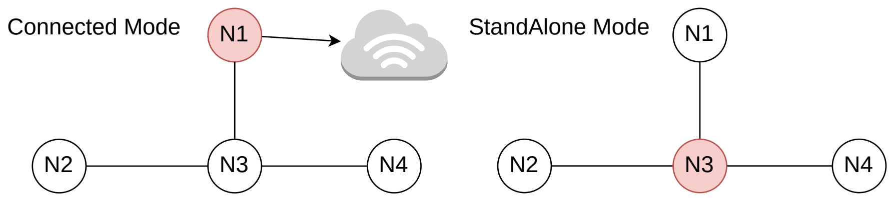
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2022
http://excel.fit.vutbr.cz

Dynamic Mesh network in Micropython on ESP32

Jindřich Šesták



Abstract
The aim of this project is to implement a mesh network protocol on ESP32 microchips in MicroPython. It mainly focuses on the functioning of mesh in two modes, with connection to the Internet and without it. This thesis was ordered by Espressif company for improving and discovering new ways of mesh networking. The solution of this mesh network uses two network protocols. First, the ESP-NOW protocol offers low power consumption and doesn't need any network connection. The second is the common WiFi protocol which is used for data transmission. WiFi links are formed between ESP32 nodes and one of the nodes can even be connected to the Internet and offer a connection to the whole mesh. With full functionality, the mesh should be light weighted and will connect multiple nodes. It is possible to run user applications like light control on ESP32 boards on top of the mesh using WiFi. With WiFi, it is possible to transfer up to 1500 Bytes of data for applications. The work is still in progress. In this project, there are designed new innovations to ensure the formation of a structure in the mesh. The problem of how to select a root node in an environment without the WiFi Access Point (Router, AP) is presented.

Keywords: ESP32 — ESP-NOW — Mesh network — Mesh — Espressif — MicroPython — Asyncio

Supplementary Material: MicroPython ESP32 Mesh — ESP-WIFI-MESH video — Github ESP32 ESP-NOW

*xsesta05@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

The motivation for this project is to develop an easy mesh network. MicroPython programming language is very popular and it is expected that the programming community will have interest in this project for use in homes. Existing solutions are not versatile enough, they often offer mesh networks only in environments with WiFi AP or only without it. This work aims to develop a universal solution mainly for home use for IT enthusiasts and hobbyist.

The assignment from the company specifies use of MicroPython as the company aims to meet the possibilities and limitations of MicroPython on ESP32 boards. MicroPython should offer easier reprogramming and additions for more specific use-cases. It is also required to use proprietary ESP-NOW protocol. Because ESP-NOW protocol is currently supported only on ESP8266 and ESP32 microchips, the development aims only for ESP32 boards and portability on different platform is not currently possible.

A Mesh network is a network in which every node communicates with each other. This can be achieved either by flooding the messages through broadcast or by unicast routing. Solutions should be automatic and self-organising, meaning that mesh will form its connection without prior configuration. Dynamic mesh networks should be able to act on changes in the mesh. Meaning the addition of nodes in the existing mesh is possible and the mesh will reorganise on node failures, which is called self-healing. A Mesh network that routes the traffic needs a root node, which manages the mesh and is often connected to the Internet.

Right now, there are three mesh network solutions working on microcontrollers ESP32 [1]. First, ESP Bluetooth Low Energy Mesh is based on Bluetooth technology. In this mesh, nodes are connected to as many as they possibly can. The mesh is without any structure and uses flooding as the only way of transmitting messages. PainlessMesh is a library in C language that offers small and fast deployment of the mesh using a WiFi interface. Nodes form a structure therefore this mesh is not fully connected therefore routing to reduce the number of packets is used. The third solution is ESP-WIFI-MESH, which also uses a WiFi interface in mesh and routes packets. This solution is more reliable and faster. These solutions are described in detail in section 2.

Our solution uses a combination of two technologies. ESP-NOW protocol [2] is used to collect information about nodes in the mesh. Prior to WiFi connection and transmitting of data, the mesh is formed based on the collected information from ESP-NOW. The mesh requires a root node. After the root node is elected, it manages and directs the further forming of the mesh. In the process of formation, nodes connect to each other through mentioned WiFi. Node is connected to only a subset of nodes it sees and the aim is to form connections with nodes with the best signal.

