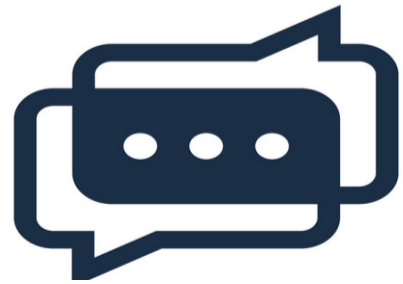
ESP32-GROUP CHAT USING MESH PROTOCOL



PROJECT REPORT

EkLAVYA MENTORSHIP PROGRAMME (2022)

AT

SOCIETY OF ROBOTICS AND AUTOMATION, VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE MUMBAI.

ACKNOWLEDGMENT

It was truly a great experience to work with the seniors. We are extremely grateful for their guidance which helped us reaching our goal. We thank SRA for giving this opportunity and also for making us aware of different domains through their domain sessions. It was a great experience altogether and learnt a lot new things in this 2-month journey.

Special Thanks to:

- Rishikesh Donadkar
- Krishna
- Viraj

Team Members:

- Vansh Panchal

  vansh5chal0308@gmail.com

  +91 8291202342

- Aryan Karawale

  askarawale@gmail.com

  +91 7977488078

Project Overview:

Project Aim:

In this project our aim is to establish a communication between 3 and more ESP32 using MESH Protocol. The network build will between self-healing and repairing that is if one of the devices disconnects due to power losses or any other issue the network will reestablish itself. Each ESPs will receive messages from user, from the prompt, and will send messages to other ESPs. Only the Parent ESP will be connected to the router and all other will establish a tree topology.

TECHNOLOGY USED:

- **ESP-WIFI-MESH:**

ESP-WIFI-MESH is a networking protocol built atop the Wi-Fi protocol. ESP-WIFI-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-WIFI-MESH is self-organizing and self-healing meaning the network can be built and maintained autonomously. ESP-WIFI-MESH is also less susceptible to overloading as the

number of nodes permitted on the network is no longer limited by a single central node. ESP-WIFI-MESH allows nodes to simultaneously act as a station and an AP. Therefore, a node in ESP-WIFI-MESH can have multiple downstream connections using its softAP interface, whilst simultaneously having a single upstream connection using its station interface. The connections in an ESP MESH leads to a TREE topology.

- **FreeRTOS:**

FreeRTOS is a market-leading real-time operating system (RTOS) for microcontrollers and small microprocessors**.** FreeRTOS is built with an emphasis on reliability and ease of use. The scheduler in a Real Time Operating System (RTOS) is designed to provide a predictable (normally described as *deterministic*) execution pattern.

INTRODUCTION

❖ BASIC PROJECT DOMAIN:

- Embedded C
- Networking
- IOT

❖ THEORY:

➢ WHY TO USE ESP-MESH:

- **Self-forming and Self-Healing:**

In the auto-routing mode, the ESP-WIFI-MESH network gets formed automatically, according to the signal strength values of peers seen by the nodes. This mode also facilitates the automatic reconnection between different nodes, whenever a parent node goes off.

- **No Extra Gateway Required:**

ESP-WIFI-MESH does not require any additional equipment to form a network. It also scales well with a low-capacity Wi-Fi access point, since the access point is completely unaware of the existence of ESP-WIFI-MESH nodes.

- **IP Connectivity:**

All the nodes in the ESP Mesh network can get IP connectivity and communicate both with each other and the external world. The internet access of these nodes is provided by a root node acting either as a NAT or a bridge.

- **Secure By Design:**

ESP-WIFI-MESH is based on standard Wi-Fi connectivity and it can use standard WPA2 network security among the mesh nodes to ensure communication security.

