other without needing a router or internet.

- The function esp_now_register_recv_cb(onReceive) registers a callback function, **onReceive**, which is called when data is received from other devices (A1 or A2).
- When **ESP32_A1** or **ESP32_A2** sends data, the **onReceive()** function is triggered.

4. Data Handling and Acknowledgment

- **Objective:** ESP32_B acknowledges receipt of data and processes it for forwarding to Adafruit IO.
- **Implementation:**
  - In the onReceive() function:
    - **ESP32_B** reads the incoming data from A1 or A2 and converts it into a string format for easy handling.
    - **Acknowledgment (ACK):** ESP32_B sends an acknowledgment message back to the sending node (A1 or A2) to confirm that it received the data correctly.
      - The sendReply() function handles sending back the acknowledgment message.
    - ESP32_B determines which node sent the data by comparing the **MAC address** of the sender (info->src_addr).
    - If the data is from **ESP32_A1** or **ESP32_A2**, it sends a personalized ACK to the sender.

5. Publishing Data to Adafruit IO

- **Objective:** Forward received data to the cloud (Adafruit IO) for further processing or visualization.
- **Implementation:**
  - The publishToAdafruit() function is responsible for publishing data to **Adafruit IO**.
  - Based on the sender's **MAC address**, ESP32_B chooses the appropriate MQTT **feed** (for example, **feed_bin1** or **feed_bin2**) to publish the data.
  - If the data contains information about the **bin fill level**, it publishes the data to the corresponding feed on Adafruit IO.
  - The function client.publish() sends the data to the cloud.
  - **Rate limiting:** To avoid sending too many messages in a short time, a small delay (2.5 seconds) is added between messages.

6. Peer Configuration (ESP-NOW Peers)

- **Objective:** Define which devices ESP32_B should communicate with using ESP-NOW (i.e., A1 and A2).
- **Implementation:**
  - The esp_now_add_peer() function is used to add **ESP32_A1** and **ESP32_A2** as peers.
  - It ensures ESP32_B is aware of the MAC addresses of the devices it needs to receive data from.
  - By adding the peers, **ESP32_B** only listens to messages from A1 and A2 and ignores others.

7. Looping and Reconnection (MQTT Keep-Alive)

- **Objective:** Ensure the MQTT connection stays alive and handle disconnections.
- **Implementation:**
  - In the loop() function, ESP32_B continuously checks if it is connected to the MQTT server.

- If it is disconnected, it attempts to reconnect using the connectMQTT() function.
- client.loop() ensures the MQTT client keeps checking for new messages or any required actions.

Conclusion of the Implementation:

- **ESP32_B** serves as an **edge node** that receives sensor data from two other ESP32 devices (A1 and A2) via ESP-NOW.
- It processes the data and forwards relevant information (like bin fill level) to **Adafruit IO** for cloud storage/monitoring.
- The node acknowledges receipt of the data back to the sender and ensures the system operates reliably by maintaining an MQTT connection.
- This setup allows for **low-latency communication** between sensor nodes (A1 and A2) and the cloud, enabling real-time monitoring and decision-making based on the sensor data.

RESULTS:

