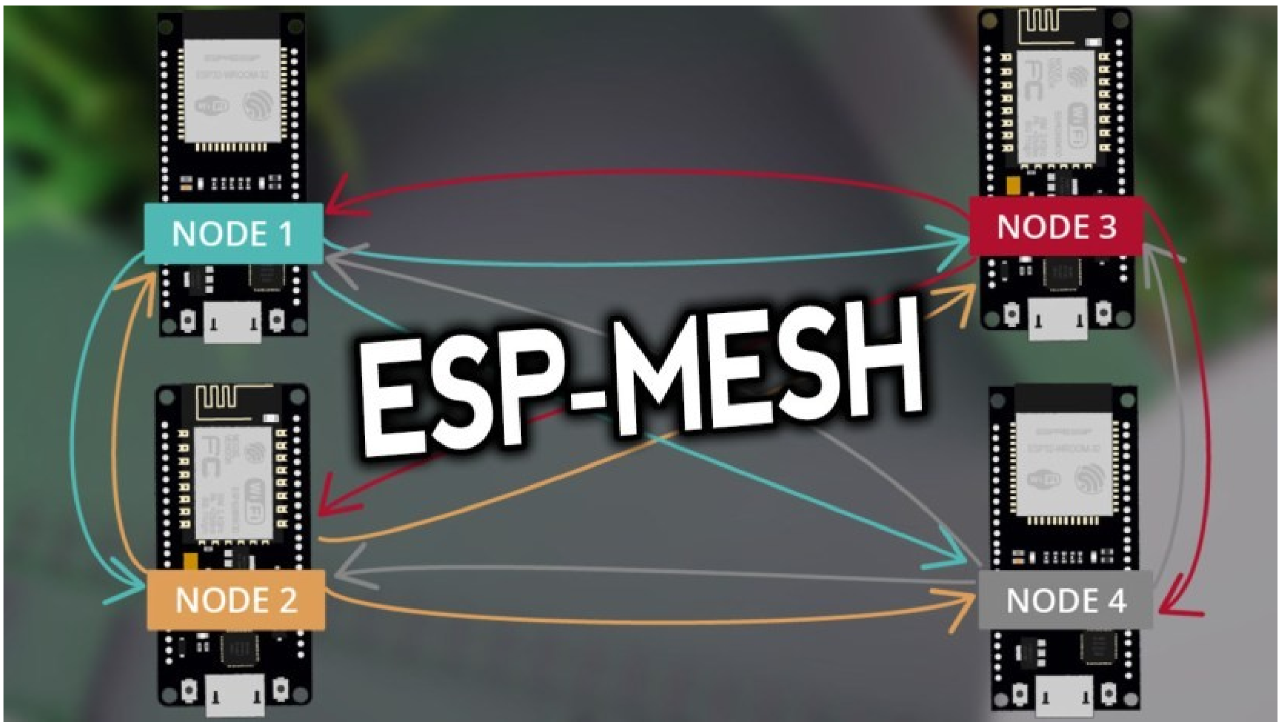
Part 10
-
ESP-MESH

Version: 2021-01-09

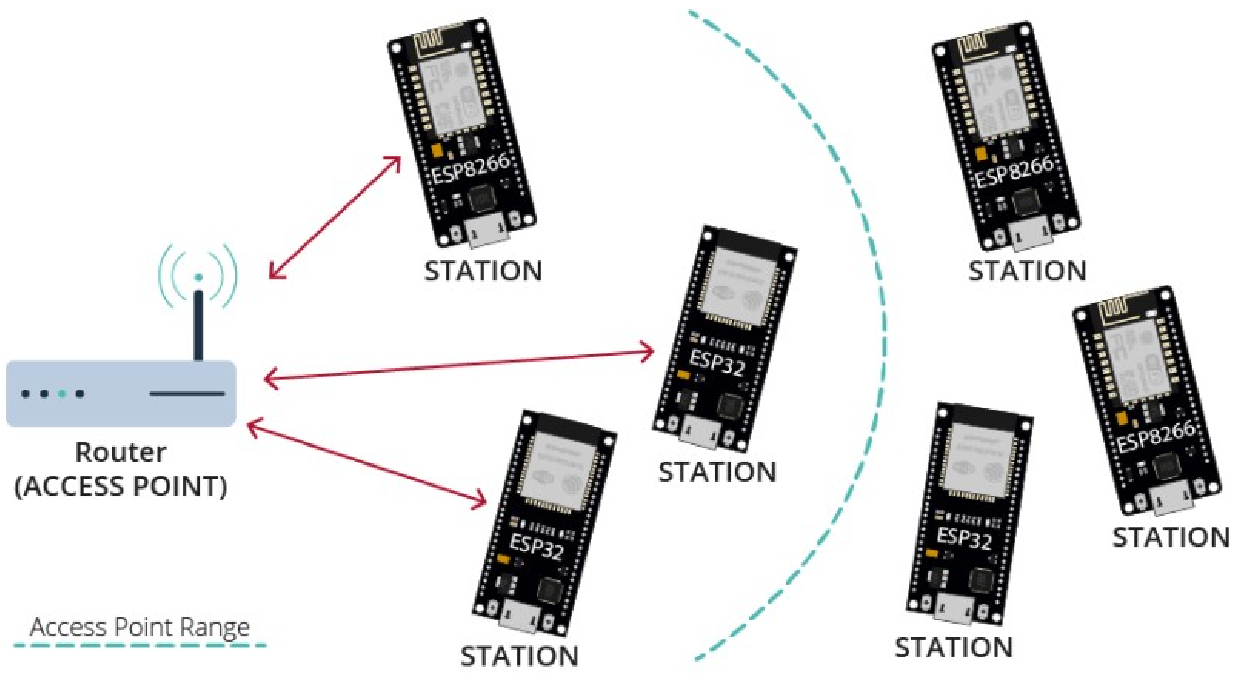
ESP-MESH with ESP32

Getting Started
Learn how to use ESP-MESH networking protocol to build a mesh network with the ESP32 boards. ESP-MESH allows multiple devices (nodes) to communicate with each other under a single wireless local area network. It is supported on the ESP32. We'll show you how to get started with ESP-MESH using the Arduino core.

