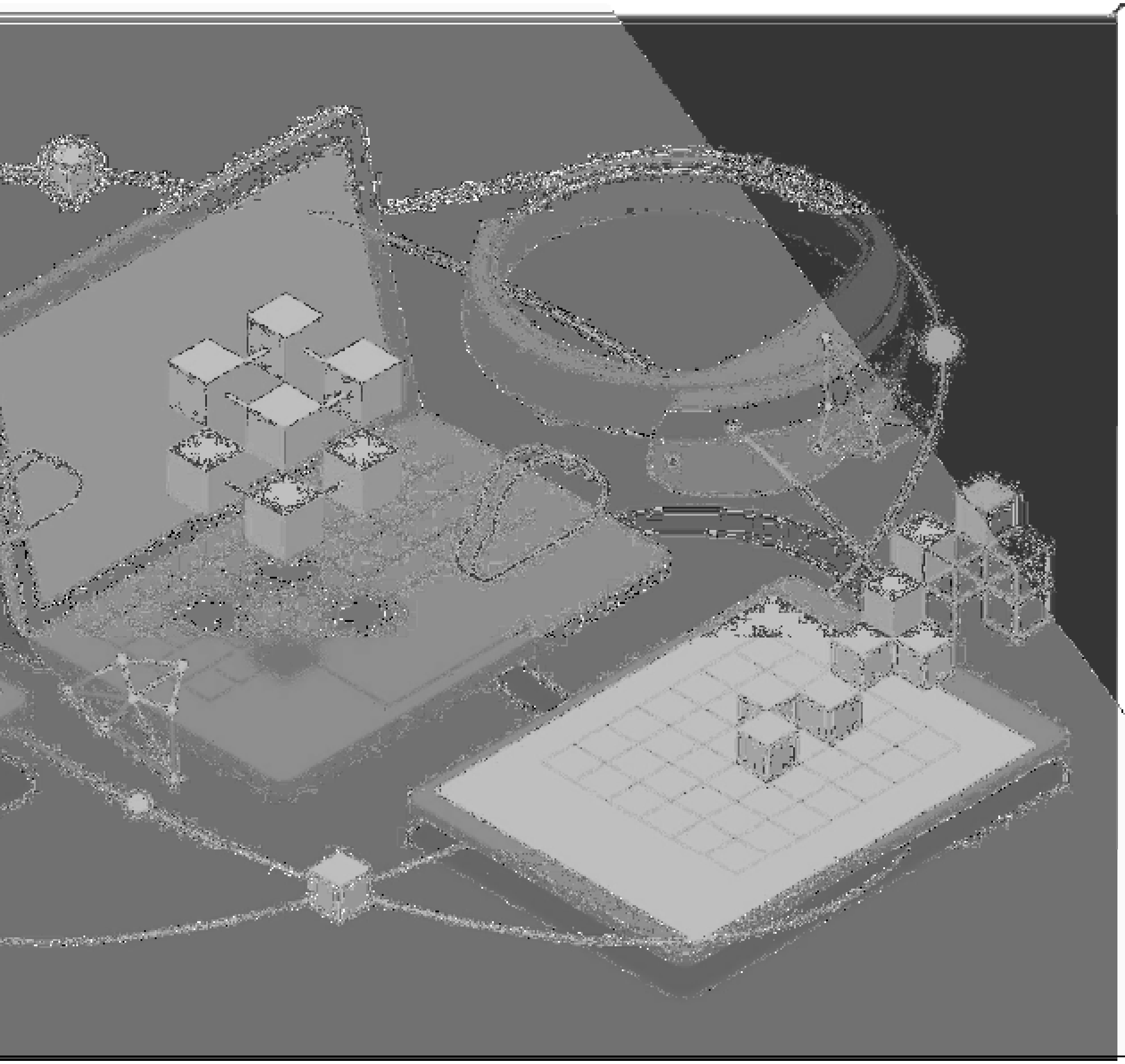
ESP NOW

Table of contents

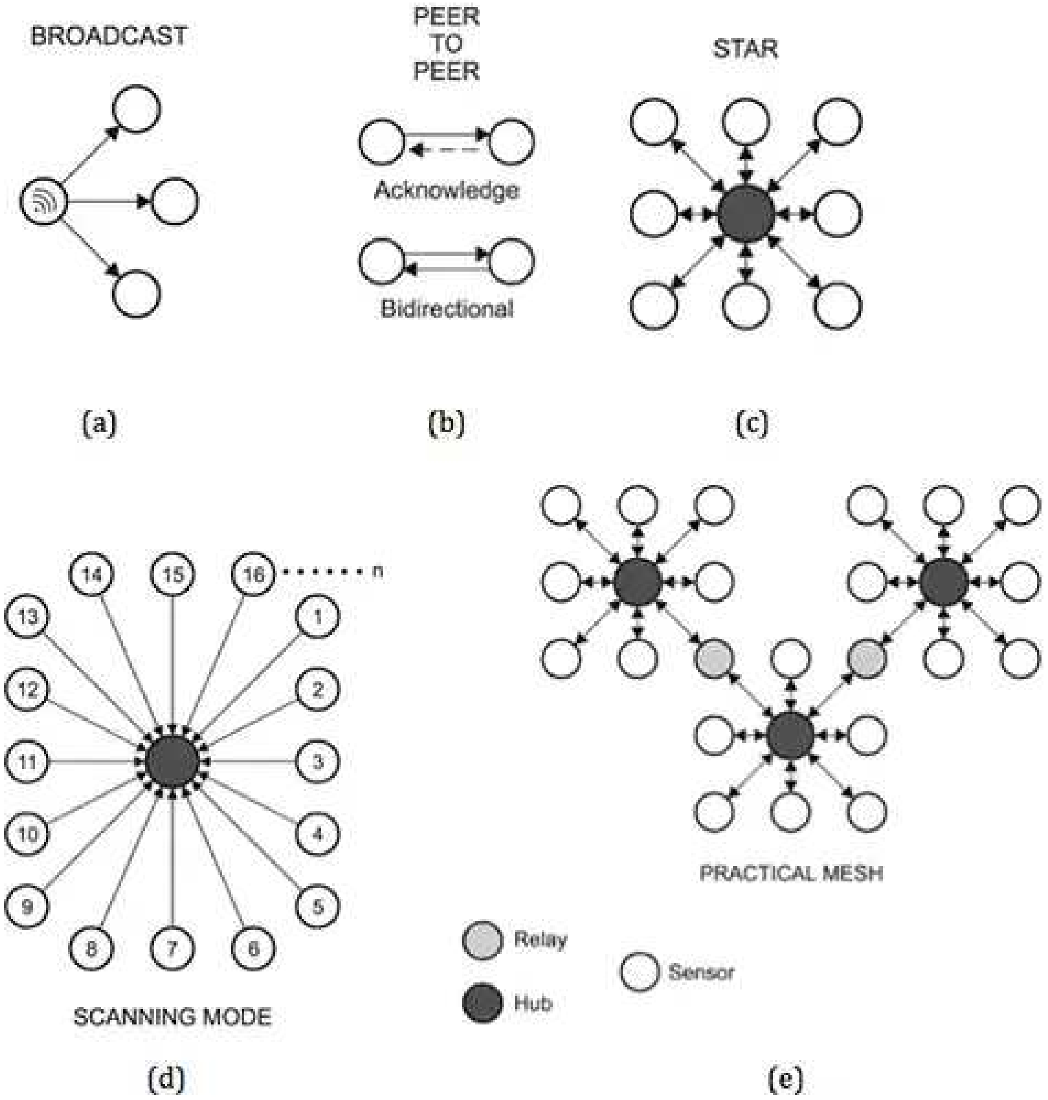


01

Pengenalan

GOAL

- Memahami sistem ESPNow dan metode koneksinya
- Membuat program sensor node dengan master ESPNOW
- Membuat jaringan mesh



