



**POLITECNICO  
DI TORINO**

**POLITECNICO DI TORINO**

Master Degree course in Electronic Engineering

Master Degree Thesis

# **Development and Enhancement of a Mesh Network on Microcontrollers for Vehicles in Challenging Environments**

**Relatore**

Prof. PAOLO GIACCONE

**Mojtaba Norouzi**

Mojtaba Norouzi

ACADEMIC YEAR 2024–2025

I would like to express sincere appreciation to Prof. Paolo Giaccone for his Valuable guidance, support and kind feedback throughout this work. I would like to extend my gratitude to EURIX for providing the opportunity to conduct my thesis in a professional environment. Finally, I am deeply grateful to my family and friends for their support and motivation through this journey.

## **Abstract**

In this work, the limitations of the standard Wireless mesh standard are studied. The limitations in the standard wireless network impose a negative impact on the functionality of the network. The flowing ad-hoc network is studied as a solution to address the limitations of the standard network, which is implemented through a microcontroller and a specific library that provide tools to improve the reliability of the true-ad-HOC network and offer smooth message transmission by providing a multitask microcontroller. Subsequently, a method called fog computing is implemented, which enhances the true ad-hoc network functions by offloading the intensive tasks and allowing the microcontroller to focus on handling the connections and routing of the network. The enhancements are validated by some tests to present the true ad-hoc nature of the implemented enhanced network, and at the end some recommendations for further work are offered.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Wireless mesh network (WMN) . . . . .	6
1.1.1	Enhancements on WMN limitations . . . . .	7
1.2	thesis structure . . . . .	9
<b>2</b>	<b>General concept: Enhancing wireless mesh network limitations</b>	<b>11</b>
2.1	Challenging environment for WMN . . . . .	13
2.1.1	Enhancing the deficiency due to challenging environments . . . . .	13
2.2	Enhanced wireless mesh network . . . . .	14
2.2.1	PainlessMesh library and dependencies . . . . .	14
2.2.2	Arduino JSON library . . . . .	14
2.3	Related works . . . . .	15
<b>3</b>	<b>Developing a true ad-hoc wireless mesh network</b>	<b>17</b>
3.1	Ad-hoc mesh network using ESP8266 Wi-Fi microcontroller . . . . .	17
3.2	Developing an Ad-Hoc Network using ESP8266 . . . . .	18
3.2.1	ESP8266 processor . . . . .	20
3.3	Employment of Painless Mesh Library to implement True Ad-Hoc WMN on ESP8266 . . . . .	23
3.4	Task scheduler overview . . . . .	24
3.5	ArduinoJson . . . . .	26
3.6	LuckFox Pico mini A overview . . . . .	27
<b>4</b>	<b>Design and implementation of WMN on PainlessMesh</b>	<b>31</b>
4.1	Programming ESP8266 WiFi MCU using PainlessMesh library . . . . .	31
4.2	Implementation of fog computing Using Luck Fox Pico Mini A in WMN . . . . .	37
4.2.1	Configuration and implementation of Luck Fox board . . . . .	38
<b>5</b>	<b>Implementation results and validation</b>	<b>47</b>
5.1	Performance analysis enhanced WMN . . . . .	47
5.2	Experimental validation . . . . .	54
5.2.1	Experimental test of WMN under a simulated overload condition . . . . .	60
5.2.2	Evaluation and measurement of key metrics . . . . .	61

<b>6 Conclusion and recommendation</b>	69
6.1 other recommended solutions . . . . .	69



# Chapter 1

## Introduction

In recent decades employment of electronic devices revolutionized every aspect of human life. Communication Networks provide the capability of data sharing and interaction between electronic devices, which further transform their application by the creation of intelligent and controlled environments. With the widespread adoption of communication networks in daily life, many cutting-edge technologies rely on these networks for their implementation. However, some restrictions such as the limited number of devices in a network and the covered distance range in the wired network, raise the need to have wireless networks (WN) that tackle this restriction. Nowadays, Wireless networks (WN) are the most general type of network in human daily life and are an ideal replacement for traditional wired networks. This network improves crucial Parameters such as scalability speed and ease of connectivity. A common application of WN in our daily life is the Wi-Fi network. At home, Wi-Fi provides seamless communication and data sharing between the devices of the Wi-Fi network. Wi-Fi networks appear as a significant part of numerous daily tasks. For instance, Wi-Fi is used in smart and automated homes, remote work, video streaming, and Real-Time data monitoring in industrial, medical, and urban management. The wireless network is categorized based on the node's arrangement in the network. The arrangement and form of connection are expressed as Topology. Each topology offers certain features, Based on the specific needs and the environment the appropriate topology will be chosen for network implementation. In most of the Topologies, a central node controls all the connections and interactions of sub-nodes, which makes the network slow and vulnerable in case the central node fails. On the other hand, The Mesh network is arranged in a decentralized form. In a mesh network, each node relies on the nearest node, and in terms of poor connection to the neighbor node it will switch and rely on other nodes Mesh topology delivers better scalability and reliability in comparison to other topologies. The mentioned decentralized form of connection in mesh topology makes it a scalable, reliable network with lower latency in comparison to other technologies. The beneficial features provided by the mesh network make it an ideal choice for many cutting-edge technology products, such as "Smart grids" for energy distribution networks that connect transformers and sensors [1].

**Motivation** : Wireless Mesh Network (WMN) is an outstanding network in terms of reliability and scalability among other networks, but there are some limitations that might cause degradation in the robustness of a WMN. Some limitations, like High power consumption and difficulties in handling many connections in a large network, deteriorate the functionality of the network, Particularly in challenging environments. In this work, we are aiming to implement a WMN for vehicular networks that are battery-powered. In a network for such an application, the reliability and stability of the network are important. However, the limitation on handling connections in large networks and high power consumption discard the efficiency of the network and may impose a significant negative impact on it. The purpose of this work is to enhance the efficiency and functionality of the Mesh network to make it an ideal network for battery-powered devices in challenging environments and large networks. This will be achieved by implementing specific hardware and software that optimize crucial parameters like latency, size of shared messages, and control power consumption. We utilized a library named `painlessMesh` and an ESP8266 Microcontroller. The Microcontroller and `painless mesh` library help to optimize the power consumption and the complexity of handling node connectivity in large-scale networks. While the ESP8266 Microcontroller will be used to form WMN, A board with a powerful CPU will process the user tasks and share the process data with the network. Using a Board along with the network to process is Fog computing and will improve the speed of the Microcontroller that is dedicated to forming a WMN to handle the dynamic switch in connection in large-scale networks. .

## 1.1 Wireless mesh network (WMN)

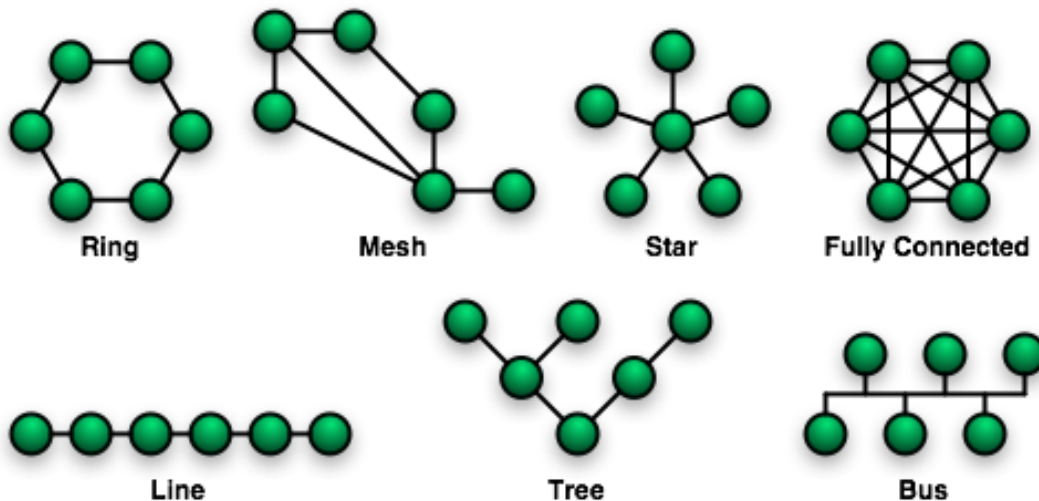


Figure 1.1. Network Topologies



Network topology refers to rules that determine the physical arrangements of nodes of a network. Common types of topologies are illustrated in figure 1.1 In a network, each node is an electronic device that is connected to other nodes of the network to communicate for a specific task. A common example of a network in our daily lives is “Wi-Fi,” which is a wireless network of devices such as phones and laptops, etc. The devices are connected to a modem central node of this Wi-Fi network to provide internet for devices in the Wi-Fi network. Topologies are standards with different features that are based on the desired application; the most ideal type of topology will be chosen to implement the network for the desired application.

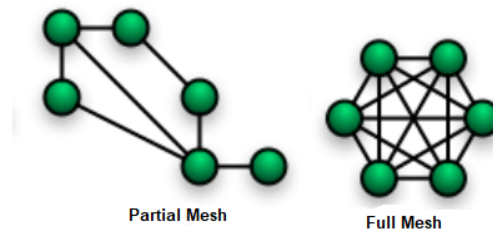


Figure 1.2. Partial Mesh vs Full Mesh

As in figure 1.1 we can see most of the topologies have a central node to handle the network connection. "Wi-Fi" is implemented as a star network and Modem is the central node. Mesh networks unlike other topologies have centralized connections, and each node is connected to all other nodes of the network. In a Mesh network nodes are connected in two forms Full mesh and partial mesh. As illustrated in figure 1.2 the general description of the mesh network represents a full mesh in which all the nodes are interconnected, while in Partial mesh each node is connected to the nearest node and it forms a different arrangement. A partial arrangement of a mesh network is formed in large-scale networks, and due to the fundamental difference in node connections, message transmission and routing is completely different. While in full mesh topology, a node establishes a connection to all other nodes in a partial mesh network, each node finds the best and closest node to join the network. In reference to the fact that each node will find the closest node to establish a strong connection, in case of finding a node with the better condition the node will switch to another node, and it periodically is repeated. In partial mesh Messaging in partial mesh from a source node to a destination, the node passes through intermediate nodes.

### 1.1.1 Enhancements on WMN limitations

**Limitation of WMN for vehicular network in harsh environments** : The mesh communication network is a reliable and scalable, network with low latency, but the mentioned beneficial features are provided by forming a complex network. The complex architecture of the Mesh network is that large networks pose some limitations to the applications that are implemented through this network. The initial configuration is

expensive and a time-consuming task, and in case of disconnection of a node, establishing a connection to rejoin the network will add delay. While each node is responsible for periodically processing many tasks to check the connection along with sending messages, it makes the node slow, and the slow execution of tasks can cause problems in message transmission with sufficient speed. In a mesh network, delay can cause failure in sending messages and maintaining connectivity. High power consumption is another issue in mesh networks that makes it a non-ideal choice for battery power applications. The other significant concern of a mesh network is that, based on its dynamic connection and routing, challenging environments can disturb the proper functionality of this network. Also, remote area environments, such as urban environments, need battery-powered nodes, and the high power consumption of WMN imposes significant negative impacts on the functionality.

**Challenging environments for WMN** : In communication, a Challenging In this communication, a challenging environment refers to an environment that negatively affects data communication between nodes. In challenging environments such as urban areas, and remote areas due to having limited infrastructure to provide Wi-Fi, and large distances between nodes, communicational networks may encounter latency. Additionally in large-scale networks with a considerable number of nodes and data loss rate is increased because of the high volume of data transmission communicational network congestion occurs. The decentralized and dynamic form of connection of a mesh network and also the self-healing and dynamic routing capability of WMN, make it an ideal network challenging environment to provide a resilient network in comparison to other types of connection. The vehicular network is an example of a network The beneficial features of WMN leverage its functionality, but on the other hand, the WMN network limitation degrades the functionality of this network. Is a network of vehicles that are connected to share data like navigation and video streams through the network in remote areas and in dynamic environmental conditions. Generally, vehicular networks are battery-powered, and as vehicles are moving and changing their position with other vehicles, connections between them are dynamically switching. Such applications demand low-power consumption and large-scale networks to function as reliable and real-time networks.

**Enhancement of WMN limitation** : In order to make WMN an ideal network for vehicular networks that function properly in challenging environments, we need to tackle the stated problems and limitations of traditional WMN. The root of the main concern of WMN is the added delay due to intensive tasks for handling dynamic connections and routing and the high power supply of each node. As explained in the beginning of this chapter, each node is an electronic device that is working in a network based on the rules that topology defines. Microcontrollers are inexpensive electronic devices with controlled functions by the user. The ESP8266 Microcontroller” is an inexpensive but strong microcontroller that can transfer data through a wireless connection. Additionally, this microcontroller can be set to lower power consumption while it is not working in the network, which can enhance the power consumption concern. The microcontroller is equipped with a powerful processor to leverage the speed of handling dynamic connections

and routing.ESP8266 can be programmed by using programming language, which provides the probability of using software solutions to tackle the WMN problems.” Painless Mesh” is a software tool that can be run on the microcontroller to implement a WMN. This software tool uses many techniques that aim to help the microcontroller work as a multitask device to avoid slow processing and be capable of handling and maintaining connections; additionally, in the Painless mesh library, software is implemented to reduce the size of data to decrease demand for processing and make the node work faster in the network. The other notable solution that is used to improve the network and make it an ideal network to be used for vehicular networks in challenging environments is fog computing. Fog computing is a method in which we use a very powerful device along with the nodes of the network to process any additional task to help each node be dedicated to handling WMN tasks. All the solutions improve the crucial parameters related to delay overload and power supply to make the WMN reliable and ideal for a network of vehicles used in challenging environments.

Crucial Parameters	Traditional WMN(Limitaions)	Enhanced WMN (Improvements)
Node discovery	Slow Automatic Discovery	Fast Self-Discovery
Routing Mechnisem	Complex	Self adjustment(Dynamic)
Re-Routing	Slow	Fast Self-Healing
Real-Time communication	Synchronous with Delay	Asynchronous and Multi Task (Real-Time)
Power Cunsomption	High Power Consumption	Low powr Consumption (Dynamic Adjustment)

Table 1.1. Traditional WMN vs Enhanced WMN

## 1.2 thesis structure

This thesis is structured into 6 chapters to enhance the limitations of wireless mesh networks (WMN). In the two beginning chapters, the required background to provide into the limitations of the standard mesh network, such as complex connections and high cost and high power supply limitations, are provided. Chapter 3 discusses the methodology to address the limitations of the traditional network, and the microcontroller and the library employed in this work are introduced in detail. Furthermore, fog computing and the Luckfox board that is used to implement fog computing are examined. The subsequent chapter presents the hardware connection and software configuration of the devices used in this work to address the limitations of WMN and enhance this problem. Chapter 5 is dedicated to the validation of the implementation, which examines the result and validates the improvement of the WMN by some experimental tests. In the last chapter, the conclusion of this study and some recommendations for further work are discussed.