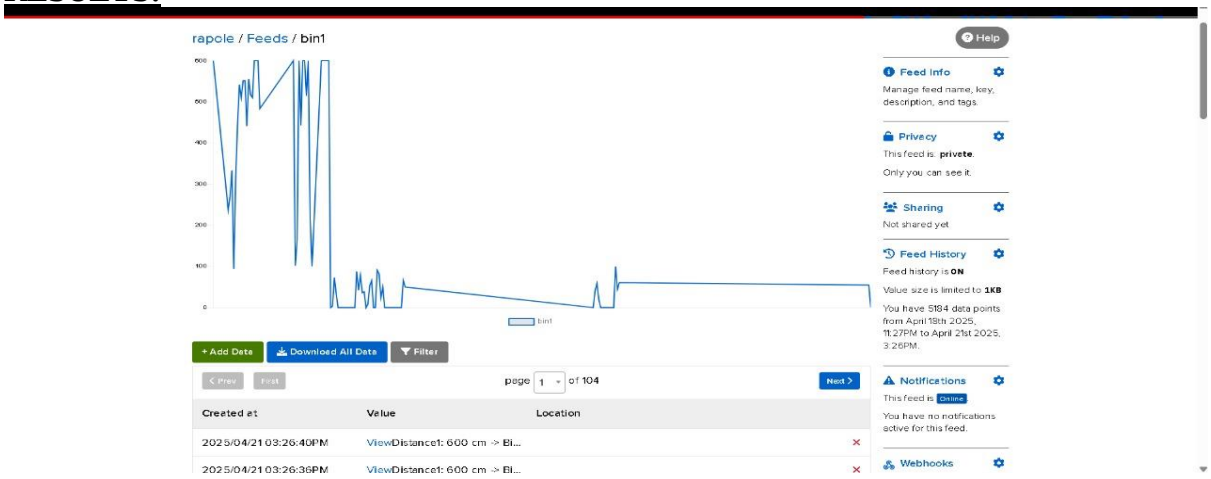


Fig: esp32A1 values send to Adafruit feed value bin1 via edge node

Fig: esp32A2 values send to Adafruit feed value bin1 via edge node

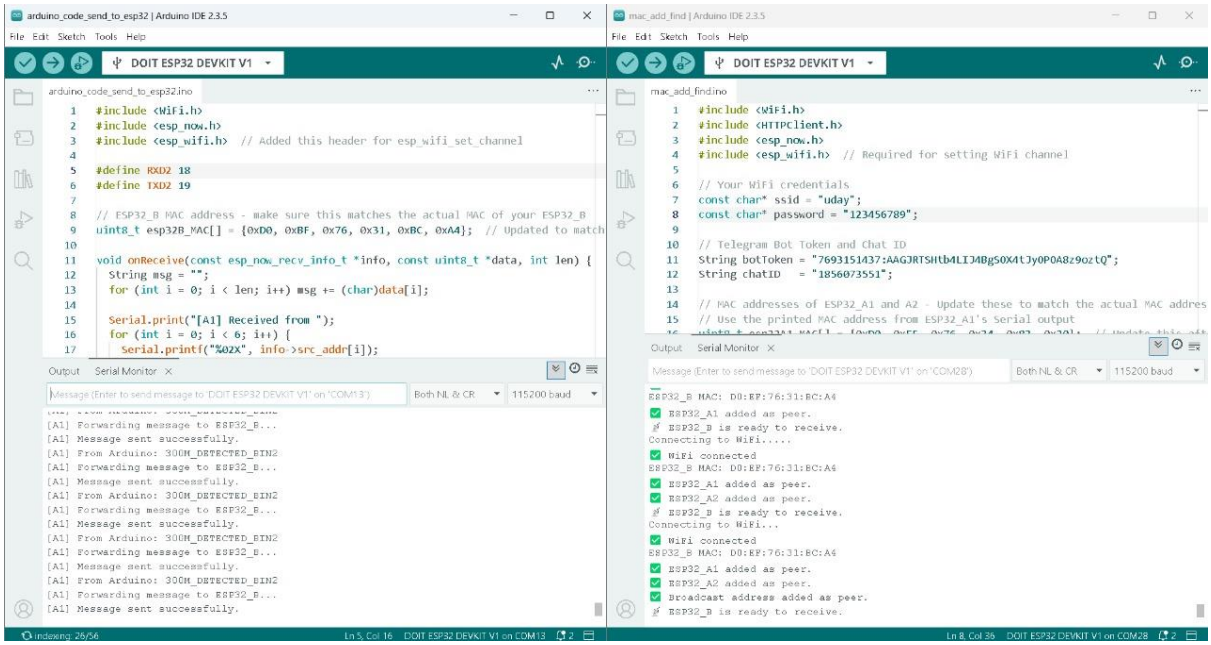


Fig: ESP32 code Output for edge node

Experiment No: 10,11,12 **Date:21-04-2025**

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:

Programming Languages: Python, JavaScript (AJAX)
Frameworks: Flask (Python), Bootstrap (Frontend)
Cloud Platform: Adafruit IO (for storing and fetching IoT data)
APIs: Adafruit IO API, Requests (for HTTP requests)
Libraries: Regex (Python), PubSubClient (for MQTT-based communication)
Development Tools: Flask Development Server, HTML/CSS/JS for frontend

IMPLEMENTATION:

This Flask-based web application provides a **real-time dashboard** for monitoring two smart waste bins' fill levels. It communicates with the **Adafruit IO** cloud platform, where the fill level data is stored and retrieved.

1. Flask Web Application:

The Flask framework is used to create a simple web application that provides endpoints to display the **latest** and **average fill levels** for two smart waste bins.

a. Routes and Views:

- **Route /:**
  The root route (/) serves the **HTML page** which is the dashboard interface for the user. The render_template function renders index.html, which displays the dashboard with **live data** from two waste bins.
- **Route /bin1/latest**:
  This route fetches the **latest data** for Bin 1 from Adafruit IO and returns it as a JSON object. The data includes the latest recorded fill level.
- **Route /bin2/latest**:
  Similar to /bin1/latest, this route fetches the **latest data** for Bin 2 from Adafruit IO.
- **Route /bin1/average**:
  This route fetches the **average fill level** for Bin 1 based on the last 15 recorded fill levels. The average is computed by processing the data received from Adafruit IO.
- **Route /bin2/average**:
  Similarly, this route calculates and returns the **average fill level** for Bin 2.

2. Communication with Adafruit IO:

The system uses **Adafruit IO**, a cloud service that stores and provides real-time data for IoT devices. The Flask application makes HTTP requests to Adafruit IO to get the fill levels for the bins. The application uses the **Adafruit IO API** for this communication.

API Call (fetch_latest_value and fetch_average functions):

- **fetch_latest_value(feed)**:
  This function fetches the latest **fill level data** from a specified feed (e.g., bin1, bin2) on Adafruit IO. It retrieves the most recent entry and extracts the fill level information.
- **fetch_average(feed)**:
  This function fetches the **last 15 data points** from the given feed and calculates the **average** of the values. It filters out invalid or incomplete data, ensuring that the calculated average is based only on valid fill level readings.

3. Data Extraction:

- The **extract_fill_level(msg)** function uses regular expressions to extract the full text (e.g., '**Bin Fill Level: 45%**') from the message received from Adafruit IO.
- The **extract_numeric_level(msg)** function extracts just the numeric value (e.g., **45**) from the message to facilitate average calculation.

4. Frontend (HTML + JavaScript):

The frontend uses **HTML**, **Bootstrap**, and **JavaScript** to create a **responsive, user-friendly dashboard**. It displays the latest and average fill levels for Bin 1 and Bin 2. The user can refresh the data by clicking the "Refresh" buttons.

JavaScript Functions (AJAX):

- **loadBin1()** and **loadBin2()** are JavaScript functions that use the **Fetch API** to make asynchronous calls to the Flask backend. They request the latest and average fill level data for each bin and update the HTML page accordingly.
    - The **innerText** of the corresponding elements (bin1-latest, bin1-average, etc.) is updated dynamically with the fetched data.
- These functions are called automatically when the page loads, ensuring that the dashboard displays the most current data on startup.

5. Backend (Python):

The Python part of the application handles the logic for communicating with the **Adafruit IO API** and processing the responses to extract the relevant fill level data.

Key Backend Functions:

- **fetch_latest_value(feed)**:
  Makes an HTTP GET request to Adafruit IO, fetches the most recent fill level data, and returns the extracted fill level information (e.g., "Bin Fill Level: 45%").
- **fetch_average(feed)**:
  Fetches the last 15 fill level data points and computes the average by summing the values and dividing by the count.

RESULTS:

Fig: The above picture shows the UI to get the latest bin levels and average bin fill percentage

```
[Running] python -u "c:\Users\c.naga sai varun\Desktop\iotnew\app.py"
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 111-810-064
```

Fig: The above figure shows the local server is hosted on http://127.0.0.1:50

Smart Wastage Management System Using IOT

Chapter1: Introduction

In today's rapidly urbanizing world, effective waste management has become a critical issue for municipalities and governments. With growing population density and increased consumption, waste generation has escalated at an alarming rate, putting immense pressure on traditional waste collection systems. These systems often rely on fixed schedules for waste collection, which can lead to problems such as overflowing bins, inefficient resource utilization, unpleasant odors, and unhygienic conditions that threaten public health and the environment.

To address these challenges, technology-driven solutions such as the Smart Waste Management System using Internet of Things (IoT) are being developed and adopted. IoT enables real-time monitoring, automation, and data-driven decision-making by connecting physical devices like sensors, microcontrollers, and communication modules to the internet. In the context of waste management, smart bins equipped with ultrasonic sensors can detect the level of waste inside them. Additionally, gas sensors can identify the presence of hazardous gases like methane or carbon monoxide, while temperature and humidity sensors can assess environmental conditions that may indicate the decomposition of waste.

The system works by collecting data from various sensors installed in waste bins and transmitting it to a central cloud server or mobile application using Wi-Fi, GSM, or LoRa communication modules. This information allows municipal authorities or waste management services to prioritize bin collection based on real-time needs, thus optimizing the collection routes, saving fuel, reducing labor costs, and minimizing environmental impact.

Moreover, the integration of features such as GPS tracking, automatic alerts, and user-friendly dashboards can further enhance the efficiency of the waste collection process. Citizens can also participate by reporting issues via mobile applications, leading to increased community involvement and awareness.