Network topology

COMMUNICATION PROTOCOL

WiFi

Bluetooth

ZigBee

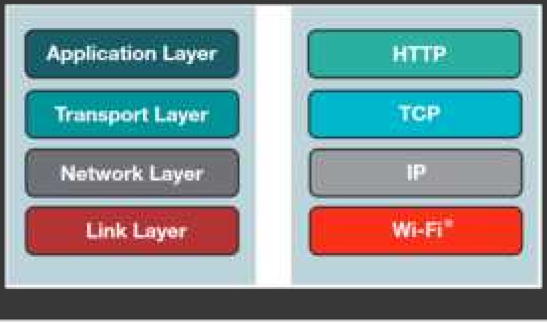
Sub-1GHz

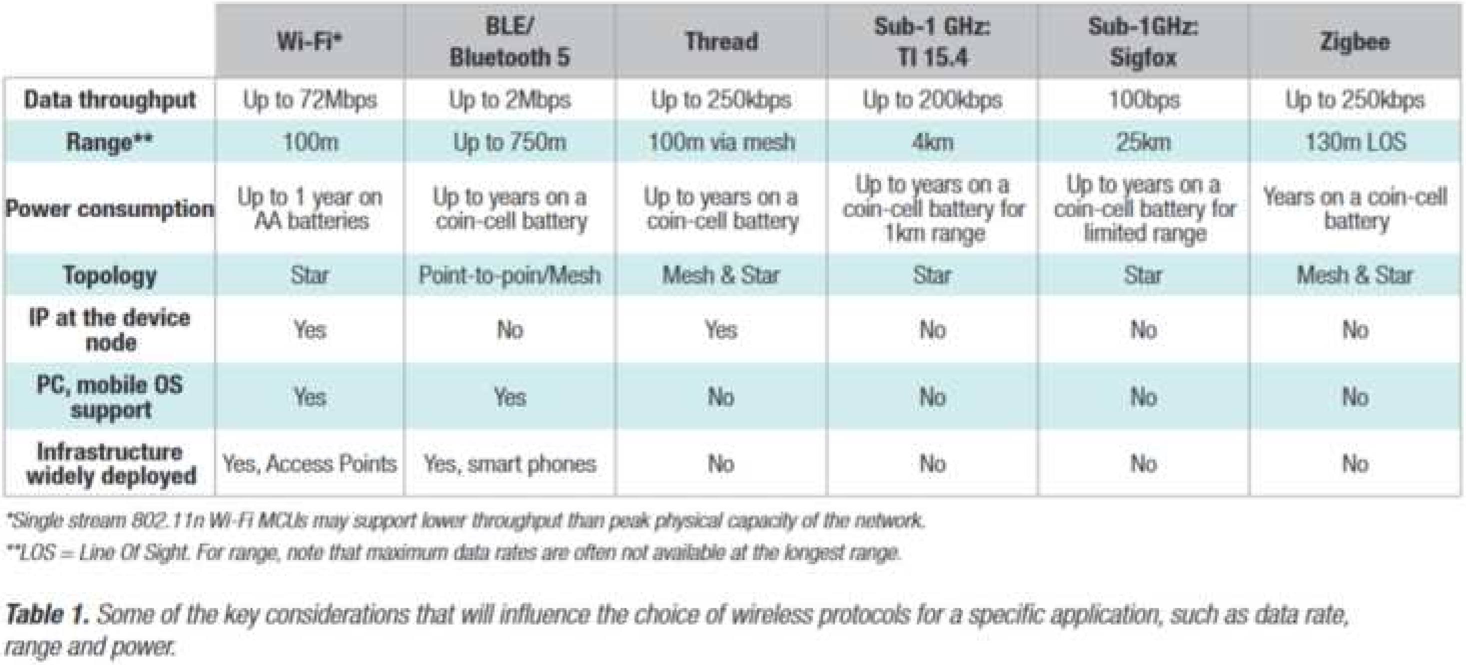
6LoWPAN

sigfox

THREAD

WirelessHART





	Wi-Fi*	BLE/ Bluetooth 5	Thread	Sub-1 GHz: TI 15.4	Sub-1GHz: Sigfox	Zigbee
Data throughput	Up to 72Mbps	Up to 2Mbps	Up to 250kbps	Up to 200kbps	100bps	Up to 250kbps
Range**	100m	Up to 750m	100m via mesh	4km	25km	130m LOS
Power consumption	Up to 1 year on AA batteries	Up to years on a coin-cell battery	Up to years on a coin-cell battery	Up to years on a coin-cell battery for 1km range	Up to years on a coin-cell battery for limited range	Years on a coin-cell battery
Topology	Star	Point-to-poin/Mesh	Mesh & Star	Star	Star	Mesh & Star
IP at the device node	Yes	No	Yes	No	No	No
PC, mobile OS support	Yes	Yes	No	No	No	No
Infrastructure widely deployed	Yes, Access Points	Yes, smart phones	No	No	No	No

*Single stream 802.11n Wi-Fi MCUs may support lower throughput than peak physical capacity of the network.
**LOS = Line Of Sight. For range, note that maximum data rates are often not available at the longest range.

Table 1. *Some of the key considerations that will influence the choice of wireless protocols for a specific application, such as data rate, range and power.*

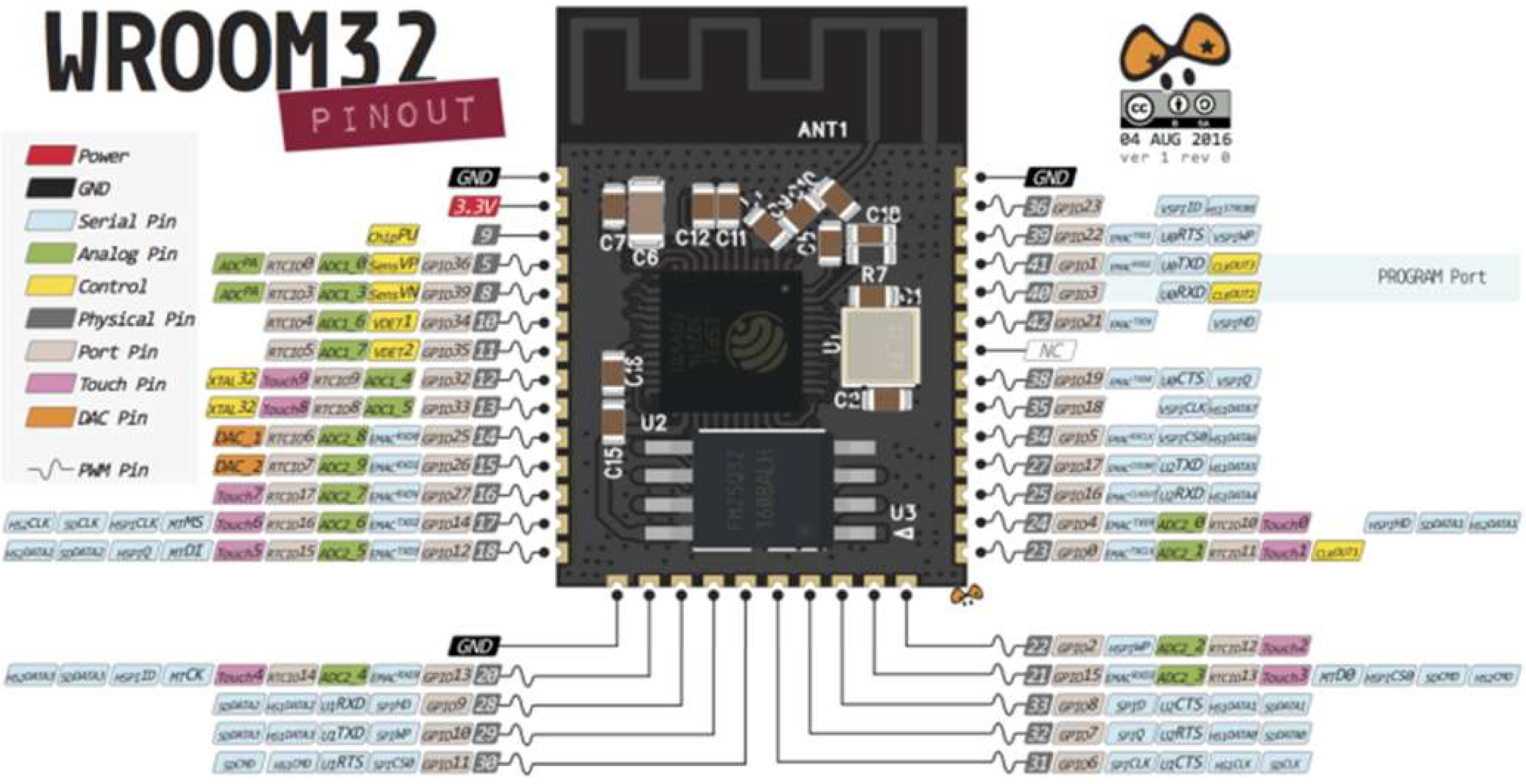
Wireless connectivity for the Internet of Things: One size does not fit all', Nick Lethaby, Texas instruments, October 2017