➢ APPLICATIONS OF ESP-WIFI-MESH:

SOME APPLICATIONS ARE:

- Smart Lighting: smart lights, lighting networks
- Smart Home: smart switches, sockets, plugs, etc.
- Automation: big parking lots, small factories, shared offices

What is Tree Topology:

The tree network topology consists of one root node, and all other nodes are connected in a hierarchy. The topology itself is connected in a star configuration. Many larger Ethernet switch networks, including data centre networks, are configured as trees.

ESP-IDF Project:

An ESP-IDF project can be seen as an amalgamation of a number of components. ESP-IDF makes these components explicit and configurable. To do that, when a project is compiled, the build system will look up all the components in the ESP-IDF directories, the project directories and (optionally) in additional custom component directories. It then allows the user to configure the ESP-IDF project using a text-based menu system to customize each component. After the components in the project are configured, the build system will compile the project.

The example Project Directory tree might look like this:

- myProject/
  - CMakeLists.txt
  - sdkconfig
  - components/ - component1/ - CMakeLists.txt
    - Kconfig
    - src1.c
  - component2/ - CMakeLists.txt
    - Kconfig

```
                        - src1.c
                        - include/ - component2.h
- main/       - CMakeLists.txt
              - src1.c
              - src2.c

- build/
```

➢ What is Scheduling and Scheduler:

The **scheduler** is the part of the kernel responsible for deciding which task should be executing at any particular time. The **scheduling policy** is the algorithm used by the scheduler to decide which task to execute at any point in time.

➢ What is TaskHandle_t?

Type by which tasks are referenced. For example, a call to xTaskCreate returns (via a pointer parameter) an TaskHandle_t variable that can then be used as a parameter to vTaskDelete to delete the task.

➢ What is Queue and Its some basic Modules:

Queue is a data structure that helps exchange data between different tasks or between tasks and interrupts. In most cases they are used as thread safe FIFO (First in First Out)

buffers with new data being sent to the back of the queue, although data can also be sent to the front.

- **xQueueCreate:**

```
QueueHandle_t xQueueCreate (UBaseType_t uxQueueLength,
                            UBaseType_t uxItemSize);
```

Creates a new queue and returns a handle by which the queue can be referenced.

If the queue is created successfully then a handle to the created queue is returned. If the memory required to create the queue could not be allocated then NULL is returned.

- **xQueueSend:**

```
BaseType_t xQueueSend (
                          QueueHandle_t xQueue,
                          const void * pvItemToQueue,
                          TickType_t xTicksToWait
                      );
```

This is a macro that calls xQueueGenericSend (). Post an item on a queue. The item is queued by copy, not by reference.

Returns pdTRUE if the item was successfully posted, otherwise errQUEUE_FULL.

- **xQueueReceive:**

```
BaseType_t xQueueReceive (
                             QueueHandle_t xQueue,
                             void *pvBuffer,
                             TickType_t xTicksToWait
                         );
```

This is a macro that calls the xQueueGenericReceive () function. Receive an item from a queue. The item is received by copy so a buffer of adequate size must be

provided. The number of bytes copied into the buffer was defined when the queue was created.

Returns pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

STAGES OF PROGRESS / WORK FLOW:

- Understanding the LED Blink and Hello World Example of ESP-IDF and learn to build and flash the code.

  - First set the idf by typing **get_idf**.
  - Then build to build the code type **idf.py build**.
  - To flash the code type **idf.py flash monitor**.

- Our code uses various functions from the following ESP MESH code.

  https://github.com/espressif/esp-idf/blob/master/examples/mesh/internal_communication/main/mesh_main.c

  Thus, with the help of these functions, a mesh is

  successfully set up. This allows message transfer between two esps connected in the mesh network. but there is a problem that only a hardcoded message can thus be transferred.

- Thus arises the need to develop an interactive console to take string messages from the terminal. We used the following reference to do so.

https://github.com/espressif/esp-idf/tree/master/examples/system/console/advanced

This includes building blocks needed to develop an interactive console over serial port.

The above console component is configured such that it will only take commands from the prompt and act accordingly.

These commands are predefined. We made some required changes in the code to accept messages from the user.