## Chapter 2

# General concept: Enhancing wireless mesh network limitations

A network's topology generally refers to the rules that determine the logical and physical arrangement of its nodes. Based on this arrangement, networks are categorized into different types. The topology model of a network arrangement plays a significant role in its performance; determining the proper network that suits the intended application increases the efficiency of the application. The most common network topologies: 1-Ring Topology 2-mesh Topology (Full -Partial) 3-Star Topology 4-line Topology 5-Tree Topology 6-Bus topology. A wireless mesh network (WMN) is a network in which nodes are arranged in a decentralized form. In WMN, each node plays a different role in the network, and based on the node activity and position in the network, they can be considered as routers, gateways, clients, and end-points, which are a network's main components. Each component has different specified roles in the network to accomplish data transmission from the transmitter node to the destination node. This type of arrangement allows mesh technology to have multiple paths to communicate data; this feature brings the outstanding advantage of self-redundancy and self-fault tolerance to the network, which is the resilience of this network compared to other networks.

**Mesh components** : In a mesh network, routers play a role as the primary infrastructure and manage and control routing and secure connectivity. In the mesh network are managing and controlling routing and securing connectivity. Gateways connect the mesh network to external networks and may also be responsible for establishing a connection to an external network to provide the Internet for the network in some scenarios. Endpoints do not have networking responsibilities in the network and simply receive data sent from the repeaters and do not pass data along. [2]

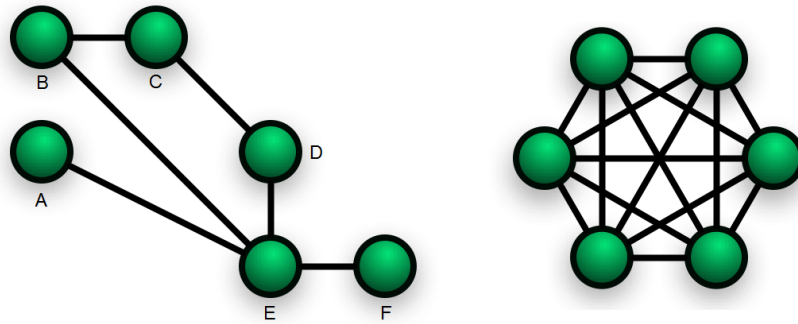


Figure 2.1. Partial mesh vs. Full mesh

**Partial mesh and full mesh in WMN** : Mesh network is further divided into two types of full mesh and partial mesh based on the type of arrangement. The figure 2.1 above two Configurations vary between advantages and disadvantages; choose any of these according to your needs. To manipulate or quantify based on the performance required or the number of nodes and so on, and the best criteria can deploy a Mesh network.

**Full mesh** : In this configuration, each node is connected directly to every other node in the network. Although some of the advantages of full mesh include the existence of many paths and the improvement of fault tolerance, the increase in the number of nodes in the full mesh creates problems such as communication congestion and complexity of connections, and the full mesh becomes inefficient in scalability-required applications.

**Partial mesh** : In a partial mesh, a few nodes are connected directly to all the other nodes, but a few other nodes do not have connections to all other nodes in the network and are connected through intermediate nodes. This type of connection reduces the number of connections in a mesh network and lessens the complexity. A partial mesh can be more acceptable than a full mesh for a larger network, it is a high-performance, scalable network. [3]

**WMN routing** : The routing method in a WMN is individualized as a result of its differing architecture. The architecture provides Real-time data among nodes, however, which is not enough for such networks; hence, WMN DHT, or a decentralized network formed by the combination of such networks, is also different from other classical networks. In WMN, no central node is responsible for managing the connection of nodes; the network is self-healable. This means that to join a network, each node needs to connect up to its closest nodes within that network. Regarding reliability, The strength of the signal will take precedence in The node to determine the best neighbor node and to choose it as an intermediate to connect with other nodes. As it compares other nodes and decides to meet with the one having higher signal strength, in the case it finds the new node superior, it removes its previous current neighbor node. It repeats the process many times as the nodes keep on moving and forming new clusters around. In this way, as the connections

are reliable, the message transfer is also reliable. A Node shall receive a message; it will get forwarded through neighboring nodes as a route towards a destination node. [4]

## 2.1 Challenging environment for WMN

**challenging environment in wireless mesh networks (WMN)** A challenging environment for communicational networks refers to conditions that are distributing the communication network in terms of connectivity and reliability. In some cases, the environment in which a network is used can cause external or internal distortion. The distribution may be the result of limited infrastructure, large distances between nodes, or inappropriate functioning conditions. Wireless Mesh Networks (WMN) are mostly known for their capability to provide reliable communication. But challenging environments impose some negative impacts, Namely the latency and data loss of a network, which can lead to higher data congestion in the network. The challenges for WMN in these environments can be met in 3 types: Physical challenges, infrastructural challenges, and functional challenges. Each of these types is an obstacle that must be studied to guarantee the establishment of a resilient and reliable network.

**Functional challenges in a harsh environment** The dynamic nature of the mesh network is the main cause of functional issues in WMNs. Functional challenges are classified by The wireless connectivity of this network, which is tolerant to many kinds of physical and environmental interference and has a determinant impact on WMN performance. Both in urban and remote areas, the network's performance could be affected by natural events like weather conditions, electromagnetic interference, and geographical barriers. By way of example, high humidity and rainfall can diminish signal strength, effectively decreasing the potential communications range of the network and resulting in increased latency and data loss. Likewise, elevated temperatures create thermal stress on devices, resulting in noise with the risk of static issues.

### 2.1.1 Enhancing the deficiency due to challenging environments

**Synchronization** :In challenging environments with a lot of interference or node switching, a synchronized network that is composed of asynchronous nodes improves the real-time functionality of networks that have nodes with dynamic switching nodes, communications including autonomous vehicles, where even a minor latency can impose considerable impacts.

**Power optimization** : The use of dynamic power consumption and utilization of energy-efficient hardware elements in the implementation of the network can increase the lifetime of battery-powered mesh nodes in devices that are powering on battery in harsh environments. For example, Environmental situations can lead to congestion of data in a network, which can render latency for critical applications. If necessary, implement prioritization of the critical data, which in turn means some high-priority data like emergency data or communication or real-time data can be transferred without

latency. The goal is to have an efficient, real-time network by accelerating the delivery of important communications.

## 2.2 Enhanced wireless mesh network

**Ad-hoc Wireless Mesh Network** It is a decentralized, self-healing network that is designed based on mesh network standards; some features are enhanced to provide a resilient network for Wi-Fi applications implemented by ESP microcontrollers. The modified Mesh network is enhanced by employing techniques and C++ libraries that improve critical parameters in Wi-Fi communication such as bit rate, data packet size, reliability, and real-time and synchronized performance. Painless Mesh network also simplifies message routing by managing network formation and message routing. [5]

**True ad-hoc networking** :A mesh standard network is an ad hoc network; in ad hoc networking, nodes directly communicate with each other without relying on a central access point. A Painless-Mesh network is a true ad-hoc network that provides an outstanding feature in comparison to an ad hoc network, and nodes are self-organized into a network without needing manual configuration. In case a device is added or removed, the network adjusts automatically. In a true ad hoc network, All devices are equal, and there is no "master" or "server" in the network. When a node that is not part of any network is powered up, it's ready to form or join a mesh network. Nodes automatically and dynamically discover and link with nearby nodes to form a network without any requirement for a predefined structure or external controller. [5]

**ID-based WiFi network** :Mesh networks doesn't rely on TCP/IP for addressing, mesh networks use IP as the address of each node in networks, but painlessMesh is not IP networking and is ID-based. The ID is a 32-bit chip ID dedicated to each ESP Microcontroller. ID-based addressing can be a significant advantage of a Painless-Mesh network that eliminates the overhead of TCP/IP to gain a lightweight and efficient communication network. [6]

### 2.2.1 PainlessMesh library and dependencies

The Painless-Mesh library is a C++ library that enables a WiFi-based mesh network utilizing ESP8266 and ESP32 microcontrollers. Painless-Mesh is Built on ESP-NOW and some WiFi libraries from Espressif or streamlines the procedure of developing and managing a wireless mesh network without the need for an external router. [5]

### 2.2.2 Arduino JSON library

JSON is a verbose and not binary format that modifies and optimizes messages, ArduinoJson which is a dependency of the Painless-Mesh library optimizes message size by utilizing some techniques to minimize unnecessary data and it makes ESP communication more efficient. [5] [7]



**TaskScheduler library** :Task-scheduler is a lightweight library that is implemented for cooperative multitasking for microcontrollers like ESP8266 and ESP32. Real-time control and execution of tasks are crucial. Task Scheduler allows ESP to be a multi-task Microcontroller with some additional features like Periodic task execution, deterministic number of iterations of tasks, and prioritizing task execution that can help to enhance mesh network by optimizing data loss, latency, and power consumption and also increasing reliability of a network. [8]

**ESPAsyncTCP(ESP8266) library** : In regard to the significant demand to control and manage multiple connections or tasks without delaying synchronized multitasking, it is a crucial tool to have a resilient Wireless Mesh network. ESPAsyncTCP library is a TCP library that allows ESP to handle multiple tasks asynchronously. ESPAsyncTCP is a preemptive scheduler library that makes the task scheduler an asynchronous function in the ESP Microcontroller to deliver asynchronous TCP communication. ESPAsyncTCP(ESP8266) library is dedicated to the ESP8266 microcontroller. It enables non-blocking network communication and the ESP8266 can manage multiple tasks like transmitting, receiving, and controlling connections without blocking the main program. Non-blocking communication guarantees the mesh network remains responsive even under heavy traffic and manages multiple connections. These libraries use interrupts and callbacks of events in the ESP Microcontroller to handle events like data transmission or connection status and enhance the network to have better scalability, Reduce Latency, meanwhile optimize power consumption by reducing the probability of data loss and the need to resent data. [9]

## 2.3 Related works

Wireless mesh networks (WMN) have been widely studied because of their robust capabilities, such as reliability and adaptability in different environments. Several studies analyzed the performance of WMN and enhanced the network by developing the WMN using ESP8266, which is a low-cost and low-power-consumption microcontroller. Additionally, Fog computing is implemented to improve the efficacy and throughput of WMN by reducing delay. In this section, we review the related works on developed WMN using the ESP8266 microcontroller and the role of fog computing in enhancements of WMN. One of the notable studies is by Martinez [10] which evaluates the performance of a WMN implemented by an ESP8266 microcontroller and Painless Mesh library. In this study, crucial metrics such as round-trip and one-way delay of packet and delivery ratio are measured under heavy network loads. Their finding indicates that the ESP8266 nodes show an increase in latency and a decrease in throughput under heavy network loads. Another similar study by [11] examines the self-organization and synchronization mechanism of a network developed by ESP8266. This study highlighted the complexity of routing and data congestion management in the network. In contrast, the mentioned studies not only measure and evaluate one-way delay and round-trip delay, but also the true ad-hoc nature of the developed WMN using ESP8266 presented by exacting the JSON-based control messages. By extracting the communication of nodes, we analyze

the self-configuration, synchronization, and routing of nodes in the developed network. Moreover, we experimentally tested the enhanced WMN under the overloaded condition to determine an optimal payload size vs. transmission interval to address the expected overload in realistic application of the network in IOT applications. A study related to fog computing by Shabir.Ali [12], which studied software-defined fog computing developed for wireless mesh networks in semi-permanent environments, highlighted the role of fog computing in reducing the delay and also the consumed bandwidth by performing in network processing, which leads to improved scalability and efficiency. In our work, we experimentally test the probability of using the Luck Fox PICO mini A as a fog node to offload the user tasks from the ESP microcontroller and monitor shared data by the WMN to ensure real-time functionality of the nodes. The experimental test suggests offloading ESP nodes to provide a real-time network and improve network efficiency by dynamically balancing computational loads and reducing data congestion. By synthesizing the insights from the mentioned related works and extending them through our experimental validations, our study contributes to the development of robust and true ad-hoc WMN that aligns with the requirements of vehicular networks used in challenging environments.

## Chapter 3

# Developing a true ad-hoc wireless mesh network

Standard WiFi mesh networks offer high data rates and extensive range coverage. However, some key characteristics of this protocol can be considered weak points and cause significant issues in challenging environments for communicational networks. In WiFi mesh, Protocol one such issue is l, which agency is regarded as an overhead of this protocol that may cause some delay in data Packet transmission due to using of TCP/IP stack, as in Mesh protocol Nodes are linked to each other. In a large network with many nodes with an excessive number of nodes interacting with each node, it may receive several data packets that are more than tolerant of the node, which can be another root of latency in this protocol and cause network congestion. In a decentralized message routing network, this message transmitting method can be considered as vulnerable route massaging as in case of failure and in consequence of this failure of the central node data loss will be increased and meanwhile, network congestion will occur and

In challenging environments, features of a protocol that specify the arrangement of the network and data transmission rule the network, like real-time communication, resilience network, and dynamic routing to increase area coverage and reduce time of interaction and network congestion, are critical. Painless-Mesh optimizes these aspects of the protocol to reduce latency by decentralizing the Wi-Fi mesh, a lightweight protocol that, overhead, improves self-healing through dynamic node discovery. The inclusion of the Painless-Mesh library in regular Wi-Fi mesh networks will bring significant latency, reliability, and general performance improvements. It offers a resilient, lightweight, optimized power consumption and reliable solution for use in challenging environments.

### 3.1 Ad-hoc mesh network using ESP8266 Wi-Fi microcontroller

The standard WMN is highly reliable, and its scalable architecture provides robust connectivity for large-scale applications that need a resilient and wide-range cover network. However, In standard WMN, some limitations, such as high power consumption, complex configuration, reconnecting, and "routing," along with the high cost of required

infrastructure, make this network impractical for some applications, such as "Energy constraint mobile networks" applications that the network of the vehicle is a type on this application. In energy-constrained networks with battery-powered nodes, having Low power consumption is vital. Accordingly, the high power consumption of Traditional WMN is a big shortcoming. Besides, The complex configuration and high-cost infrastructure of traditional WMN impose limitations on the service of WMN for some functions and make them Non-economical. To resolve these problems, we propose implementing the WMN on the ESP8266 microcontroller (MCU) shown in figure 3.1 as a fundamental building block of the WMN. In the following study, we examine the ESP8266 and its peripherals, and we discuss the ESP8266 features that are employed to develop a low-cost and low-power consumption network WMN that is self-organized. These enhancements on WMN provide fundamental enhancements on the standard WMN to eliminate the issues regarding configuration, connection, and routing.

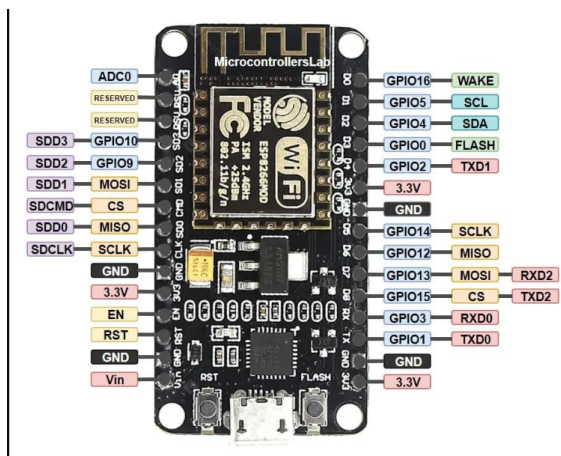


Figure 3.1. ESP8266 pin-out reproduce from [13]

### 3.2 Developing an Ad-Hoc Network using ESP8266

**Ad-hoc network** : An ad-hoc network refers to a network that has a decentralized architecture, such as a mesh network where a node is self-organized to dynamically establish a peer-to-peer connection, while it doesn't rely on any external infrastructure. An individual node that is not part of a network autonomously scans and discovers the nodes in its range, which results in a self-organized network. Power-up or disconnected nodes can join a network to form a self-configured network. In an ad-hoc network, each node works individually without relying on infrastructure to provide Wi-Fi to the node, and each node of an ad-hoc network can be a host (endpoint) or router to forward data to other nodes. Self-organized Routing is another feature of an ad hoc network. That is possible by communicating nodes to find the best path for data transmission. In a True ad-hoc network, each node is scanning frequently to find new nodes, and by comparing the signal strength of the probable connection, the node discovers the most optimal path

to transfer the message. Regarding improving the standard mesh to an ad-hoc network, each node should have a built-in Wi-Fi transceiver and be capable of being controlled to behave in a controlled algorithm. The ESP8266 is a Wi-Fi microcontroller (MCU) that is composed of a Processor and a Wi-Fi transceiver.