Espnow connection

Tentang ESPNOW

WROOM32

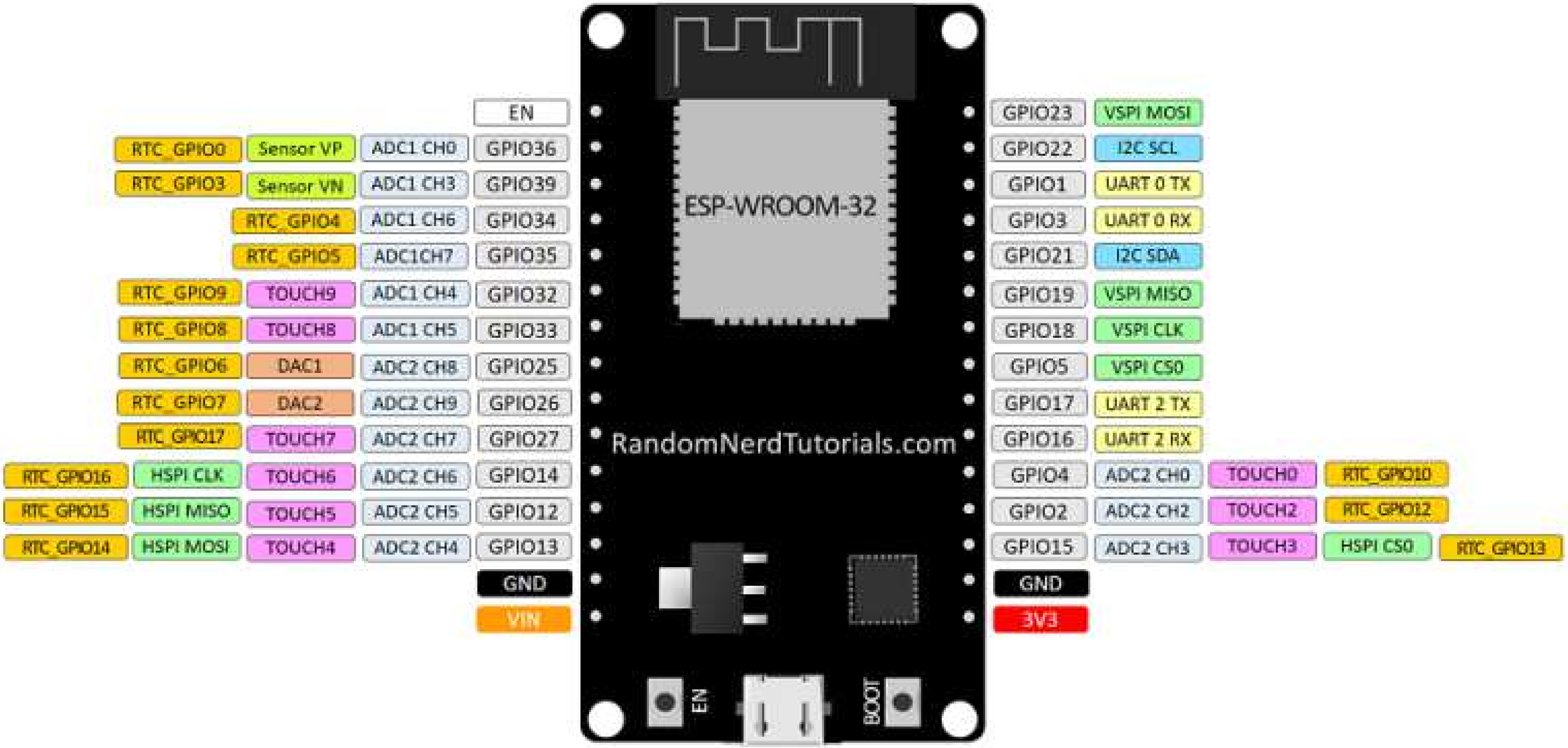
PINOUT



Power
GND
Serial Pin
Analog Pin
Control
Physical Pin
Port Pin
Touch Pin
DAC Pin
PWM Pin
ANT1
C7
C6
C12
C11
C18
R7
C2
C16
C15
U2
U3
04 AUG 2016
ver 1 rev 0
GND
3.3V
ChipPU 9
ADC_PA RTCIO0 ADC1_0 SensVP GPIO36 5
ADC_PA RTCIO3 ADC1_3 SensVN GPIO39 8
RTCIO4 ADC1_6 VDET1 GPIO34 10
RTCIO5 ADC1_7 VDET2 GPIO35 11
XTAL32 Touch9 RTCIO9 ADC1_4 GPIO32 12
XTAL32 Touch8 RTCIO8 ADC1_5 GPIO33 13
DAC_1 RTCIO6 ADC2_8 EMAC_RXD0 GPIO25 14
DAC_2 RTCIO7 ADC2_9 EMAC_RXD1 GPIO26 15
Touch7 RTCIO17 ADC2_7 EMAC_RX_DV GPIO27 16
HS2_CLK SD_CLK HSPICLK MTMS Touch6 RTCIO16 ADC2_6 EMAC_TXD2 GPIO14 17
HS2_DATA2 SD_DATA2 HSPIQ MTDI Touch5 RTCIO15 ADC2_5 EMAC_TXD3 GPIO12 18
GND
36 GPIO23 VSPI_ID HS1_STROBE
39 GPIO22 EMAC_TXD1 U0RTS VSPI_WP
41 GPIO1 EMAC_RXD2 U0TXD CLK_OUT3
40 GPIO3 U0RXD CLK_OUT2
PROGRAM Port
42 GPIO21 EMAC_TX_EN VSPI_HD
NC
38 GPIO19 EMAC_TXD0 U0CTS VSPIQ
35 GPIO18 VSPI_CLK HS1_DATA7
34 GPIO5 EMAC_RX_CLK VSPI_CS0 HS1_DATA6
27 GPIO17 EMAC_CLK_180 U2TXD HS1_DATA5
25 GPIO16 EMAC_CLK_OUT U2RXD HS1_DATA4
24 GPIO4 EMAC_TX_ER ADC2_0 RTCIO10 Touch0 HSPI_HD SD_DATA1 HS2_DATA1
23 GPIO0 EMAC_TX_CLK ADC2_1 RTCIO11 Touch1 CLK_OUT1
GND
HS2_DATA3 SD_DATA3 HSPI_ID MTCK Touch4 RTCIO14 ADC2_4 EMAC_RX_ER GPIO13 20
SD_DATA2 HS1_DATA2 U1RXD SPI_HD GPIO9 28
SD_DATA3 HS1_DATA3 U1TXD SPI_WP GPIO10 29
SD_CMD HS1_CMD U1RTS SPI_CS0 GPIO11 30
22 GPIO2 HSPI_WP ADC2_2 RTCIO12 Touch2
21 GPIO15 EMAC_RXD3 ADC2_3 RTCIO13 Touch3 MTD0 HSPI_CS0 SD_CMD HS2_CMD
33 GPIO8 SPID U2CTS HS1_DATA1 SD_DATA1
32 GPIO7 SPIQ U2RTS HS1_DATA0 SD_DATA0
31 GPIO6 SPI_CLK U1CTS HS1_CLK SD_CLK

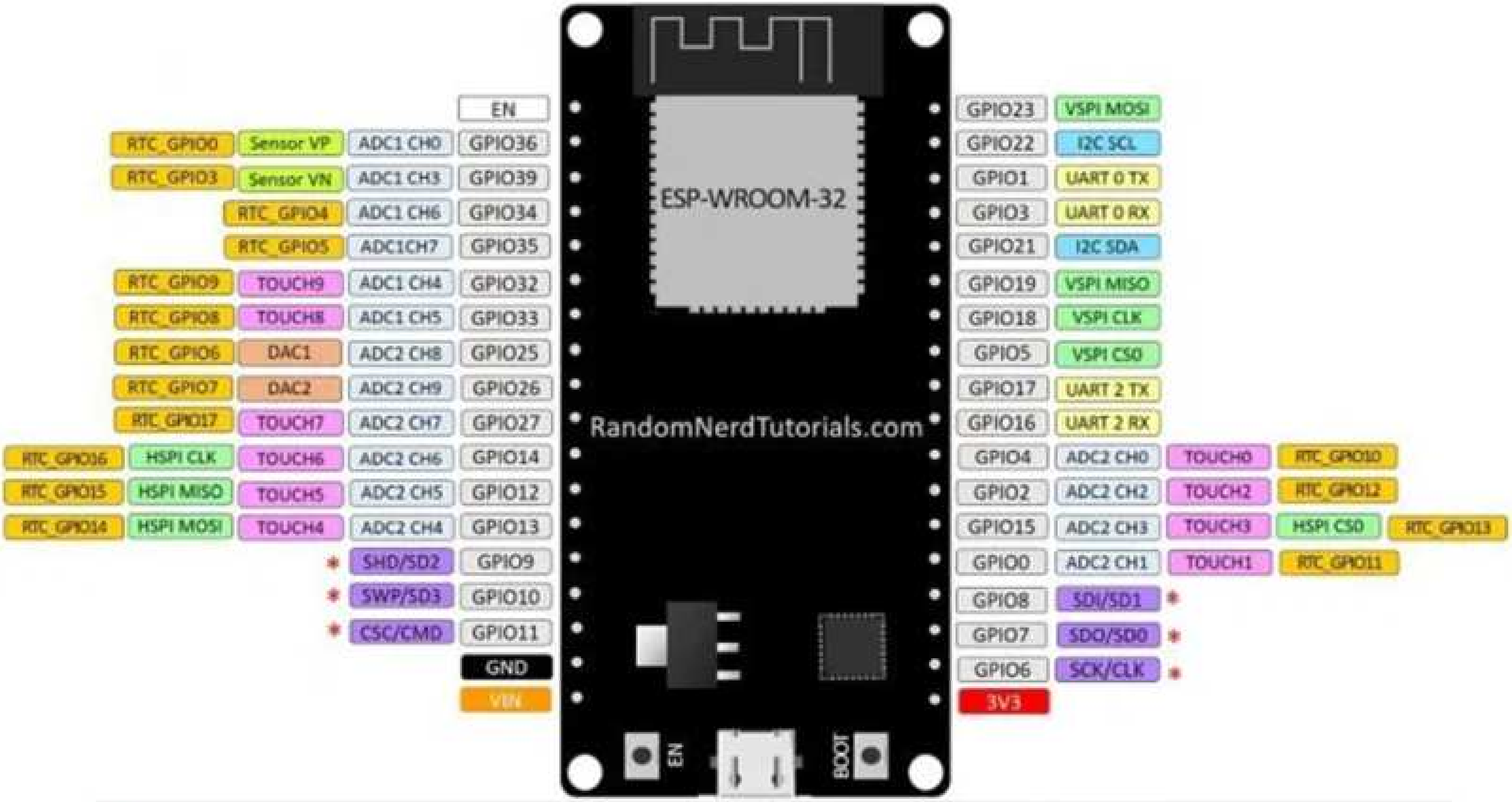
ESP32 DEVKIT V1 – DOIT

version with 30 GPIOs


EN GPIO23 VSPI MOSI
RTC_GPIO0 Sensor VP ADC1 CH0 GPIO36 GPIO22 I2C SCL
RTC_GPIO3 Sensor VN ADC1 CH3 GPIO39 GPIO1 UART 0 TX
ESP-WROOM-32
RTC_GPIO4 ADC1 CH6 GPIO34 GPIO3 UART 0 RX
RTC_GPIO5 ADC1CH7 GPIO35 GPIO21 I2C SDA
RTC_GPIO9 TOUCH9 ADC1 CH4 GPIO32 GPIO19 VSPI MISO
RTC_GPIO8 TOUCH8 ADC1 CH5 GPIO33 GPIO18 VSPI CLK
RTC_GPIO6 DAC1 ADC2 CH8 GPIO25 GPIO5 VSPI CS0
RTC_GPIO7 DAC2 ADC2 CH9 GPIO26 GPIO17 UART 2 TX
RTC_GPIO17 TOUCH7 ADC2 CH7 GPIO27 GPIO16 UART 2 RX
RandomNerdTutorials.com
RTC_GPIO16 HSPI CLK TOUCH6 ADC2 CH6 GPIO14 GPIO4 ADC2 CH0 TOUCH0 RTC_GPIO10
RTC_GPIO15 HSPI MISO TOUCH5 ADC2 CH5 GPIO12 GPIO2 ADC2 CH2 TOUCH2 RTC_GPIO12
RTC_GPIO14 HSPI MOSI TOUCH4 ADC2 CH4 GPIO13 GPIO15 ADC2 CH3 TOUCH3 HSPI CS0 RTC_GPIO13
GND GND
VIN 3V3
EN
BOOT

ESP32 DEVKIT V1 – DOIT

version with 36 GPIOs



* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

ESP-NOW

ESP-NOW is a kind of connectionless Wi-Fi communication protocol that is defined by Espressif. In ESP-NOW, application data is encapsulated in a vendor-specific action frame and then transmitted from one Wi-Fi device to another without connection. CTR with CBC-MAC Protocol(CCMP) is used to protect the action frame for security. ESP-NOW is widely used in smart light, remote controlling, sensor, etc.

ESP-NOW

Connectionless communication protocol developed by espressif

Short packet transmission (up to 250 bytes)

Komunikasi tanpa menggunakan Wi-Fi

Mirip komunikasi 2.4Ghz perangkat low power seperti mouse wireless

System menggunakan pairing tanpa hand shake

Maksimum 20 node open dan 10 node jika dengan enskripsi

No router atau dhcp server

No overhead

No lost time to connect

ESP-NOW ADALAH

Protoocol komunikasi yang cepat yang dapat digunakan untuk pertukaran data kecil (up to 250 bytes) sesame ESP32 board

HEADER FILE

components/esp_wifi/include/esp_now.h

- esp_now_init() Initializes ESP-NOW. You must initialize Wi-Fi before initializing ESP-NOW.
- esp_now_add_peer() Call this function to pair a device and pass as an argument the peer MAC address.
- esp_now_send() Send data with ESP-NOW.
- esp_now_register_send_cb() Register a callback function that is triggered upon sending data. When a message is sent, a function is called – this function returns whether the delivery was successful or not.
- esp_now_register_rcv_cb() Register a callback function that is triggered upon receiving data. When data is received via ESP-NOW, a function is called.