The implementation of a Smart Waste Management System not only ensures cleanliness and public hygiene but also aligns with the vision of smart cities, sustainability, and resource optimization. By leveraging the power of IoT, this system serves as a step forward in creating eco-friendly urban infrastructures and a cleaner future.

Chapter2: Literature Review

Over the past decade, the integration of Internet of Things (IoT) technologies into waste management systems has gained substantial attention in both academic research and industrial applications. Several studies and projects have been conducted to address the inefficiencies in traditional waste collection methods and to promote sustainable urban development.

1. Al Mamun et al. (2016) proposed a smart waste bin that uses ultrasonic sensors to monitor bin levels and transmits data through GSM modules. Their system helped in reducing unnecessary bin pickups and improved operational efficiency. However, it lacked integration with gas sensors for monitoring hazardous waste emissions.
2. Longhi et al. (2012) introduced a multi-sensor-based smart waste management system using wireless sensor networks (WSN). Their approach focused on bin fill-level detection and data aggregation, but it was limited by the highpower consumption and range limitations of Zigbee-based communication.
3. Kanchan et al. (2014) developed a smart garbage monitoring system using RF modules and microcontrollers. Though cost-effective, the system had limitations in terms of real-time data access and lacked scalability for larger cities.
4. S. Islam and F. Wahid (2018) designed a waste management framework using IoT and cloud computing where data from multiple smart bins was visualized on a central dashboard. This study highlighted the importance of cloud platforms for remote monitoring and control but faced delays due to network latency.
5. Thakker and Narayanamoorthy (2015) implemented an IoT-based smart waste system using ultrasonic sensors and GSM modules. Their system sent SMS alerts when bins were full, reducing the need for manual bin checks. However, the system was not capable of analyzing other environmental parameters like gas presence or temperature.

These studies collectively underscore the potential of IoT in transforming waste management systems. Most of the previous work focused on basic fill-level detection and notification systems. However, there is a growing need to expand functionality by integrating multiple environmental sensors (e.g., gas, temperature, humidity), incorporating cloud computing, and utilizing low-power communication protocols like LoRa or Wi-Fi for improved scalability and coverage.

The current project aims to overcome the limitations identified in previous studies by designing an IoT-enabled Smart Waste Management System that not only monitors bin levels but also detects harmful gas emissions and environmental conditions in real time. The system will utilize microcontrollers like the Raspberry Pi Pico W or ESP32 and communicate data to a cloud interface for efficient waste collection and management

Chapter3: Objectives

- Develop a Smart Waste Management System that separates wet and dry waste using moisture sensors and monitors bin fill levels using ultrasonic sensors.
- Implement Communication Between End Nodes and Edge Node: Use ESP-NOW protocol for low-power, short-range wireless communication between end nodes (waste bins) and the edge node (ESP32).
- Ensure Data Processing at the Edge: The Edge Node (ESP32) processes and filters the data received from multiple end nodes to ensure relevant information (like bin fill levels) is sent to the cloud.
- Enable Cloud Integration: Use MQTT protocol for communication between the Edge Node and Adafruit IO cloud platform for storing and visualizing bin data in real-time.
- Design a Local Server for Data Retrieval: Develop a local server (using Flask or Django) to retrieve data from Adafruit IO and make it accessible to the user interface.
- Create a Web Interface for Users: Build a web-based user interface to visualize and interact with the waste bin data, showing metrics like current bin fill levels, average bin fill levels, and historical data.
- Implement Real-Time Monitoring: Ensure real-time data transmission and visualization for effective monitoring of the waste management system.
- Enable Data Filtering and Visualization: Allow for the filtering of sensor data (e.g., separating waste types or monitoring bin thresholds) and visualize the data via graphs, gauges, and other UI components.
- Provide Alerts for Full Bins: Trigger visual alerts (e.g., blinking LEDs) when the bins are 100% full or when motion is detected near the bins.
- Support Scalability and Extensibility: Design the system to easily scale by adding more end nodes (bins) or extending the edge computing capabilities.
- Optimize Power Consumption: Ensure that communication protocols and the entire system are optimized for low power usage, suitable for battery-operated devices.

Chapter3: System Overview

Smart Waste Management System Using IOT– System Overview

1. Core Objective:

To monitor and manage garbage bins in real time by detecting waste type (wet/dry), bin fill level, and human presence — and communicate this information efficiently through edge computing, ESP-NOW, MQTT, and a local UI via Flask/Django.

2. System Components:

End Nodes (2 Bins – Node 1 & Node 2):

Each end node includes:

- Moisture Sensor – Determines whether waste is wet or dry.
- Ultrasonic Sensor 1 – Measures bin fill level.
- Ultrasonic Sensor 2 – Detects human presence.
- Red LED – Glows when bin is full.
- Yellow LED – Glows when a person is detected and dust is placed.
- ESP32 Node MCU – Collects sensor data and sends it to the edge via ESP-NOW protocol.

Edge Node (ESP32):

- Receives data from both end nodes using ESP-NOW (a fast, connectionless Wi-Fi communication protocol).
- Performs filtering, e.g., only forwarding certain types of data like bin fill levels.
- Sends filtered data to Adafruit IO using MQTT protocol and stores it in separate feeds (one per bin).

Adafruit IO (Cloud Platform):

- Acts as an MQTT broker and cloud data storage.
- Each bin's data is stored in a separate feed (e.g., bin1_level, bin2_level).
- Can be accessed using REST API to fetch historical or real-time data.

Local Server (Flask or Django App):

- Hosted on a local machine.
- Uses Adafruit IO REST API to pull latest or historical data.
- Displays data via a web-based UI with buttons:
    - Latest value of bin1.
    - Latest value of bin2.
    - Average of bin1 data.
    - Average of bin2 data.

3. Communication Flow

Source	Destination	Protocol
End Node ESP32 → Edge ESP32	ESP-NOW	Transfer all sensor data
Edge ESP32 → Adafruit IO	MQTT	Send selected data (e.g., bin levels) to cloud
Flask/Django Server → Adafruit IO	REST API (HTTP)	Fetch real-time/latest data and history for UI display

4. Edge Computing Role

- Acts as an intermediate processing unit.
- Performs filtering (e.g., only sending bin level, not moisture/human data).
- Converts protocol: ESP-NOW (end nodes) → MQTT (cloud).

- Manages multiple incoming node data streams and directs each to the appropriate Adafruit feed.