**implementation of ad-hoc WMN using ESP8266 Wi-Fi MCU** : In order to address the limitations of standard WMN, we implement the network through a Wi-Fi microcontroller to be used as each node of the WMN network. The ESP8266 MCU is a low-power, low-cost, and low-weight MCU. This MCU is composed of two main chips, which are a processor and a Wi-Fi transceiver. Employing this MCU as a node can mitigate the cost of the network by eliminating the need for any Wi-Fi router and enhance the power consumption of the network through its low-power processor, which will be examined in more detail in the flow. Additionally, considering that the ESP8266 MCU is a node of the WMN, we can program the microcontrollers, which are representing as nodes of the network, to be self-organized in terms of configuring to join a network and handle connections and routing. The approach and features to having a self-organized node will be studied in the next section 3.2 . A self-organized node allows us to reduce the complexity of configuring, Connection, and routing. With the intention of implementing a self-organize node, we leverage a C++ library that is debated in the section 3.3. Implementation of a WMN with self-organizing nodes results in an ad hoc and self-configure network. Below in figure 3.2 on the left figure you can see the architecture of a standard mesh network that is dependent on an infrastructure, which is indicated as a red node to provide Wi-Fi for the nodes in the mesh, and in the right for each node a ESP8266 Wi-Fi MCU is implemented to form an Ad-hoc mesh network that is in-dependable of infrastructure.

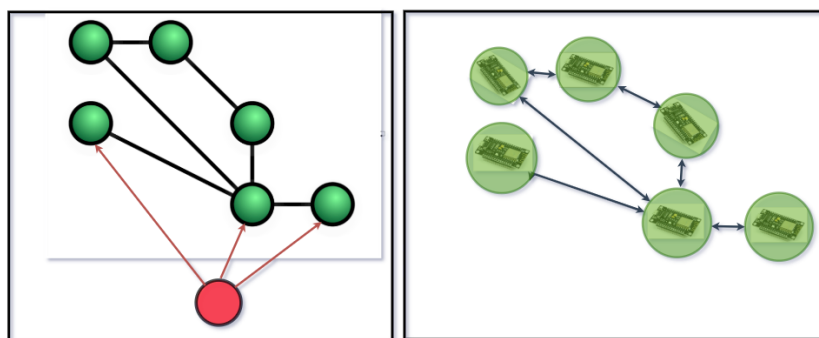


Figure 3.2. infrastructure depended Mesh vs. Ad-Hoc Mesh Network Implementation

**Introduction to ESP8266 Microcontroller** : In the following, we discuss in detail some of the built-in features of the ESP8266, such as the 32-bit chip with high frequency, dynamic power management, QOS to control priority of message type to improve real-time functionality, etc. To describe how they leverage the functionality of the traditional WMN. As the ESP8266 Wi-Fi MCU is composed of two main parts, which are the processor and Wi-Fi, we divided the explanation into two parts, and each describes the processor and

Wi-Fi and describes in detail the features it provides to enhance the traditional WMN.

### 3.2.1 ESP8266 processor

The ESP8266 MCU is composed of "Processor", "Wi-Fi radio,"memory, etc. Here we discuss the processor, which is the "Tensilica L106" processor." The 32-bit chip offers high-frequency processing for ESP8266 that allows us to manage multiple messages simultaneously, consequently reducing the latency of data transmission and having efficient packet processing. The Tensilica L106 is a 32-bit RISC processor core that is the brain of the ESP8266 MCU. The RISC (Reduced Instruction Set) architecture makes the processor simple but fast with the 32-bit size registers and bus and memory. The 32-bit size allows the processor to address memory up to 4 gigabytes and handle larger numbers in processing in each clock cycle. Some advantages abilities in Painless-Mesh-like message routing, are dependent on this and the 32-bit core of ESP8266 and help to properly handle the required processing. The high frequency of this processor is another notable feature of the ESP8266 MCU. The default speed is 80 MHz and can be increased to 160MHz by overclocking . The high frequency along with the 32-bit size of the register in the processor, Improves the robustness of the Mesh network regarding processing and handling tasks for configuring, connection, and routing.

**Low power consumption** : The Tensilica L106 processor is an energy-efficient processor which makes it an ideal choice for IOT and networking. The processor includes different power modes that can dynamically adjust the power consumption of the network based on the state of each node of a network. The three power modes of the Tensilica L106 processor are deep sleep mode ( $\tilde{20}$  mA), light sleep mode(0.4ma), and modem sleep mode(15 mA)

**Adjust power consumption** : Dynamic power consumption in a network implemented by ESP8266 consumes 0.5 W while they are active, and less than 0.1 W in sleep mode under 0.1 W in sleep modes, In large-scale networks, this capability can cause excellent optimization in power consumption and makes it an ideal choice for applications that are powered on battery.(cite-ESP8266 datasheet). [14]

### ESP8266 Wi-Fi Transceiver

:The ESP8266 Wi-Fi transceiver is an integrated RF module that supports 802.11 b/g/n standard. The IEEE802.11 defines rules for networks to communicate in a Wireless network. The standard has various frames that contain the node's status and are used for different purposes, such as establishing connections, message transmission, synchronization, and sharing power management status. IEEE 802.11 offers features that allow us to have an ad-hoc network by employment of these features, and we will discuss in flow. As the Wi-Fi transceiver is a practical part of the ESP8266 Microcontroller in our work to form a WMN, we can refer to its low power consumption feature as one of the most significant features of this transceiver that perfectly aligns with our purpose to resolve the problem of high power consumption in traditional WMN. The ESP8266 transceiver is capable of handling multiple connections and can provide reliable Wi-Fi connections in noisy and

challenging environments, which increases the reasonability of choosing it as a Wi-Fi microcontroller to implement a WMN in a challenging environment. Moreover, the ability to handle multiple channels can allow the ESP8266 to manage its connections with other nodes with higher flexibility and make connection and routing more reliable in a WMN.

**Implementation of Ad-hoc network using IEEE802.11 on ESP8266** : The initial Steps to have an Ad-Hoc network is build a self-configure, and the nodes of this network be able to work synchronized. In the flowing some feature of the Wi-Fi transceiver are discussed that can be implemented to have a self-configure and sync network.

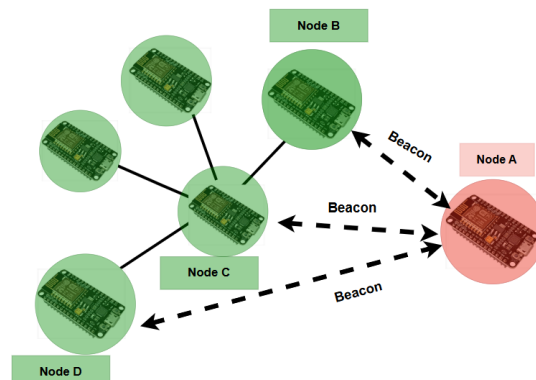


Figure 3.3. Beacon Broadcast to Find Nodes

**Develop an Self-Configure Network Using Beacon Frame** : To develop an ad hoc network, the first step is to provide the ability to have a self-configuration network. It is possible to broadcast a signal to all in-range nodes to find them and start the initial communication to form a network. Beacon is a frame in the IEEE 802.11 standard. ESP8266 supports the IEEE 802.11 standard, and the Beacon frame helps to design a self-configure network. Illustrated in figure 3.3A WMN is presented while a node powered up node starts to broadcast a beacon frame to announce its presence to in-range nodes. While other nodes receive the beacon request, a response is transmitted to confirm the presence. In figure, Node A broadcasts a beacon request to Node A, B and C and announces its presence. The 3 in-range nodes respond by sending a prob beacon to confirm their presence. These steps provide a self-configured network to enhance the problem-complex configuration in mesh networks. Synchronization between nodes of WMN is the next step to achieve an ad-hoc WMN that works in real-time.

**Synchronizing Nodes of WMN using TSF (Time Synchronization Function)** :While a node of a network works in a non-synchronized network, the messages relating to connections and routing are not Real-time, and each node establishes connections and routes based on delayed data, which raises problems such as frequent disconnection and data collision. Accordingly, the synchronization feature is an essential factor in the implementation of an ad hoc network. TSF (Time Synchronization Function) is

another feature of IEEE 802.11. The TFS mechanism can leverage the development of a synchronized network. Each ESP8266 MCU is equipped with a Timer based on a clock provided by a crystal. The ESP8266 uses this timer to count time in microseconds or nanoseconds. Each ESP8266 MCU has a timer that works based on its crystal. A new node should be synced with other nodes of the network before joining the ad hoc network, so before establishing a connection, The new node will share its time to find the gap in reference time. The difference in time is referred to as offset, and the nodes in a network are functioning as a sync node while the offset is below the predefined threshold that will be defined in the code that is used to program the ESP8266, and we will examine this synchronization and the offset with more detail in 3.3 that the library will use to implement the discussed features to form an ad-hoc network. To provide a sync network in the initial steps of establishing a connection, the new node will share its time to find the offset. The time adjustment will occur until the offset reaches below the threshold and the new node will be synced to other nodes of WMN. [14]

**ESP8266 Features to Enhance Messaging Efficiency** : Along with features discussed in developing an ad hoc network, ESP8266 offers some features to increase the efficiency of the network by reducing data loss and latency. One of the notable features is Delivery Traffic Indication Message (DTIM) which avoids data congestion to reduce latency. The DTIM acts as a signal from a system for client devices to signal the delivery of broadcast beacon data. DTIM allows devices to define optimal times for wake-up from low-power modes to receive buffered data that enhances power optimization while keeping network responsiveness and avoiding data congestion. Another feature in the Wi-Fi transceiver of the ESP8266 is Guard Interval (GI), which aims to reduce the time interval between transmissions. By reducing the latency in transmission, we can improve the data throughput by sending more messages in a shorter time with no significant increase in data loss rate. When data are sent in small packets, this Guard Interval (GI) is added between them to avoid any interference between them by adding a short pause. SGI adds an 800 nanosecond delay between two message transmissions. It minimizes the number of message transmissions and reduces failure in message transmission; moreover, further more data packets are transmitted to the browser and also faster data transmission, hence leading to a high data rate. Real-Time nodes: When we create a network that shares some crucial data (like navigation data or monitoring data extracted by sensors on the vehicles), it has to be Real-Time. Some data are more critical than others; in this regard, several applications, such as vehicular network navigation data, play a critical role in updating navigation; therefore, the network should give priority to them to be transmitted. ESP8266 has a feature called Quality of Service (QoS) which helps to define the priority for every type of data. By utilizing QoS in ESP8266, critical data packets such as real-time control signals or sensor updates will be given precedence over less important transmissions, which assists in minimizing data loss and preserving reliable network performance. QoS ensures timely and accurate message transmission in applications where message delivery speed is operationally important, such as for IOT automation or emergency response systems. Scheduling data transmission with the QoS avoids any kind of data collision and helps maintain the WMN connection's reliability and stability, even in large-scale networks.



The quality of service (QOS) provides optimal traffic control of the data and carries the WMN to operate as a steady network even in disaster situations. [14]

Attributes	Description	Effect   Advantage	Enhancement in Traditional WMN
Beacon Signal (Presence Signal)	announce presence	Self-Configure nodes	Self-Configure WMN
Beacon Signal (Communication)	Control Messages (Connections)	Self-Organized Nodes	Auto-Connect Nodes Auto-Route Messaging
TFS	Synchronization of Nodes	synchronized Nodes	Latency & Bitrate
QOS	Prioritizing Data	Real-Time WMN	Latency
DTIM	Data Delivery Signal	Real-Time WMN	Multi-Task
FEC	Forward Error Correction	Lower Data Loss	Reliability

Table 3.1. ESP8266 features and their Enhancements

### 3.3 Employment of Painless Mesh Library to implement True Ad-Hoc WMN on ESP8266

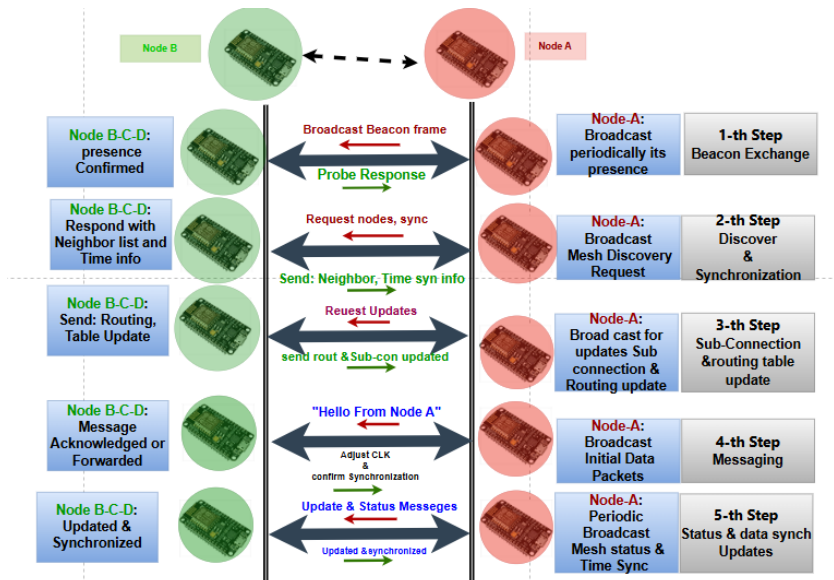


Figure 3.4. UDP of Painless Mesh based network

**User Datagram Protocol in Painless Mesh** :Generally the communicational network operates based on the standard transmission protocol(TCP) but as some IOT applications aim to reduce the power supply and increase data transmission speed a user datagram is used for message transmission. The Painlessmesh communicates with other nodes of the network based on the specific UDP used. The figure 3.4 illustrates all the steps of establishing a connection and messaging presented.

**Step 1- Announcing the presence of beacon exchange** :In the initial step of establishing a connection, the powers-up node broadcasts a beacon signal to the in-rang nodes. As presented in the figure Node A is transmitting a beacon signal to Node B to announce its presence and start a peer-to-peer connection. Afterward, Node-B receives the beacon signal and confirms its presence to Node-A by sending an acknowledgment signal.

**Step 2- Discovery and synchronization** :In the Second step, Node-A sends a discovery request that contains time information and existing nodes. When data is received by node B the node with fewer sub-connections, Is set as an access point, and its time is set as the reference time, Moreover node-B transmits the list on sub-connection to node-A to allow node-A to update its connection table with the sub-connection list.

**Step3-Sub-connection and routing information** : The third step is dedicated to routing and connections, Where the access point shares the table of routing information and sub-connections to allows the node-A to updates its information before connecting to the mesh network.