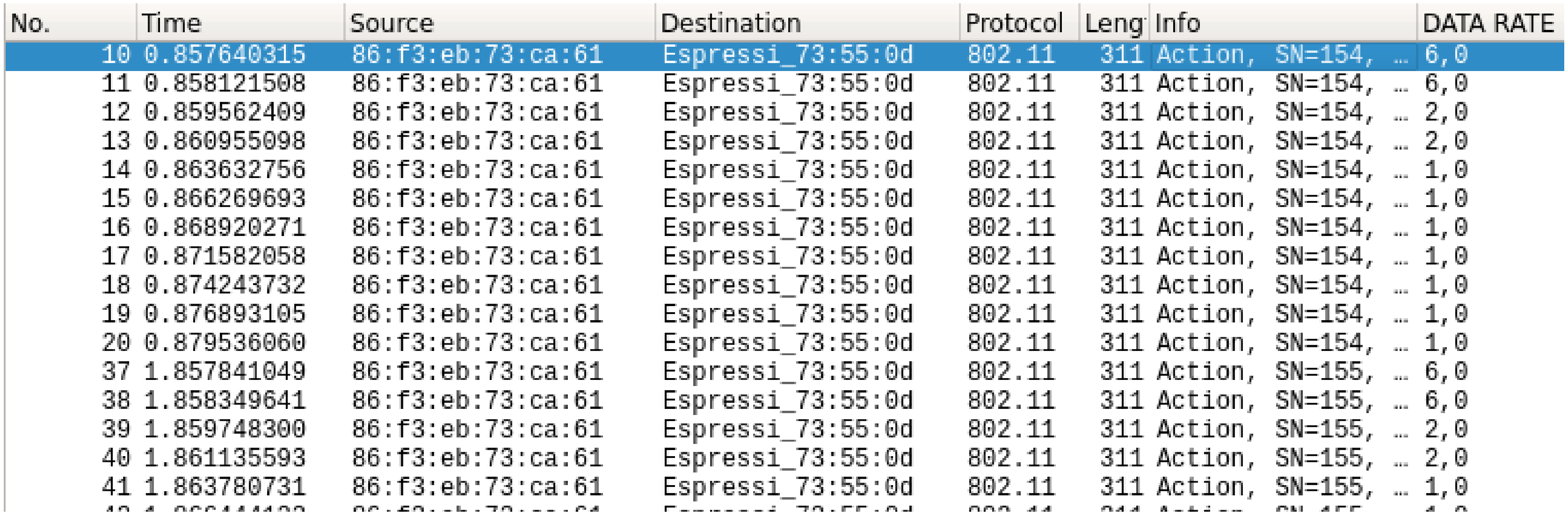
CARA MENGIRIM

- Initialize ESP-NOW;
- Register a callback function upon sending data – the OnDataSent function will be executed when a message is sent. This can tell us if the message was successfully delivered or not;
- Add a peer device (the receiver). For this, you need to know the receiver MAC address;
- Send a message to the peer device.

CARA DARI PENERIMA

- Initialize ESP-NOW;
- Register for a receive callback function (OnDataRecv). This is a function that will be executed when a message is received.
- Inside that callback function, save the message into a variable to execute any task with that information.

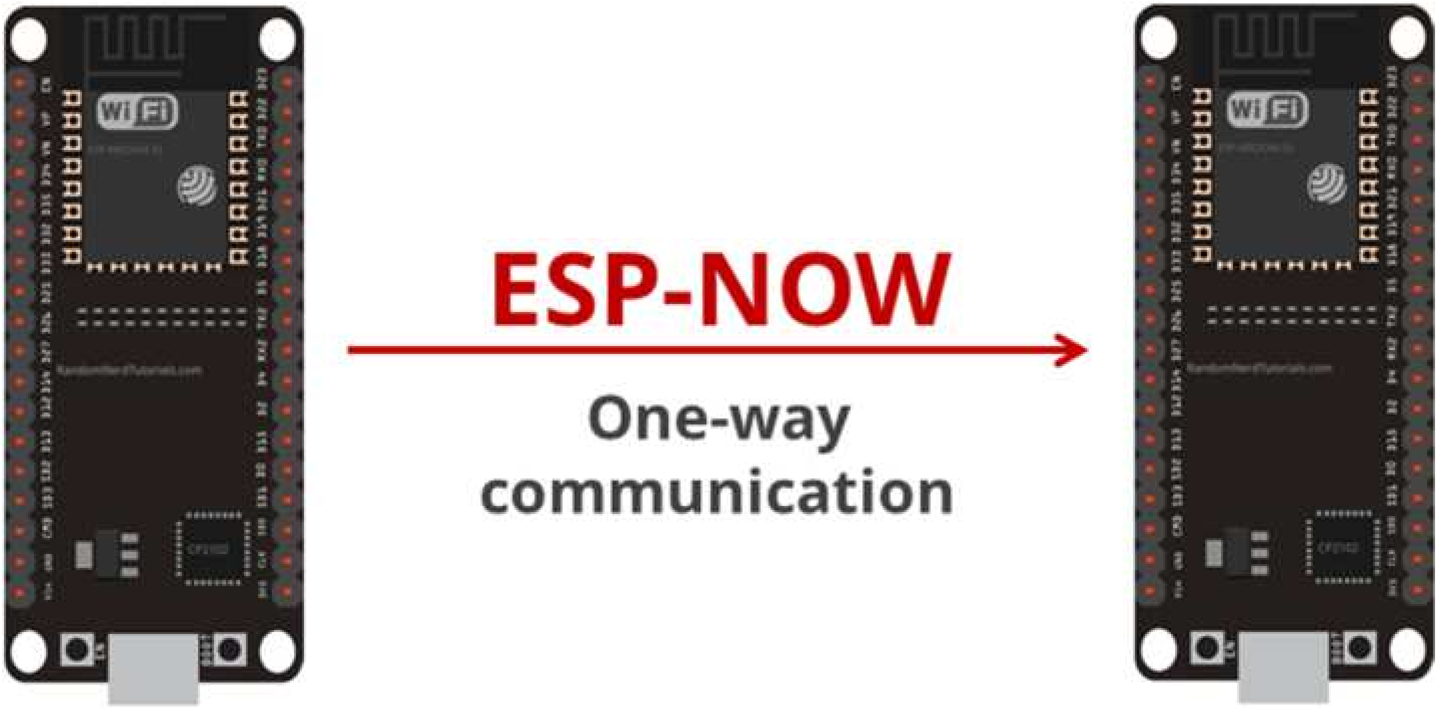
Esp now packet



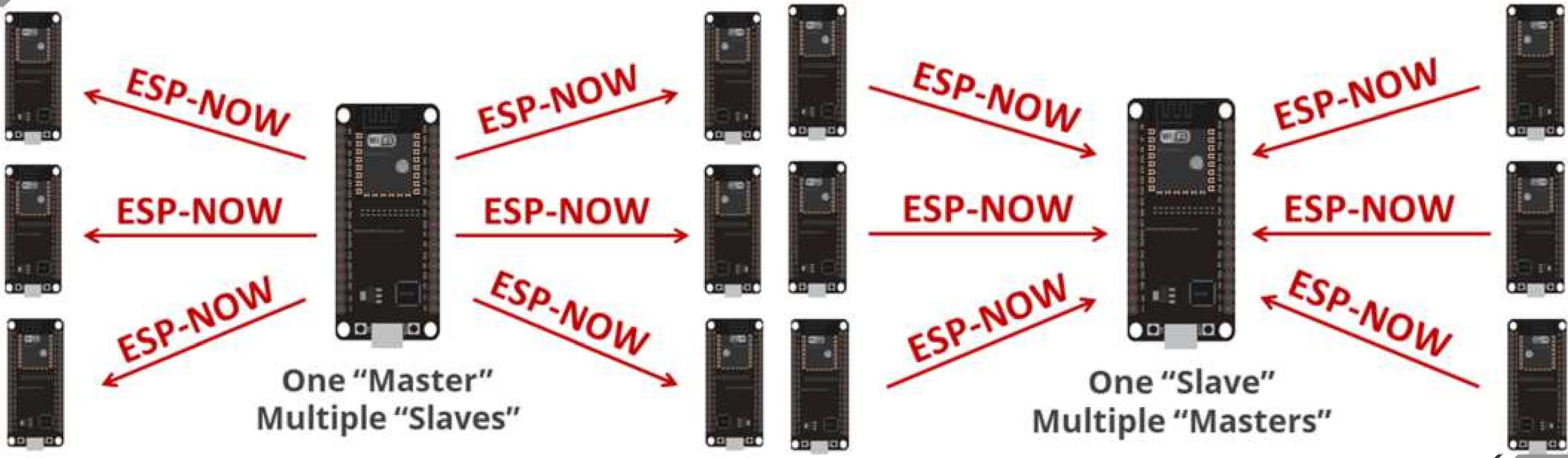
No.	Time	Source	Destination	Protocol	Leng	Info	DATA RATE
10	0.857640315	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	6,0
11	0.858121508	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	6,0
12	0.859562409	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	2,0
13	0.860955098	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	2,0
14	0.863632756	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	1,0
15	0.866269693	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	1,0
16	0.868920271	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	1,0
17	0.871582058	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	1,0
18	0.874243732	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	1,0
19	0.876893105	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	1,0
20	0.879536060	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	1,0
37	1.857841049	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=155, …	6,0
38	1.858349641	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=155, …	6,0
39	1.859748300	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=155, …	2,0
40	1.861135593	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=155, …	2,0
41	1.863780731	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=155, …	1,0

```
-----------------------------------------------------------------------------------------------------------
| MAC Header | Category Code | Organization Identifier | Random Values | Vendor Specific Content |   FCS   |
-----------------------------------------------------------------------------------------------------------
 24 bytes          1 byte                      3 bytes                 4 bytes                    7~255 bytes        4 bytes
```

