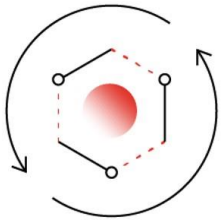
ESP-MESH



The ESP-MESH protocol enables communication between multiple ESP8266 and ESP32 microcontrollers on a local area network. The ESP-MESH network provides single (one-to-one), broadcast (one-to-many) and mesh (many-to-many) communication between devices. All ESP-MESH nodes do not have to have direct contact with all other nodes, as the ESP-MESH protocol ensures that a message reaches the destination node by routing the message through the network (see Figure 1). The node topology is updated every three seconds and the network self-organises the connection of a new node to the network. Messages are sent as strings, with variable and value pairs in JSON format. Details of ESP-MESH are available at docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/mesh.html.

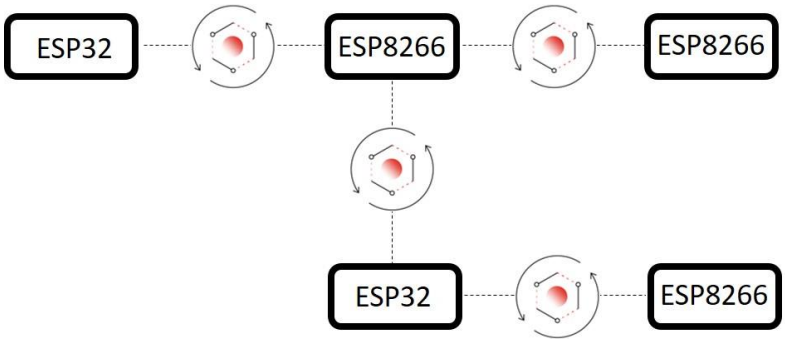


Figure 1 *ESP-MESH topology*

The *painlessMesh* library by Coopdis *et al* is available in the Arduino IDE. The *painlessMesh* library references the *ArduinoJson* and *TaskScheduler* libraries plus the *ESPAsyncTCP* or *AsyncTCP* library for an ESP8266 and ESP32 microcontroller, respectively, and the corresponding `#include<library.h>` instructions are not required. [gitlab.com/painlessMesh/painlessMesh. Coopdis, Scotty Franzyshen Edwin van Leeuwen, Germán Martín,Maximilian Schwarz and Doanh Doanh]

A message is broadcast to all nodes in the ESP-MESH network with the instructions:

```
String sendMsg = "message";                    // message
mesh.sendBroadcast(sendMsg);                   // broadcast to all nodes
```

The message is received and displayed on the Serial Monitor with the identity of the sender node using the instructions:

```
mesh.onReceive(&recvMessage);
void recvMessage(uint32_t sender, String &recvMsg)     // sender is node identity
{
  Serial.print(recvMsg);                       // display message and
```

```
Serial.print(" from ");Serial.println(sender);          // sender node identity
```

A message is transmitted to a specific node identity by the `sendSingle` instruction. If the node identity is defined as a string, then the C++ *strtoul* instruction converts the string to a 32-bit integer with base 10. For example, a message is transmitted to *node* with the instructions:

```
String node = "3943544243";                             // node identity
uint32_t receiver = strtoul(node.c_str(),NULL,10);      // converted to a number
mesh.sendSingle(receiver, sendMsg);                     // transmit message
```

An example ESP-MESH network consists of three nodes, with the node 1 microcontroller setting the states, using switches, of a relay and an LED connected to the node 2 and node 3 microcontrollers, respectively (see Figure 2).