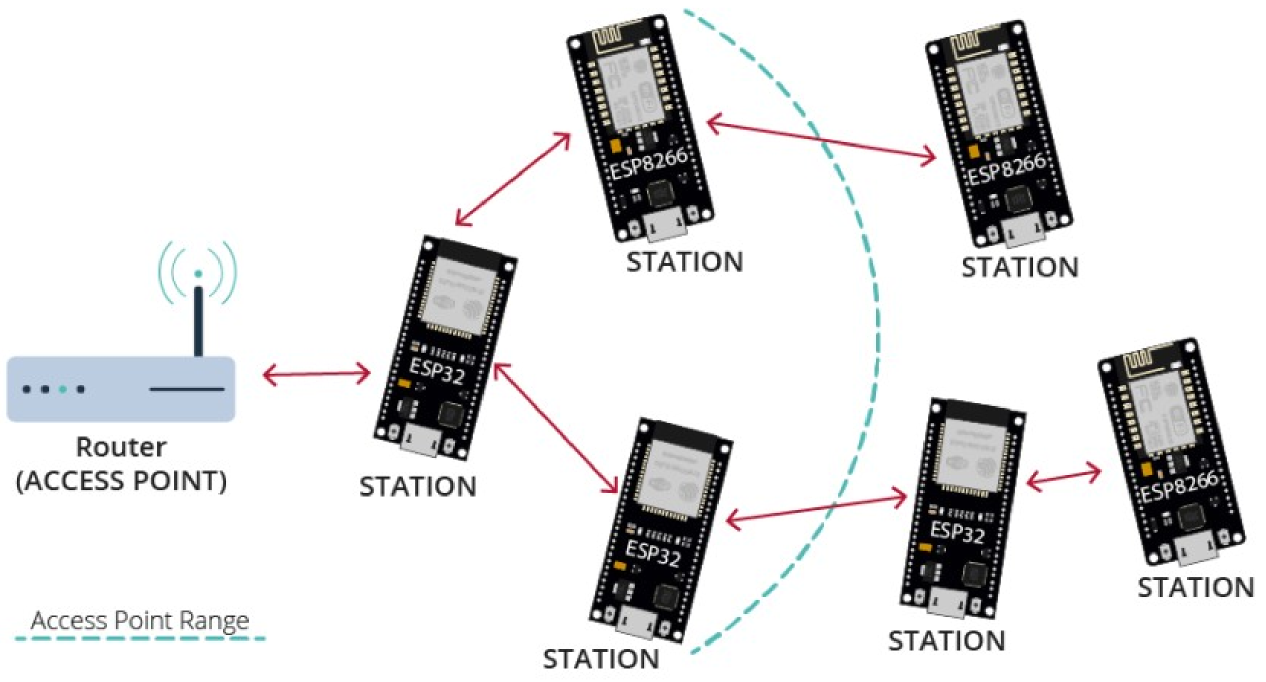


Introducing ESP-MESH
Accordingly to the Espressif documentation:
"*ESP-MESH is a networking protocol built on top the Wi-Fi protocol. ESP-MESH allows numerous devices (referred to as nodes) spread over a large physical area (both indoors and outdoors) to be interconnected under a single WLAN (Wireless Local-Area Network).*
ESP-MESH is self-organizing and self-healing meaning the network can be built and maintained autonomously." For more information, visit the ESP-MESH official documentation.

Traditional Wi-Fi Network Architecture
In a traditional Wi-Fi network architecture, a single node (access point – usually the router) is connected to all other nodes (stations). Each node can communicate with each other using the access point. However, this is limited to the access point wi-fi coverage. Every station must be in the range to connect directly to the access point.

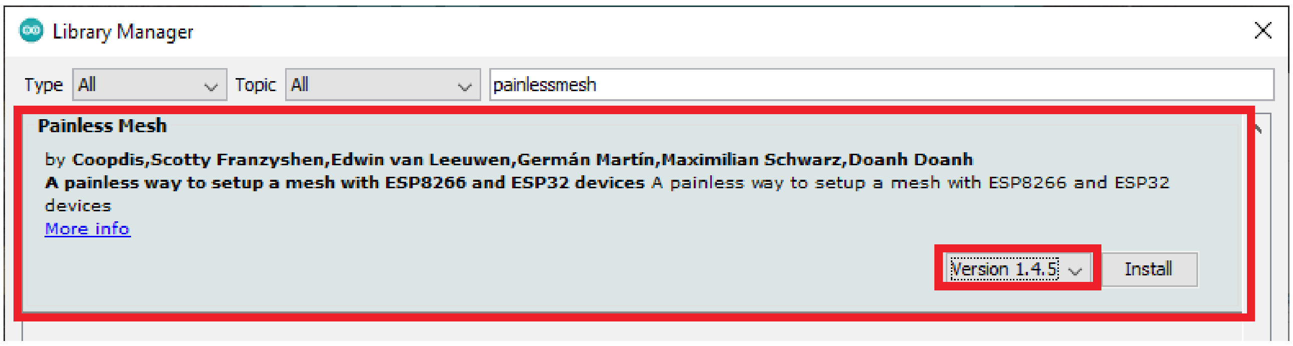


ESP-MESH Network Architecture