TEST



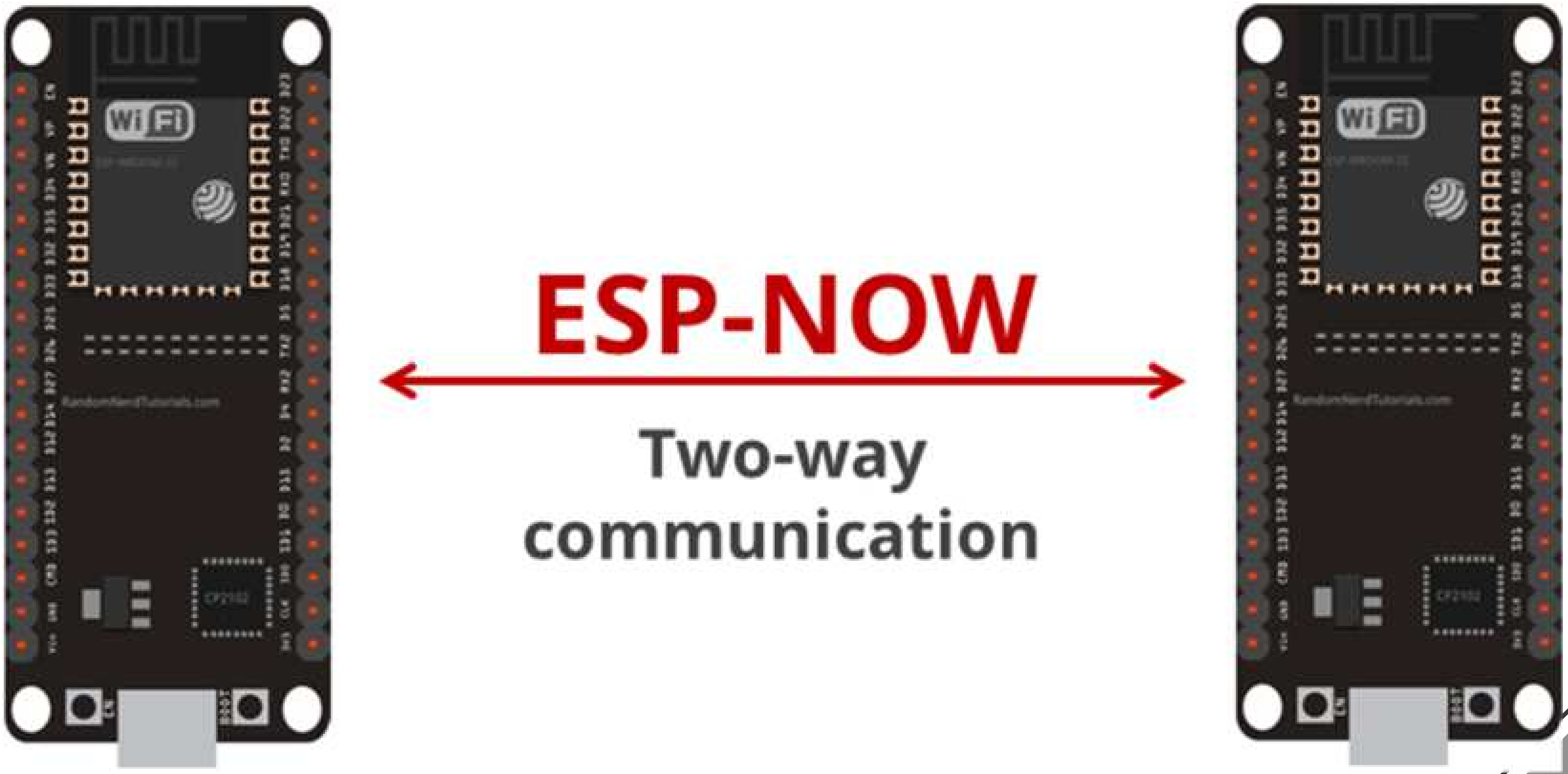
TOPOLOGY STAR



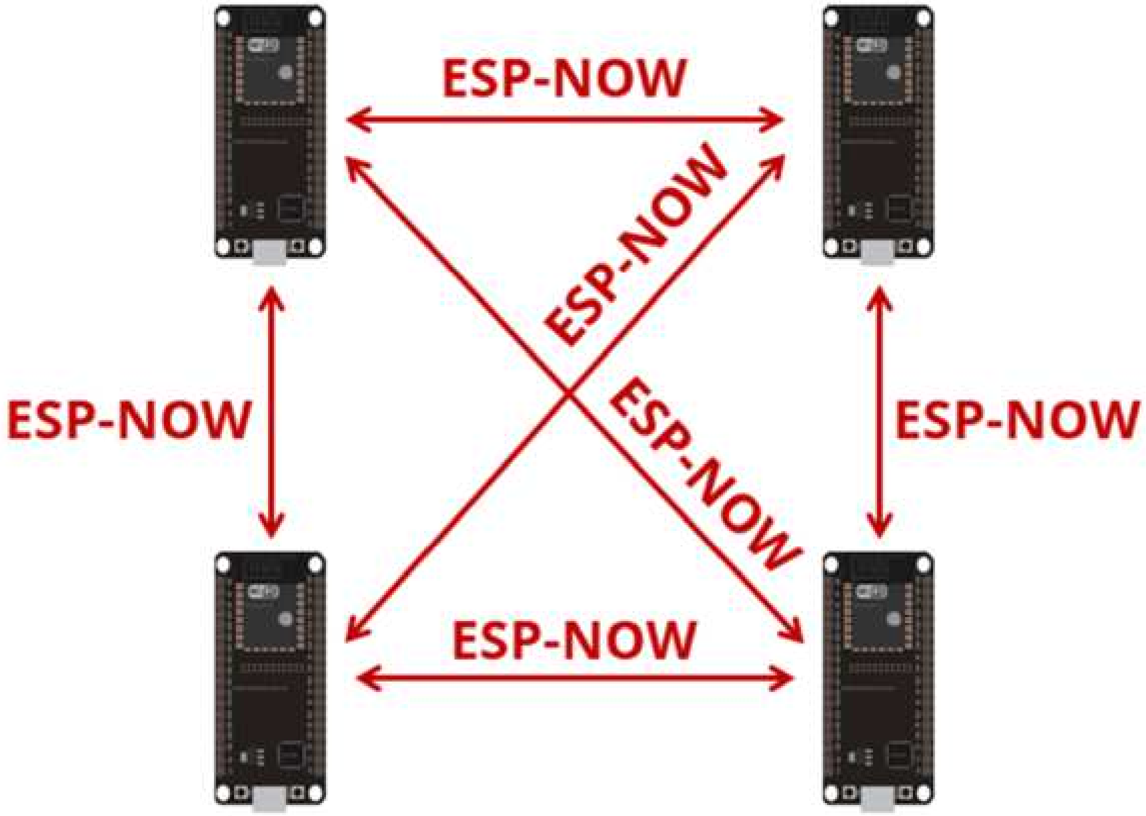
Platform IO

```ini
monitor_speed = 115200
upload_port = COM6
monitor_port = COM6
lib_deps =
    adafruit/Adafruit Unified Sensor@^1.1.4
    adafruit/Adafruit BMP280 Library@^2.4.2
```

TEST



TOPOLOGY MESH



STRUCTURE IN C

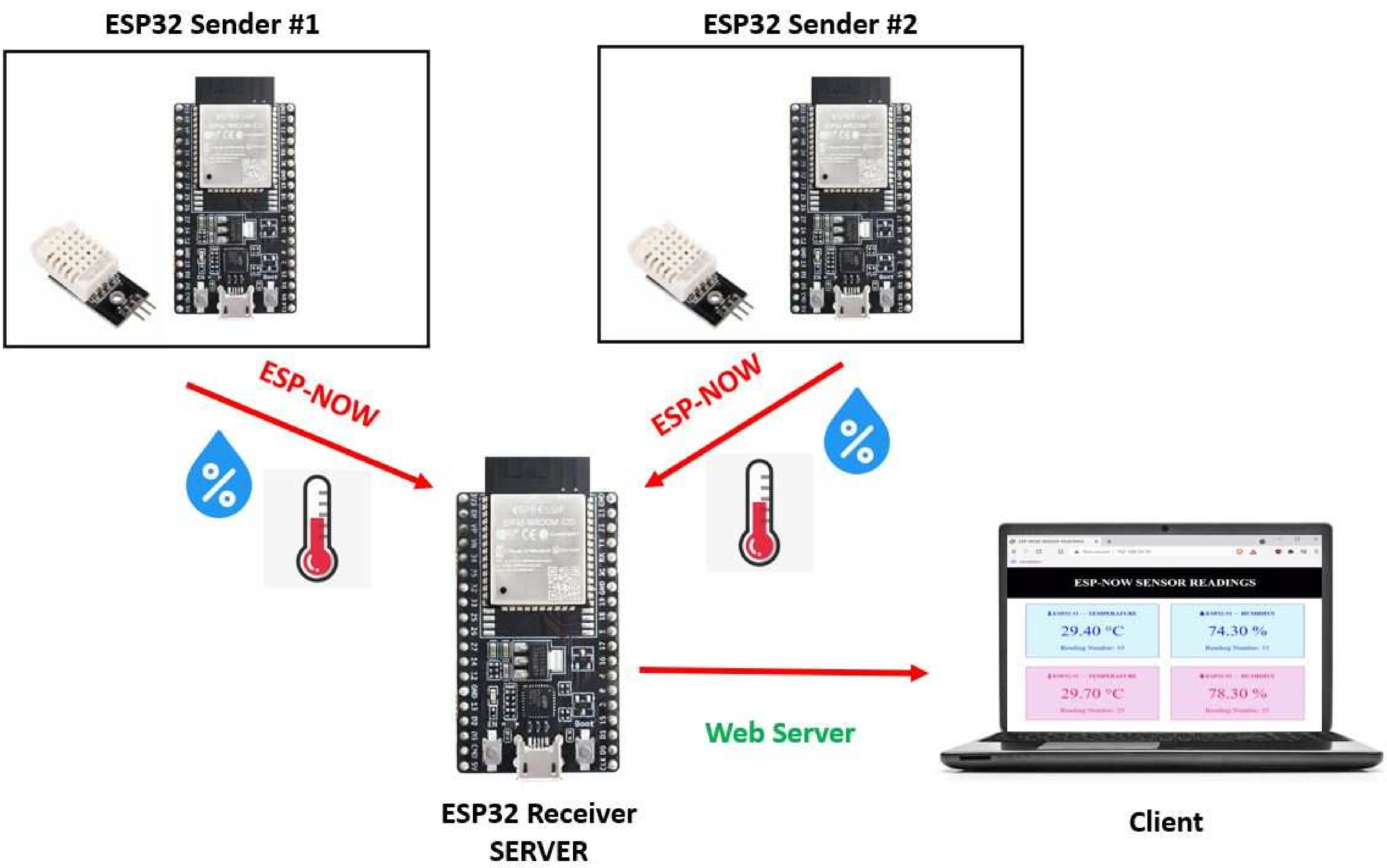
```c
typedef struct struct_message {
    uint8_t id;
    float temp;
    float pres;
} struct_message;

// Create a struct_message to hold incoming sensor readings
struct_message incomingReadings;
```