- We made a project having console as main.c and mesh_main.c in components. With the help of Build System, the project is built.

Now the next challenge would to pass the data from console to espmesh for it to be sent. We will pass the string using the "FreeRTOS QUEUE", which helps provides a pipeline between esp-mesh and console. A queue is a data structure that follows the first come first leave approach.

In total, we have created 3 tasks:

1. Console
2. Send
3. Receive

The console task helps take input from the user and then sends it to the espmesh via the queue. The console task triggers the send task which sends the data packet to the immediate parent node of the current node and all its child nodes. The third task is received which is already

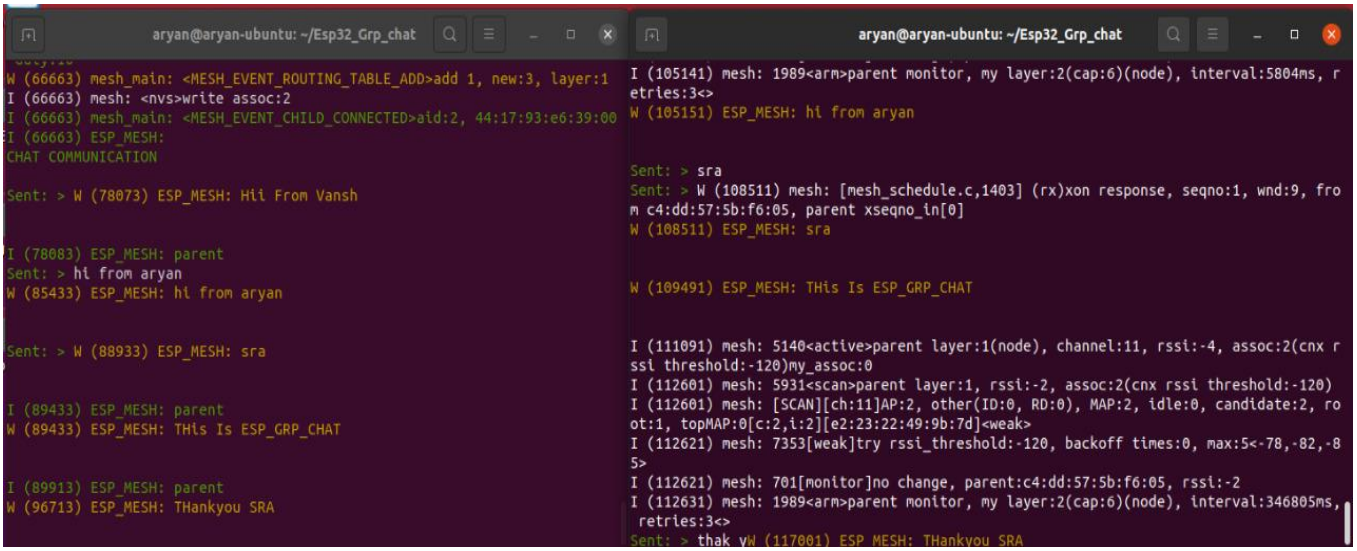
triggered before the send task has been triggered. The receive task prints the message received on the monitor.

But this still had an issue as the message was not being broadcast wherein there was no communication between two child nodes. To solve

