A logo for a university

AI-generated content may be incorrect.

**Mesh Networking with NodeMCU ESP8266 and painlessMesh**

**Course: CSE 406**

**Section: 1**

**Lab: 4**

**Submitted To: Dr. Raihan Ul Islam**

**Associate Professor, CSE Dept.**

**Submitted by: Nawreen Islam**

**ID: 2021-3-60-052**

**1. Introduction**

In this lab, we explored how to create a self-healing, decentralized **mesh network** using **NodeMCU ESP8266** boards and the **painlessMesh** library.  
Unlike traditional Wi-Fi setups that rely on a central router, a mesh network lets each device communicate directly with others, passing messages along through intermediate nodes when needed.  
The focus was on two key skills:

1. Understanding and interpreting the main **network event callbacks** from painlessMesh.
2. Modifying the code to send messages directly to one specific node instead of broadcasting to all.

This hands-on experience builds the foundation for more complex IoT systems where reliability, adaptability, and range matter — such as smart homes, distributed sensors, or emergency communications.

**2. Summary of Procedure**

1. Installed **ESP8266 Board Support** and the **painlessMesh** library in Arduino IDE.
2. Connected **three NodeMCU boards** via USB for programming.
3. Flashed all boards with the same mesh credentials (MESH\_PREFIX, MESH\_PASSWORD, MESH\_PORT).
4. For **Task 1**, ran the basic broadcast example and monitored the **Serial Monitor** for callback messages.
5. For **Task 2**, modified the sendMessage() function to use mesh.sendSingle() for targeted communication.
6. Verified that the direct messages appeared **only** on the intended recipient’s serial output.

**3. Task 1 — Understanding Callback Messages**

When running the example code, painlessMesh displayed several types of events in the Serial Monitor. Here’s what they mean in plain language:

| **Event** | **What It Means** | **Example Output** |
| --- | --- | --- |
| **New Connection** | A new device has joined the mesh and is now directly connected to this node. This means the mesh has expanded. | --> startHere: New Connection, nodeId = 324578912 |
| **Connection Change** | The network layout changed — maybe a device joined, left, or a connection link changed. The mesh updates its internal map to keep communication flowing. | Changed connections |
| **Adjusted Time** | The node’s clock synced with the mesh’s global time, keeping all devices on the same schedule. | Adjusted time 17234. Offset = -23 |

**Task 2 — Direct Messaging Code:**

#include "painlessMesh.h"

#define MESH\_PREFIX "MeshNetwork"

#define MESH\_PASSWORD "meshpassword"

#define MESH\_PORT 5555

Scheduler userScheduler;

painlessMesh mesh;

uint32\_t targetNodeId = 298374623; // Replace with your actual target node ID

// Send message directly to a node

void sendMessage() {

String msg = "Direct hello from node " + String(mesh.getNodeId());

if(mesh.isConnected(targetNodeId)) {

mesh.sendSingle(targetNodeId, msg);

Serial.println("Sent direct message to node " + String(targetNodeId));

} else {

Serial.println("Target node not connected!");

}

}

// Task for sending messages periodically

Task taskSendMessage(TASK\_SECOND \* 5, TASK\_FOREVER, &sendMessage);

void receivedCallback(uint32\_t from, String &msg) {

Serial.printf("Received from %u msg=%s\n", from, msg.c\_str());

}

void newConnectionCallback(uint32\_t nodeId) {

Serial.printf("--> New Connection, nodeId = %u\n", nodeId);

}

void changedConnectionCallback() {

Serial.println("Changed connections");

}

void nodeTimeAdjustedCallback(int32\_t offset) {

Serial.printf("Adjusted time %u. Offset = %d\n", mesh.getNodeTime(), offset);

}

void setup() {

Serial.begin(115200);

mesh.setDebugMsgTypes(ERROR | STARTUP | CONNECTION); // Optional

mesh.init(MESH\_PREFIX, MESH\_PASSWORD, MESH\_PORT);

mesh.onReceive(&receivedCallback);

mesh.onNewConnection(&newConnectionCallback);

mesh.onChangedConnections(&changedConnectionCallback);

mesh.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);

userScheduler.addTask(taskSendMessage);

taskSendMessage.enable();

}

void loop() {

mesh.update();

}

**Conclusion**

This lab clearly showed how painlessMesh makes it simple to create a **self-healing, multi-device network** with no central router.  
From Task 1, I learned how to interpret important network events like new connections, topology changes, and time synchronization.  
From Task 2, I successfully modified the code to send a message directly to a single node and verified that it worked as expected.

**Why mesh over star topology?**

* Mesh networks don’t have a single point of failure — if one link breaks, the network can find another path.
* Range can be extended using intermediate nodes.

**Potential applications:**

* Smart lighting systems where each bulb relays commands.
* Agricultural monitoring over large fields.
* Disaster-relief communications where infrastructure is damaged.