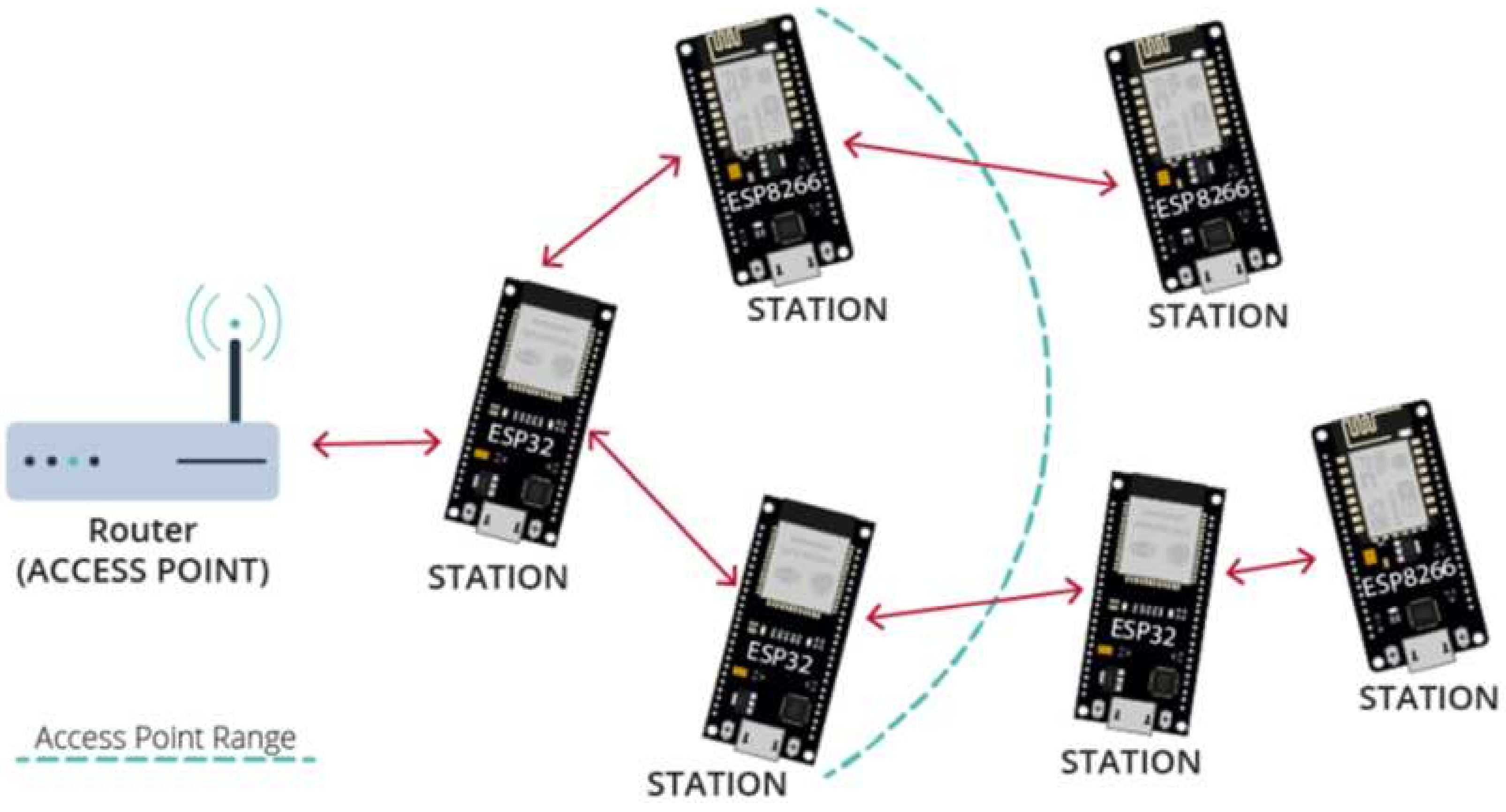
Size (type)

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Json Data point

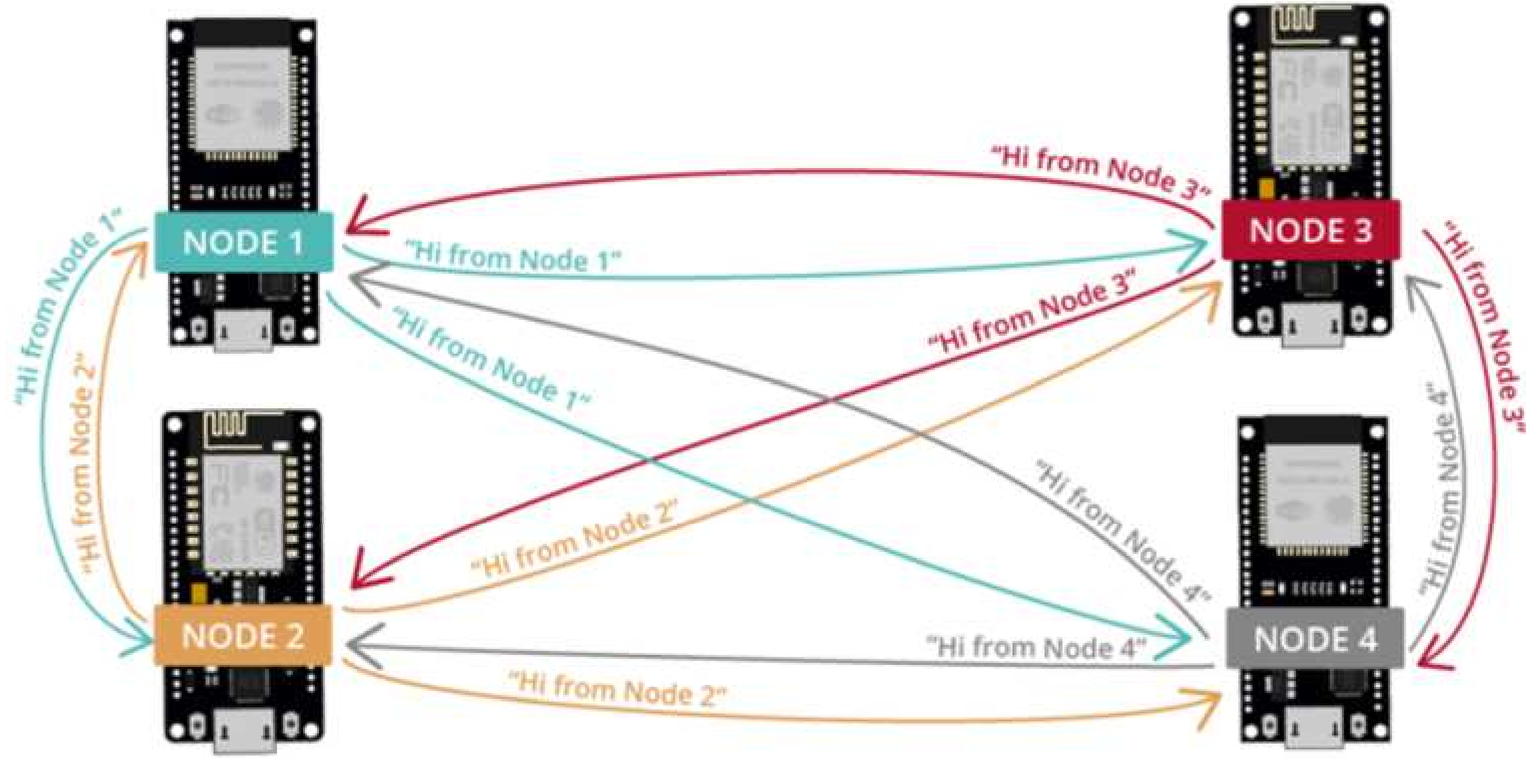
```
{
"temperature": 42.2,
"humidity": 70,
"hvacEnabled": true,
"hvacState": "IDLE",
"configuration": {
                "someNumber": 42,
                "someArray": [1,2,3],
                "someNestedObject": {"key": "value"}
                }
}
```


ESP32 Sender #1
ESP32 Sender #2
ESP-NOW
ESP-NOW
ESP-NOW SENSOR READINGS
29.40 °C
74.30 %
29.70 °C
78.30 %
Web Server
ESP32 Receiver
SERVER
Client

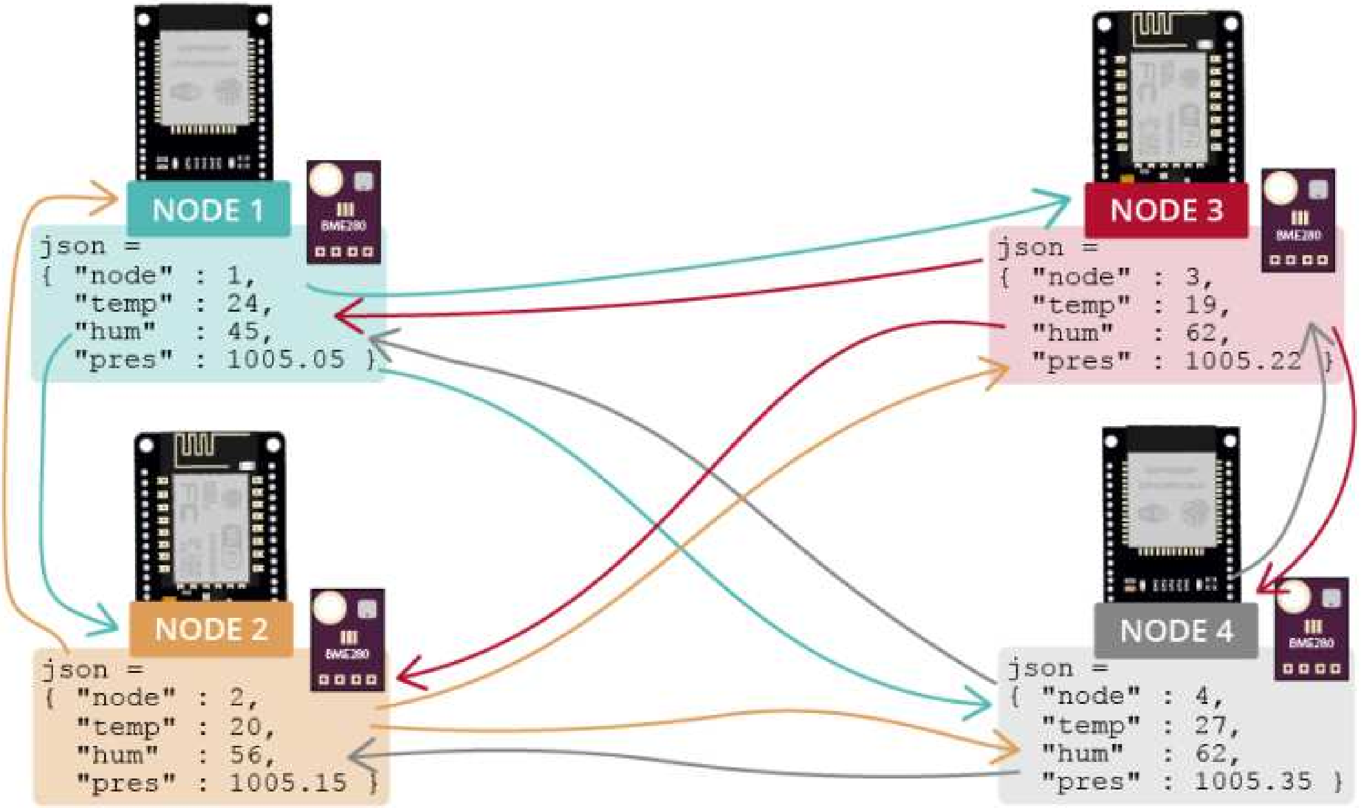

STATION
STATION
Router
(ACCESS POINT)
ESP8266
ESP8266
ESP32
STATION
ESP32
ESP32
ESP8266
STATION
STATION
STATION
STATION
Access Point Range