This project brings another solution in mesh networks using affordable ESP32 microcontrollers. With the use of MicroPython, it aims to become more popular for community projects and spread to more users. A new way of forming the mesh is presented. Additionally, this solution can work either with connection to the Internet or without it, while there is no need for manual reconfiguration. The mesh is formed without any prior setup except key credentials.

2. Previous works

Programmers from Espressif company have already been working on mesh networks using ESP32 microcontrollers and they have come up with three official solutions.

The ESP Bluetooth Low Energy MESH [3][4] is optimised for large scale networks. Bluetooth standard offers connectivity to many different devices with different Bluetooth versions. The use of Bluetooth interface keeps the WiFi Station interface free to connect to the WiFi AP, however, they cannot be Access Points themselves. This means that the node can be part of the

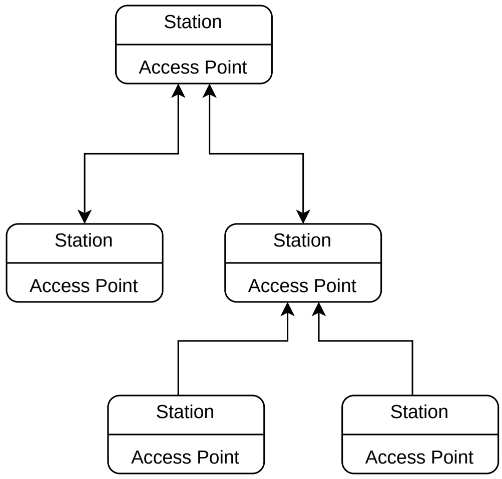


Figure 1. ESP32 microcontroller has two independent WiFi interfaces. Nodes can combine these two interface to create network structure or hierarchy.

mesh while still being connected to the Internet, while the mesh as a whole is not connected. The mesh is not structured and there is no root node and messages are broadcasted to everyone. A special node called Relays can forward and broadcast messages further to the distant nodes. There is a need of provisioning the node with the credentials, which is done by smartphone with a mobile application. The provisioning is needed to perform on each node.

The PainlessMesh library [5] written in C++ provides an easy solution for small mesh network projects. It uses both WiFi interfaces, Access Point and Station mode. Nodes can connect to other nodes' Access Point interface while still acting as Access Point for other nodes as shown in figure 1, therefore creating a star or tree-like structure. It is ensured that there are no loops in the structure. Nodes exchange topology information with each other hence every node knows the whole topology. All nodes are equal in this mesh and have the same information. As nodes connect to the Access Point interface with the best RSSI signal there is no need for the root node, but it is allowed and recommended to manually set the root in the mesh. Not setting the root node can lead to a bad topology and can lead to the creation of several small and isolated meshes like in figure 2. However, PainlessMesh cannot form a connection to the outside network like the Internet, because it is not known which node is to become the first node that has no upstream connection and therefore it is not known which node has a free Station interface. Messages are in JSON format for better human readability and clarity.

The third solution is the ESP-WIFI-MESH [6] project built on top of WiFi protocol while nodes use

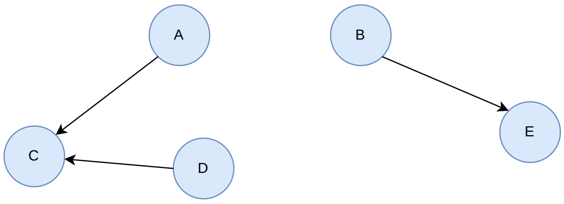


Figure 2. PainlessMesh with automatic root election can cause inconvenience mesh topology. It can lead to creation of several independent meshes, which is not desirable.

both WiFi interfaces same as in the PainlessMesh solution. But this solution requires the presence of WiFi AP in the range of at least one node because this protocol is used to offer Internet connections for the remote nodes outside the range of WiFi AP. Based on the strength of the signal RSSI to the WiFi AP, the root node is selected or it can be set manually. Only the root is connected to the WiFi AP while it offers a WiFi connection to the other (child) nodes and thus enlarges the Internet connection to the other nodes. The mesh forms a tree structure in which nodes are not equal. Nodes higher in hierarchy knows more about the topology than the leaf nodes with the root node that knows the exact whole topology. Nodes use routing of messages, therefore, reducing the load on the network. Nodes connect to the parent nodes based on two conditions. Firstly node takes into consideration the depth of the possible parent and chooses the shallowest one, which reduces the depth of a tree. Secondly, it chooses parents with the fewest child nodes already connected, aiming to more balanced trees.

