Project Report: painlessMesh Network

Md Ashik
2020-3-60-032
Project: A detailed exploration of painlessMesh callbacks, direct messaging protocols, and the principles of multi-hop routing
Task 1:Mesh Callback Messages

Observations and Explanations

1. **New Connection**
   - **Explanation:** This callback is a highly specific event, triggered only on the local node when it successfully establishes a direct, physical-layer link with a new neighbor. This is the foundational event for mesh formation, representing a successful "handshake." When this occurs, the new neighbor's unique nodeId is added to the local node's list of direct connections, which is the first step in building its routing table.
   - **Example from Serial Log:**
     --> startHere: New Connection, nodeId = 3486689028
   - **Analysis:** This log is not just a notification; it's a confirmation that a new, direct communication path has been opened. The local node now knows it can send data to 3486689028 in a single transmission without needing a relay.
2. **Changed Connections**
   - **Explanation:** This is a more general, higher-level callback that signifies a change in the **entire network's topology**. It is triggered on *all* nodes in the mesh whenever a node joins, a node leaves (either gracefully or by timing out), or a direct link between any two nodes is broken. Its purpose is to signal that every node's internal "map" of the network is potentially outdated and must be recalculated to ensure mesh stability and find the most efficient routes for messages.
   - **Example from Serial Log:**
     Changed connections
   - **Analysis:** Receiving this callback is a cue for the node to update its complete list of sub-connections (the entire routing tree). This self-healing mechanism ensures that if a path becomes unavailable, the mesh can autonomously reroute future messages through alternative nodes.
3. **Adjusted Time**

- **Explanation:** This callback confirms that the local node's internal clock has been synchronized with the mesh's master time. painlessMesh autonomously elects one node to act as the "master" time source. Other nodes then periodically request the time, calculate the round-trip delay, and apply an offset to their local clocks. This distributed timekeeping is critical for timestamping events, coordinating scheduled tasks across different nodes, and ensuring that timeouts and other time-sensitive operations are consistent throughout the network.
- **Example from Serial Log:**
  Adjusted time 1723404732. Offset = -3

- **Analysis:** The offset value is the calculated correction in seconds. A negative offset means the local clock was running fast compared to the mesh time. This process ensures a unified temporal reference frame for the entire distributed system.

Task 2: Implementing Direct Messaging

The objective was to modify the firmware to send a message to a single, specific node using mesh.sendSingle(), contrasting its behavior with the default mesh.sendBroadcast() used in the tutorial's basic example.

Code Modification

The sendMessage() function from the basic example was updated to target a specific, hardcoded nodeId. Crucially, a conditional check using mesh.isConnected(targetNodeId) was added. This check queries the node's routing table to see if a path to the target exists before attempting to send the message, preventing unnecessary transmissions.

```
// Define the specific Node ID you want to send a message to.
// This ID is discovered dynamically when nodes connect.
uint32_t targetNodeId = 3486689028;

void sendMessage() {
  String msg = "Direct message from Node ";
  msg += mesh.getNodeId();

  // First, verify that a route to the target node exists in the mesh.
  if (mesh.isConnected(targetNodeId)) {
    // If a route exists, send the message directly. The mesh handles routing.
```

```
    mesh.sendSingle(targetNodeId, msg);
    Serial.printf("--> Sent direct message to %u\n", targetNodeId);
  } else {
    // If the target is not in the routing table, inform the user and save resources.
    Serial.printf("--> Target node %u not found. Message not sent.\n", targetNodeId);
  }
}
```

Verification of Results

The code was tested with two nodes: a sender (ID: 2484433821) and the designated target (ID: 3486689028).

- **Sender Node (ID: 2484433821) Serial Log:**
  --> Sent direct message to 3486689028

- **Target Node (ID: 3486689028) Serial Log:**
  Received from 2484433821: Direct message from Node 2484433821

Analysis: The logs confirm the successful implementation of unicast (one-to-one) messaging. Unlike sendBroadcast, which floods the message to every node, sendSingle creates a targeted packet that is only processed by the intended recipient. In a two-node network, this demonstrates the fundamental difference in command intent. In a larger network, this method would be significantly more efficient, reducing overall network traffic and saving processing power on nodes that are not part of the communication.

Task 3: Infeasibility of Multi-Hop Messaging with Two Nodes

Task 3 required demonstrating multi-hop routing, where a message is relayed through an intermediate node to reach a destination that is out of the sender's direct wireless range.

Reason for Infeasibility

This task is **impossible to complete with only two nodes**.