5. Data Flow Example

[End Node 1]
  ├─ Moisture: Wet
  ├─ Bin Level: 90%
  └─ Human Detected

↓ ESP-NOW

[Edge ESP32]
  ├─ Receives all data from Node 1
  ├─ Filters: Only sends "Bin Level"
  └─ Publishes to Adafruit MQTT → Feed: bin1_level → Payload: 90

[Flask App]
  └─ Fetches data from Adafruit (via REST API) → Displays on UI

6. Web UI Functionality

local web interface includes:

- Button Bin1: Get latest value of Bin 1
- Button Bin2: Get latest value of Bin 2
- Button average value: Get average of Bin 1 levels
- Button average value: Get average of Bin 2 levels

7. Technologies Used

- ESP-NOW – For node-to-edge wireless communication.
- MQTT – For edge-to-cloud messaging.
- REST API – For server-to-cloud communication.
- Edge Computing – Processing and filtering data at the edge.
- Adafruit IO – IoT cloud platform for data storage and visualization.
- Flask/Django – Web framework for local data UI.

Chapter4: Methodology

Smart Waste Management System Using IOT

1. End Node Setup and Local Data Acquisition

Each waste bin is embedded with an End Node, which is responsible for sensing, basic processing, and transmitting data to the Edge Node. The hardware components and their respective functionalities include:

- Moisture Sensor: Differentiates between wet and dry waste. This assists in segregating waste for eco-friendly disposal and recycling.
- Ultrasonic Sensor 1: Measures the bin's fill level. If the bin is 100% full, the system triggers a Red LED alert.
- Ultrasonic Sensor 2: Detects if a person is near the bin for waste disposal. This triggers a Yellow LED to indicate interaction.
- Arduino Microcontroller: Interfaces with all sensors and handles raw data collection and simple logic.
- ESP32 Microcontroller: Communicates the processed sensor data to the Edge Node using the ESP-NOW protocol, a proprietary, lightweight wireless communication protocol developed by Espressif.

2. ESP-NOW Communication Protocol

- ESP-NOW is used for low-latency, peer-to-peer communication between the ESP32s at each End Node and the Edge Node.
- Since ESP-NOW supports MAC address-based communication, each end node sends data specifically to the Edge ESP32.
- This local communication does not require Wi-Fi or internet, which makes it suitable for low-power IoT environments.

3. Edge Node Responsibilities

The Edge Node, also an ESP32, serves as a middleware gateway between the local sensor network and the internet/cloud. Its responsibilities include:

- Listening to incoming data from all End Nodes via ESP-NOW.
- Identifying source nodes using predefined MAC addresses.
- Parsing and filtering the data — for example, selecting only the bin level information to be uploaded.
- Protocol Conversion: The Edge Node converts the ESP-NOW data into MQTT messages to be sent to the cloud.
- Publishing Data to Adafruit IO:
    - bin1_level → Data from Node 1
    - bin2_level → Data from Node 2
- This selective upload optimizes cloud bandwidth and helps in focused analysis.

4. Cloud Communication via MQTT (Adafruit IO)

- The Edge Node connects to the internet using Wi-Fi and uses the MQTT (Message Queuing Telemetry Transport) protocol.
- MQTT is ideal for IoT due to its lightweight overhead, publish-subscribe model, and real-time delivery.
- Data is sent to Adafruit IO, which acts as the cloud backend and visualization dashboard.
- Adafruit IO stores the data into two separate feeds:
    - bin1_level
    - bin2_level
- This helps in tracking individual bins and analysing waste management performance.

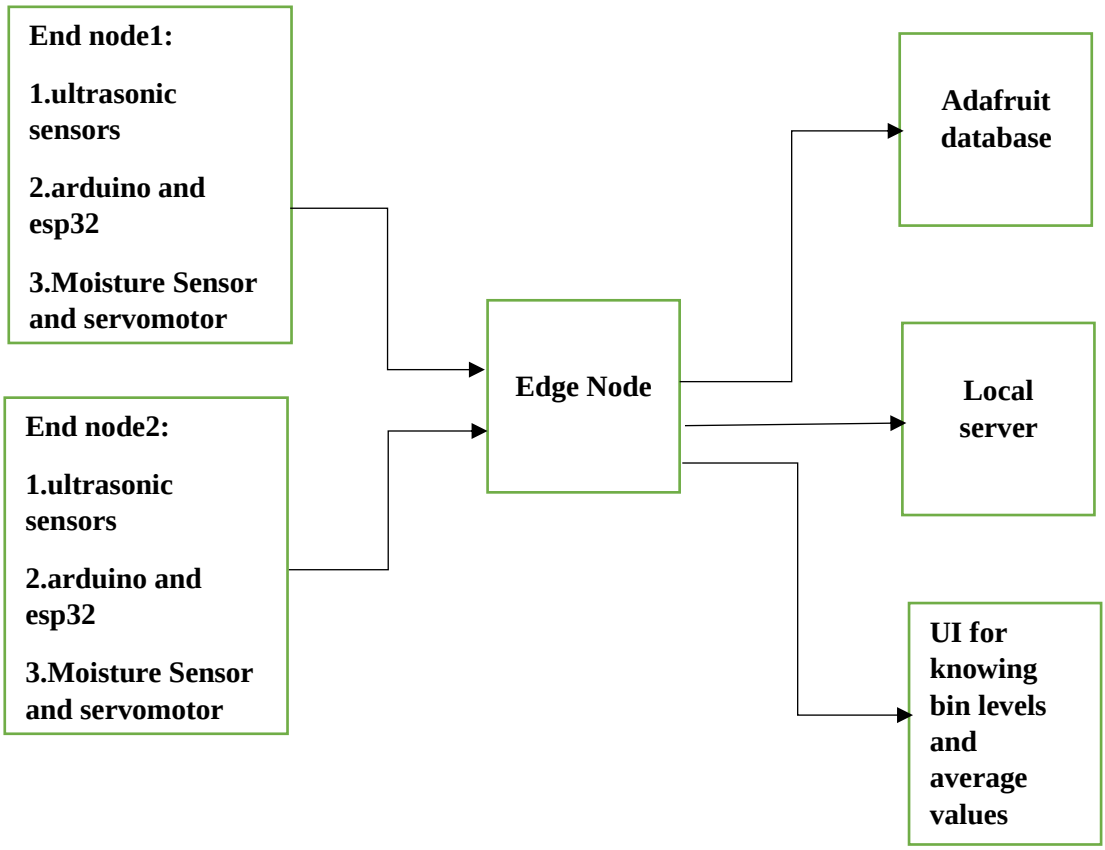
5. Local Server and Web Interface (UI)

A Python-based local server (developed using Flask or Django) is created for user interaction and data visualization. Here's how it functions:

- Periodically pulls the latest data from Adafruit IO using its RESTful HTTP API.
- Offers a Web Interface (UI) hosted on localhost or a local IP.
- The UI contains 4 buttons:
    - Bin 1 – Latest Level: Fetches and displays the latest value from bin1_level.
    - Bin 2 – Latest Level: Fetches and displays the latest value from bin2_level.
    - Average – Bin 1: Calculates and shows average level of bin 1 over time.
    - Average – Bin 2: Calculates and shows average level of bin 2 over time.
- This interface allows local administrators to monitor bins without needing to log into Adafruit IO manually.

