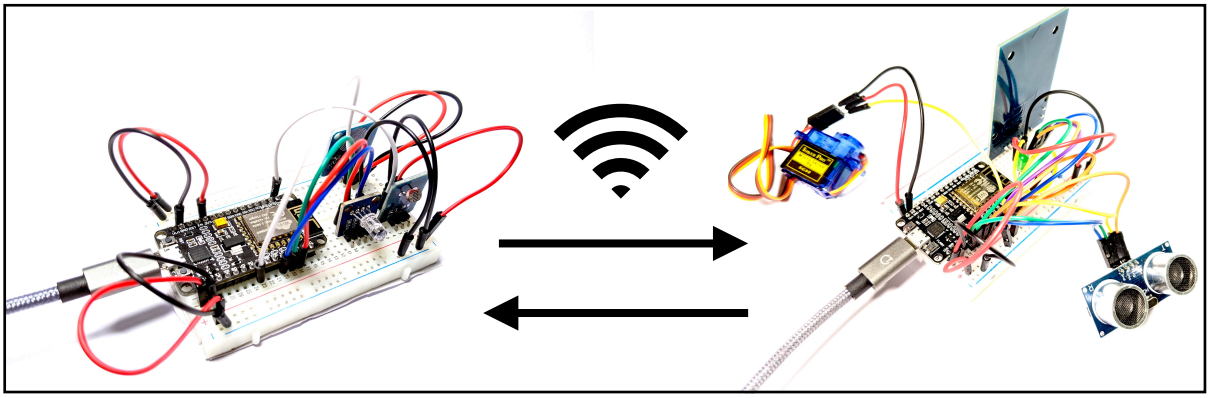
Mesh Networking

A single wireless local area network

Project description:

In this project, you will explore and implement one final communication protocol: a local mesh network. Precisely, you will work with the smart chair and the smart door circuits to setup a local area network protocol. This will allow you to send and receive data wirelessly between your ESP nodes. There is no need for you to build a separate circuit for this exercise. You will use the smart chair circuit and the smart door circuit that you have built during the course and upload a different code sketch.

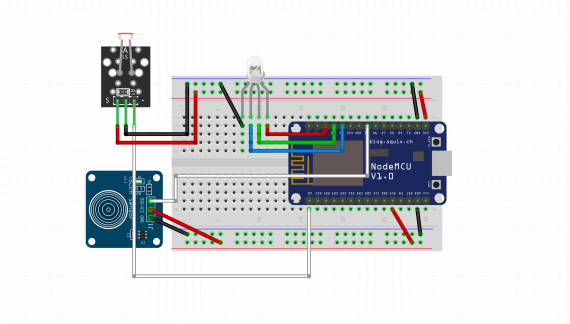


Project objectives:

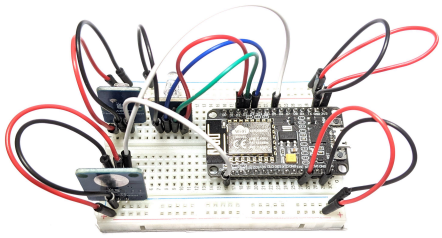
- Create a local area network with more that one ESP node
- Use the PainlessMesh library to setup the network
- Exchange data between two or more ESP nodes in the network

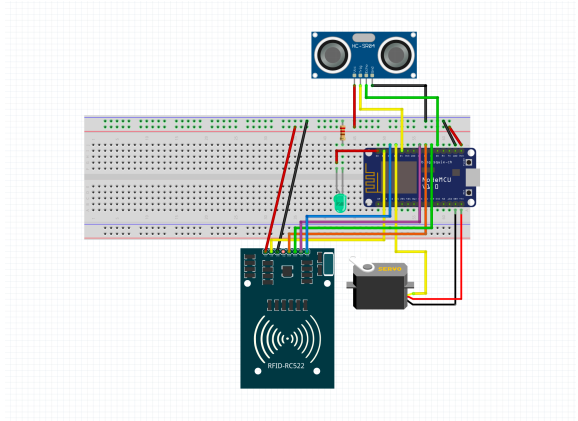
Project components:

Component Reference	Component Quantity	Component Name	Component Description
A	1	Smart Chair circuit	The smart chair circuit which features the photoresistor, the RGB LED, and the touch sensor
B	1	Smart Door circuit	The smart door circuit which features the servo, the green LED, the distance sensor, and the RFID sensor
C	1	Micro USB Cable	A USB cable to power and upload instructions to a microcontroller

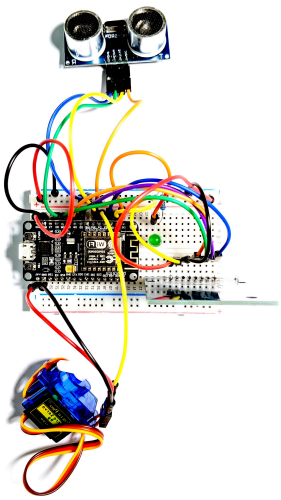


A





B

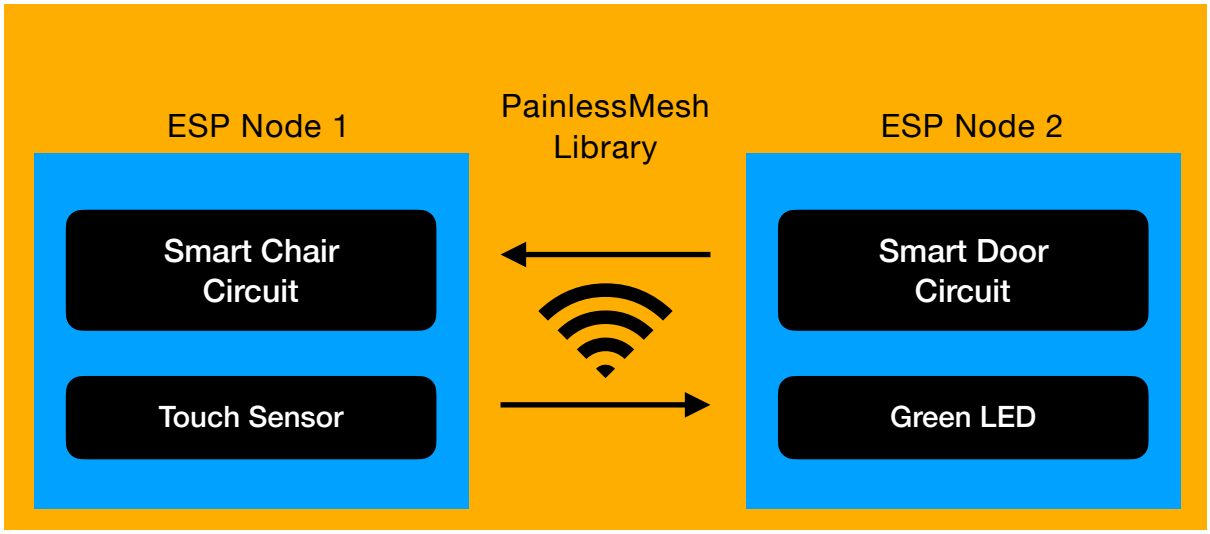


C

Step One
Project Introduction

The idea behind this exercise is for you to practice what you learnt in the video lectures and implement a simple mesh networking protocol.

Here, you will create the following mesh network:



ESP Node 1: This is the smart circuit ESP board which has the touch sensor.

ESP Node 2: This is the smart door ESP board which has the green LED.

PainlessMesh Library: This is the library that allows you to create a local area network between the **ESP Node1** and the **ESP Node 2**.

You will create a local area network on your existing two circuits. The smart chair circuit will broadcast a message to the smart door circuit with the value of the touch sensor. The smart door circuit will receive such message and turn ON, or OFF, the green LED accordingly. The mesh network will allow you to have multiple ESP nodes communicating with each other wirelessly.