With ESP-MESH, the nodes don't need to connect to a central node. Nodes are responsible for relaying each others transmissions. This allows multiple devices to spread over a large physical area. The Nodes can self-organize and dynamically talk to each other to ensure that the packet reaches its final node destination. If any node is removed from the network, it is able to self-organize to make sure that the packets reach their destination.



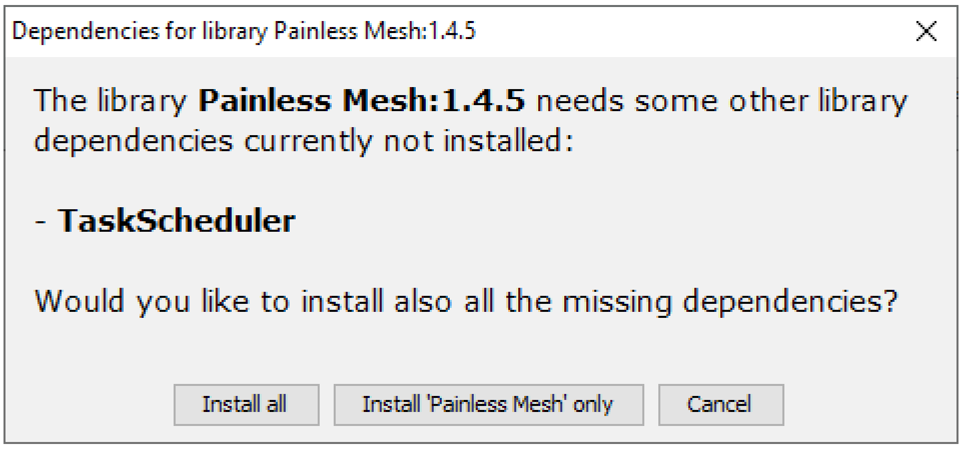
painlessMesh Library
The painlessMesh library allows us to create a mesh network with the ESP32 boards in an easy way.
"painlessMesh is a true ad-hoc network, meaning that no-planning, central controller, or router is required. Any system of 1 or more nodes will self-organize into fully functional mesh. The maximum size of the mesh is limited (we think) by the amount of memory in the heap that can be allocated to the sub-connections buffer and so should be really quite high."