Summary: Edge Intelligence and Efficiency

- The system implements edge computing by performing protocol conversion and data filtering at the Edge Node.
- Only relevant data is pushed to the cloud, conserving bandwidth and improving efficiency.
- Real-time communication, intelligent filtering, and cloud integration make this system scalable, responsive, and ready for smart city environments.



The above block diagram shows our implementation

Chapter5: Tools & Systems

Here's a structured section for your "Tools & Systems" that were used Smart Waste Management System Using IOT:

Tools & Systems Used:

Hardware Components

Component	Purpose
ESP32 (End Nodes & Edge Node)	Wireless microcontroller used for ESP-NOW (local communication) and MQTT (cloud communication).
Arduino UNO	Reads sensor data and communicates with ESP32 via UART.
Ultrasonic Sensor (HC-SR04)	Measures bin fill level and detects human interaction.
Moisture Sensor	Detects whether the waste is wet or dry for segregation.
LEDs (Red & Yellow)	Visual indicators for full bin and person detection.
Jumper Wires, Breadboard	For prototyping and sensor interfacing.
Power Supply	Powers the microcontroller and sensors.

Software Tools

Tool/Platform	Purpose
Arduino IDE	Writing and uploading Code to Arduino and ESP32 devices.
Python (Flask/Django)	Backend development for the local server and web UI.
Adafruit IO	Cloud IoT platform for storing and visualizing sensor data.
MQTT Protocol	Used by the Edge Node to transmit sensor data to Adafruit IO.
ESP-NOW Protocol	Peer-to-peer wireless communication protocol between ESP32 devices.
JSON Format	Used for structuring sensor data packets between devices.
Web Browser	For accessing the locally hosted web interface.

Cloud & Networking

System	Function
Adafruit IO Feeds	Stores bin data separately (bin1_level, bin2_level) for tracking.
Wi-Fi Network	Enables Edge Node to connect to the internet and publish MQTT messages.
MAC Addresses	Unique identifiers for each ESP32 device in ESP-NOW communication.

Chapter6: Nodes

Nodes in the Smart Waste Management System Using IOT:

1. End Nodes (Waste Bin Sensors)

Role:

The End Nodes are responsible for monitoring and collecting data from the sensors placed in each waste bin. These nodes handle local sensing, simple data processing, and initial communication with the Edge Node.

Key Components:

- Microcontroller: An Arduino UNO microcontroller is used to interface the sensors and process the initial data. It communicates with the ESP32 using UART (Universal Asynchronous Receiver-Transmitter).
- Sensors:
    - Ultrasonic Sensors (HC-SR04): Measure the bin's fill level. If the bin reaches 100% capacity, it triggers a Red LED to signal that the bin is full.
    - Moisture Sensor: Determines the type of waste (wet or dry), which aids in waste segregation for recycling and proper disposal.
    - Human Detection: Another Ultrasonic Sensor detects when a person is interacting with the bin, which then triggers a Yellow LED as an alert to indicate the bin is in use.
- Data Collection & Processing: The End Node aggregates the sensor data and sends it to the Edge Node. The data includes:
    - Fill Level: Percentage or numerical data (e.g., 85% full).
    - Waste Type: Wet or Dry.
    - Person Detection: Boolean (True/False).

Communication with Edge Node:

The End Nodes send their sensor data to the Edge Node using the ESP-NOW protocol, a low-latency, peer-to-peer communication protocol. The data is sent wirelessly, and each End Node is assigned a unique MAC address to identify its data.

Data Format:

```
{
  "node_id": bin1,
  "bin_level": 85,
  "moisturevalue": "780",

}
```

2. Edge Node (Data Aggregator)

Role:

The Edge Node serves as an intermediary that connects the End Nodes to the cloud (Adafruit IO). It is responsible for receiving data from the End Nodes, processing it, filtering out unnecessary information, and then publishing the relevant data to the cloud via MQTT.

Key Components:

- ESP32: A powerful microcontroller with Wi-Fi capabilities, used as the Edge Node. It handles both receiving data from End Nodes (using ESP-NOW) and sending the data to the cloud (using MQTT over Wi-Fi).
- Data Processing:
    - The Edge Node listens for incoming data from all End Nodes via ESP-NOW.
    - It filters the data, e.g., selecting only the fill level of the bins, and then formats the data for transmission to the cloud.
- Protocol Conversion: The Edge Node converts the data from ESP-NOW (used locally between End Nodes and Edge Node) to MQTT (for cloud communication). This is

done by packaging the data into a structured format (e.g., JSON) and using the MQTT protocol to send the data to Adafruit IO.

- Communication with End Nodes:
  - The Edge Node receives data via ESP-NOW.
  - It identifies which End Node the data came from by using the MAC address of the sending End Node.
- Communication with Cloud:
  - The Edge Node connects to the internet via Wi-Fi and uses MQTT to send data to the cloud (Adafruit IO).
  - The Edge Node sends data from each End Node to separate MQTT feeds (e.g., bin1_level, bin2_level).

Data Format Sent to Adafruit IO:

```
{
  "bin1_level": 85,
  "bin2_level": 90
}
```

3. Cloud Platform (Adafruit IO)

Role:

The Cloud Platform serves as the data storage and visualization system. It receives data from the Edge Node and provides real-time monitoring through its dashboard.

Key Components:

- MQTT Feeds: Data is stored in specific MQTT feeds on Adafruit IO. For instance, one feed might be dedicated to Bin 1's fill level and another for Bin 2's fill level.
- Data Visualization: Adafruit IO offers dashboards and UI elements for visualizing data, including charts, graphs, and gauges. Users can monitor bin levels and receive alerts based on the data received.

Data Format in Adafruit IO:

- Each bin's fill level is stored in a unique feed, e.g., bin1_level and bin2_level.
- The data can be accessed through Adafruit's API or visualized on the dashboard.

Flow of Data Between Nodes

1. Data Sensing (End Node):
   - The End Node detects the bin's fill level and waste type using the Ultrasonic and Moisture Sensors.
   - The data is packaged and sent to the Edge Node using ESP-NOW.
2. Data Aggregation & Filtering (Edge Node):
   - The Edge Node receives the data from multiple End Nodes.
   - It processes the data and filters out irrelevant information, selecting only the essential details like bin fill levels.
   - The filtered data is then sent to the cloud (Adafruit IO) via MQTT.
3. Cloud Storage & Visualization (Adafruit IO):
   - The data is stored in MQTT feeds on Adafruit IO, where it can be monitored in real-time via the cloud dashboard.

4.Local Server Interface:

   - A local server (Python-based, using Flask or Django) is used to pull data from Adafruit IO and present it in a web interface.
   - The web UI allows for interaction with the data, such as retrieving the latest bin levels or calculating average values.