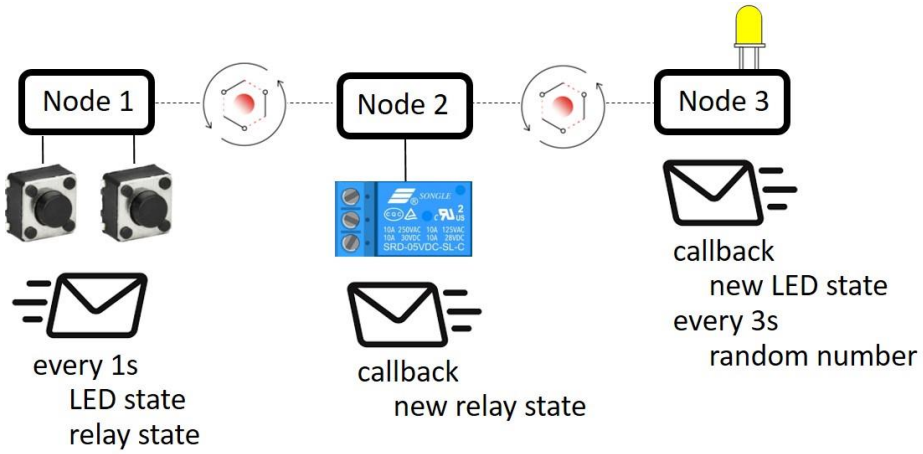


Figure 2 *ESP-MESH layout and schedule*

Every second, node 1 broadcasts to nodes 2 and 3 a JSON document containing the updated relay and LED states. Node 3 broadcasts at 3s intervals a random number, to simulate sensor data, that is included in the message as *value*. Node 1 immediately receives callback messages from nodes 2 and 3 containing the updated relay and LED states, but only when the relay or LED state is changed, which node 1 displays on the OLED display. Without the callback messages, node 1 would only receive the updated relay and LED states from the message broadcast by node 3 after the set interval of 3s.

Connections for the three microcontrollers are shown in Figure 3. The switches are connected to the WeMos D1 mini development board pins *D3* and *D4*, which have built-in pull-up resistors, so the switches are active *LOW*.

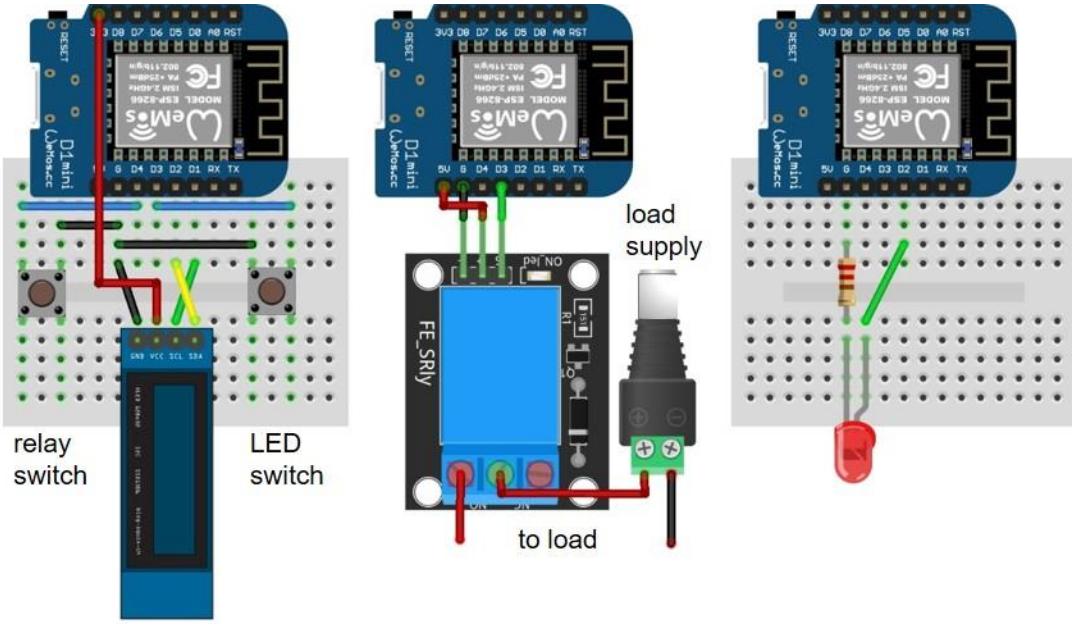


***Figure 3** ESP-MESH*

An example sequence of events is node 1 instructing a change in the LED state and then a change in the relay state. During the sequence, node 3 changes the variable *value*. The received messages by node 1 from nodes 2 and 3 are:

```
from node 3 {"LED":0,"relay":0,"value":83}
from node 3 {"LED":0,"relay":0,"value":74}      node 3 updated value
from node 3 {"LED":1,"relay":0,"value":74}      node 1 changed LED state
from node 3 {"LED":1,"relay":0,"value":78}      node 3 updated value
from node 2 {"LED":1,"relay":1,"value":78}      node 1 changed relay state
```

The first section of Listing 1 includes the ESP-MESH name and password with the default port number of *5555*. The *taskSend* function defines message transmission timing. In the *setup* function, the ESP-MESH functions are declared. The *setDebugMsgTypes* function instruction precedes the *init* function to display messages when the ESP-MESH is established. The *sendMessage* function defines the relay and LED states that are converted from JSON formatted name and value pairs to the message as a string, with the transmission interval defined by the `setInterval` instruction. On receipt of a message from the ESP-MESH, node 1 converts the message string to name and value pairs in the *recvMessage* function and displays the named values on the OLED screen.

