IoT Lab: Mesh Networking with NodeMCU ESP8266 and painlessMesh

Lab Overview

This lab introduces students to mesh networking in IoT applications using NodeMCU ESP8266 microcontrollers and the painlessMesh library. Mesh networks allow devices to communicate directly with each other without a central router, enabling self-healing and extended range through multi-hop routing. Students will start with a basic broadcasting example, observe network events, and modify the code to implement targeted messaging. The lab emphasizes understanding callback messages, code modification for direct communication, and observing routing behavior in a multi-node setup.

Learning Objectives

By the end of this lab, students will be able to:

- Set up a mesh network using painlessMesh on ESP8266 devices.
- Demonstrate message broadcasting across multiple nodes.
- Explain key network event callbacks in painlessMesh.
- Modify code to send messages directly to specific nodes.
- Observe and analyze multi-hop routing based on signal strength in a mesh topology.

Materials Required

- 4–6 NodeMCU ESP8266 boards (minimum 3 for multi-hop demonstration).
- Micro-USB cables for programming and power.
- Computers with Arduino IDE installed (version 1.8 or later).
- painlessMesh library (install via Arduino Library Manager: Search for "painlessMesh" by Coopdis).
- Serial monitor tools (built into Arduino IDE).
- Optional: Breadboards, LEDs, or sensors for visual feedback (e.g., to blink on message receipt).

Prerequisites

- Basic knowledge of Arduino programming and ESP8266 setup.

- Install the ESP8266 board support in Arduino IDE (via Boards Manager: Add "https://arduino.esp8266.com/stable/package_esp8266com_index.json" to Additional Boards Manager URLs).
- Ensure all NodeMCU boards are flashed with the same Wi-Fi credentials in the code (MESH_PREFIX, MESH_PASSWORD, MESH_PORT).

Background

Mesh networks are decentralized systems where nodes relay data for each other, improving reliability and coverage. The painlessMesh library simplifies this for ESP8266/ESP32 by handling connections, routing, and time synchronization automatically. It uses callbacks to notify about network events:

- Messages are sent via broadcast (to all nodes) or single (to one specific node).
- Routing is based on a dynamic topology, considering factors like signal strength to choose optimal paths.
- Time synchronization ensures coordinated actions across the mesh.

The provided code (below) sets up a basic mesh where each node broadcasts a "Hello" message at random intervals (1–5 seconds). Received messages and network events are printed to the serial monitor.

Lab Tasks

Complete the following tasks individually or in groups. Document your observations, code changes, and explanations in a lab report. Include screenshots of serial outputs and diagrams of your node topology.

Task 1: Explain the Meanings of Different Messages

Based on observations from the serial monitor and painlessMesh documentation (refer to the library's GitHub or examples), explain the following callback messages in your own words. Provide examples from your serial logs:

- **New Connection**: This callback is triggered when a new node joins the mesh and establishes a direct connection with the current node. It indicates the mesh is expanding. Example log: "--> startHere: New Connection, nodeId = [ID]".
- **Connection Change**: This occurs when the overall network topology changes, such as a node joining, leaving, or a link breaking/forming. It prompts the mesh to update its routing table. Example log: "Changed connections".

- **Adjusted Time**: This callback fires when the node's internal clock is synchronized with the mesh's global time (using NTP-like offsets). It ensures all nodes share a common time reference for timed tasks. Example log: "Adjusted time [time]. Offset = [offset]".

Task 2: Update the Code to Directly Send a Message to a Specific Node

Modify the provided code to send a message directly to one specific node instead of broadcasting to all. Use mesh.sendSingle(targetNodeId, msg) where targetNodeId is the ID of another node (hardcode it or add input via serial).

- Replace or add to the sendMessage() function.
- Test by sending a custom message (e.g., "Direct hello from node [ID]") and verify it appears only on the target node's serial monitor.
- Handle cases where the target node is not connected (use mesh.isConnected(nodeId) if needed).

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

With at least 3 nodes, configure the setup so that not all nodes are in direct range (e.g., place Node A and Node C far apart, with Node B in between).

- Use the modified code from Task 2 to send a direct message from Node A to Node C.
- Observe that the message routes through Node B (painlessMesh automatically handles hops based on signal strength and routing tables).
- Monitor serial outputs or enable debug messages (uncomment mesh.setDebugMsgTypes(...) with additional flags like CONNECTION | COMMUNICATION) to trace the path.
- Experiment: Move nodes to change signal strengths and note how routing adapts (e.g., weaker signals may cause rerouting or failures).
- Explain in your report how painlessMesh determines hops (hint: It builds a routing tree considering RSSI/signal quality).

Lab Report Guidelines

- Include introduction, procedure summary, description of each tasks, code snippets, serial logs, and conclusions.

- Discuss advantages of mesh over star topologies (e.g., resilience) and potential applications (e.g., smart homes, sensor networks).
- Submit by [due date].

This lab builds foundational skills in IoT networking—extend it by adding sensors for real data transmission!