Chapter8: Network

Network Architecture of IoT-Based Smart Waste Management System

1. End Nodes to Edge Node Communication

Communication Protocol: ESP-NOW

- ESP-NOW is a proprietary protocol developed by Espressif (the maker of ESP32), which allows direct communication between ESP32 devices without the need for a Wi-Fi router. It is a low-power, low-latency communication protocol that is ideal for scenarios where devices need to communicate wirelessly over short distances with minimal overhead.

Flow of Data:

- The End Nodes (Waste Bins) consist of an Arduino UNO interfacing with sensors like Ultrasonic and Moisture Sensors. These sensors capture data related to the fill level of the bin and the type of waste (wet or dry).
- The End Nodes send this sensor data to the Edge Node (ESP32) via ESP-NOW. The data is packaged into a structured format (e.g., JSON) for easy interpretation.

Key Characteristics:

- Peer-to-Peer Communication: The End Nodes (Arduino + ESP32) send data to the Edge Node directly using the ESP-NOW protocol.
- Low Power: ESP-NOW is optimized for low power consumption, making it suitable for battery-powered IoT applications.
- Short-Range Communication: Typically used within a local area (e.g., within the range of an office or industrial workspace).

2. Edge Node to Cloud Communication

Communication Protocol: MQTT

- Once the Edge Node (ESP32) receives the data from the End Nodes, it processes and filters the data as needed (e.g., extracting only relevant information like bin fill levels).
- The Edge Node then sends the processed data to the cloud platform (Adafruit IO) using MQTT.

Flow of Data:

- The Edge Node connects to the internet using Wi-Fi (e.g., through a router or hotspot) and communicates with Adafruit IO via the MQTT protocol.
- The data is sent in MQTT messages to the appropriate feeds (e.g., bin1_level, bin2_level) on the Adafruit IO platform.
- MQTT is an efficient protocol for sending small amounts of data over the internet. It operates on a publish/subscribe model, where the Edge Node publishes data to specific feeds that can be subscribed to by any interested parties (e.g., web applications or users).

Key Characteristics:

- Publish/Subscribe Model: The Edge Node publishes sensor data to specific MQTT topics (feeds) on Adafruit IO, which can be accessed and visualized by subscribed clients.
- Low Bandwidth: MQTT is designed to minimize network usage, making it ideal for IoT applications where bandwidth is limited.
- Real-Time Communication: The Edge Node sends data in real time, and Adafruit IO updates its dashboard instantly.

3. Local Server Communication

Communication Protocol: HTTP/REST API

- The Edge Node sends data to Adafruit IO over MQTT, where it can be visualized. However, for local monitoring and interaction, a local web server is used to retrieve and display the data from Adafruit IO.

Flow of Data:

- A local server (using Flask or Django with Python) communicates with Adafruit IO to retrieve data from the MQTT feeds.
- The local server acts as an intermediary between the cloud platform and the user interface (UI). It provides data through HTTP requests to a web browser.
- The web interface (UI) enables users to monitor the status of the waste bins, such as current fill levels and the average fill levels for the bins. The server serves this data through API endpoints.

Key Characteristics:

- RESTful API: The server exposes API endpoints to query the latest sensor data from Adafruit IO.
- User Interface: A web interface (e.g., built with HTML, JavaScript, and Bootstrap) provides visual representation of the data, such as gauges, charts, or values indicating the bin's status.

4. Communication Flow Overview

1. End Nodes (Waste Bin Sensors):
    - Sensors (Ultrasonic, Moisture) monitor the fill level and type of waste in the bins.
    - The data is transmitted to the Edge Node using ESP-NOW.
2. Edge Node (Data Aggregator):
    - The Edge Node (ESP32) receives data from multiple End Nodes via ESP-NOW.
    - The data is processed and filtered (e.g., bin fill levels), and sent to Adafruit IO using MQTT.
3. Cloud (Adafruit IO):
    - The data is stored in MQTT feeds on Adafruit IO. Feeds like bin1_level and bin2_level allow real-time data visualization and analysis.
    - The data can be accessed via the Adafruit IO Dashboard or API.
4. Local Server & User Interface:
    - The local server (Flask/Django) retrieves the data from Adafruit IO using the MQTT API and exposes it to a web interface.
    - The web UI allows users to monitor bin fill levels, check the status of waste bins, and view historical data.

Summary of Network Architecture:

- End Nodes communicate with the Edge Node using ESP-NOW, sending sensor data wirelessly.
- The Edge Node aggregates the data, filters it, and sends it to Adafruit IO over MQTT.
- Adafruit IO stores the data in MQTT feeds, which are accessible via a local server running Flask or Django.
- The local server provides the data to a web-based UI, where users can interact with the system and view real-time data.

Chapter9: Results & Analysis

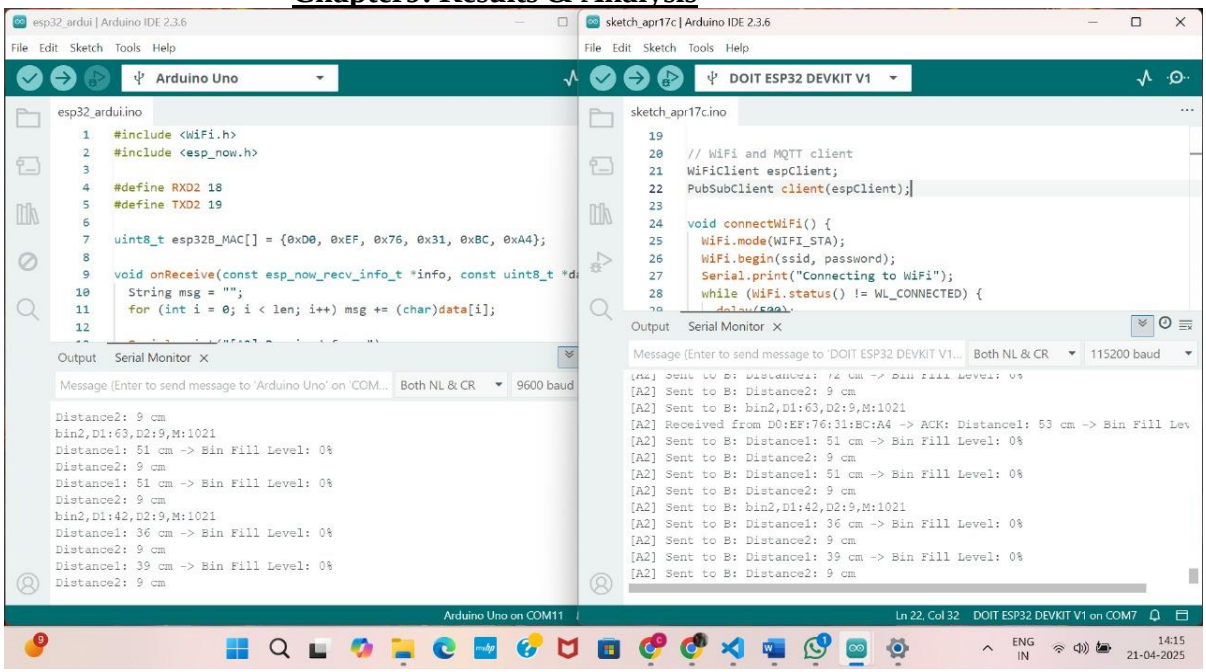


Fig: Figure shows the Arduino is sending to data to end Esp32 Both are the outputs for end node