**Step4-Messaging** :In this phase, Node-A is connected to the mesh with updated information on routing and sub-connections, and the initial messages are transmitted and Node-A relies on Node-B to forward the message as an intermediate node. The initial mesh message announces the presence of Node-A and the other nodes of the Network update list of connections.

**Step5- Synchronization updates** :The last step is synchronization to maintain a real-time and reliable connection by frequently transmitting and checking the offset time between Node-A and Node-B. While the offset is below  $10\mu$  the connection is validated, and in case of exceeding 10 microseconds, the synchronization steps are executed since the offset meets the threshold which normally takes 3 or 4 synchronization steps.

### 3.4 Task scheduler overview

A scheduler is a lightweight multitasking system that controls various tasks in microcontrollers such as the ESP8266. Unlike free-RTOS, which is a real-time operating system for embedded devices, the task scheduler is an easy-to-implement system to ensure cooperative scheduling of tasks for systems. The task scheduler provides outstanding features to the microcontroller, such as periodic task execution with variable time intervals. Prioritizing

the scheduled task and execution of tasks based on interrupting signals leads to a real-time and low power-consuming system. The task scheduler provides a C++ library that can be used on Arduino, and the Painless Mesh library takes advantage of this library to offer multitask ESP8266 microcontrollers to implement a WMN. Additionally, the multitask functionality of each node results in smooth message by handling both transmission and receiving. As mentioned, this library allows setting priority for the undergo message transition that leads to lower data loss and having real-time WMN.

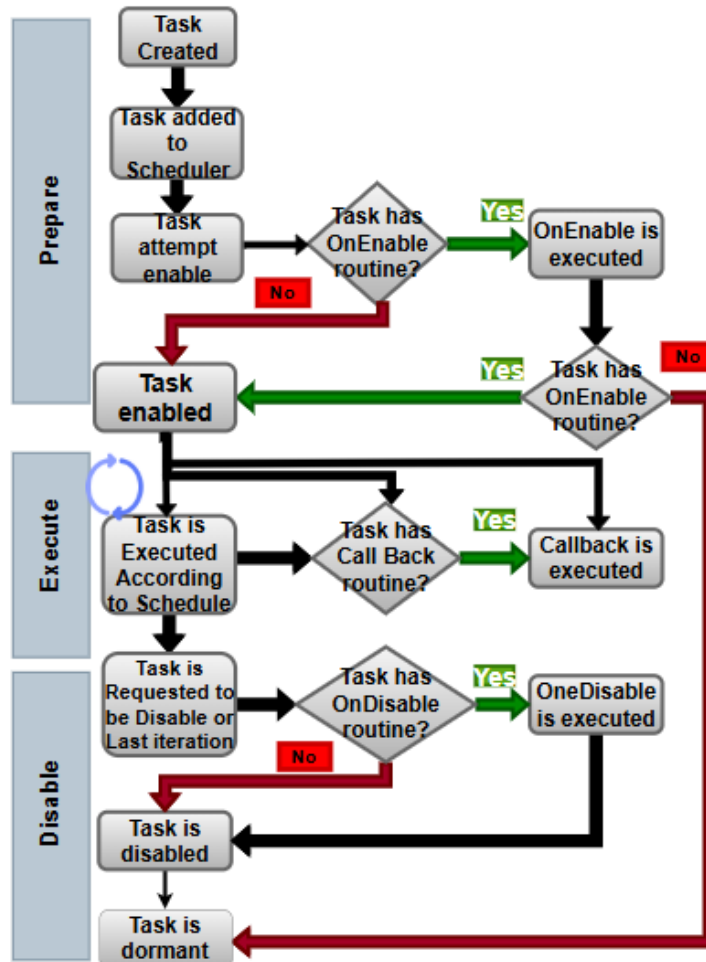


Figure 3.5. Task Scheduling Flow-chart to present it routine function

**Task Scheduler flow chart** : The figure 3.5 To provide a clear insight into the function routine of the task scheduler a flow chart is provided. The Flowchart is divided into three Phase, 1-Prepare the task, 2-Execute the task, and Disable phase. The process from the first phase to the last phase is explained in the following. **Prepare phase**

1. -Task creation: in the first step a task is created by defining the code.

2. -Task Added to be enabled: in this step, the defined task is registered to the Scheduler queue and waits to be enabled.
3. -Task attempt enabled:The scheduler attempts to activate the task that is in the queue by setting it as Onenable.
4. -Check for ONEnable Routine:In case, the task is an enable routine it will be executed by the system.
5. -Evaluate OnEnable return:the next step is to check the returned feedback, if the feedback was TRUE, the is enabled, otherwise it will be disable.

#### **Execution phase**

1. Task Enable: Now the task is activated by the system.
2. Task Executes According to Schedule: The active task executes according to the defined time interval by the user.
3. Check For Callback routine:the system checks the assignment of any callback, If any callback is assigned the callback is executed.

#### **Disable phase**

1. Task Required To be Disable: Any task can be stopped in case any manual request for stop is sent or if the task execution finishes its iteration. .
2. Check for OnDisable Routine: If any OnDisable tasks remain, The task is called before disabling it.
3. Task Disabled: The task is marked to stop execution.
4. The task is Dormant: the task is in memory but does not execute until it becomes enabled for another time.

**ID-based networking** :As we discussed in the last part of this work The painless mesh works on a specified protocol and is not implemented by TCP and IP configuration and calculation is eliminated in its protocol that allows the microcontroller to dedicates its processing power on the managing connections. The painless Mesh works based on the ID of nodes. Each node contains a 32-bit Mack address which is used as its ID for addressing and identifying different nodes.

### **3.5 ArduinoJson**

The Arduino JSON is a C++ library for data comparison in microcontrollers such as ESP8266 MCU. This library is a dependency of the PainlessMesh Mesh library to parse and convert messages before transmitting or broadcasting to WMN. ArduinoJson manipulates the structure to optimize data structure efficacy by two main tasks namely, serialization and de-serialization. In serialization the structure of the message is converted to a JASON

format structure to reduce the size before transmission. When the message is received by the receiver node, The receiver de-serializes the message, and data are parsed back to a usable structure. In WMN, Arduino JSON results in a reduction of processing data which optimizes managing the network by the microcontroller, latency and impacting real-time and synchronization in WMN. [15]

### 3.6 LuckFox Pico mini A overview



Figure 3.6. Luckfox pico Mini A reproduce from [16]

**Introduction to LuckFox Pico Mini A** Luck Fox Pico Mini A is a small-sized and powerful single-board computer designed to offer high-performance computing, that is hown in 3.6. The Rock-chip processor of this board is dedicated is equipped with a neural-network coprocessor to be used in machine learning applications, and intensive processing tasks. Moreover, despite the powerful processing capability, the risk-v architecture of this board makes it a low-power Microcontroller which makes it ideal for battery-power applications and mesh networks with a high-power consumption network. These features make it a powerful yet efficient choice for various edge-computing applications. "Fog computing" is a method by which we can have data processing close to the data generation source to reduce the tasks that the data generator performs.

**Fog computing in WMN using Luckfox board** :Fog computing is a popular method in IOT to increase capability in Real-time functionality, and fast decision-making. In this work, we add Luck Fox mini PICO A to process the data instead of overloading ESP8266 to process all the data and improve the Real-time Performance of the Enhanced WMN. In a network of vehicles, we have a central control unit that controls the behavior of the network by sending commands after processing the received data from the network. While we process data by fog computing before sending data to the central control unit, we reduce the amount of processing in the central control unit. The Luck Fox Mini A is an ideal choice for Fog Computing, it utilizes a powerful CPU and low power consumption, which align with the need of this work to address the problem of high power consumption in Traditional WMN. Additionally, The powerful CPU helps to bypass the user tasks from the ESP8266 MCU and process them on the Luck Fox board, and The machine learning capability of this work provides Real-Time data analysis. Luck Fox Board is designed in small size and low-weight compact and can be used for applications in remote areas and applications where weight is a crucial parameter. This board is run on the Ubuntu operating system, and it provides the probability of using a wide range of software, such as compilers and machine learning software, to leverage the analysis of data. Along with software dedicated to Ubuntu, the "Software Development Kit (SDK)" allows running software that is designed for other operating systems. Some Software is designed for operating systems with different architectures. By "Cross-Compile" ability, we make software that is designed for another operating system to be compatible with Luck Fox. The SDK provides cross-compiling software for Luck Fox. All the drivers and libraries that we need to run it on the Luck Fox board. In the following, we discuss the steps of configuring a Luck Fox. [16]

**LuckFox Pico Mini A key features** : The main feature of Luck Fox Pico Mini Pico Mini A offers some unique hardware and software features that make it a versatile development board. LuckFox Pico Mini A has a RISC-V architecture onboard processor, that is Pareto in case performance vs power consumption and it makes it an ideal choice for our work.

**Rockchip RV1103 chip** : Rockchip RV1103 chip is based on the high integration IPC visual processing SoC, for AI-related applications. The RV1103's processor is a single-core model of the ARM Cortex-A7 32-bit with NEON and FPU support and has a built-in Neural Processing Unit (NPU) 0.5 TOPS for INT4/INT8/INT16 mixed operations. which makes it more suitable for AI tasks such as image recognition, real-time inference, etc. The new image processor is supported by a hardware ISP(Image Signal Processor) with enhanced algorithm accelerators such as HDR, 3A, 3DNR, 2DNR, and sharpening, for a high-end shooting experience.

**16-bit DDR2 DRAM** :It uses 16-bit DDR2 DRAM for enhanced memory bandwidth and adds Power-On Reset (POR), audio codec, and MAC PHY.To conclude, the LuckFox Pico series is a Turbo 80 processor on a board with cost efficiency and Maximum image processing and memory position per mac2(8by) per unit. It is perfect for intelligent

cameras, edge computing, internet of Things (IoT) devices, and similar applications. The board support 64MB of DDR2 RAM which is capable of lightweight Linux-based applications and data processing tasks.

**dual-core CPU** : The board consists of a dual-core CPU made up of a Cortex A7 running at 1.2 GHz and a RISC-V core. This combination of performance and power usage also makes the chip an excellent option for scenarios such as edge computing or IoT-connected devices. Moreover, The unified NPU supports the data types int4, int8, and int16, enhancing the efficiency of machine learning and AI tasks. This characteristic makes the board perfect for real-time inference applications, for example, image processing and sensor data analysis. [16]

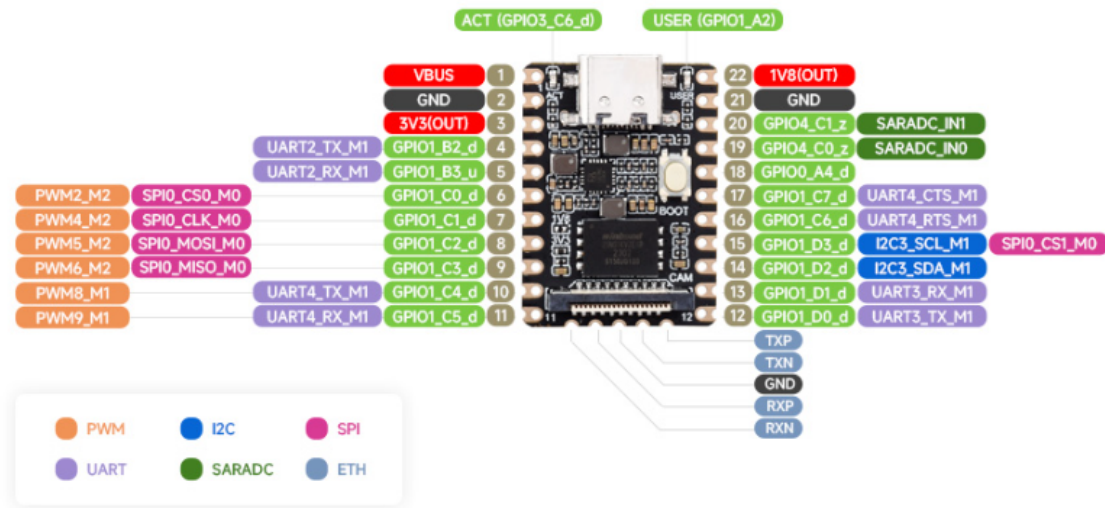


Figure 3.7. Pin definition Of Luck Fox Pico Min A reproduce from [16]

**Pin definition** :The figure 3.7 board has 17 GPIO pin which are adoptable and can be figured as peripheral interfaces, including UART, SPI I2C, UART, Ethernet, PWM and Analog Digital Converter(ADC) which are indicated with specific color in the figure.





## Chapter 4

# Design and implementation of WMN on PainlessMesh

In this section, all the steps to implement an enhanced WMN using an ESP8266 MCU and Painless Mesh library are presented. As we need to develop the WMN on ESP8266 MCU by the Painless Mesh C++ library, first we will cover the installation of Arduino IDE and in the following, the library will be added to the Arduino IDE. Furthermore, the functions of the Painless Mesh library to develop WMN are explained. Finally, we will configure Luck Fox Pico Mini A to receive the data from WMN by UART and print the received data by a serial monitor application that is provided using Python. To cover all the required tasks for a vehicular network an image processing application cross-compiled by SDK is run and the result is illustrated.

### 4.1 Programming ESP8266 WiFi MCU using PainlessMesh library

The initial step for programming the ESP8266 is to provide the IDE for programming by Painless Mesh library. The successive sections include steps to install the Arduino IDE and add the PainlessMesh library, followed by explanations of PainlessMesh functions and C++ code for forming WMN on ESP8266 MCU.