Step Two

Installing the PainlessMesh Library

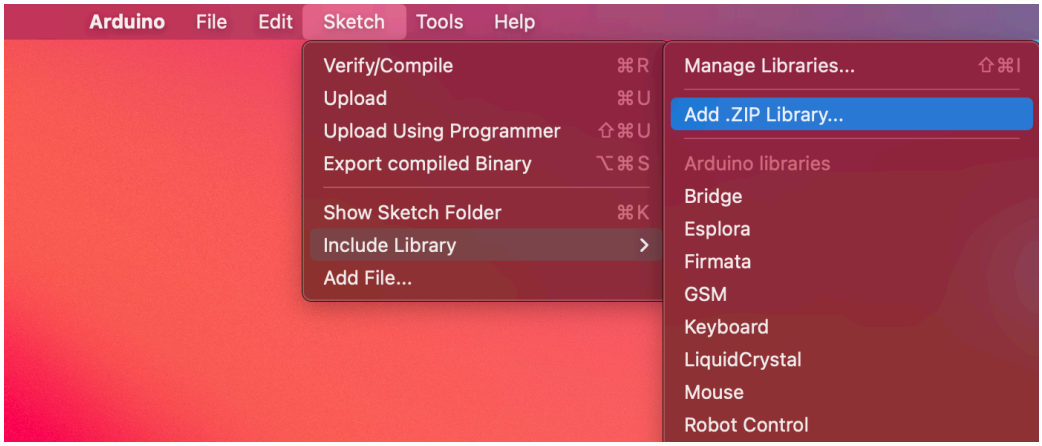
In this section of the exercise, you will install the PainlessMesh Library. This is a library that takes care of the particulars of creating a simple mesh network using ESP8266 and ESP32 hardware.

Unfortunately, the process of installing the library from the Arduino IDE library manager might result in packages conflict.

The safest way to install the library, and its dependencies, is via ZIP import.
We have included all the zip packages that you should import in the exercise reading. Alternatively, you can find and download the ZIP files at the following links:

PainlessMesh (The main library)
https://gitlab.com/painlessMesh/painlessMesh
AsyncTCP (Required)
https://github.com/me-no-dev/AsyncTCP
ESPAsyncTCP (Required)
https://github.com/me-no-dev/ESPAsyncTCP
ArduinoJson (Required)
https://github.com/bblanchon/ArduinoJson
TaskScheduler (Required)
https://github.com/arkhipenko/TaskScheduler

You can install the above zip packages using the Arduino IDE “Add .ZIP Library…” menu as shown below: **Sketch** -> **Include Library** -> **Add .ZIP Library…**



Go ahead and install the **PainlessMesh** library and all the required packages.

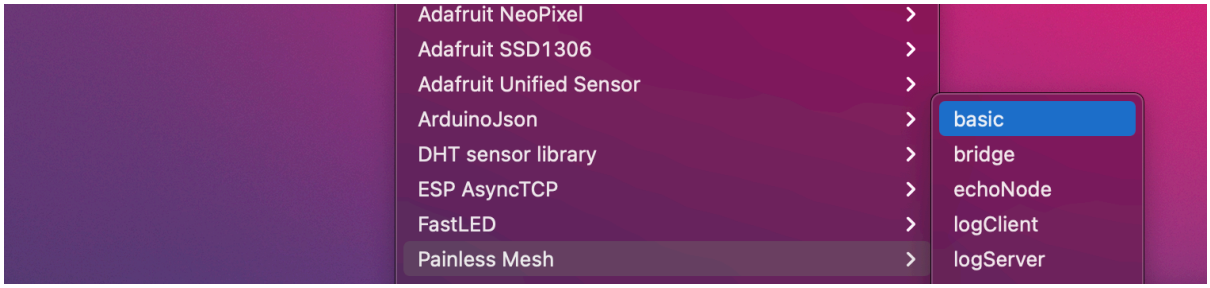
Step Three

Test the library and the example sketch

Now that you have imported the library and all the required packages, let's check that the import was successful. You will do this by running one of the sketch examples that the library provides.

Go ahead and open the following Arduino sketch example:

File > Examples > Painless Mesh > basic



Feel free to save the sketch and rename it to something sensible:
mesh_network_protocol for instance.

This sketch provides the mesh network infrastructure logic where each node in the network sends a message to all other nodes in the network. At the same time, each node in the network listens for incoming messages.

The message is a simple "Hello from node <NodeID>" text that it is broadcasted to all other nodes in the local area network.

Let's first verify that the **PainlessMesh** library and required packages were installed correctly. Compile the example sketch and see if you receive any errors from the Arduino console.

If the compilation is successful, you have correctly install all the packages and you are ready to upload the code to your ESP nodes.

