EAST WEST UNIVERSITY

Department of Computer Science and Engineering

Course Title: Internet Of Things

Code: CSE406

Section: 1

LAB-04

Submitted To:

Dr. Raihan Ul Islam

Associate Professor

Department of Computer Science & Engineering

Submitted by

Name: Maisha Rahman

ID: 2022-1-60-371

Date of Submission

10.08.2025

Lab-04

1. Introduction:

Mesh networking is a decentralized communication method in which each device, known as a node, can connect directly to other nodes without the need for a central router. This enables self-healing capabilities and extended communication range through multi-hop routing. In this lab, NodeMCU ESP8266 microcontrollers are used alongside the painlessMesh library to create a mesh network. The library manages node connections, routing, and time synchronization automatically, allowing for both broadcast and direct messaging between nodes. The lab starts with a basic broadcasting setup, progresses to understanding callback messages, and then modifies the code to implement targeted communication between specific nodes. Finally, multi-hop routing behavior is observed and analyzed.

2. Objectives:

The primary goals of this lab are to:

1. Set up a mesh network using painlessMesh on ESP8266 devices.
2. Demonstrate broadcasting messages across multiple nodes.
3. Explain the key network event callbacks in painlessMesh.
4. Modify the code to enable direct messaging to a specific node.
5. Observe and analyze multi-hop routing based on signal strength in a mesh topology.

3. Requirements (Hardware and Software):

- 4–6 NodeMCU ESP8266 boards (minimum of 3 for multi-hop demonstration).
- Micro-USB cables for programming and power supply.
- Computers with Arduino IDE (version 1.8 or later).
- painlessMesh library (:install via Arduino Library Manager).
- Serial monitor (built into Arduino IDE).
- Optional: Breadboards, LEDs, or sensors for visual feedback.

4. Prerequisites:

- Basic knowledge of Arduino programming and ESP8266 board setup.
- ESP8266 board support installed in Arduino IDE:

  Added this to Boards Manager URLs:

  https://arduino.esp8266.com/stable/package_esp8266com_index.json

- All NodeMCU boards must have the same Wi-Fi credentials in code:

  MESH_PREFIX, MESH_PASSWORD, MESH_PORT.

5. Background:

A mesh network is a decentralized communication system where each node (device) can send and receive data directly with others or relay data through intermediate nodes. Unlike a star topology that depends on a central hub, mesh networks improve reliability and coverage through multiple communication paths. The painlessMesh library simplifies mesh network creation on ESP8266/ESP32 devices by automatically handling:

- Node connections and routing
- Time synchronization across the network
- Dynamic topology updates when nodes join, leave, or move

In painlessMesh, communication can occur in two ways:

1. **Broadcast Messaging:** Sends a message to all nodes in the mesh.
2. **Single (Direct) Messaging:** Sends a message to one specific node identified by its unique Node ID.

Routing decisions are based on a dynamic topology that considers signal strength (RSSI) to determine optimal paths. If direct communication is not possible, messages are forwarded through intermediate nodes (multi-hop routing). Additionally, painlessMesh supports time synchronization, allowing all nodes to maintain a shared clock reference, which is essential for coordinated tasks. In this lab, the provided base code broadcasts a "Hello" message from each node at random intervals (1–5 seconds) and logs received messages and network events to the serial monitor.

6. Lab Tasks:

Task 1: Explain the Meanings of Different Messages

Objective:

Interpret and explain different callback messages from the painlessMesh library based on serial monitor observations and documentation.

Procedure:

1. Upload and run the provided base broadcasting code on all NodeMCU devices.
2. Open the Arduino Serial Monitor for each node to observe real-time network events.
3. Identify and document the meaning of the following callbacks:

New Connection: Triggered when a new node joins and establishes a direct connection with the current node.

Example Log:

```
--> startHere: New Connection, nodeId = [ID]
```

Connection Change: Triggered when network topology changes, such as nodes joining/leaving or link status changes. Updates routing tables.

Example Log:

```
Changed connections
```

Adjusted Time: Triggered when the node synchronizes its clock with the mesh's global time.

Example Log:

```
Adjusted time [time]. Offset = [offset]
```

Expected Output:

- Clear understanding of how painlessMesh notifies about network changes and synchronization.
- Serial monitor screenshots showing each event type.

Task 2: Direct Messaging to a Specific Node

Objective:
Modify the code to send a message to one specific node instead of broadcasting.

Procedure:

1. Identify the **target node ID** by reading it from the serial monitor logs.

Modify the sendMessage() function to use:

mesh.sendSingle(targetNodeId, msg);

2. Hardcode the targetNodeId or allow input via serial command.
3. Upload the modified code to the sender node.
4. Verify that the message appears only in the target node's serial monitor.

5. Implement a check using:

if (mesh.isConnected(targetNodeId)) { ... } to handle cases where the target node is not available.

Expected Output:

- Direct message displayed only on the target node's monitor.
- No broadcasted messages to other nodes.

Task 3: Demonstrate Multi-Hop Direct Messaging Based on Signal Strength

Objective:

Demonstrate that painlessMesh can route direct messages through intermediate nodes automatically.

Procedure:

1. Arrange at least 3 nodes (Node A, Node B, Node C) so that:
   - Node A and Node C cannot directly communicate.
   - Node B is in between and can communicate with both.
2. Use the modified direct messaging code from Task 2 to send a message from Node A to Node C.

3. Enable debug messages to observe routing:

mesh.setDebugMsgTypes(CONNECTION | COMMUNICATION);

4. Monitor serial logs to confirm that the message passes through Node B.

5. Change node positions to observe routing changes when signal strengths vary.

6.Record cases where routing adapts or fails due to weak connections.

Expected Output:

- Serial logs showing multi-hop routing.
- Demonstrated adaptability when node positions change.
- Explanation of how painlessMesh builds its routing tree based on RSSI and connection quality.

GitHub Link:

https://github.com/RahMaisha/Mesh-Networking-with-NodeMCU-ESP8266-and-painlessMesh.git

Reference:

ESP-MESH with ESP32 and ESP8266: Getting Started | Random Nerd Tutorials