**software configuration and code to implement WMN** :The Arduino Integrated Development Environment (IDE) is versatile software designed to simplify programming. It also contains a series of function menus to connect a board and compile and test the programmed board. It provides a text editor for writing code, compiling and uploading code on boards, and debugging the output by printing it on a serial monitor tool embedded in the Arduino IDE. The Arduino IDE consists of a library manager to add a library from an extensive number of libraries that are developed for use in the Arduino IDE, a compiler, and a serial monitor that all work together to streamline the implementation of a work. In this work, we used the Arduino IDE to program ESP8266 by the Painless Mesh library, which is a library in the Arduino IDE to implement the enhanced WMN. In

the following section, the progress of installing Arduino and adding the demanded library is explained. Subsequently, the functions of the painless mesh library to implement the enhanced WMN are explained . [17]

**Arduino IDE installment** :The initial step is to install the Arduino IDE from Arduino official website [18] All the steps to install the Arduino IDE is illustrated in figure 4.1

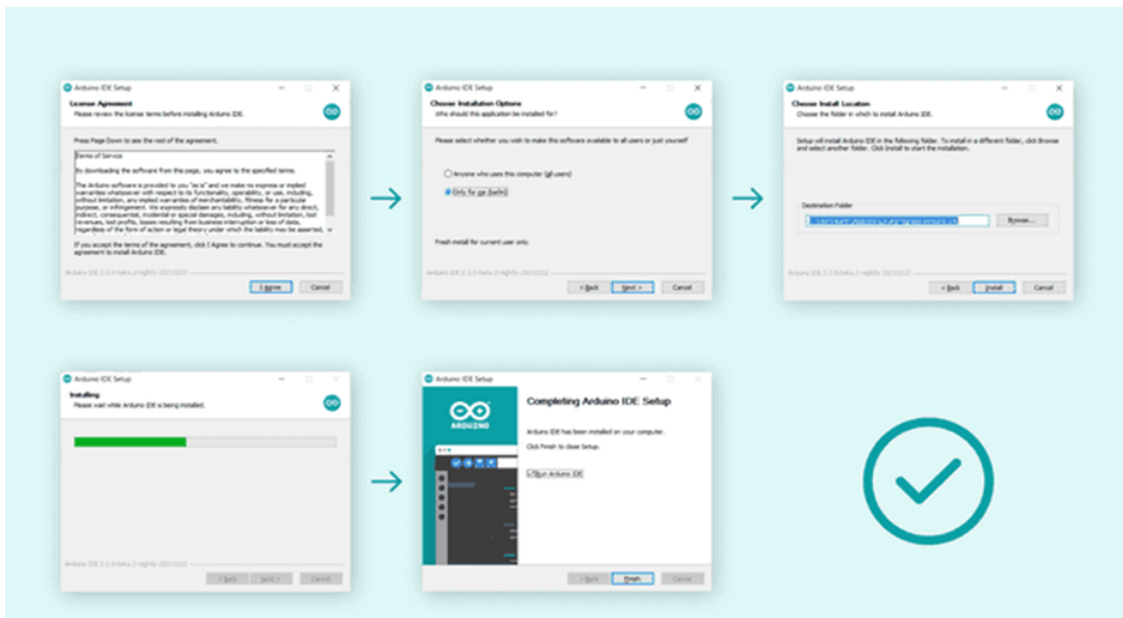


Figure 4.1. Arduino Setup

In subsequent step after the installation of the Arduino IDE , We need to add the Painless Mesh library using the Library Manager toolbar. From the Tools drop down option in the IDE, By selecting Manage library function. The PainlessMesh library can be search and last version of the library should be added.

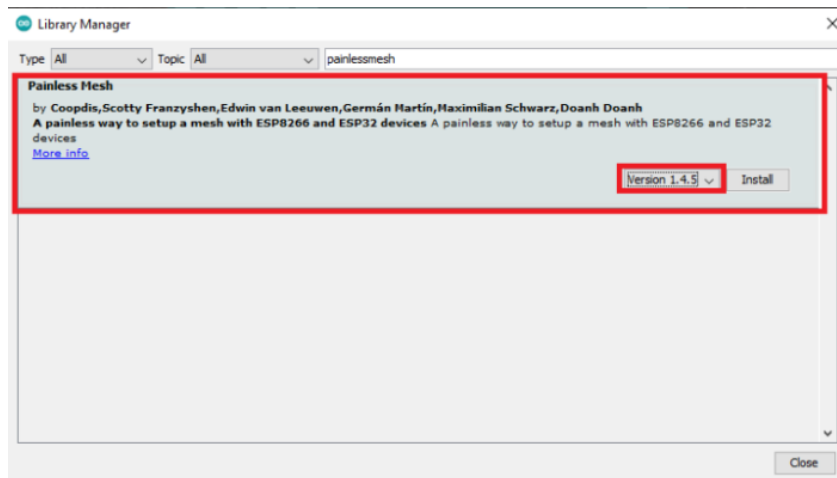


Figure 4.2. Painless-Mesh library

The Painless-Mesh library some primitives are needed to implement a WMN. These dependencies that are shown 4.3 are ESPAsyncTCP to form an asynchronous network using ESP8266. The other preventives are ArduinoJson (by bblanchon) to compress messages and TaskScheduler to develop a multi-task microcontroler.

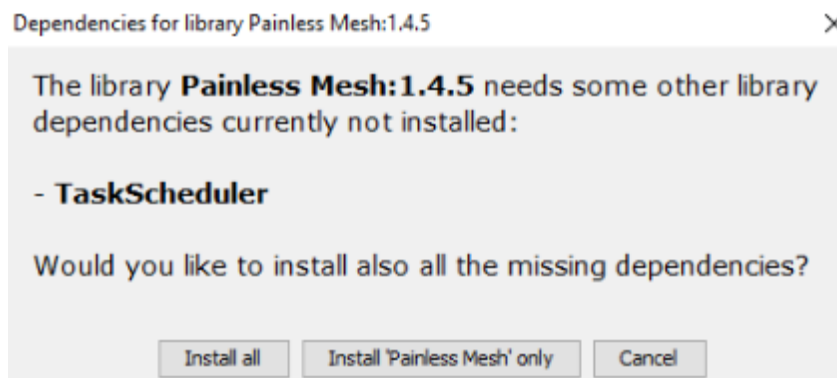


Figure 4.3. PainlessMesh dependencies

After adding the library and the dependencies, the Arduino IDE is ready to be used for programming the ESP8266 by PainlessMesh library. The next step of WMN implementation is to write a C++ code using the IDE and employing the Painless mesh library in this regard. In the following, the functions to develop a WMN and the C++ code are discussed.

**C++ Code to develop a WMN on three ESP8266 using the Painless-Mesh library** : In the text editor of Arduino IDE, the first line is dedicated to including the library, as shown in figure 4.4 by include "painlessMesh. h command the Painless-Mesh

library is included, which contains all the dependencies and sub-libraries that we need to implement and The functions of this library are available to be used. Like any other Wi-Fi devices to keep the connection safe, we set a username and password which is a similar username and password for all the 3 ESP8266 that work in the same network. The command define MESH-PREFIX "Mojtaba44" defines the user name and the next line is the password of the network. The port number is set to 5555, a non-standard port that is not a common port used by standard communication service devices, the implementation of a network on an uncommon port helps to decrease the probability of conflict with standard communication services.

```
#include "painlessMesh.h"

#define MESH_PREFIX "Mojtaba44"
#define MESH_PASSWORD "M44444M"
#define MESH_PORT 5555
```

Figure 4.4. Username and password

**Developing multi task nodes by using task scheduler** :As discussed we should avoid using any delay to implement a network that works smoothly on transmission and receiving messages while concurrently it works on the background process to maintain the connection. The task scheduler is a function of the task-scheduler library to avoid using delay and implement a smooth multitask node for the WMN. In order to use the employment task scheduler, the object *userScheduler* is used as a parameter that contains the code's undergone tasks, and the defined object propagates the information to the sub-library to be processed. Figure 4.5 illustrates the definition of the object for Scheduler to control the tasks. In the same figure, *mesh* is The other defined object is to handle all the required functions for connection in WMN such as Node discovery and connection management , messaging , routing and synchronization, Handling network.

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

Figure 4.5. Definition of objects for Task Scheduler and PainlessMesh

**Send-Message task:** The function shown in figure 4.6 is responsible for broadcasting messages to the other node of WMN. The function constructs a message that contains the string *HiFromNode1* including the *Node – ID*. The interval is defined to set a tension time between 0 to 5 seconds, The reason for the interval is to prevent message collision and data congestion. This task will be scheduled by a routing to be frequently executed and transmit the constructed message by the defined routine.

```

void sendMessage() {
  String msg = "Hi from node1";
  msg += mesh.getNodeId();
  mesh.sendBroadcast( msg );
  taskSendMessage.setInterval( random( TASK_SECOND * 1, TASK_SECOND * 5 ) );
}

```

Figure 4.6. Send message Function in Painless Mesh

To have a stable bit-rate and controlled network we need to schedule a routing for the message transmission task. The task scheduler function shown in Figure 4.7 is implemented to schedule a task to be executed with the scheduled routing that is defined in the input of this task. The has 3 input parameters to set a routine, The first one is *TASK – SECOND \* 1* which defines the time interval to send a message, Setting value of this parameter to \*1 indicates that the ESP8266 transmitted the message every second. The second one is the parameter defined to set the number of repeats of the task, which is set to *TASK – FOREVER* to transmit the message indefinitely. The last parameter of the object that contains the content of the message that the ESP8266 transmits.

```

Task taskSendMessage( TASK_SECOND * 1 , TASK_FOREVER, &sendMessage );

```

Figure 4.7. Message Transmission Function in Painless Mesh

`newConnectionCallback()` shown in figure 4.8 is called to signal when a new node has joined the network. This function prints the chip ID of the joined node in the output. Also in case, any node was disconnected or the routing was updated in the frequent scanning of the node, the new connection callback function shown in figure 4.8 prints the ID of the new node.

```

void newConnectionCallback(uint32_t nodeId) {
  Serial.printf("--> startHere: New Connection, nodeId = %u\n", nodeId);
}

```

Figure 4.8. new connection callback

```
void changedConnectionCallback() {
  Serial.printf("Changed connections\n");
}
```

Figure 4.9. changed Connection Callback

The other notable function in PainlessMesh library is `nodeTimeAdjustedCallback()`. As the name represented it signals any offset more the threshold and call the synchronization sub-routine to adjust the offset between the nodes. The function and the printed output is presented below in figure 4.10.

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

Figure 4.10. nodeTimeAdjustedCallback Function in PainlessMesh

In the PainlessMesh library receive message is implemented by using event interrupts to a smooth network. When a message is delivered the interrupt signals the ESP8266 to read the received data. The function below in the figure 4.11 is the function to receive data by using interrupt, and it extracts the `&msg` from the received messages and print it in the output by UART. The printed output is the shared message in the WMN.

```
// Needed for painless library
void receivedCallback( uint32_t from, String &msg ) {
  Serial.printf("startHere: Received from %u msg=%s\n", from, msg.c_str());
}
```

Figure 4.11. Callback Function in PainlessMesh

## 4.2 Implementation of fog computing Using Luck Fox Pico Mini A in WMN

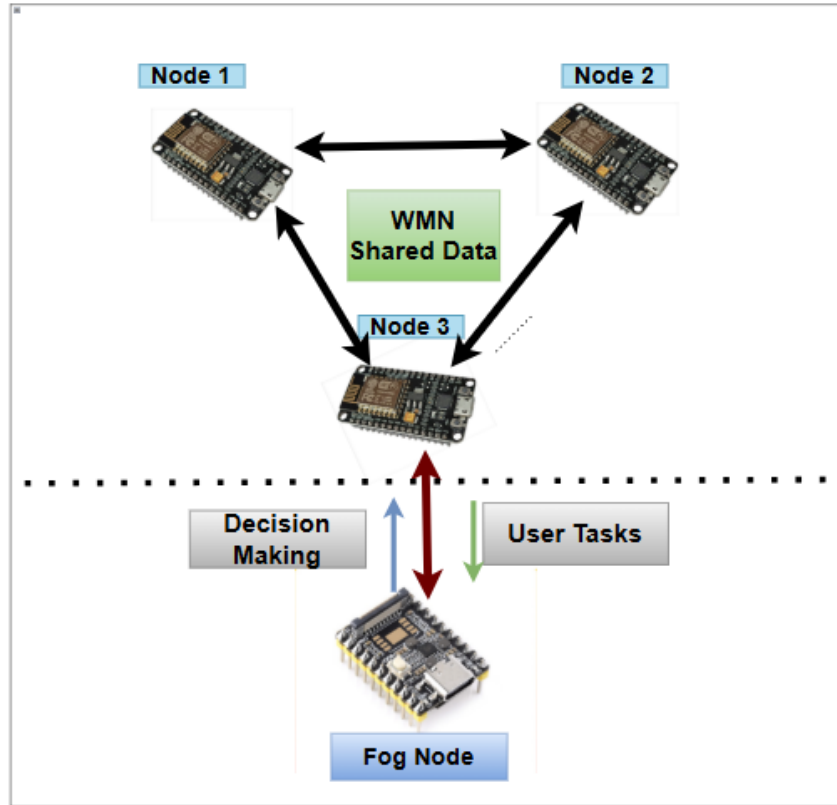


Figure 4.12. Fog Computing in WMN

**Fog computing overview** : Fog computing is a data processing method in that the generated data are processed close to the source and not transmitted to the cloud server that is at a distance. Fog computing helps reduce the latency of data transmission and offload processing overload. In this work, we use a Luck Fox Pico Mini A to work as a node fog, that takes the intensive user tasks from the ESP8266 microcontrollers and offloads any foreseen overload from the WMN network. Offloading overload from microcontrollers allows the WMN to focus on communication while the Luckfox board manages the processing and decision-making, the WMN functions as a real-time and reliable network [19]. As illustrated in figure 4.12 the three ESP8266 formed a WMN to share data. In this network each node shares its data to other nodes, and node 3 receives the data sent by the other two. Node 3 works as an intermediate node and transmits the data by UART to the Luck Fox board, which is node fog, and is processing the user task to offload the overload from the ESP8266 microcontrollers. The processed data and the decisions are transmitted from the Luck Fox board to the WMN.

### 4.2.1 Configuration and implementation of Luck Fox board

In the subsequent section, we will present the hardware and software configuration of the Luck Fox board. Initially, steps to install the Ubuntu Operating System (OS) and login to the system are presented. Hereafter, the Software configuration of UART and the hardware setup for UART connection between an ESP microcontroller and the luck fox board are discussed. Additionally, to print the received data from WMN and present the proper functionality of the setup, A serial monitor application using Python code is compiled on the Luck Fox board to receive the UART data. Finally, an image processing project built for the Android operating system is cross-compiled by the software development kit (SDK), and the application is run on the Luck Fox board to use the image processing capability of this powerful board.

**Installation of operating system** :Initially, we installed all the required drivers to use the rock chip board on Windows by the Driver Assistant tool that is shown in figure 4.13. This driver assistant identifies the architecture of the Rock-chip chipset and installs all the correct drivers on the operating system.

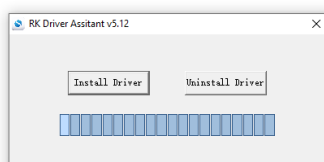
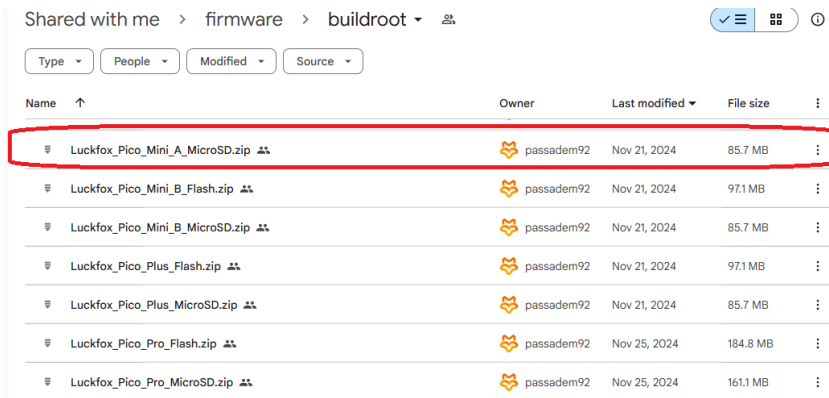


Figure 4.13. Driver assistant software

The first step is to download the operating system from the suggested link provided by the Luck Fox official website [16]. Among the operating system dedicated to Luck Fox Mini A, the built root version that is indicated Operating system in figure 4.14 is downloaded to burn SD card using the application that rock chip designed to burn OS built by this company.