3. Proposed Mesh network using ESP-NOW and WiFi

The goal is to create one mesh protocol that can function in two modes autonomously, connected to the WiFi AP and stand-alone without the Internet connection. This versatile approach is lacking in existing solutions. As for any mesh network, it should implement self-healing and self-organising features for autonomous functioning.

For a collection of information about nodes and for adding nodes into the mesh network, the proprietary ESP-NOW protocol is used. This protocol is built on top of IEEE 802.11 Management Frames. For topology updates and application data transmitting, a common WiFi protocol is used. This means that for periodic messages the mesh uses low power protocol ESP-NOW and for bigger data transmitting the WiFi is used. With these two wireless protocols, the mesh uses both broadcast flooding in ESP-NOW and unicast routing for WiFi.

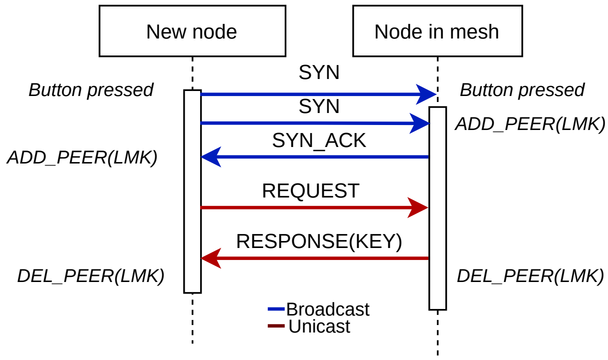


Figure 3. Mesh Protected Setup (MPS) for exchanging security key for message signing uses broadcast for registering a peer. Secure key is transfered via ciphered unicast. After this exchange new node can participate in mesh.

For easy and not time demanding provisioning of nodes, we proposed the Mesh Protected Setup (MPS) method of adding nodes to already installed mesh. There is still a need to manually set the key credentials for message signing and encryption but only on the first node. The addition of nodes to the mesh is done by pressing the button on ESP32 boards. Using handshake both new node and node with credentials register each other with predefined LMK security key for encryption in ESP-NOW protocol and securely exchange key credentials. They are registered only for this exchange process due to the limit of registered devices, therefore one node in mesh can one by one send credentials all the other nodes. The message is considered valid and accepted for further processing if and only the HMAC [7] hash signed with received key credentials matches. Otherwise, messages are dropped.

Nodes with key credentials send periodic updates through broadcast. Receiving nodes update their node database and retransmit these advertisements. This way nodes collect information about all the nodes with credentials, ergo nodes in the mesh. If nodes didn't receive advertisements about certain nodes for some amount of time, it considers him disconnected and wipes out the record from the database. The ESP-NOW protocol needs a WiFi interface to be active, because of that nodes can see WiFi AP interfaces networks of other nodes. A node can compare these WiFi networks it sees with his database of nodes and from the subset where MAC addresses match it can compute the strength of the signal to his neighbours. It also sees the WiFi Access Point of the router with an Internet connection and RSSI signal to it. Based on these two values, the root is elected. The root is the one with the best signal to WiFi AP router similar to the ESP-WIFI-

MESH solution mentioned earlier. But when there is no WiFi Access Point presented in the environment, it doesn't take it into the account. Instead, it elects root based on the signal to its neighbours. We can assume, that this value represents the centrality of the node, how much in the centre of the mesh is it. A node that sees only one neighbour would not be a good root. Instead, a node most in the centre of the mesh is selected.