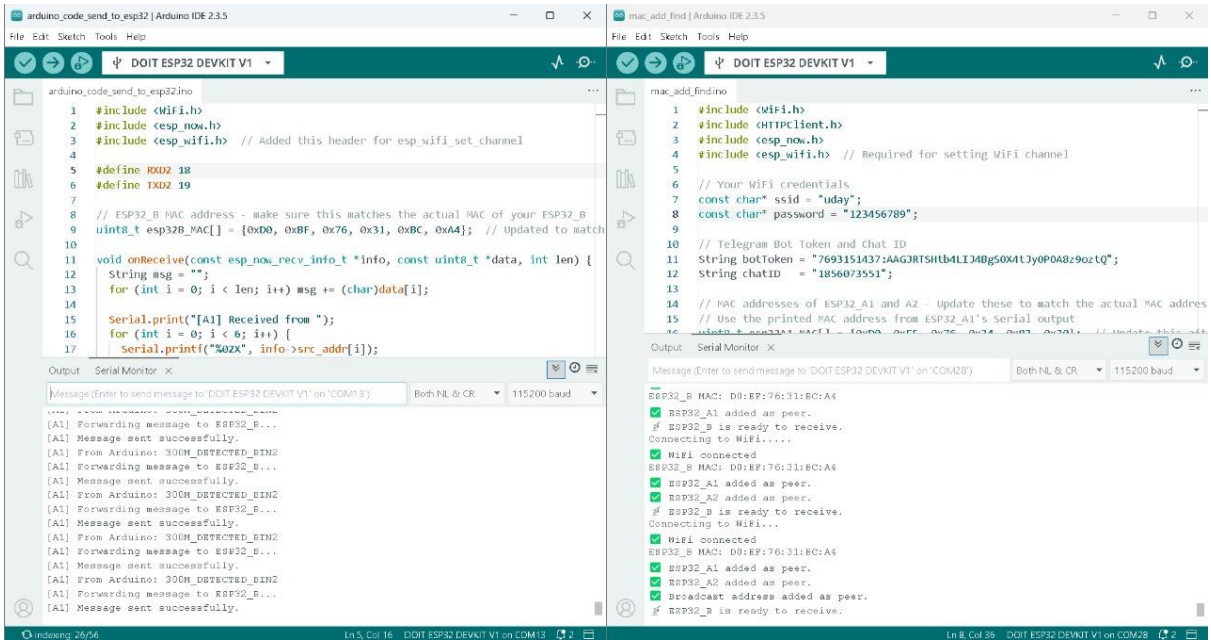


Fig: The figure shows the output for edge Node

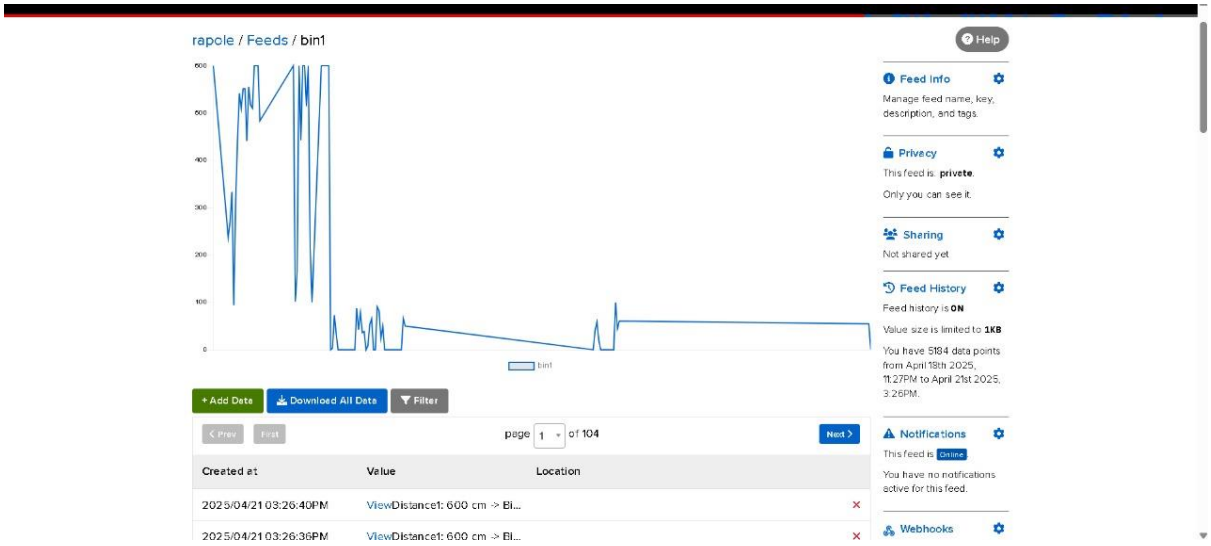


Fig: the above figure represents the feed data of bin1 which is sent form edge node

Fig: the above figure represents the feed data of bin2 which is sent form edge node

```
[Running] python -u "c:\Users\c.naga sai varun\Desktop\iotnew\app.py"
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 111-810-064
```

Fig: The above figure shows the local server is hosted on http://127.0.0.1:5000

Fig: Shows the output of UI which shows the latest bin fill level and average of fill percentage