Name	Owner	Last modified	File size
Luckfox_Pico_Mini_A_MicroSD.zip	passadem92	Nov 21, 2024	85.7 MB
Luckfox_Pico_Mini_B_Flash.zip	passadem92	Nov 21, 2024	97.1 MB
Luckfox_Pico_Mini_B_MicroSD.zip	passadem92	Nov 21, 2024	85.7 MB
Luckfox_Pico_Plus_Flash.zip	passadem92	Nov 21, 2024	97.1 MB
Luckfox_Pico_Plus_MicroSD.zip	passadem92	Nov 21, 2024	85.7 MB
Luckfox_Pico_Pro_Flash.zip	passadem92	Nov 25, 2024	184.8 MB
Luckfox_Pico_Pro_MicroSD.zip	passadem92	Nov 25, 2024	161.1 MB

Figure 4.14. Luckfox directory overview reproduced from [16]

In the following The chip that the board is equipped by is choose and the OS is burned on the SD card. The downloaded OS is burning on the formatted SD card Using the "Rockchip Soc toolkit" that is illustrated in 4.15 .

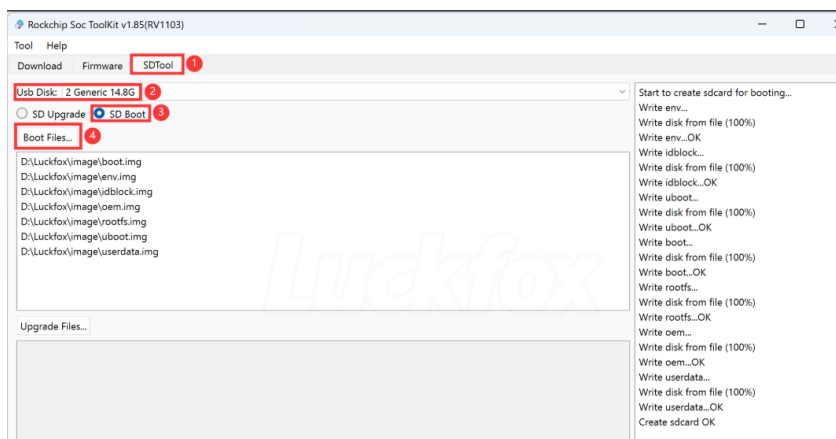


Figure 4.15. Create SD Card

To power run the Luck Fox board using the burned OS, the SD card is inserted into the SD card reader. Shown in 4.16 The board can be used by all the operating system and in this work we used windows and command promote to login to the Ubuntu environment.

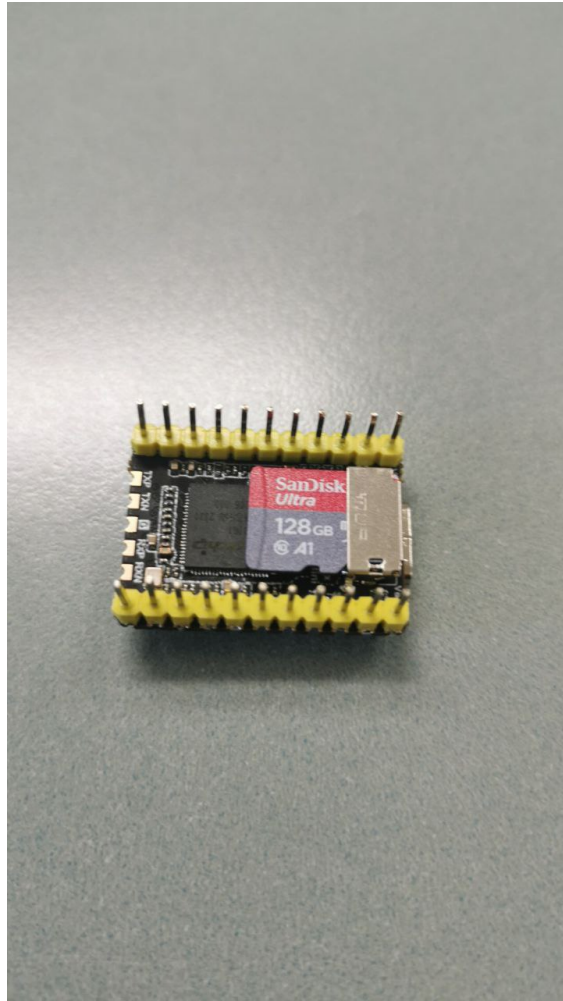


Figure 4.16. SD card slot on Luckfox Board

**ADB login** :The remote access and debugging to the Luckfox board through ADB (Android Debug Bridge) are provided. The Android Debug Bridge is a command-line tool that facilitates communication with the device. In figure below [4.17](#) the ADB command line environment and all the files in the build-root OS and the OS version are shown.

```
[root@luckfox ]# ls
bin                linuxrc            run
core-750-luckfox_pico_rt  lost+found        sbin
data              media             sys
dev              mnt              tmp
etc              oem              userdata
install          opt              usr
lib              proc              var
lib32            rockchip_test
lib64            root

[root@luckfox ]# cat /etc/os-release
NAME=Buildroot
VERSION=-g955ff6aa5-dirty
ID=buildroot
VERSION_ID=2023.02.6
PRETTY_NAME="Buildroot 2023.02.6"
[root@luckfox ]#
```

Figure 4.17. ADB environment in Luck fox board

**Configuring UART on Luckfox Pico Mini A** : The luck fox board is ready to be used, the next step is to configure the UART peripheral. Luck-fox Pico Min A supports UART(Universal Asynchronous Receiver-Transmitter), Which is one of the most common protocols for serial data. UART is important for debugging, firmware updates, and communication between peripherals, And we are using UART to establish a serial connection between Luck Fox and the ESP8266 network. The Luckfox-Pico-Mini-A includes dedicated UART pins (TX for transmission and RX for reception) on the GPIO header. In the default setting the UART0 is used for debugging, instead we use UART3 and UART4. In figure 4.18 we use the command *Luckfox – Config* we can access to to the advance setting of luck fox board , change the setting of UART and select UART3 or UART4 and Enable the UART by selecting enable option.

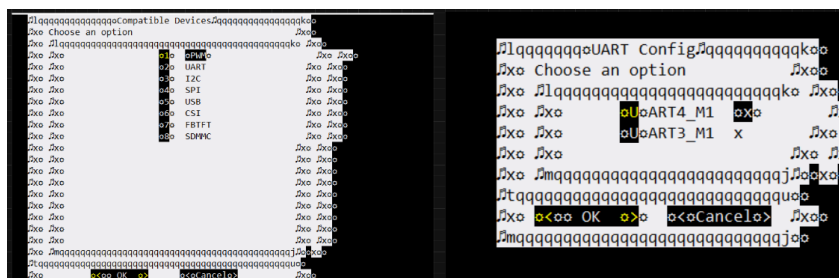


Figure 4.18. UART configuration in Luck Fox

Now we can see the pin configuration by using the command `luckfox-config show`. The figure below shows the pin configuration of the luck fox board after enabling the UART.

Figure 4.19 is the pinout of the luck fox board and we can see The UART3 and UART4 are marked by \* which indicates both of the GPIOs are configured as UART.

				+ -USB- +	
- FIQTTY_TX	- GPIO1_B2		VBUS	-	-
- FIQTTY_RX	- GPIO1_B3		VSYS	-	-
	- GND		GND	-	-
PWM11_M1 - UART4_M1_CTS	- GPIO1_C7		3V3_EN	-	-
PWM10_M1 - UART4_M1_RTS	- GPIO1_C6		3V3_OUT	-	-
PWM9_M1 - UART4_M1_TX	- GPIO1_C5		NC	-	-
PWM8_M1 - UART4_M1_RX	- GPIO1_C4		NC	-	-
	- GND		GND	-	-
PWM0_M1 - I2C3_M1_SDA	- GPIO1_D2		GPIO4_C1 - SARADC_M1	-	-
PWM11_M2 - I2C3_M1_SCL	- GPIO1_D3		GPIO4_C0 - SARADC_M0	-	-
PWM0_M0	- GPIO1_A2		NC	-	-
PWM2_M2 - SPI0_M0_CS0	- GPIO1_C0		NC	-	-
	- GND		GND	-	-
PWM4_M2 - SPI0_M0_CLK	- GPIO1_C1		GPIO4_A4	-	-
PWM5_M2 - SPI0_M0_MOSI	- GPIO1_C2		GPIO4_A3	-	-
PWM6_M2 - SPI0_M0_MISO	- GPIO1_C3		GPIO4_A2	-	-
PWM1_M0	- GPIO0_A4		GPIO4_A6	-	-
	- GND		GND	-	-
PWM3_M2 - *UART3_M1_TX	- GPIO1_D0		GPIO4_B0	-	-
PWM10_M2 - *UART3_M1_RX	- GPIO1_D1		GPIO4_B1	-	-
				+ - + - +	

SDMMC=1

Figure 4.19. Luck Fox Pin-out Configuration

**Python code Luck Fox Pico mini A to monitor UART port** In the previous Step, we initialize the UART 3 and UART 4 and we use UART 3 of the Luck Fox Pico Mini A to establish a real-time connection between the Enhanced Mesh network based on the Painless-Mesh library and using ESP8266. The Python Code illustrated in figure 4.20 creates a UART (Universal Asynchronous Receiver-Transmitter) a serial monitor to receive and print the received data on UART 3.

In order to receive messages, the function Closing and Listening for messages reads the receiving messages by UART 3 port, and in the Python code, The function "Serial Connection Setup" initializes the UART 3 with baud rate speed 115200, 8 data bits, 1 stop bit, and no parity. To prevent the program from facing a pseudo cycle while reading data, a 1-second timeout is specified. In the first part of the code, we start using UART 3 by setting the baud rate to 115200, 8 data bits, 1 stop bit, and no parity, which are similar to setting in the UART port in the ESP8266 MCU. In the following, the code constantly transmits the string "Hello World" on the UART 3 to test the transfer of any data by Luck Fox.

```
1 import serial
2 import time
3
4 try:
5     with serial.Serial(
6         "/dev/ttyS3",
7         baudrate=115200,
8         bytesize=serial.EIGHTBITS,
9         stopbits=serial.STOPBITS_ONE,
10        parity=serial.PARITY_NONE,
11        timeout=1,
12    ) as uart3:
13        while True:
14            uart3.write(b"Hello World!\n")
15            buf = uart3.read(128)
16            print("Raw data:\n", buf)
17            data_strings = buf.decode("utf-8", errors="ignore") # Avoid
18                               decoding errors
19            print("Read {:d} bytes, printed as string:\n {:s}".format(len(buf),
20                               data_strings))
21            time.sleep(1) # Adjust sleep time as needed
22
23 except KeyboardInterrupt:
24     print("Program terminated manually.")
```

Figure 4.20. Python-serial Monitor

**ESP8266 and Luckfox board setup** : To construct a setup for fog computing and receiving the serial output, The three ESP8266 form a WMN, one of the three microcontrollers shares the WMN data through UART by Luck Fox board. The Transfer channel of the ESP8266 is connected to the RX pin of Luck Fox to receive the data from WMN and print it on the output by the serial monitor application that we designed by Python in the previous section that is shown in 4.20 and the TX on the UART 3.0 of Luck fox board is transmitting a message frequently to be received by a VOVSOFT serial monitor to simulate a cloud server or a station that receives the processed data. Figure 4.21 presents the hardware setup of the WMN, communicating through UART to the Luck Fox board. On the right, the connection is simulated for better comprehension of the setup, and the shared data of the WMN received by UART3.0 is indicated by red indicator, while the transmitting serial data is indicated by green and received by a PC through a serial port.

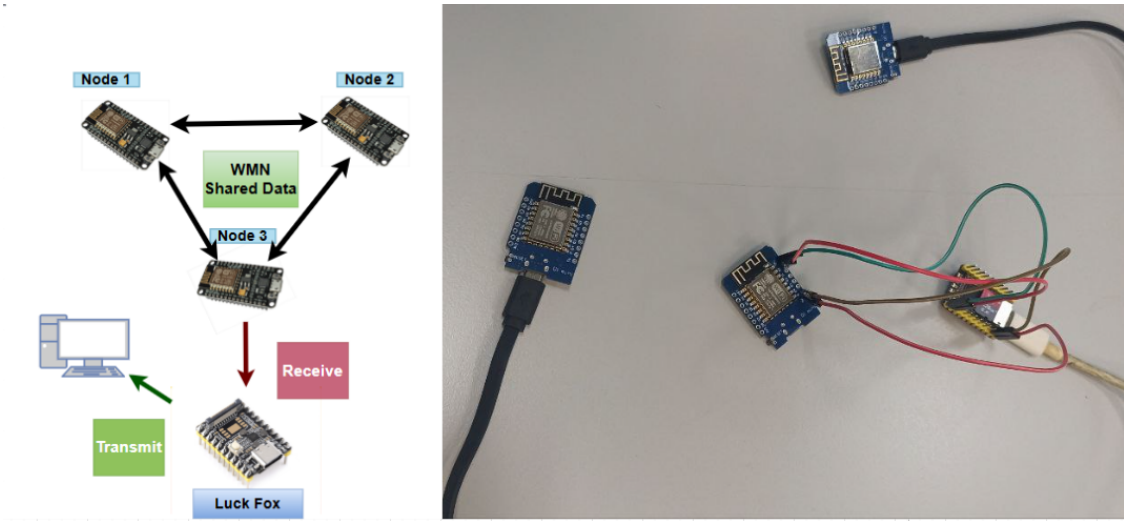


Figure 4.21. hardware confirmation of Luck Fox board as node fog in the implemented WMN

**SDK environment deployment** : SDK is a "complete software development kit" for developing and customizing applications on the Luckfox-Pico-Mini-A and other Luckfox-Pico series devices based on this chipset. The SDK allows developers to have all the tools, libraries, and documentation needed to write, debug, and deploy software at Luckfox-Pico. We use this feature to cross-compile any Project that is not compatible with Luck Fox Processor architecture, The SDK provides all the libraries, Drivers, and tools that we need to run the project on Luck Fox Pico Mini A. By using SDK, we can cross-compiler applications that are not designed for rock chip architecture. The Luckfox-Pico platform simplifies development by streamlining the process of customizing the firmware. In the first step we clone the luckfox-Pico-rtsp-open-cv shown in figure 4.22 from GitHub directory. [20] created by luck fox.

Platform Support			
DEMO	CPU	system	Camera
luckfox_pico_rtsp_opencv	RV1103(resolution adjustment required),rv1106	buildroot	sc3336
luckfox_pico_rtsp_opencv_capture	RV1103,RV1106	buildroot	sc3336
luckfox_pico_rtsp_retinaface	RV1103,RV1106	buildroot	sc3336
luckfox_pico_rtsp_retinaface_osd	RV1103,RV1106	buildroot	sc3336
luckfox_pico_rtsp_yolov5	RV1106	buildroot	sc3336

Figure 4.22. The image processing application created by open-Cv, reproduced from [20]

In the subsequent step, we cross-compile the cloned application by using the ./build.sh

command in the directory of the application, followed by the name of the application to start the cross-compile. After the cross-compiling all the required drives and libraries are generated, and the application can run on the Luck Fox. The figure bellow 4.23 illustrate the output of the executed application on the luck fox board. The access to the memory and camera represent that the application was executed successfully.

```
[root@luckfox install]# cd luckfox_pico_rtsp_opencv_demo/
[root@luckfox luckfox_pico_rtsp_opencv_demo]# ls
lib                               luckfox_pico_rtsp_opencv
[root@luckfox luckfox_pico_rtsp_opencv_demo]# chmod a+x luckfox_pico_rtsp_opencv
[root@luckfox luckfox_pico_rtsp_opencv_demo]# ./luckfox_pico_rtsp_opencv
Stop Application ...
killall: rkipc: no process killed
killall: udhpc: no process killed
rkipc exit
ls: /oen/usr/etc/init.d/S??*: No such file or directory
Create Pool success !
rkaiq log level ff0
ID: 0, sensor_name is U, iqfiles is /etc/iqfiles
Segmentation fault (core dumped)
[root@luckfox luckfox_pico_rtsp_opencv_demo]#
```

Figure 4.23. Open-CV on Luckfox

In conclusion, the node fog communicates to WMN through UART to receive the data. The node fig is capable of executing any application to implement its neural network and machine learning features to process the revived data from WMN through UART. As a result, concerning later examinations, the implemented capabilities of the Luck Fox board reduce the latency and enhance the reliability of the WMN, while all the ESP8266 can be focused on executing tasks related to the WMN, and the node fog handles the more intensive user task than we need for managing vehicular networks.