painlessMesh

Topology mesh ESP32



Topology mesh ESP8266

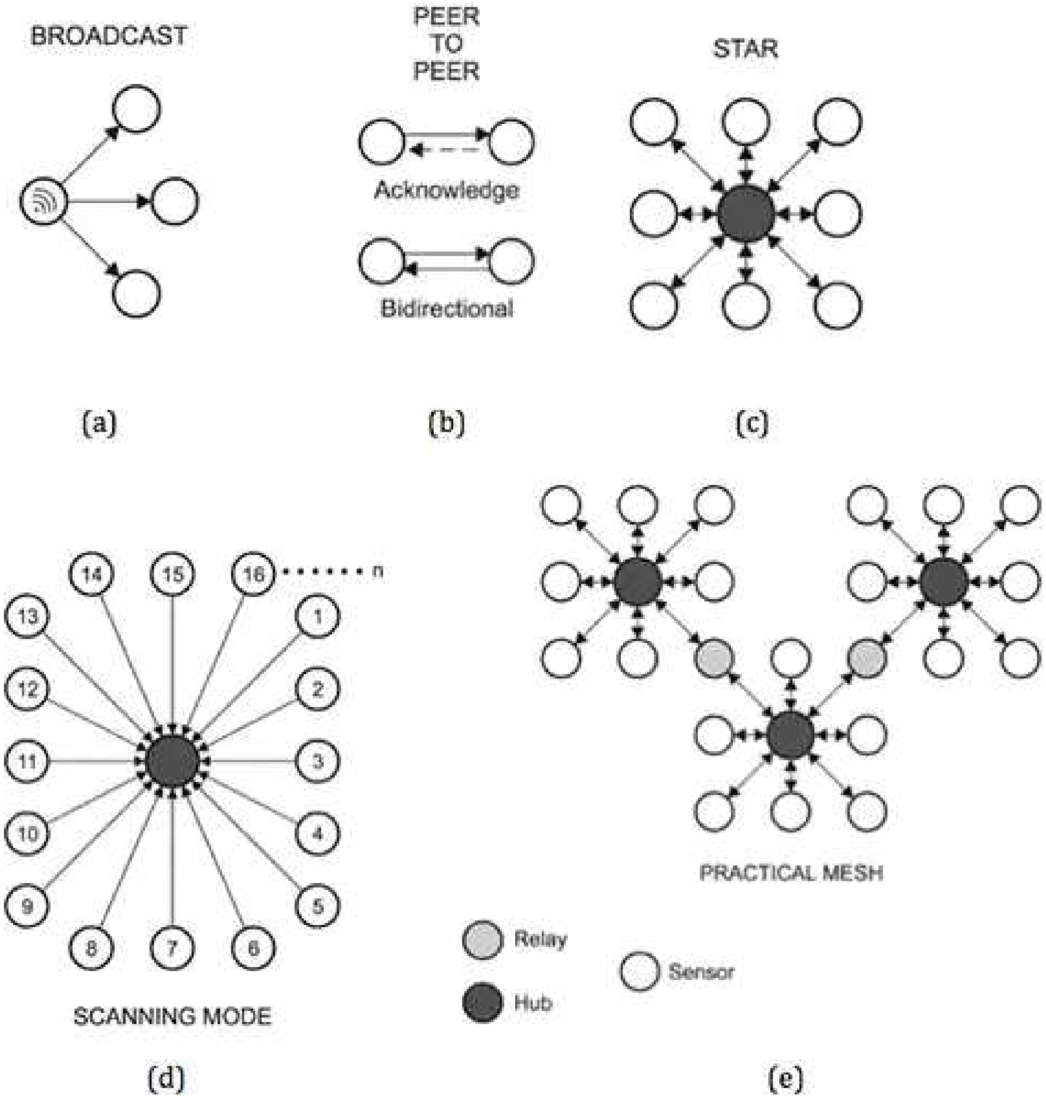


```
json =
{ "node" : 1,
  "temp" : 24,
  "hum"  : 45,
  "pres" : 1005.05 }
```

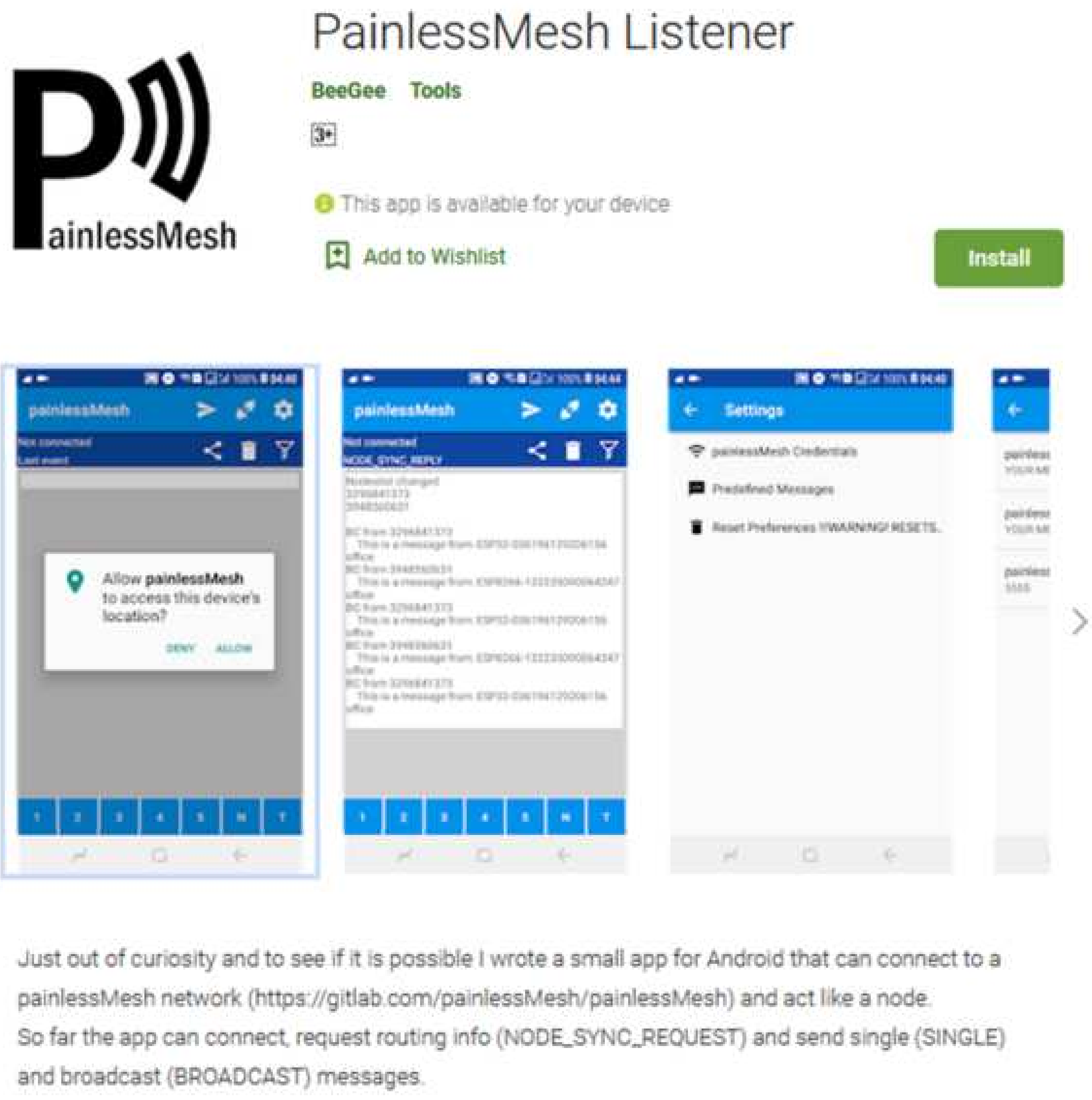
```
json =
{ "node" : 3,
  "temp" : 19,
  "hum"  : 62,
  "pres" : 1005.22 }
```

```
json =
{ "node" : 2,
  "temp" : 20,
  "hum"  : 56,
  "pres" : 1005.15 }
```

```
json =
{ "node" : 4,
  "temp" : 27,
  "hum"  : 62,
  "pres" : 1005.35 }
```



NETWORK TOPOLOGY



PainlessMesh Listener

BeeGee Tools

3+

This app is available for your device

Add to Wishlist

Install

Just out of curiosity and to see if it is possible I wrote a small app for Android that can connect to a painlessMesh network (https://gitlab.com/painlessMesh/painlessMesh) and act like a node.
So far the app can connect, request routing info (NODE_SYNC_REQUEST) and send single (SINGLE) and broadcast (BROADCAST) messages.