Node connection changes to the ESP-MESH network are monitored with the *newConnectionCallback* function, that displays node identities of new connections. The *changedConnectionCallback* function generates a list of ESP-MESH connections with the `SimpleList<uint32_t> list = mesh.getNodeList()` instruction, with the number of

nodes equal to `list.size()`. The identity of the node calling the function to obtain network connection details is obtained with the instruction `getNodeId()`.

For example, when a node is connected to the ESP-MESH network, the following information is displayed:

```
New connection nodeID 2885393143      node identity
number of nodes 2             number of nodes excluding the node calling the function
Connection list
node1 3943544243                      node 1 identity
nodeID 3956052891                     existing node identity
nodeID 2885393143                     new node identity
```

Listing 1 *Microcontroller with switches and OLED (node 1)*

```
#include <painlessMesh.h>                           // include painlessMesh library
String ssid = "meshSSID";                           // ESP-MESH name & password
String password = "meshPass";
int port = 5555;                                    // ESP-MESH port
int LEDswitchPin = D3;
int relaySwitchPin = D4;                            // LED and relay pins
int LEDstate, relayState;
DynamicJsonDocument jsonDoc(1024);                  // JSON document

#include <Adafruit_SSD1306.h>                 // include Adafruit SSD1306 library
int width = 128;                                    // OLED width and height
int height = 32;
Adafruit_SSD1306 oled(width, height, &Wire, -1);

Scheduler scheduler;                                // associate scheduler with library
painlessMesh mesh;                                  // associate mesh with library
void sendMessage();
Task taskSend(TASK_SECOND * 1, TASK_FOREVER, &sendMessage); // task timing of 1s

void setup()
{
  Serial.begin(115200);
  mesh.setDebugMsgTypes(ERROR | STARTUP);           // before init instruction
  mesh.init(ssid, password, &scheduler, port);
  mesh.onReceive(&recvMessage);                    // set recvMessage function
  mesh.onNewConnection(&newConnectionCallback);     // ESP-MESH functions
  mesh.onChangedConnections(&changedConnectionCallback);
  mesh.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);
  scheduler.addTask(taskSend);                      // schedule and enable
  taskSend.enable();                                // send message function

  oled.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  oled.clearDisplay();                              // clear OLED display
  oled.setTextSize(1);                              // characters 6×8 pixels
  oled.setTextColor(WHITE);
  oled.display();                                   // update display
}
```

```
void sendMessage()                                  // function to send message
{                                                   // change LED state
  if (digitalRead(LEDswitchPin) == LOW) LEDstate = 1-LEDstate;
  jsonDoc["LED"] = LEDstate;                        // update JSON document
  if (digitalRead(relaySwitchPin) == LOW) relayState = 1-relayState;
  jsonDoc["relay"] = relayState;
  String sendMsg;                                   // sendMsg defined in function
  serializeJson(jsonDoc, sendMsg);                  // convert to message
  mesh.sendBroadcast(sendMsg);                      // broadcast message
  taskSend.setInterval((TASK_SECOND * 1));          // message timing of 1s
}
                                                    // function to receive message
void recvMessage(uint32_t sender, String &recvMsg)
{
  Serial.print("from ");Serial.print(sender);       // sender identity
  Serial.print("\t");Serial.println(recvMsg);       // Serial display message
  DeserializationError error = deserializeJson(jsonDoc, recvMsg);
  if (error)
  {
    Serial.print("deserializeJson() failed: ");
    Serial.println(error.c_str());
  }
  oled.clearDisplay();                              // update OLED screen
  oled.setCursor(0,0);
  oled.print("LED ");                               // LED as ON or OFF
  if(jsonDoc["LED"].as<long>() == 1) oled.print("ON");
  else oled.print("OFF");
  oled.setCursor(0,8);
  oled.print("relay ");                             // relay as ON or OFF
  if(jsonDoc["relay"].as<long>() == 1) oled.print("ON");
  else oled.print("OFF");
  oled.setCursor(0,16);                             // generated sensor data
  oled.print("value ");oled.print(jsonDoc["value"].as<long>());
  oled.display();
}

void newConnectionCallback(uint32_t nodeID)         // new node connected
{                                                   // display node identity
  Serial.print("New connection nodeID ");Serial.println(nodeID);
}

void changedConnectionCallback()                    // function to list connected nodes
{
  SimpleList<uint32_t> list = mesh.getNodeList();           // number of nodes
  Serial.print("number of nodes ");Serial.println(list.size());
  Serial.println("Connection list");
  Serial.print("node1 ");Serial.println(mesh.getNodeId()); // node 1 identity
  SimpleList<uint32_t>::iterator node = list.begin();
  while (node != list.end())                        // list of nodes
  {
    Serial.print("nodeID ");Serial.println(*node);        // *node is identity
    node++;                                         // increment node
  }
}
```

```
void nodeTimeAdjustedCallback(int32_t offset)     // display synchronised timing
{
  Serial.printf("adjust %u offset = %d\n", mesh.getNodeTime(), offset);
}

void loop()
{
  mesh.update();                                  // handle ESP-MESH
}
```