## Chapter 5

# Implementation results and validation

In the preceding chapter, the procedure for implementation of an enhanced WMN based on the ESP8266 MCU by painless mesh library was presented. Subsequently, configuration and initialization of the Luck Fox Board were discussed, and a Python serial monitor was designed to illustrate the received data from WMN in the output. In this chapter, the results of the implementations are presented; thereafter, the implemented mythology is validated through theoretical and experimental tests to present the enhancements on the limitations of the standard WMN. Initially, the printed output of WMN is discussed, which illustrates forming a full mesh WMN by 3 ESP8266 MCU.

### 5.1 Performance analysis enhanced WMN

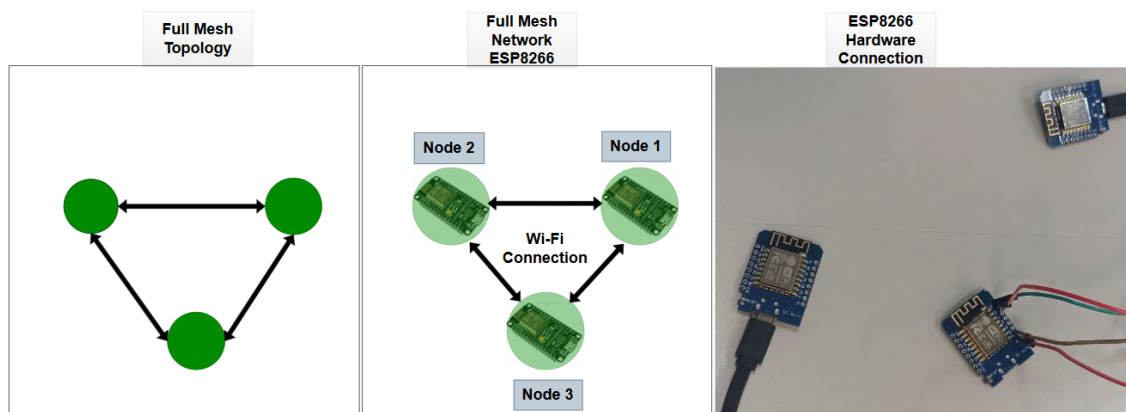
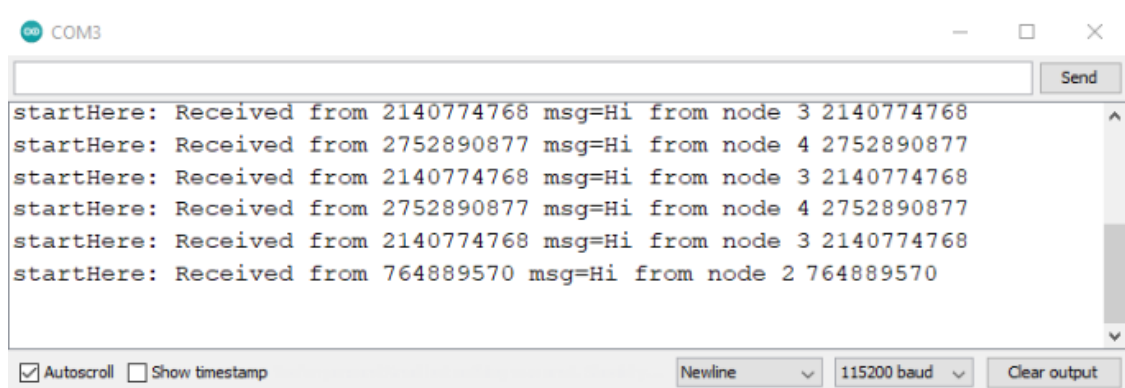


Figure 5.1. Hardware configuration of WMN

**Monitor mesh performance on UART** : The figure 5.1 illustrates the hardware configuration of the WMN by 3 ESP8266 that was discussed and presented in the previous chapter, which forms a full mesh wireless mesh network. Figure 5.1 contains three pictures, and Full mesh topology by 3 nodes has presented the network that we implement in our work. The picture full mesh network ESP8266 represents the diagram of the hardware connection that is shown in the last picture indicated as ESP8266 hardware connection. As figure 5.3, for each ESP8266 the number of the Node is different to recognize any change related to the node in the printed output. The output of the implemented WMN is shown in the 5.2. To monitor the output, the serial monitor of the Arduino is used. In the output, we can see that the output of the WMN is that in the first row the first value represents the ID of the node and the second value is the assigned number of each node. As in figure presented the 3 ESP8266 are working in the the WMN that we implemented by the Painless Mesh library. The true Ad-Hoc nature of this network is validated by experimental tests to present a self-configure network and the measurement of crucial metrics.



```
COM3
startHere: Received from 2140774768 msg=Hi from node 3 2140774768
startHere: Received from 2752890877 msg=Hi from node 4 2752890877
startHere: Received from 2140774768 msg=Hi from node 3 2140774768
startHere: Received from 2752890877 msg=Hi from node 4 2752890877
startHere: Received from 2140774768 msg=Hi from node 3 2140774768
startHere: Received from 764889570 msg=Hi from node 2 764889570
```

Autoscroll  Show timestamp  Newline  115200 baud  Clear output

Figure 5.2. WMN serial monitor

The formed WMN by ESP microcontrollers is connected to the Luck Fox Board which is implemented to receives data on UART and print the WMN data in the output using the application compiled on Luckfox using python. The The figure 5.3 shows the received data that are transmitted by one of the ESP8266 which forms a WMN and transmits the data of the other nodes to the UART of the Luck-Fox board.

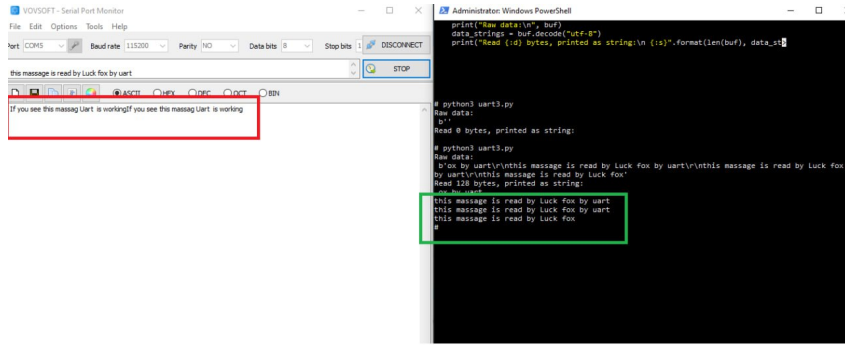


Figure 5.3. Serial Monitor on Luck Fox Board and PC to illustrate Fog Node function

## Formula and theoretical evaluation

In this section, we discuss and analyze the synchronization steps in PainlessMesh UDP. Furthermore, theoretically evaluate the payload size of standard mesh and Painless Mesh.

**synchronization in PainlessMesh** : Time synchronizing in Painless Mesh synchronizes all nodes within a network to have the same clock which allows nodes to perform operations in lock-step mode. This is implemented by using a distributed synchronization protocol, where each node periodically adjusts its clock local clock. Synchronization process aims to be highly precise within several milliseconds, and this precision must be sustained over time.

**synchronization Mechanism in Painless Mesh** : In PainlessMesh, the time synchronization Mechanism procedure is Periodically Synchronizing nodes of the network. The Periodic Time Syncing is Every 10 minutes which is just one part of the synchronization processor and in case of jitter that exceeds the  $10\mu s$  threshold, it will reduce the jitter to be lower than the threshold. Or so, the system initiates a time sync process as it expects all clocks in the network to be synchronized, in regard to prevent collisions of the synchronization requests, a random delay in the range of  $\pm 35\%$  of the interval is introduced to each synchronization cycle. Besides occasional synchronization, whenever a new node joins the network or connects to an AP, the time synchronization is also performed to ensure that new directly connected nodes are quickly synchronized with the rest of the network. Regarding sending a message, each node synchronizes its time only with its neighbor before the message transfer. In the first step, a synchronization request is sent from one node to another to decide if there is a need for a clock adjustment, and the originating node knows whether to adjust its clock or its neighbor's clock. To log this decision some rules should be considered regarding priority and number of connections to handle synchronization:

1. The node with fewer connections and sub-connections adopts itself time to sync with the network. The motivation is that if a jitter more than the threshold is detected consequently, connections and sub-connections must adopt their timestamp

with their neighbor so the node with fewer connections and sub-connections has fewer successive updates and will minimize the total network latency and improve convergence time.

2. If two nodes have an equal number of connections, the Access Point (AP) becomes the time master. This provides stability, as the AP is often a gateway and other nodes synchronize to its clock.

### Synchronization Triggered by Events in JSON :

Synchronization in a Painless-Mesh network is conducted by Jason message interaction between nodes, where timestamps and time requests are deployed to obtain synchronization. In the implemented mechanism for synchronization, as with every other task in Painless-Mesh to establish interaction between peers, JSON message is used for synchronization, so accordingly, before theoretical analysis of synchronization, we will analyze the JSON message schema in Painless-Mesh for synchronization.



Figure 5.4. Synchronization using JSON message, reproduce from [5]

**Synchronization using JSON message:** The figure 5.4 illustrates the JSON message schema that is used by PainlessMesh library for messaging. Trought the figure that rereset basic schema, and chages message in time synchronozation, we discuss and analyze this presedure to have a clear insight on JSON message-type.

**JSON message schema** : The schema of a JSON-Object message in the Painless-Mesh is shown in. Understanding JSON schema helps to comprehend better the synchronization of nodes of a Painless-Mesh network. In the JSON schema, "dest" refers to the destination node, "From" refers to the last intermediate node that this message passes through, and "Type" represents the type of the message. "Timestamp" is another parameter that is attached to this message, which indicates the time that this message is sent. The "timestamp" is a crucial parameter for synchronization, later on, it will be used in this work to evaluate trip travel(round Trip) and also one-way delay. Each node of WMN is an ESP8266 Wi-Fi Microcontroller equipped with an internal or external crystal to produce a resonance frequency that ESP8266 is used as a reference signal for the Microcontroller clock. To have a synchronized network, we need a time to be used as a parameter to compare and verify any time operation offset. For this purpose, each node has a timer, which generates a time that is attached to the JSON message to be used for different events of interactions of nodes of a network, like sending a message, requesting, or responding to an event to evaluate offset. The offset is used to synchronize a node with its neighbor node and keeps the communication synced during network operation.

When a new node joins it When a node joins a mesh network or connects to an access point (AP) it will check its time with the new connection.in case it does not need time adoption, it will send a request to the other node and request to start time synchronization by sending the message in figure ?? and the node that receives this message will check the synchronization by the other nodes that might have offset.

**Basic JSON schema** : On the other hand, if there is no need for time adoption the origin node will ask the other node a request time message that is shown in ??, the request packet contains an internal timestamp ( $t_0$ ) that indicates when the packet was generated then it receives that request.

**Time request** : :Now the receiver also adds two more timestamps, and the time of receiving the request and response time are attached to the message which contains  $T_0$ .

- $T_1$ : is The time that the request was received.
- $T_2$ : is The time when the response was generated

**Recipient Timestamps** :The response message is subsequently sent to the node that requested it, which records another timestamp ( $t_3$ ) when it receives the response. Accordingly, now we have a start time of request and the time that it arrives which can be considered as a way delay. Also, we have the time the reptant node transmits the response and the time that it arrives at a node that requests the time, which is a round-trip delay.To evaluate the time on using the internal clock, we will calculate both the offset at which the adopter is in the network and the round trip delay spent. By covering for the transmission delays, these values lead to correct time alignment between the nodes.

- $T_0$ : is the internal clock value of the adopter when the request is generated
- $T_1$ : is the time the responder receives the request

- T2: is the time the response is generated
- T3: is the timestamp when the response comes back to the adopter

**one-way delay evaluation** : Trip delay is the time that a message is broadcasting from a node until it arrives at the destination. By using Trip travel we also measure Round trip travel (RTT). RTT measurement is used to synchronize a message that travels from node A to node B and then travels back to node A.

$$\text{Trip Delay} = (T3 - T0) - (T2 - T1) \quad (5.1)$$

**Offset** : The offset time also can be evaluated by The difference between the two nodes' internal clocks. The above process is done iteratively until the computed offset falls within a threshold that is generally less than  $10\mu s$  and it typically takes about 2 to 4 rounds of initial synchronization. As mentioned before This process gets repeated roughly every 10 minutes by a tolerance of  $\pm 35\%$  to keep everything in sync over time.

$$\text{OFFSET} = \frac{(448585896 - 32990)}{2} + \frac{(448596056 - 63221)}{2} \quad (5.2)$$

$$= 448542871\mu s \quad (5.3)$$

$$\text{Trip-Delay} = (63221 - 32990) - (448596056 - 448585896) \quad (5.4)$$

$$= 20071\mu s \quad (5.5)$$

Trip Delay estimates the network latency between nodes. The Trip Delay is employed along with the offset for adjustment between nodes to enable accurate clock synchronization across the mesh network because it compensates for the time taken for messages to propagate to each node. [21]

**Payload size evaluation** : The payload is a segment of a data frame. Payload refers to two types of payload: 1-data payload and 2-malware payload. Malware payload is defined as a code that is used by malware to fully exploit and compromise IT networks and systems, but here we focus on data payload. A data payload is described as a payload associated with the transfer of data across a network. To calculate payload size, we should exclude the overhead of the data packet, as the payload is a segment of a data frame.

In other words, the data payload of a single network packet or other protocol data unit (PDU) is the data transferred within a packet that is sent from the source endpoint, and as well, network protocols limit the maximum length of the payload in the packet. The actual payload is then encapsulated in a packet that includes the media access control address and IP information, quality of service tags, time-to-live data, and checksum, among other things.

The payload size is the maximum amount of user data that can be transmitted in a single packet. Excluded overheads that the protocol adds, such as headers, footers, and other control information, are required for a data frame. In IP-based networks, The

maximum payload size of an IP packet is determined by the Total Length of the dedicated field in the IP packet header, which is 16 bits long, which means that the maximum acceptable value for the field is 216, so consequently the maximum permitted value of packet length is 65,535. Straightforwardly, no payload can exceed the determined size of a payload minus the header size, and if it does, for instance, if we have 1500 bytes and 100 headers, the payload size will be :

$$\text{Header size} = 1500 - 100 = 1400 \quad (5.6)$$

The maximum transmission unit (MTU) and Overhead size The formula to calculate payload size is :

$$\text{Payloadsize} = \text{MTU} - \text{Overhead} \quad (5.7)$$

- MUT : the maximum transmission unit(MUT)) refers to the maximum size of a packet that can be transmitted.
- overhead: The size of the headers and other control information added by the protocol are excluded of the payload.