PainlessMesh

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.

https://gitlab.com/painlessMesh/painlessMesh

API

```
#include <painlessMesh.h>
painlessMesh  mesh;


void painlessMesh::init(String ssid, String password, uint16_t port = 5555,
WiFiMode_t connectMode = WIFI_AP_STA, _auth_mode authmode =
AUTH_WPA2_PSK, uint8_t channel = 1, phy_mode_t phymode =
PHY_MODE_11G, uint8_t maxtpw = 82, uint8_t hidden = 0, uint8_t
maxconn = 4)
```

API

```
void painlessMesh::stop()
void painlessMesh::update( void )

void painlessMesh::onReceive( &receivedCallback )
        void receivedCallback( uint32_t from, String &msg )
void painlessMesh::onNewConnection( &newConnectionCallback )
        void newConnectionCallback( uint32_t nodeId )
void painlessMesh::onChangedConnections( &changedConnectionsCallback )
        void onChangedConnections()

bool painlessMesh::isConnected( nodeId )

void painlessMesh::onNodeTimeAdjusted( &nodeTimeAdjustedCallback )
        void onNodeTimeAdjusted(int32_t offset)
void onNodeDelayReceived(nodeDelayCallback_t onDelayReceived)
        void onNodeDelayReceived(uint32_t nodeId, int32_t delay)
```

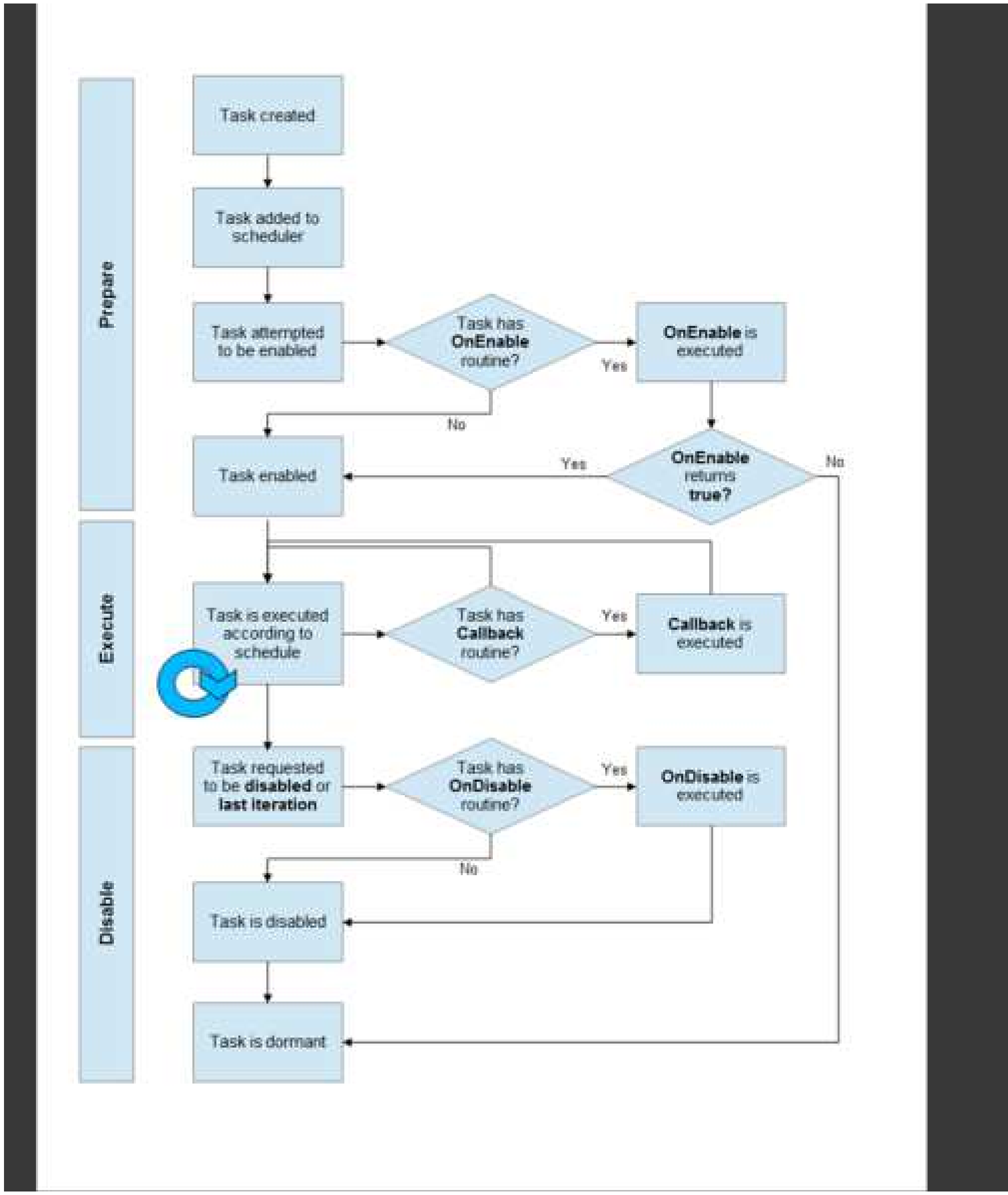
API

bool painlessMesh::sendBroadcast(String &msg, bool includeSelf = false)
bool painlessMesh::sendSingle(uint32_t dest, String &msg)

String painlessMesh::subConnectionJson()
uint32_t painlessMesh::getNodeId(void)
void painlessMesh::stationManual(String ssid, String password, uint16_t port, uint8_t *remote_ip)

NO DELAY

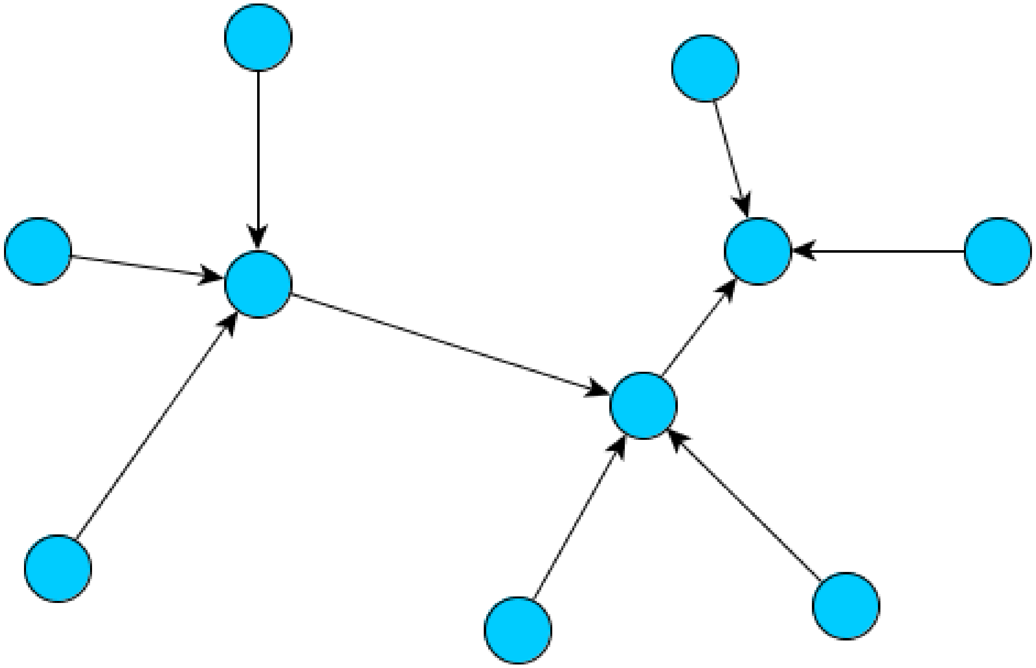
But TaskScheduler

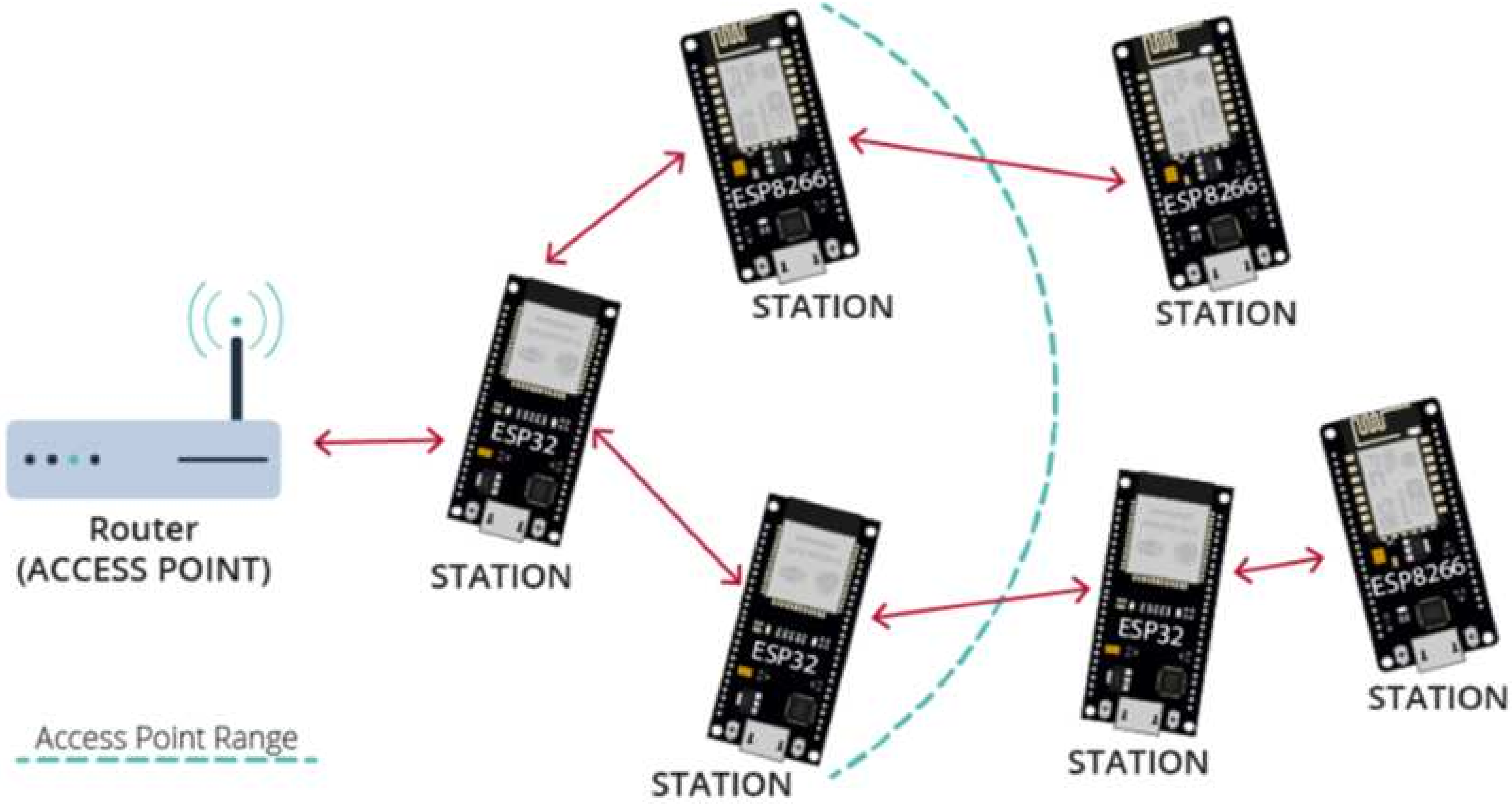


MeshNet

```
#define   MESH_PREFIX     "meshnet"
#define   MESH_PASSWORD   "meshnet123"
#define   MESH_PORT       5555
```

Network Layout




STATION
STATION
ESP8266
ESP8266
Router
(ACCESS POINT)
ESP32
STATION
ESP32
ESP32
ESP8266
STATION
STATION
STATION
STATION
Access Point Range

THANKS

Do you have any question?

hasbiida@gmail.com

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, infographics & images by Freepik