For node 3, the LED state, included in the message from node 1, is updated in the *recvMessage* function. If the LED state has changed, an acknowledgment or callback message is transmitted only to node 1, the message sender, by the *sendSingle* instruction. To simulate data collection by node 3, a random number between 1 and 99, inclusive, is generated and incorporated in the message broadcast every 3s by the *sendMessage* function.

For node 2, the relay state is updated in the *recvMessage* function, as with node 3. The sketch for node 2 is based on Listing 2 with the relay replacing the LED in the *recvMessage* function. No message is transmitted on a regular basis by node 2, so the *sendMessage* function contains no instructions.

Listing 2 is for the node 3 microcontroller with the first section of the sketch similar to Listing 1 for the node 1 microcontroller, but without the OLED and relay pin definitions. The *setup* function includes the `pinMode` instruction for the LED. The *newConnectionCallback*, *changedConnectionCallback* and *nodeTimeAdjustedCallback* functions are not required by node 2. In the *sendMessage* function, the variable *value*, for the generated random number, is updated and incorporated in the JSON formatted message, which is broadcast every 3s.

Listing 2 *Microcontroller with LED and generated data (node 3)*

```
#include <painlessMesh.h>                         // differences from Listing 1
String ssid = "meshSSID";
String password = "meshPass";
int port = 5555;
int LEDpin = D2;                                  // define LED pin
int LEDstate, oldLEDstate;                        // new and old LED states
DynamicJsonDocument jsonDoc(1024);

Scheduler scheduler;
painlessMesh mesh;
void sendMessage();
Task taskSend(TASK_SECOND * 1 , TASK_FOREVER, &sendMessage);

void setup()
{
  Serial.begin(115200);
  pinMode(LEDpin, OUTPUT);                        // LED pin as OUTPUT
```

```
  mesh.init(ssid, password, &scheduler, port);
  mesh.onReceive(&recvMessage);
  scheduler.addTask(taskSend);
  taskSend.enable();
}

void sendMessage()                                  // function to send message
{
  float value = random(0, 100);                     // generate random number
  jsonDoc["value"] = value;                         // update name and value pair
  String sendMsg;
  serializeJson(jsonDoc, sendMsg);
  mesh.sendBroadcast(sendMsg);
  taskSend.setInterval((TASK_SECOND * 3));                  // message timing of 3s
}

void recvMessage(uint32_t sender, String &recvMsg)
{
  deserializeJson(jsonDoc, recvMsg);
  LEDstate = jsonDoc["LED"];                                // update LED state
  digitalWrite(LEDpin, LEDstate);                           // and LED pin state
  if(LEDstate != oldLEDstate)                               // turn on or off  LED
  {                                                         // if LED state changed
    oldLEDstate = LEDstate;                                 // update old LED state
    mesh.sendSingle(sender, recvMsg);                       // send message to node 1
  }
}

void loop()
{
  mesh.update();
}
```

ESP-MESH and Bluetooth (1)