After the root node is elected he became the moderator of the mesh. It sends to the close neighbours with good signal credentials and they connect to its WiFi AP interface, therefore becoming its child nodes. After that, the root node and the child node communicate via WiFi protocol while still advertising in ESP-NOW. Then the child nodes report to the root node about their close neighbours with a request to claim them as their child nodes. The root node collects these requests from all the child nodes and allows the node with the best connection to claim a new node as its own. In the same spirit, the formation of tree structure continues completing the process of self-organising. The root node also sends periodic topology updates to everyone in the structure, thus every node knows exactly where in topology it stands. When some node is detected inactive through the loss of its WiFi signal, it is considered a failed node. Every node in topology is informed about this fail down and descendants of this failed node can see that their connection is lost. They disconnect from it and wait for another node in topology to claim them, thus ensuring a self-healing process as can be seen in figure 4.

4. Implementation with asyncio in MicroPython

The project is written in MicroPython port for ESP32 devices according to assignment. MicroPython is an implementation of a Python3 programming language optimised to run on microcontrollers. Some of the core Python libraries are part of this language, but it also includes modules that allow low-level hardware access to the programmer like library *machine*. For this project, the essential libraries are *espnow* for ESP-NOW protocol operations and library *network* for directing and managing WiFi network interfaces.

In MicroPython and Python, there are several ways to achieve concurrency computing, like multi-core parallelism or threading. A new and more suitable way is using module *asyncio* [8]. In this implementation, coroutines (==tasks) are scheduled to run in an overlapping but non-blocking manner using cooperative multitasking on single-core processors, similar to threading. But the biggest advantage over threading is, that in asyncio the programmer himself decides when and where should one task yield its resources like CPU to the other tasks. Furthermore, the task cannot be interrupted in the middle of computing unless it wants to, therefore there is no need to worry about locks, mutexes, race conditions and deadlocks, unlike threading. Asynchronous event loop manages tasks and schedules tasks to be run.

The project consists of several modules and supporting files. Functionality is achieved by dividing into two main cores. One core (*espnowcore.py*) manages ESP-NOW protocol, processing of messages, exchange of credentials and root election. This core is initialised and executed within second core (*wificore.py*) which takes care of WiFi messaging, tree topology formation.

User can build application in which he initialises WiFi core and run its *start()* function. This function further invokes full functionality of mesh and both ESP-NOW and WiFi cores. User has to define class with at least two functions for his application. One for sending or invoking application messages and one for processing them (must be named exactly *process()*). Application messages are in JSON predefined format inside class *AppMessage(srcMac, dstMac, message)*.

After startup of the node, the WiFi core *start()* function must be called. It invokes ESP-NOW core and waits. In ESP-NOW core advertisements and MPS processes are executed. In MPS procedure it is allowed for 45 seconds to send and process unsigned messages only regarding MPS. After node receives credentials for signing in MPS procedure, it can start processing and sending all ESP-NOW messages. After 29 seconds without any new advertisement additions to a database of nodes, we assume all node are started and root election can take place.

WiFi core waits for node to be a root (only one node) or for node to receive AP credentials from his parent node. In case of root node, he creates tree topology. In case of received AP credentials, it connects to parent ESP32 AP and creates socket connection to parent node. After that node waits until he receives tree topology update from parent node. The root node doesn't have to wait because he has the tree topology which he previously created. Consequently, the node configures its AP WiFi interface, open and listens on socket for child nodes, this also invokes function for claiming children which sends node's randomly generated WiFi AP credentials (ssid, password) to child nodes so they can connect to him.