Upload the sketch to both the smart chair circuit and the smart door circuit. Make sure that the network credentials remain the same for both uploads:

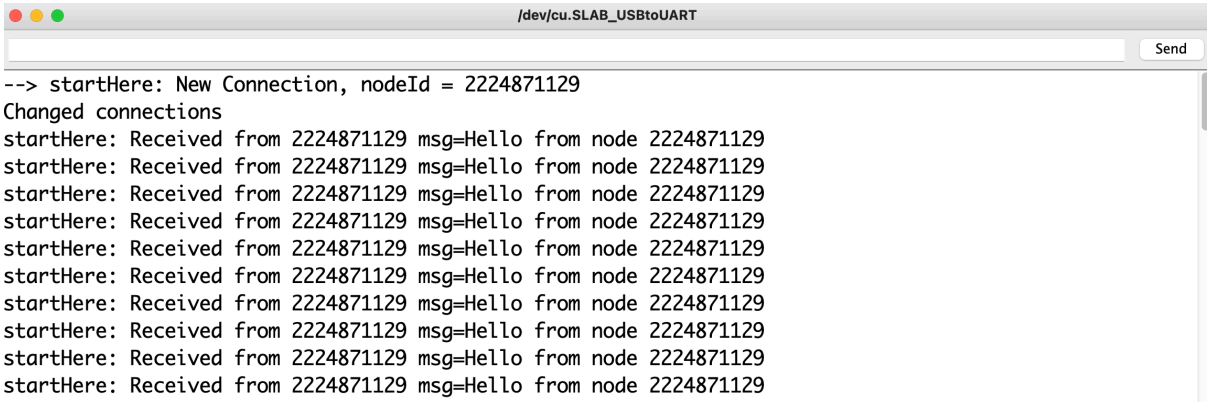
```
#include "painlessMesh.h"

#define   MESH_PREFIX     "whateverYouLike"
#define   MESH_PASSWORD   "somethingSneaky"
#define   MESH_PORT       5555
```

You can of course change both the **MESH_PREFIX** and **MESH_PASSWORD** values but you want to make sure that they remain the same for each ESP upload. This allows both the smart char circuit and the smart door circuit to be in the same local network and communicate with each other.

Once the upload is finished, you can either power both circuits via USB or power them with an external power supplier (e.g. power bank).

In order to test the example sketch, you need to have at least one of the circuits connected via USB to your computer.

Now, open the Serial Monitor at the correct baud rate and you should see the following:



Great, you can see that the ESP node connected via USB receives the “Hello from node” message from the other ESP node. Try to switch the circuit connected to the USB and you will see a different ID on the Serial Monitor.
The two circuits are now communicating wirelessly, isn’t it great?
You might already picture all the wonderful projects that you could create with such a protocol.

Now that the mesh networking protocol is working, you will modify the code so that you can control the smart chair green LED from the touch sensor on the smart chair circuit.

Step Four

Controlling components

You saw the the example sketch allowed the smart chair and the smart door circuits to send each other a message wirelessly via a mesh network protocol. This is great but not really exciting. What about controlling components using the same protocol?

In this part of the exercise you will modify the example sketch code so that you can control the green LED of the smart door circuit from the touch sensor on the smart chair circuit. When you press on the touch sensor, the green LED should light up.

There are few things that you need to change/add to the example sketch before you can upload it to both your smart chair and your smart door ESP circuits:

- Add the logic to read the touch sensor value
- Add a reference to the green LED pin
- Add the logic to send out a message with the touch sensor value
- Add the logic to read the incoming message and turn ON/OFF the green LED

The network credentials must be the same for both uploads. You can either leave them as they are or change the **MESH_PREFIX** and **MESH_PASSWORD** values to your choice:

```
#include "painlessMesh.h"

#define   MESH_PREFIX     "whateverYouLike"
#define   MESH_PASSWORD   "somethingSneaky"
#define   MESH_PORT       5555
```

The next step is for you to add the references to the touch sensor and the green LED pins. The touch sensor is attached to pin D5 and the green LED to pin D0:

```
// Initialise the touch sensor pin
// sensor_value to store the value of the touch sensor
const int touch_sensor = D5;
int sensor_value;

// LED pin
const int  led_pin = D0;
```

Add the above code just below the network credentials.