ESP-MESH enables transmission of messages between Bluetooth devices, such as Android tablets, over a greater range than with only Bluetooth communication. For example, an Android tablet is connected to an ESP32 microcontroller by Bluetooth and messages are transmitted over the ESP-MESH network (see Figure 4). Listings 1 and 2 correspond to the ESP8266 and ESP32 microcontrollers, or node 1 and node 3 respectively, in Figure 4

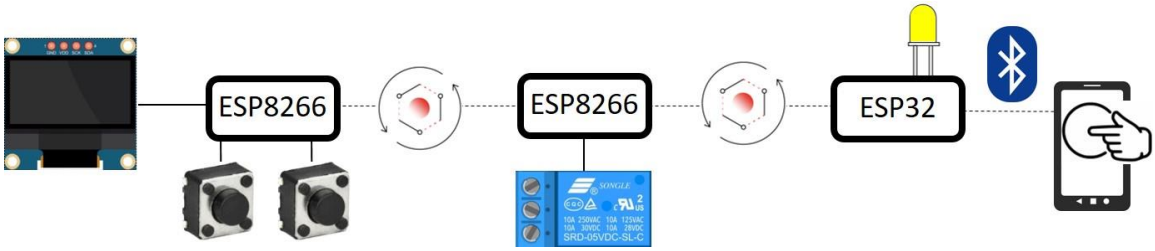


Figure 4 *ESP-MESH and Bluetooth communication (1)*

Bluetooth communication with an ESP32 microcontroller, node 3, requires a Serial Bluetooth connection, which is established with the instructions

```
#include <BluetoothSerial.h>;                    // include Bluetooth library
BluetoothSerial SerialBT;                        // associate SerialBT with library
SerialBT.begin("ESP32 Bluetooth");               // identify Bluetooth device
```

The message from the transmitting Bluetooth device received by the transmitting ESP32 microcontroller is converted to a string and included in the JSON document. The only changes to Listing 2 for the ESP32 microcontroller or node 3 are inclusion of the instruction:

```
if(SerialBT.available()) BTmsg = SerialBT.readString();
```

in the *loop* function to detect the received message, with *BTmsg* defined as a string, and replacing the instruction `jsonDoc["value"] = value` with the instruction `jsonDoc["msg"] = BTmsg` in the *sendMessage* function to allocate the received Bluetooth message to the JSON document.

The receiving ESP8266 microcontroller or node 1 converts the received JSON formatted message to a character array, which is displayed on the OLED screen with the instructions:

```
oled.print("msg ");oled.print(jsonDoc["msg"].as<char*>());
```

and replaces the penultimate line in the *recvMessage* function of Listing 1.

ESP-MESH and Bluetooth communication (2)

Communication between several Bluetooth devices, such as Android tablets or mobile phones, independent of a WLAN (Wireless Local Area Network) or an ISP (internet service provider) is achieved with an ESP-MESH network. In an example linear ESP-MESH network, messages are transmitted between the two Android tablets connected to ESP32 microcontrollers by Bluetooth. The left-side tablet transmits, by Bluetooth to the left-side ESP32 microcontroller, text messages or command letters to control the relay and the LED attached to the other microcontrollers. The left-side ESP32 microcontroller or node 1 broadcasts the information with ESP-MESH (see Figure 5). The right-side ESP32 microcontroller or node 3 generates a random number, to simulate sensor data, which is broadcast across the ESP-MESH network as in Figure 5.

There are several Bluetooth communication applications to download from Google Play and the *Bluetooth Terminal HC-05* app by MightyIT is recommended. In the *Bluetooth Terminal HC-05* app, button settings are configured with a long press to enter the *Button Name* and the corresponding ASCII Command letter. For example, *Button Names* of *LED*

and *relay* are configured with the command letters of *L* and *R*. Pressing a *Bluetooth Terminal HC-05* app button broadcasts the command letter across the ESP-MESH network and the LED or relay is turned on or off by the connected microcontroller receiving the message.