Installing painlessMesh Library
You can install painlessMesh through the Arduino Library manager. Go to `Tools > Manage Libraries`. The Library Manager should open.
Search for "**painlessmesh**" and install the library. We're using Version 1.4.5



Painless Mesh
by **Coopdis,Scotty Franzyshen,Edwin van Leeuwen,Germán Martín,Maximilian Schwarz,Doanh Doanh**
A painless way to setup a mesh with ESP8266 and ESP32 devices A painless way to setup a mesh with ESP8266 and ESP32 devices

This library needs some other library dependencies. A new window should pop up asking you to install any missing dependencies. Select "Install all".


Dependencies for library Painless Mesh:1.4.5
Install all
Install 'Painless Mesh' only
Cancel

The library **Painless Mesh:1.4.5** needs some other library dependencies currently not installed:

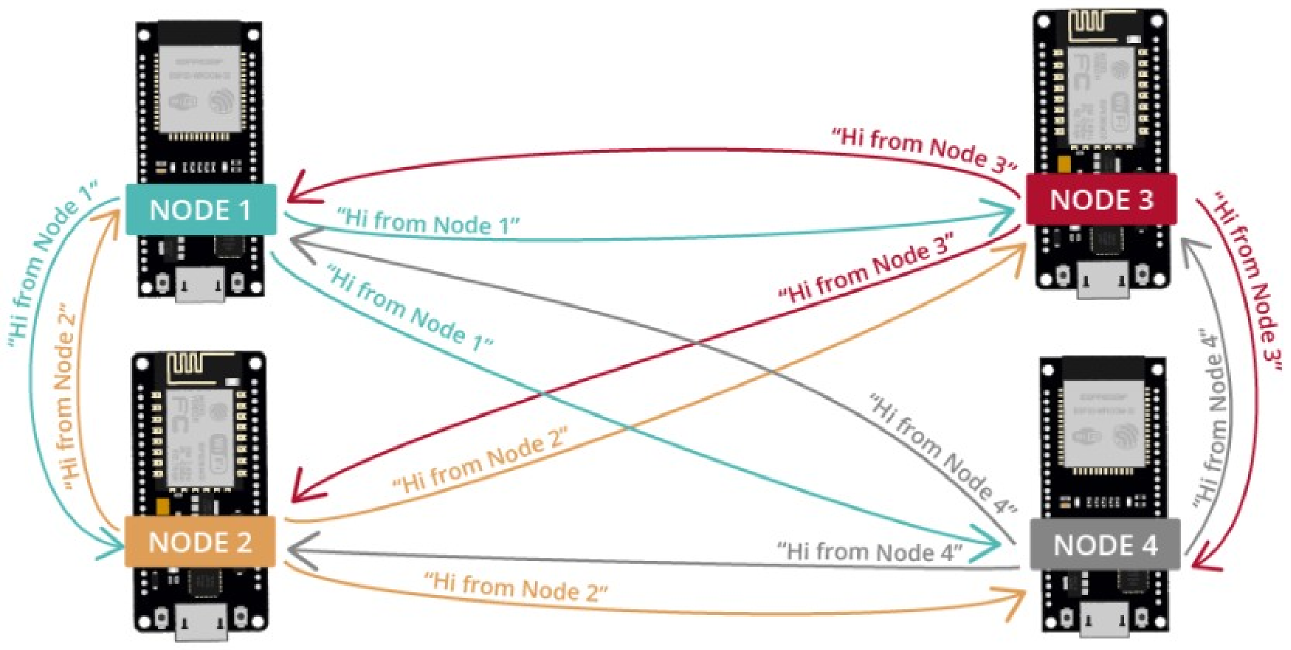
- **TaskScheduler**

Would you like to install also all the missing dependencies?