Now, you need to set the mode of both pins inside the **setup()** function. You want to read the value from the touch sensor (INPUT) and set the value of the green LED (OUTPUT):

```
void setup() {
  Serial.begin(115200);

  // Touch sensor as INPUT
  pinMode(touch_sensor, INPUT);
  // Led pin as OUTPUT
  pinMode(led_pin, OUTPUT);

//mesh.setDebugMsgTypes( ERROR | MESH_STATUS | CONNECTION | SYNC | COMMUNICATION | GENERAL | MSG_TYPES | REMOTE ); // all types on
  mesh.setDebugMsgTypes( ERROR | STARTUP );  // set before init() so that you can see startup messages

  mesh.init( MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT );
  mesh.onReceive(&receivedCallback);
  mesh.onNewConnection(&newConnectionCallback);
  mesh.onChangedConnections(&changedConnectionCallback);
  mesh.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);

  userScheduler.addTask( taskSendMessage );
  taskSendMessage.enable();
}
```

Add the above code just below the **Serial.begin(1152200)** statement.

The setup() function is mainly responsible for setting up the mesh network protocol and for managing the call back functions:

receivedCallback: listens for incoming messages from all other nodes

newConnectionCallback: new connection detected

changedConnectionCallback: connection topology has changed

nodeTimeAdjustedCallback: adjusts local time with the mesh

taskSendMessage: broadcast the message to all other nodes

The next step is to create the utility function to read the touch sensor value and call it in the **loop()** function:

```
// Read the value of the sensor
void touchSensorValue(){
  sensor_value = digitalRead(touch_sensor); // Read the value of the touch sensor (0--1)
}
```

Add the **touchSensorValue()** utility function just below the **loop()** function.

```
void loop() {
  // it will run the user scheduler as well
  mesh.update();
  // Check the value of the touch sensor
  touchSensorValue();
}
```

Now, simply call the **touchSensorValue()** inside the **loop()** function to get a constant reading from the touch sensor. The **loop()** function is responsible to run the mesh network: **mesh.update().**

At this point, you have the code to read the value from the touch sensor. **sensor_value** will store a **0** when the sensor is not touched and a **1** when the sensor is touched.

You now need to broadcast **sensor_value** as a message to all the other nodes and finally read the received message to either turn ON or OFF the green LED.

Modify the **sendMessage()** function as follow:

```
// Send the touch sensor value message
void sendMessage() {
  String msg = "";
  msg += String(sensor_value);
  mesh.sendBroadcast( msg );
  taskSendMessage.setInterval( random( TASK_SECOND * 1, TASK_SECOND * 5 ));
}
```

Here the function sends the value of the **sensor_value** variable at random intervals to all the other nodes in the network. In our case, we want to send the value to the smart door circuit.

Finally, you want to read the incoming message from other nodes to either turn ON or OFF the green LED. Modify the **receivedCallback()** function as follow:

```
// Needed for painless library
void receivedCallback( uint32_t from, String &msg ) {
  Serial.printf("Message:");
  Serial.printf(msg.c_str());
  // Check the response
  String response = String(msg.c_str());
  if (response == "1"){
    digitalWrite(led_pin,HIGH);
    Serial.printf("LED ON");
  }else{
    digitalWrite(led_pin,LOW);
    Serial.printf("LED OFF");
  }
}
```

Here, the **receivedCallback()** function checks the incoming **sensor_value** message. If the message contains a “1”, the green LED should be ON. Otherwise, the green LED should be OFF.

Hurray, the code for the mesh networking is now complete. Go ahead and upload the same sketch to both the smart chair and the smart door circuits.

You can now power up the circuits using an external power supply (power bank) or via USB.

Press on the smart chair circuit touch sensor and you should see the green LED on the smart door circuit light up. You will have to keep the touch sensor pressed for at least 5 seconds as the message is sent at random intervals.

The mesh networking protocol opens up new possibilities for you to create interesting wireless projects.

Step Four
Practice with Mesh Networking

Now that you are familiar with mesh networking and also explored how to implement the protocol, why don't you try to further adapt the **PainlessMesh** basic example sketch code to one of your circuits?

Can you create a mesh network with more than two nodes?

Currently, the example code that we implemented together broadcasts and receives messages to and from all the nodes in the network. The smart door circuit, for example, does not really have the touch sensor and therefore should not broadcast any messages linked to that sensor. Can you modify the code so that there is some sort of filtering for which messages should be broadcasted and which messages should be received by each node in the local network?