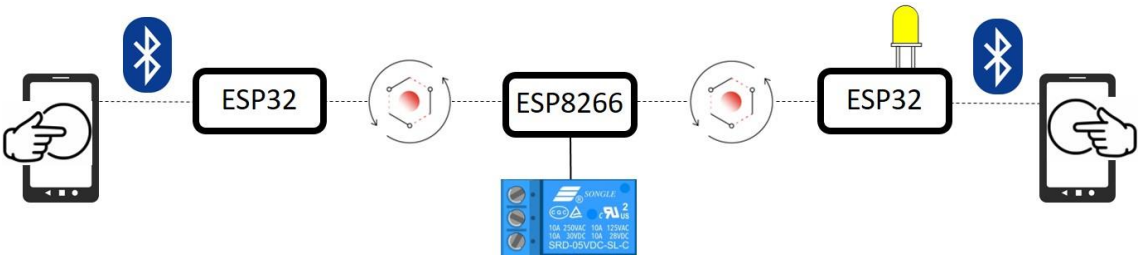


Figure 5 *ESP-MESH and Bluetooth communication (2)*

Listing 3 is for the left-side ESP32 microcontroller or node 1 in Figure 5 to transmit received messages from the Bluetooth connected Android tablet across the ESP-MESH network. The sketch includes instructions for Bluetooth communication with the app on the Android tablet and for transmission over the ESP-MESH network of messages with the other ESP32 microcontrollers. Differences from Listing 1 are commented and highlighted in yellow. Note that the JSON document is reduced to 256 bytes rather than 1024 bytes, due to memory constraints, with the instruction `DynamicJsonDocument jsonDoc(256);`.

A message received, from either Android tablet using Bluetooth communication, in the *loop* function by the corresponding ESP32 microcontroller is broadcast over the ESP-MESH by the *sendMessage* function. The message is then reset in the *sendMessage* function so that it is not repeatedly displayed on the receiving Android tablet. When the message was displayed on the OLED screen in Listing 1, the message was not reset as the last message was always displayed.

A three-character message transmitted by the left-side Android tablet indicates a command message, such as "*L\r\n*", corresponding to the command letter *L* to change the LED state. In the *loop* function of the left-hand ESP32 microcontroller or node 1, the first character of the message controls the *switch...case* function to change the LED or relay state and the corresponding message, such as *LED on* or *LED off* is transmitted by Bluetooth communication to the left-side Android tablet to display the updated LED or relay state.

In contrast, a message received over the ESP-MESH network by either ESP32 microcontroller is compared to the previous message, in the *recvMessage* function, and if different and longer than three characters, then the message is transmitted by Bluetooth communication for display on the corresponding Android tablet.

For simulating display of sensor data, the random number generated by the right-side ESP32 microcontroller or node 3 is displayed on the Serial Plotter of the left-side ESP32 microcontroller or node 1 with the `Serial.println(jsonDoc["value"].as<long>())` instruction in the *recvMessage* function.

Listing 3 *ESP-MESH and Bluetooth communication (node 1)*

```
#include <painlessMesh.h>                       // differences from Listing 1
String ssid = "meshSSID";
String password = "meshPass";
int port = 5555;
int LEDstate = 0, relayState = 0;               // define LED and relay states
DynamicJsonDocument jsonDoc(256);               // reduce JsonDoc to 256

Scheduler scheduler;
painlessMesh mesh;
void sendMessage();
Task taskSend(TASK_SECOND * 1, TASK_FOREVER, &sendMessage);

#include <BluetoothSerial.h>                    // include Bluetooth library
BluetoothSerial SerialBT;                       // associate SerialBT with library
String sendBT, oldBT = "";                      // Bluetooth message
char c;                                         // character for command letter

void setup()
{
  Serial.begin(115200);
  mesh.init(ssid, password, &scheduler, port);
  mesh.onReceive(&recvMessage);
  scheduler.addTask(taskSend);
  taskSend.enable();
  SerialBT.begin("ESP32 left");                 // identify Bluetooth device
}

void sendMessage()
{
  jsonDoc["LED"] = LEDstate;                    // update LED and relay states
  jsonDoc["relay"] = relayState;                // from Bluetooth buttons
  jsonDoc["msg"] = sendBT;                      // message to Android tablet
  String sendMsg;
  serializeJson(jsonDoc, sendMsg);
  mesh.sendBroadcast(sendMsg);
  sendBT = "";                                  // reset message to Android tablet
  taskSend.setInterval((TASK_SECOND * 1));
}

void recvMessage(uint32_t sender, String &recvMsg)
{
  deserializeJson(jsonDoc, recvMsg);
  Serial.print("value ");                       // display value on Serial
  Serial.println(jsonDoc["value"].as<long>());  //      plotter as an integer
  String recvBT = jsonDoc["msg"];               // message from Android
  if(recvBT != oldBT)                           // if a new message, then
  {                                             // transmit to receiving
```

```
    if(recvBT.length() > 3) SerialBT.print(recvBT);     //        Android tablet
    oldBT = recvBT;                                     // update message copy
  }
}

void loop()
{
  mesh.update();
  if(SerialBT.available())                              // new Bluetooth message
  {
    sendBT = SerialBT.readString();                 // read Bluetooth Serial buffer
    c = sendBT[0];                                      // first letter of message
    switch (c)                                          // switch...case on letter
    {
    case 'L':                                           // letter = L (for LED)
      LEDstate = 1-LEDstate;                            // alternate the LED state
      if(LEDstate == 1) SerialBT.println("LED ON");     // display LED state on
      else SerialBT.println("LED OFF");                 // the Android tablet
      break;                                            // end of case
    case 'R':
      relayState = 1-relayState;                        // similarly for the relay
      if(relayState == 1) SerialBT.println("relay ON");
      else SerialBT.println("relay OFF");
      break;
    default: break;                                     // no action, not L or R
    }
  }
}
```