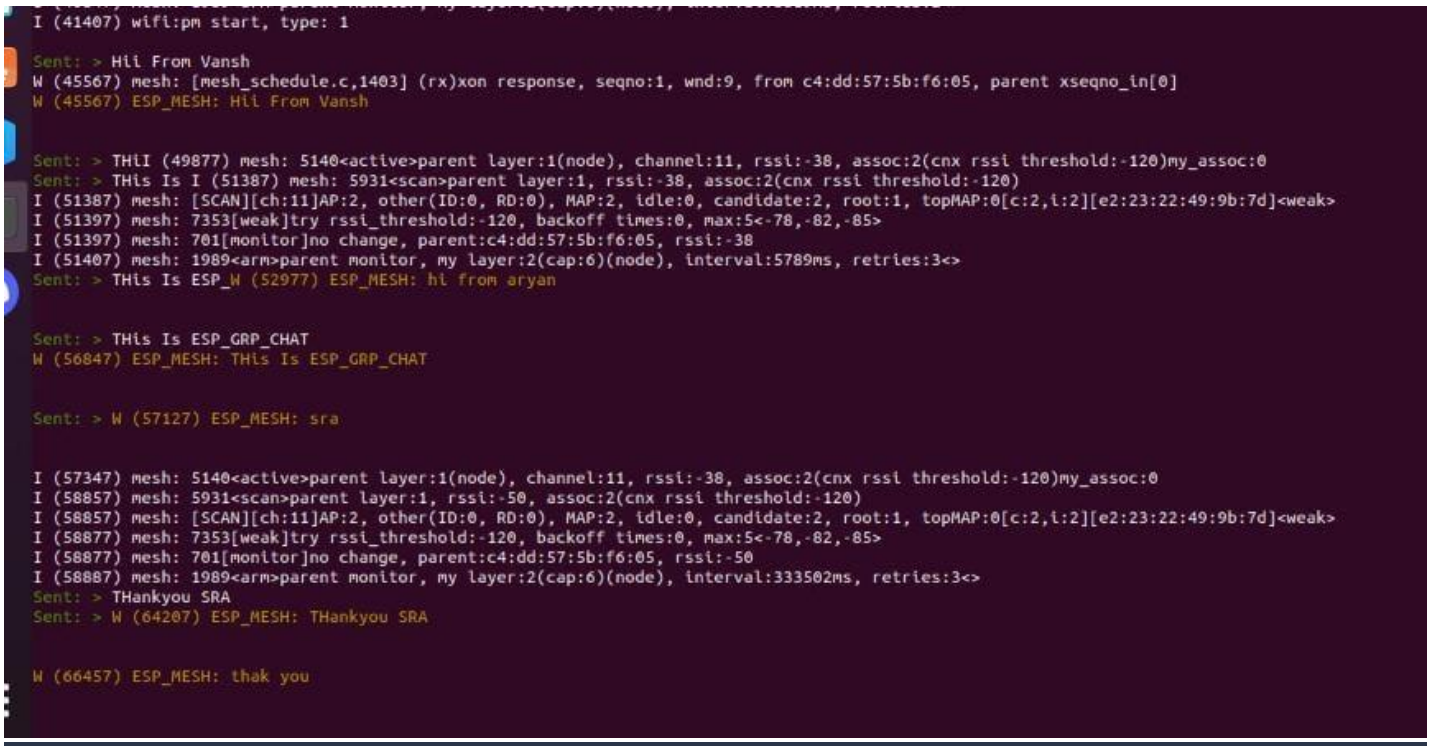
this issue if the current node is the parent node, then it triggers the send function to transfer the message to its routing table i.e. its child nodes. Thus, communication between two child nodes is established. this process is constantly repeated.

➢ Our Output:

Terminal 1 and 2:



Terminal 3:



Challenges Faced

- We faced challenges while building our project i.e., while working with Cmake.
- We faced challenges while assigning proper task preference (scheduling the tasks).
- We faced challenge while making the child nodes communicate with each other.

FUTURE WORK

- Host an http server on ESP32 along with ESP-Mesh to provide a more attractive and dynamic chat interface that can be used through a browser on mobile phone or computer.
- Add chat encryption.

Reference Links:

- Link For Basic Esp-WIFI-Mesh Terminologies and Networking basics:

https://github.com/VanshPanchal0308/Esp32_Grp_chat/blob/dev_vansh/Resources/esp32_mesh_term.md

- Basic of Networks:

https://www.geeksforgeeks.org/basics-computer-networking/

- Link For FreeRTOS:
  https://www.freertos.org/

- Link For Build System:
  https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/build-system.html

- Link for console:
  https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/console.html