If this window doesn't show up, you'll need to install the following library dependencies:

- ArduinoJson (by bblanchon)
- TaskScheduler
- AsyncTCP (ESP32)

ESP-MESH Basic Example
To get started with ESP-MESH, we'll first experiment with the library's basic example. This example creates a mesh network in which all boards broadcast messages to all the other boards. We've experimented this example with four ESP32 boards. You can add or remove boards.



Copy the following code to your Arduino IDE (code from the library examples).

```
#include "WiFi.h"
#include "painlessMesh.h"

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

String wifiMACaddress = WiFi.macAddress();

// to control your personal task
Scheduler userScheduler;
painlessMesh  mesh;

// User stub - Prototype so PlatformIO doesn't complain
void sendMessage();

// task scheduler wiil execute the function attached to it
Task taskSendMessage( TASK_SECOND * 5 , TASK_FOREVER, &sendMessage );

void sendMessage()
{
  String msg = "Hi from node ";
  msg += mesh.getNodeId();
  msg += "  (";
  msg += wifiMACaddress;
  msg += ")";
  mesh.sendBroadcast(msg);
  taskSendMessage.setInterval(random( TASK_SECOND * 3, TASK_SECOND * 5 ));
}
```

```
// ---------------------------------------------------
// Callback functions needed for painless library
void received( uint32_t from, String &msg )
{
  Serial.printf("Received from %u\n  msg=%s\n", from, msg.c_str());
}

void newConnection(uint32_t nodeId)
{
  Serial.printf("New Connection, nodeId = %u\n", nodeId);
}

void changedConnection()
{
  Serial.printf("Changed connection\n");
}

void nodeTimeAdjusted(int32_t offset)
{
  Serial.printf("Adjusted time %u. Offset = %d\n", mesh.getNodeTime(),off-
set);
}
// ---------------------------------------------------

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

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

  mesh.init(MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT);
  // hook in the callback functions
  mesh.onReceive(&received);
  mesh.onNewConnection(&newConnection);
  mesh.onChangedConnections(&changedConnection);
  mesh.onNodeTimeAdjusted(&nodeTimeAdjusted);

  // setup task scheduler
  userScheduler.addTask(taskSendMessage);
  taskSendMessage.enable();
}

void loop()
{
  // it will run the user scheduler as well
  mesh.update();
}
```

Before uploading the code, you can set up the `MESH_PREFIX` (it’s like the name of the MESH network) and the `MESH_PASSWORD` variables (you can set it to whatever you like).

How the Code Works
Start by including the `painlessMesh` library. The WiFi library is only include to obtain the WiFi MAC address so it is easier to identify the ESP32 board.

```
#include "WiFi.h"
#include "painlessMesh.h"
```

Then, add the mesh details. The `MESH_PREFIX` refers to the name of the mesh. You can change it to whatever you like.

```
#define MESH_PREFIX "whateverYouLike"
```

The `MESH_PASSWORD`, as the name suggests is the mesh password. You can change it to whatever you like.

```
#define MESH_PASSWORD "somethingSneaky"
```

All nodes in the mesh should use the same `MESH_PREFIX` and `MESH_PASSWORD`.
The `MESH_PORT` refers to the the TCP port that you want the mesh server to run on. The default is 5555.

```
#define MESH_PORT 5555
```

It is recommended to avoid using `delay()`in the mesh network code. To maintain the mesh, some tasks need to be performed in the background. Using `delay()` will stop these tasks from happening and can cause the mesh to lose stability/fall apart.
Instead, it is recommended to use `TaskScheduler` to run your tasks which is used in painlessMesh itself. The following line creates a new `Scheduler` called `userScheduler`.

```
Scheduler userScheduler;
```

Create a `painlessMesh` object called `mesh` to handle the mesh network.

```
painlessMesh  mesh;
```

Create a task called `taskSendMessage` responsible for calling the `sendMessage()` function every 5 seconds as long as the program is running.

```
Task taskSendMessage(TASK_SECOND * 5 , TASK_FOREVER, &sendMessage);
```

The `sendMessage()` function sends a message to all nodes in the message network (broadcast).

```
void sendMessage()
{
  String msg = "Hi from node ";
  msg += mesh.getNodeId();
  msg += " (";
  msg += wifiMACaddress;
  msg += ")";
  mesh.sendBroadcast( msg );
  taskSendMessage.setInterval( random( TASK_SECOND * 3, TASK_SECOND * 5 ));
}
```