Listing 4 is for the right-side ESP32 microcontroller or node 3 in Figure 5. The sketch includes instructions for Bluetooth communication with the app on the Android tablet. Differences from Listing 2, which are essentially the Bluetooth instruction, are commented and highlighted in grey.

For node 2, the relay state is updated in the *recvMessage* function, as with node 3. The sketch for node 2 is based on Listing 4 with the relay replacing the LED in the *recvMessage* function. No message is transmitted on a regular basis by node 2, so the *sendMessage* function contains no instructions. A Bluetooth device is not connected to the node 2 microcontroller, so instructions related to Bluetooth messages and the Bluetooth library are not required.

Listing 4 *ESP-MESH and Bluetooth communication (node 3)*

```
#include <painlessMesh.h>                       // differences from Listing 2
String ssid = "meshSSID";
String password = "meshPass";
int port = 5555;
int LEDpin = 27;                                // define ESP32 LED pin
int LEDstate, oldLEDstate;
DynamicJsonDocument jsonDoc(256);
```

```
Scheduler scheduler;
painlessMesh mesh;
void sendMessage();
Task taskSend(TASK_SECOND * 1 , TASK_FOREVER, &sendMessage);

#include <BluetoothSerial.h>                    // include Bluetooth library
BluetoothSerial SerialBT;                       // associate SerialBT with library
String sendBT, oldBT = "";                      // Bluetooth message

void setup()
{
  Serial.begin(115200);
  pinMode(LEDpin, OUTPUT);                      // LED pin as OUTPUT
  mesh.init(ssid, password, &scheduler, port);
  mesh.onReceive(&recvMessage);
  scheduler.addTask(taskSend);
  taskSend.enable();
  SerialBT.begin("ESP32 right");                // identify Bluetooth device
}

void sendMessage()
{
  float value = random(0, 100);
  jsonDoc["value"] = value;                     // update name and value pairs
  jsonDoc["msg"] = sendBT;                      // for value and BT message
  String sendMsg;
  serializeJson(jsonDoc, sendMsg);
  mesh.sendBroadcast(sendMsg);
  sendBT = "";                                  // reset message to Android tablet
  taskSend.setInterval((TASK_SECOND * 3));
}

void recvMessage(uint32_t sender, String &recvMsg)
{
  deserializeJson(jsonDoc, recvMsg);
  String recvBT = jsonDoc["msg"];                       // message from Android
  if(recvBT != oldBT)                                   // if a new message, then
  {                                                     //     transmit to receiving
    if(recvBT.length() > 3) SerialBT.print(recvBT);     //         Android tablet
    oldBT = recvBT;                                     // update message copy
  }
  LEDstate = jsonDoc["LED"];
  digitalWrite(LEDpin, LEDstate);
  if(LEDstate != oldLEDstate)
  {
    oldLEDstate = LEDstate;
    mesh.sendSingle(sender, recvMsg);
  }
}

void loop()
{
  mesh.update();
  if(SerialBT.available())                      // new Bluetooth message
    sendBT = SerialBT.readString();             // read Bluetooth Serial buffer
}
```