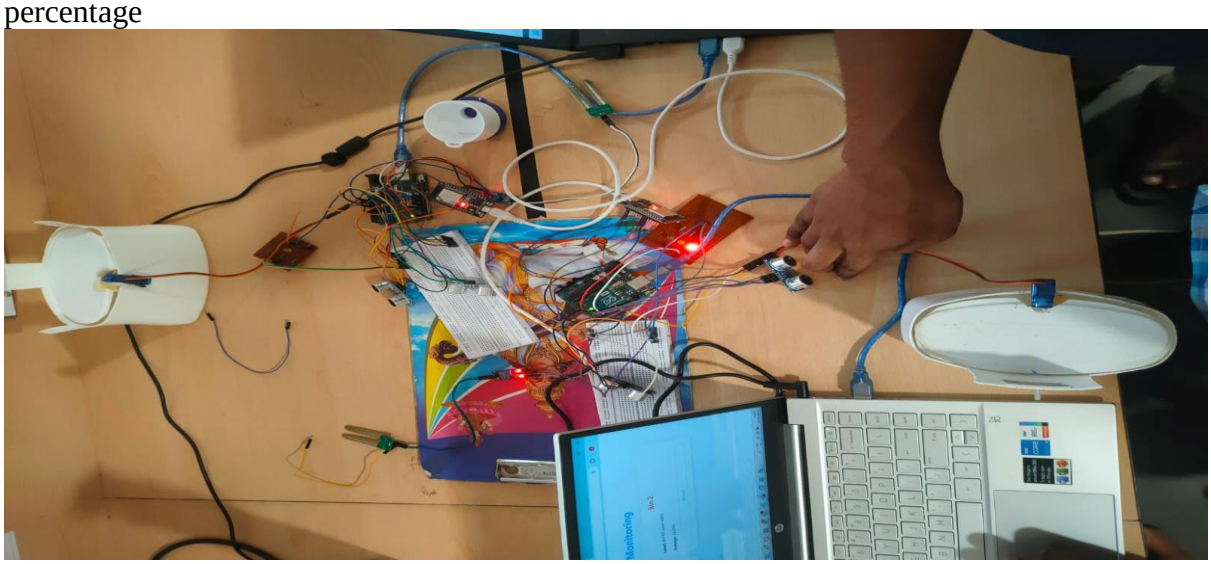


Fig: The above figure indicates the Hardware implementation of project

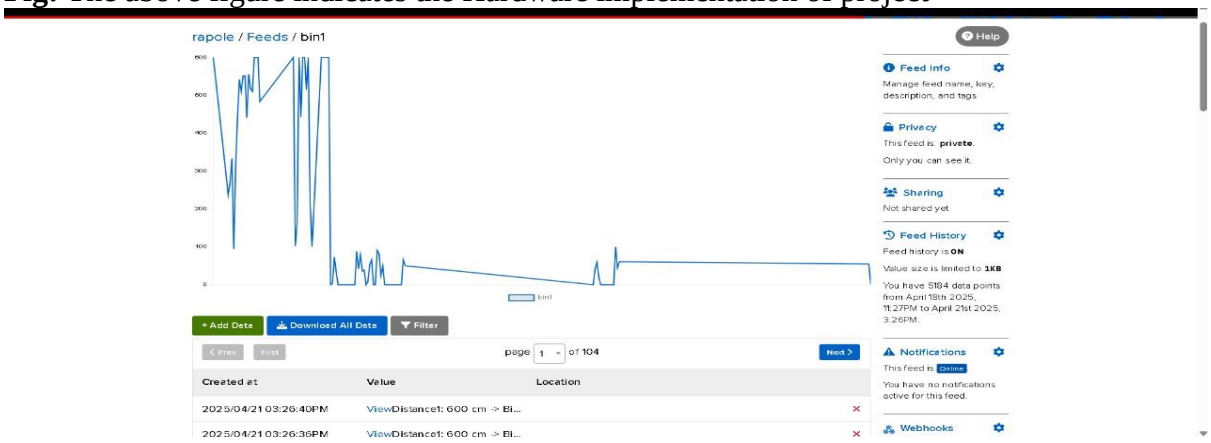


Fig: Database output for bin1

Fig: Data Base output for bin2

Analysis:

This methodology effectively combines local data processing with cloud integration for a smart waste management system. By utilizing ESP-NOW communication for low-latency, peer-to-peer interactions between the end nodes and edge node, the system reduces power consumption and network dependency. The edge node performs essential data filtering and protocol conversion, optimizing cloud bandwidth by sending only relevant data. The use of MQTT for communication ensures lightweight, real-time data transfer to Adafruit IO for visualization. Overall, the approach maximizes efficiency, scalability, and real-time responsiveness in a smart city environment.

Chapter10: Conclusion

- Smart Waste Management System Using IOT provides an automated, efficient solution for monitoring and managing waste in real-time.
- Edge Node (ESP32) collects and filters data from End Nodes (sensors), ensuring accurate data transmission and processing.
- ESP-NOW protocol enables low-power, short-range communication between End Nodes (waste bins) and the Edge Node.
- MQTT protocol is used for seamless data transfer from the Edge Node to the Adafruit IO cloud platform for storage and visualization.
- A local server (using Flask or Django) serves as an intermediary, retrieving data from Adafruit IO and displaying it on a web-based user interface.
- The web interface allows users to monitor real-time bin fill levels, track historical data, and view average values of multiple bins.
- Visual alerts (e.g., blinking LEDs) are triggered when bins reach full capacity, enhancing waste management efficiency.
- The system supports real-time monitoring of waste bin statuses, making it ideal for smart cities, industrial spaces, and residential areas.
- The low-power, wireless communication setup makes the system sustainable and suitable for long-term deployment.
- This approach reduces manual intervention in waste management and promotes better environmental sustainability.
- The system automates waste segregation, provides timely alerts, and helps in optimizing waste collection schedules.

Chapter11: References

[1] M. A. Al Mamun, M. A. Hannan, A. Hussain and H. Basri, "Wireless Sensor Network Prototype for Solid Waste Bin Monitoring With Energy Efficient Sensing Algorithm," *IEEE Sensors Journal*, vol. 15, no. 8, pp. 4275–4281, Aug. 2015.

[2] S. Longhi, D. Marzioni, E. Alidori, G. Di Buò, M. Prist, M. Grisostomi, and M. Pirro, "Solid waste management architecture using wireless sensor network technology," in *Proc. 5th Int. Conf. New Technol. Mobility and Security (NTMS)*, Istanbul, Turkey, May 2012, pp. 1–5.

[3] R. Kanchan, K. B. Mahajan, and R. S. Sangle, "RFID Based Garbage and Waste Collection System," *Int. J. Eng. Trends Technol.*, vol. 9, no. 10, pp. 456–460, Mar. 2014.

[4] S. Islam and F. Wahid, "Smart Waste Bin Management System using IoT and Cloud Technology," in *Proc. 4th Int. Conf. Electrical Engineering and Information & Communication Technology (iCEEiCT)*, Dhaka, Bangladesh, Sep. 2018, pp. 314–319.

[5] S. Thakker and S. Narayanamoorthy, "Smart and Wireless Waste Management," *Int. J. Recent Innov. Trends Comput. Commun.*, vol. 3, no. 4, pp. 123–126, Apr. 2015.

[6] F. Folianto, Y. S. Low, and W. L. Yeow, "Smartbin: Smart Waste Management System," in *Proc. IEEE TENSYMP*, Kuala Lumpur, Malaysia, Apr. 2015, pp. 1–5.

[7] L. A. Guerrero, G. Maas, and W. Hogland, "Solid waste management challenges for cities in developing countries," *Waste Management*, vol. 33, no. 1, pp. 220–232, Jan. 2013.