The message contains the "`Hi from node `" text followed by the board chip ID and the board MAC address.

```
String msg = "Hi from node ";
msg += mesh.getNodeId();
msg += " (";
msg += wifiMACaddress;
msg += ")";
```

To broadcast a message, simply use the `sendBroadcast()` method on the `mesh` object and pass as argument the message (`msg`) you want to send.

```
mesh.sendBroadcast(msg);
```

Every time a new message is sent, the code changes the interval between messages (three to five seconds).

```
taskSendMessage.setInterval(random(TASK_SECOND * 3, TASK_SECOND * 5));
```

Next, several callback functions are created that will be called when specific events happen on the mesh. The `received()` function prints the message sender (`from`) and the content of the message (`msg.c_str()`).

```
void received( uint32_t from, String &msg )
{
  Serial.printf("Received from %u\n  msg=%s\n", from, msg.c_str());
}
```

The `newConnection()` function runs whenever a new node joins the network. This function simply prints the chip ID of the new node. You can modify the function to do any other task.

```
void newConnection(uint32_t nodeId)
{
  Serial.printf("New Connection, nodeId = %u\n", nodeId);
}
```

The `changedConnection()` function runs whenever a connection changes on the network (when a node joins or leaves the network).

```
void changedConnection()
{
  Serial.printf("Changed connection\n");
}
```

The `nodeTimeAdjusted()` function runs when the network adjusts the time, so that all nodes are synchronized. It prints the offset.

```
void nodeTimeAdjusted(int32_t offset)
{
  Serial.printf("Adjusted time %u. Offset = %d\n", mesh.getNodeTime(),off-
set);
}
```

In the `setup()`, initialize the serial monitor.

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

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

Initialize the mesh with the details defined earlier.

```
mesh.init(MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT);
```

Assign all the callback functions to their corresponding events.

```
mesh.onReceive(&received);
mesh.onNewConnection(&newConnection);
mesh.onChangedConnections(&changedConnection);
mesh.onNodeTimeAdjusted(&nodeTimeAdjusted);;
```

Finally, add the `taskSendMessage` function to the `userScheduler`. The scheduler is responsible for handling and running the tasks at the right time.

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

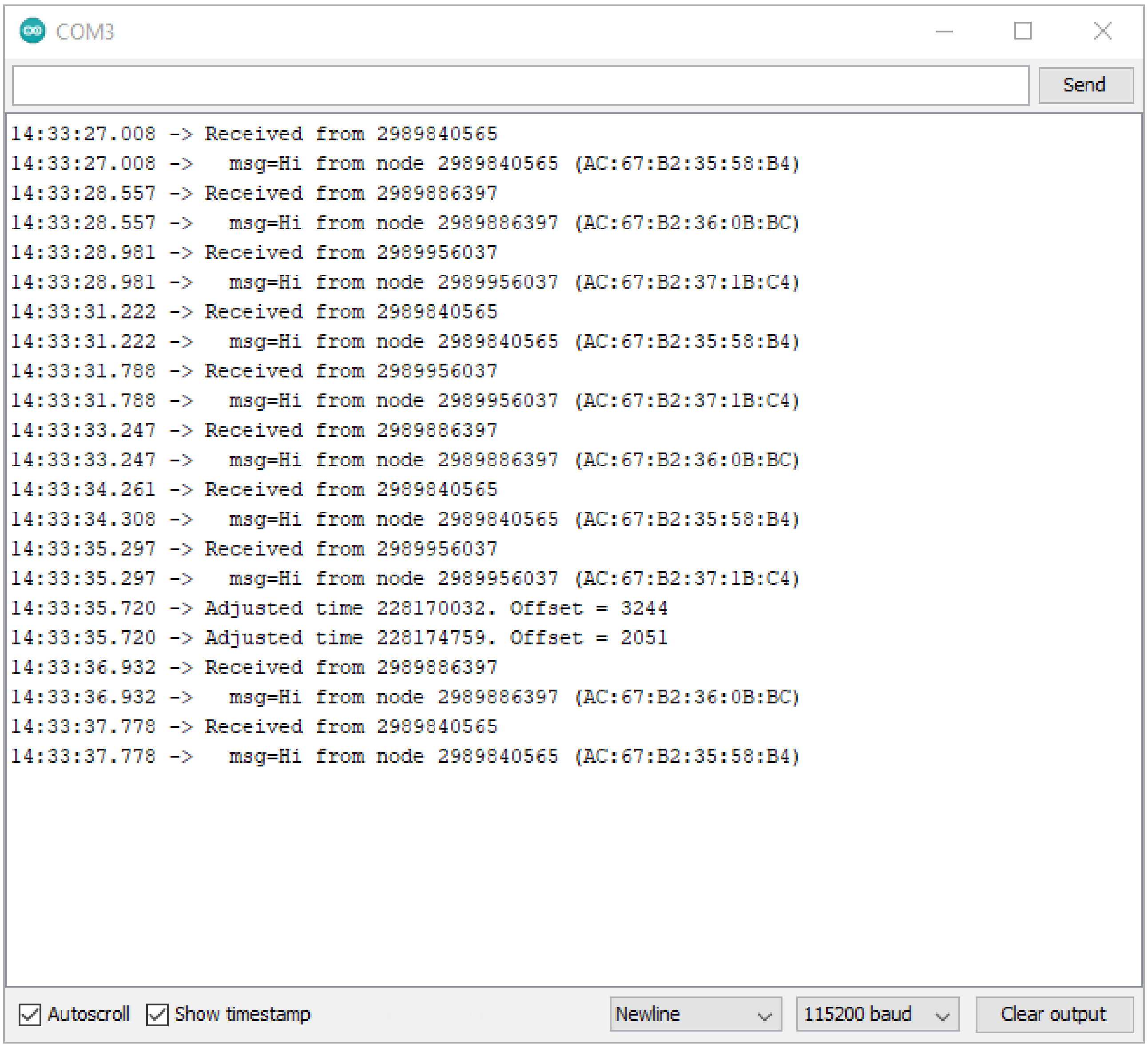
Finally, enable the `taskSendMessage`, so that the program starts sending the messages to the mesh.

```
taskSendMessage.enable();
```