Through socket connections root node sends topol-

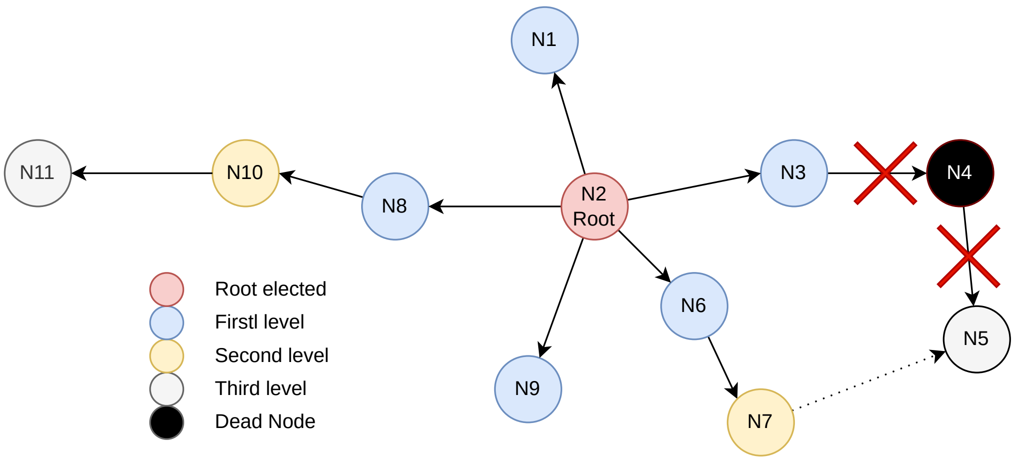


Figure 4. Without WiFi AP, root is the most central node. If some node breaks down, everyone is informed and descendants nodes can be claimed by another node.

ogy updates to his child nodes and they send it to their child nodes and so on, so every node in the mesh has the tree topology. When nodes receives new connection from child node, it reports change to the root node in order to actualise the tree topology. The same occurs when child node fails down. When parent node fails down the node must hard reset itself, because socket interface has problem with terminating and creating new socket on same port for different parent node. With hard reset the node automatically (must be defined in *main.py*) starts anew and can be claimed by some new parent.

Messages in ESP-NOW protocol are packed with module *struct* into bytes to save space in the packet because ESP-NOW allows transmitting only 250 bytes in one message. On the other hand, WiFi packets are in JSON format for human readability and offer up to 1500 bytes, which was inspired by Painlessmesh implementation. The JSON format is also better for representing the topology hierarchy in topology updates. User-defined application is to use WiFi communication with predefined JSON format. Due to ESP-NOW protocol still not being officially supported in MicroPython it is not recommended to change the behaviour of ESP-NOW part of the program.

Periodic advertisement updates in ESP-NOW protocol are send every 5 seconds, but updates from other nodes are retransmitted further to the mesh only every 13 seconds to reduce the load in the network. The root node is elected after 29 seconds with no database changes with this equation:

$$centrality = \sum_{x_i \in X} \frac{1}{\sqrt{|RSSIx_i|}}. \qquad (1)$$

WiFi topology updates are sent every 7 seconds. Time constants are selected experimentally in a hope that prime intervals are less likely to interfere with one another to cause network overload or overload in processing the messages with limits of ESP32 boards.

For message signing the HMAC library together with the SHA256 hash function is used. The digest is inserted behind the message to verify the source of the message belongs to the same mesh network.

Some critical tasks are run using *try-exception* command and exceptions are catched. Severe exceptions lead to machine hard reset and start of the node anew in order to overcome failed coroutines and undefined behaviour of the node.

5. Limitations and Drawbacks

The LMK and PMK key for secure ESP-NOW communication during MPS process of exchange signing credential must be predefined in JSON configuration file because these values has to be the same on both devices.

WiFi AP of ESP32 boards support by default 4 nodes connected and can be improved up to 10 nodes connected as child nodes. There can be by default only 10 sockets open but this can be improved up to 32 sockets. Nevertheless, maximum number of child nodes is 10 nodes.

WIFI AP and STA interface on ESP32 boards operates on same WiFi channels. Therefore ESP-NOW is on the same channel as well. This means that even though there are several WiFi AP in the mesh, everyone must operate on the same channel. When the mesh is connected to the WiFi router, the mesh must be on the same channel as WiFi router. According to the standard 802.11g, the WiFi channel has speed up to 54 Mbps. Node periodically sends 52B of ESP-NOW advertisements every 5 seconds and for every other node also 52B every 13 seconds. Also 66B + 42B x N of WiFi topology updates every 7 seconds are sent to every child node, where N is count of node in the mesh. The assignment defines at functionality on at least 10 (N=10) nodes in the mesh therefore the bit rate on the channel from all the nodes is as follows:

$$rate = (52*12+52*N*4,6)*N + ((66+42*N)*8,5)*(N-1)$$

$$rate = (624+239,2N)*N+(561+257N)(N-1)$$
$$rate = 67339B*8 = 538712bpm = 8978bps$$

This values represents traffic on the mesh during runtime after the mesh is settled and working.. In the equation, there is not counted with MPS protocol, exchange of WiFi AP credentials to child nodes and update of topology when tree is changed. Additional traffic will appear with user application.