To keep the mesh running, add `mesh.update()` to the `loop()`.

```
void loop()
{
  // it will run the user scheduler as well
  mesh.update();
}
```

Testing
Upload the code provided to all your boards. With a board connected to your computer, open a serial connection with the board. You can use the Serial Monitor, or you can use a software like PuTTY. You should see the board receives messages from the other boards.



```
14:33:27.008 -> Received from 2989840565
14:33:27.008 ->   msg=Hi from node 2989840565 (AC:67:B2:35:58:B4)
14:33:28.557 -> Received from 2989886397
14:33:28.557 ->   msg=Hi from node 2989886397 (AC:67:B2:36:0B:BC)
14:33:28.981 -> Received from 2989956037
14:33:28.981 ->   msg=Hi from node 2989956037 (AC:67:B2:37:1B:C4)
14:33:31.222 -> Received from 2989840565
14:33:31.222 ->   msg=Hi from node 2989840565 (AC:67:B2:35:58:B4)
14:33:31.788 -> Received from 2989956037
14:33:31.788 ->   msg=Hi from node 2989956037 (AC:67:B2:37:1B:C4)
14:33:33.247 -> Received from 2989886397
14:33:33.247 ->   msg=Hi from node 2989886397 (AC:67:B2:36:0B:BC)
14:33:34.261 -> Received from 2989840565
14:33:34.308 ->   msg=Hi from node 2989840565 (AC:67:B2:35:58:B4)
14:33:35.297 -> Received from 2989956037
14:33:35.297 ->   msg=Hi from node 2989956037 (AC:67:B2:37:1B:C4)
14:33:35.720 -> Adjusted time 228170032. Offset = 3244
14:33:35.720 -> Adjusted time 228174759. Offset = 2051
14:33:36.932 -> Received from 2989886397
14:33:36.932 ->   msg=Hi from node 2989886397 (AC:67:B2:36:0B:BC)
14:33:37.778 -> Received from 2989840565
14:33:37.778 ->   msg=Hi from node 2989840565 (AC:67:B2:35:58:B4)
```

You should also see other messages when there are changes on the mesh: when a board leaves or joins the network.

Note:
In order to exchange sensore readings, one should use JSON format. To make it easier to handle JSON variables, we'll use the Arduino_JSON library.
You can install this library in the Arduino IDE Library Manager. Just go to `Sketch > Include Library > Manage Libraries` and search for the library name.
Using the JSON library, you'll find example in the Examples section of the Arduino IDE and plenty of examples on the web. Here, Google is your best friend :-)