On ESP3232-Buddy which have been provided by the company for development, there is byd default 64 KB of RAM available for MicroPython. The mesh takes about 50 KB of RAM. When importing a file in MicroPython it takes RAM memory in Heap and can lead to allocation errors. This can be overcome by pre-compiling files into .mpy files which reduces overhead while importing.

Right now the root election is set statically because there is a problem with WiFi scanning networks, which takes between 2 and 2,5 seconds. Even though in MicroPython WiFi scan is defined in another thread, in RTOS it runs in the same thread as receiving of incoming packets, therefore, it blocks the receiving. Currently there is an effort to reimplement MicroPython port to be able to scan only one channel which would reduce blocking time to only 120 mili seconds. It would significantly reduce the number of dropped packets.

Because the problem with the scanning, the claiming of the child nodes with best signal was changed. In advertisements messages there was added TTL flag. And node can claim only nodes within the range using TTL flag.

After 29 seconds of no new addition through MPS into the mesh, the root is elected which takes some amount of time. Then at worst every 7 seconds new layer of child is added. In total 29+7*L seconds where L is the height of the tree. Be aware that connection to WiFi AP of parent node takes unknown amount of time.

At the moment there haven't been any power consumption measurements to know how long can device operate on battery, yet.

6. Conclusions

This paper discusses mesh networks and peeks into the existing solutions in mesh networking on ESP32 microcontrollers. The proposed new mesh network protocol is presented and its main goal is to create an independent mesh network in environments both with and without WiFi AP presence. It uses a combination of ESP-NOW proprietary protocol with broadcast communication and WiFi protocol which uses unicast.

The implementation uses MicroPython with asyncio library. Asyncio allows programmers to implement concurrency in the non-blocking state with a lower overhead than threading. The programmer also decides and directs switching between tasks as he wants. Therefore asyncio supports asynchronous I/O operations like waiting for and reading packets.

Our solution invents a new way for the root node election in environments without WiFi AP. Therefore our mesh network has always a root node and creates a tree structure of nodes in the mesh.

This project can be used as a base layer for IoT systems. Programmers can develop their own application that will operate on top of the mesh communication. Mesh uses only WiFi and there is no need for new transmitting technology thus ESP32 are affordable and ideal devices for home or small projects. But ESP32 are also very versatile and MicroPython allows the creation of complex projects. There is a need to configure a key for security and desired WiFi credentials for connecting the mesh to the Internet.

This project can be improved by adding a configuration layer in form of a simple HTTP server. Instead of manually configuring mesh in program file or JSON configuration file and uploading to the node, this webserver would allow setting and manual configuration during run time.

Acknowledgements

I would like to thank my supervisor Kamil Malinka for his advice and guidance. Big thanks belong to my technical supervisor Sergey Silnov from company Espressif for his ideas and other useful input during the development of this project.

References

[1] Systems Espressif. *ESP32 Series Datasheet*. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[2] Systems Espressif. Esp-now. https://docs.espressif.com/projects/esp-idf/

en/latest/esp32/api-reference/network/esp_now.html.

[3] Systems Espressif. Esp-ble-mesh. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/esp-ble-mesh/ble-mesh-index.html.

[4] Martin Woolley. *Bluetooth Mesh networking*, 2020. Bluetooth SIG.

[5] Edwin van Leeuwen Coopdis, Scotty Franzyshen. *PainlessMesh*. GitLab, 2019 [Online]. https://gitlab.com/painlessMesh/painlessMesh/-/wikis/home.

[6] Systems Espressif. Esp-wifi-mesh. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/esp-wifi-mesh.html.

[7] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Rfc2104: Hmac: Keyed-hashing for message authentication, 1997.

[8] John Hunt. *Concurrency with AsyncIO*. Springer International Publishing, 2019. 407–417 p. ISBN 978-3-030-25943-3.