Overhead in Mesh Networks includes MAC Layer Headers, Network Layer Headers, Transport Layer Headers, and Mesh Protocol Headers.which their size differs based on the WiFi standard.

**IEEE 802.11s standard payload size** : Wireless mesh networks can use the IEEE 802.11s mesh standard which defines a packet structure that is illustrated in 5.5 Here we assume that MTU is 1500 bytes which is typical for Ethernet and WiFi. MTU:1500 bytes (typical for Ethernet and Wifi)

Field	Frame control	Duration, id.	Address 1	Address 2	Address 3	Sequence control	Address 4	QoS control	HT control	Frame body	Frame check sequence
Length (Bytes)	2	2	6	6	6	0, or 2	6	0, or 2	0, or 4	Variable	4

Figure 5.5. IEEE 802.11 Frame

MAC header size: 30 bytes, including frame control, duration, address sequence control, and also Qos Fields.Mesh Header size: 6 bytes, which includes mesh flags, time to live, and sequence number of each data pack)

- Total Overhead calculation:
- IPV4 header size:20 bytes
- UDP header: 8 bytes
- FCS (Frame Check Sequence):4 bytes (for error detection)

$$\text{Total Overhead} = \text{MAC Header} + \text{Mesh Header} + \text{IPv4 Header} + \text{UDP Header} + \text{FCS} \quad (5.8)$$

$$= 30 + 6 + 20 + 8 + 4 = 68 \text{ bytes} \quad (5.9)$$

Accordingly, based on the explanation and formula of Payload Size Calculation:

$$\text{Payload Size} = \text{MTU} - \text{Overall Overheads} \quad (5.10)$$

$$= 1500 - 68 = 1432 \text{ bytes} \quad (5.11)$$

In this case, the maximum payload size is "1432 bytes". In other words, a "1500 byte" MTU contains 1432 bytes for user data and 68 for protocol overhead. [22]

## 5.2 Experimental validation

In the following, the ad hoc nature of WMN is validated by some experimental tests, and the results are discussed to present the evaluation of the experimental tests.

### Validation of self-organization network in challenging conditions

In this experimental test we form a WMN with two nodes with assigned names, Node 1 and Node 2. Node 1 is disconnecting frequently and the output is monitored to observe the self-organized function of Node 1 in the simulated critical condition. Figure 5.6 shows the output of that presents the functionality of Node 1. In the figure the first row of the output contains multiple values. The first value indicates the Node ID, the second value represents the time that a message is received from Node 1, and the last value of the row. In initial steps the communication is stable and we are receiving data from Node 1 almost at every 3 milliseconds and after 13 message transmissions, the "Changed connection" appears on the monitor for two times. Hereafter, the discussed UDP Phase in chapter 3 and section 3.3 is happening. **Phases of PainlessMesh user diagram protocol(UDP)**: Hereafter, all the 5 phases of UDP perform to re-establish a reliable connection, which can validate the self-organized nature of this network

**First phase of UDP ( Beacon exchange)** : The first one indicates that Node 1 is disconnected and the second one that appears immanently, which represents, a new connection to Node 1 is formed and re-joined to the network immanently and in the following by receiving its ID. The output represents the First phase of PainlessMesh UDP, which a Node 1 is broadcasting beacon signal and Node 2 confirms presence by sending an Acknowledge signal. **Second phase of UDP(Discover and synchronization)**: In the second phase Node 1 broadcasts mesh discovery message to inform all the nodes of the network. As we can see in figure 5.6 ID of Node 1 is printed after it broadcasts its message.

**Third phase of UDP(Routing and sub-connection list updates)** : Node 1 clears and updates all the sub-connections list and Node 2 also adds Node 1 to its list.



**Forth and fifth Phase of UDP(Synchronization) and messaging) :** In this Phase The new node start synchronization procedure and as it was sated the offset reaches below the threshold in 3 or 4 frequent attempt and reaches to less than 10 $\mu$ s it is presented in the first attempt the offset is too high, and 3292407678, in the next attempt the offset is -78718340 and finally it reaches to -85 that is 8.5  $\mu$ s which is below the 10  $\mu$ s .Then as is shown in figure Node 1 again keep transferring message. All the discussed steps and the presented output in figure validate the self-organize nature of the implemented enhanced network.

```

startHere: Received from 3592407678 at t1=18669681 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=21423807 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=25360340 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=27127122 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=29687179 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=32089838 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=37030247 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=40190131 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=44094607 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=46142522 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=49894886 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=50978669 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=55711257 msg=Hi from node13592407678
Changed connections
Changed connections
--> startHere: New Connection, nodeId = 3592407678
Adjusted time 7855888. Offset = -78718340
Adjusted time 8068547. Offset = -85
startHere: Received from 3592407678 at t1=11528607 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=16067701 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=20967586 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=22659371 msg=Hi from node13592407678
startHere: Received from 3592407678 at t1=26922322 msg=Hi from node13592407678

```

Figure 5.6. disconnection and re-join handling procedure

### experimental analysis of initial communication of a new connection and synchronization

In the experimental analysis of a WMN based on Painless-Mesh, two ESP8266 are programmed to create a WMN based on the Painless-Mesh, and the content data of JSON messages interacted during communication is monitored to analyze the parts of the data that present the finest steps of connection and interaction during communication.

```

COMMUNICATION: sendBroadcast(): msg=Hi from node13592407678
CONNECTION: scanComplete(): Scan finished
CONNECTION: scanComplete(): -- > Cleared old APs.
CONNECTION: scanComplete(): num = 10
CONNECTION: found : Mojtaba44, -41dBm
CONNECTION: Found 1 nodes
GENERAL: encodeNodeId():
CONNECTION: connectToAP(): Already connected, and no unknown nodes found: scan rate set to slow
COMMUNICATION: routePackage(): Recvd from 3574513645: {"type":4,"dest":3592407678,"from":3574513645,"msg":{"type":1,"to":64770987}}
COMMUNICATION: routePackage(): Recvd from 3574513645: {"type":8,"dest":0,"from":3574513645,"msg":"Hi from node13574513645"}
Received message from node 3574513645: Hi from node13574513645
COMMUNICATION: routePackage(): Recvd from 3574513645: {"type":4,"dest":3592407678,"from":3574513645,"msg":{"type":1,"to":66227443}}
COMMUNICATION: routePackage(): Recvd from 3574513645: {"type":4,"dest":3592407678,"from":3574513645,"msg":{"type":1,"to":66415915}}
COMMUNICATION: routePackage(): Recvd from 3574513645: {"type":8,"dest":0,"from":3574513645,"msg":"Hi from node13574513645"}
Received message from node 3574513645: Hi from node13574513645
COMMUNICATION: routePackage(): Recvd from 3574513645: {"type":4,"dest":3592407678,"from":3574513645,"msg":{"type":1,"to":66831909}}
COMMUNICATION: routePackage(): Recvd from 3574513645: {"type":8,"dest":0,"from":3574513645,"msg":"Hi from node13574513645"}
Received message from node 3574513645: Hi from node13574513645
COMMUNICATION: sendBroadcast(): msg=Hi from node13592407678
COMMUNICATION: routePackage(): Recvd from 3574513645: {"type":8,"dest":0,"from":3574513645,"msg":"Hi from node13574513645"}
Received message from node 3574513645: Hi from node13574513645
COMMUNICATION: sendBroadcast(): msg=Hi from node13592407678
SYNC: nodeSyncTask(): request with 3574513645
COMMUNICATION: routePackage(): Recvd from 3574513645: {"nodeId":3574513645,"type":6,"dest":3592407678,"from":3574513645}
SYNC: handleNodeSync(): with 3574513645
COMMUNICATION: sendBroadcast(): msg=Hi from node13592407678
COMMUNICATION: routePackage(): Recvd from 3574513645: {"type":8,"dest":0,"from":3574513645,"msg":"Hi from node13574513645"}

```

Figure 5.7. Logged JSON messages

The preview of the captured messages that starts from the initial steps of communication is shown in Figure 5.7 As we need to see JSON messages in the output, the previous code used for a WMN between 3 ESP8266 is modified, and adds a part of the code that sees the JSON messages.

```

// this comand enable all the mentioned data types in debugging
mesh.setDebugMsgTypes( ERROR | MESH_STATUS | CONNECTION | SYNC | COMMUNICATION | GENERAL | MSG_TYPES | REMOTE );

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();

```

Figure 5.8. Enable debug types

**Modification of ESP8266 code to monitor communication steps** as shown if figure 5.8 We replace the having (ERROR |STARTUP) with another command for Debug which in the output indicates:Error, Mesh Status, Connections, Sync, Communication, MSG-types, etc which allows us to monitor the messages of two peers in the duration of the communication.

```

void receivedCallback( uint32_t from, String &msg ) {
    // Parse the incoming JSON message and adjust the size
    StaticJsonDocument<200> jsonDoc;
    DeserializationError error = deserializeJson(jsonDoc, msg);

    // verify if the message the JSON object "type" = "Time-sync"
    if (jsonDoc.containsKey("type") && jsonDoc["type"] == "time-sync") {
        // T0 and T1 and Offset extract
        uint32_t T0 = jsonDoc["T0"];
        uint32_t T1 = jsonDoc["T1"];
        int32_t offset = jsonDoc["offset"];
    }
}

```

Figure 5.9. JSON parse code

while we receive a message an interrupt triggers and we read the received data but in normal communication, JSON messages that are serialized by the send will be de-serialized in the background and will be the demanded object of this message. as presented in the code in figure 5.9 we perform de-serialization on the received data the pass it to a filter that we create to have the messages that are related to time.

the filter checks the type of message and in case it is Time-Sync we capture it and parse T0, T1, and offset then these data will be printed in the output along with the node ID of the sender and the payload that the JSON was carried.

### First Step: Announce the presence :

*"COMMUNICATION : sendBroadcast() : msg = Hi from node13592407678"*

The figure 5.10 is the first step interactions of node 1 and node 2 are presented in which Node1 announces its presence to other nodes by broadcasting a message. When a node broadcasts it will send a message to all other nodes and not just one. The broadcasted message that is received by node2 is "COMMUNICATION: sendBroadcast(): msg=Hi from Node13592407678" which indicates node 1 is announcing its presence to other nodes and now node2 receives this packet of data. by authentication also node2 will be sure that node1 is capable of joining to node 2 and the network.

```

COMMUNICATION: sendBroadcast(): msg=Hi from node13592407678
CONNECTION: scanComplete(): Scan finished
CONNECTION: scanComplete():-- > Cleared old APs.
CONNECTION: scanComplete(): num = 10
CONNECTION:      found : Mojtaba44, -41dBm
CONNECTION:      Found 1 nodes
GENERAL: encodeNodeId():
CONNECTION: connectToAP(): Already connected, and no unknown nodes found: scan rate set to slow
COMMUNICATION: routePackage(): Recvd from 3574513645: {"type":4,"dest":3592407678,"from":3574513645,"msg":{"type":1,"t0":64770987}}
COMMUNICATION: routePackage(): Recvd from 3574513645: {"type":8,"dest":0,"from":3574513645,"msg":"Hi from node13574513645"}

```

Figure 5.10. A powered-up self-organized node interaction

### Second step: scanning for neighbor node :

*The message "CONNECTION : scanComplete() : Scan finished"*

Shown that the scan was finished and completed. Accordingly, it will delete the list of its connections from the previous Access points to ensure that the information that it will use and also be shared to other nodes for routing messaging is up to date. *CONNECTION : scanComplete() : num = 10*

**Third Step : Authentication and connection** : The message indicates that 10 WiFi networks during its scan, but the only WiFi node that with it, is just 1 in the authentication step just node 2 with SSID "Mojtaba44" with a signal strength of -41db which is acceptable and also it confirms the permission to establish a connection.

```

COMMUNICATION: sendBroadcast(): msg=Hi from node13592407678
CONNECTION: scanComplete(): Scan finished
CONNECTION: scanComplete():-- > Cleared old APs.
CONNECTION: scanComplete(): num = 10
CONNECTION:      found : Mojtaba44, -41dBm
CONNECTION:      Found 1 nodes
GENERAL: encodeNodeId():
CONNECTION: connectToAP(): Already connected, and no unknown nodes found: scan rate
COMMUNICATION: routePackage(): Recvd from 3574513645: {"type":4,"dest":3592407678,"f
COMMUNICATION: routePackage(): Recvd from 3574513645: {"type":8,"dest":0,"from":3574

```

Figure 5.11. Connection

**Fourth step: Scan-rate adjustment in stable mode** : *CONNECTION : connectToAP() : Alreadyconnected, and no unknown nodes found : scan rate set to slow*

represents that, as there is no node to be scanned, it will reduce its scan rate to a slow rate to reduce power consumption.

**Synchronization step: Time synchronization in JSON** :

*COMMUNICATION : routePackage() : Recvd from 3574513645 : "type" : 4, "dest" : 3592407678, "from" : 3574513645, "msg" : "type" : 1, "t0" : 64770987*

while nodes communicate synchronized in Painless-Mesh so that node2 receives a time synchronization request from the master node, which is T0=3574513645. In a message of type:4, a nested type:1 indicates a time sync request. The master node attaches a timestamp of its own, 64770987, to the message. This time stamp, known as t0, is the first part of the synchronization process in time. The client node sends the offset, so its timestamps (t1, t2, t3) to the server, so the server can calculate the start and end point of their clock with respect to the client clock. In Painless-Mesh, while we use JSON we have two types of messages 1-control messages and 2-user messages. Here we received a message that is a type 8 message. A type 8 is a JSON broadcast message that is type 8 which refers to a message. And ID of the transmitter which is 3574513645 and the message that was stored in the payload is attached to the message which is "Hi from Node1" and is followed by the ID node.

**Last Step: Receive a user-broadcast message in a synchronized communication**

:

```
COMMUNICATION : routePackage() : Recvdfrom3574513645 :
"type" : 8, "dest" : 0, "from" : 3574513645, "msg" : "Hi from node13574513645"
```

In a JSON message along with a time sync request, the node receives a user-defined broadcast message from master node 2, 3574513645. The message is identified by type message 8, which indicates it is a user-defined message that can be broadcast to all or unicast with a certain destination. Here, the "dest:0" field means the message is broadcast to all the nodes in the network. General-purpose broadcast messages for talking and transferring data between nodes. And we can state that communication reach is established and is stable in case of synchronization.

**Bit rate and packet loss evaluation in realistic condition**

: In this section, initially, we discuss the limitations of the enhanced WMN Mesh in in first section 5.2. In the following we also review the significant enhancement on these limitations in "Enhancement to Address Limitations in Painless-Mesh Library" 5.2 to have a clear insight of the Enhanced WMN before testing its limitations. Finally, the test setting and scenario will be defined and the logged out put of the experimental tests under challenging condition will be assessed and analyzed in the Last section 5.2.1, and concerning the analysis, some solutions are recommended in chapter 6 and section 6.1. This section plays significant role to fully understand the functionality of the WMN implemented by the Painless-Mesh library in realistic conditions.

**PainlessMesh network limitations** A WMN implemented based on a Painless-Mesh library is designed to function as a Reliable and Real-time network, but the mentioned proper functionality is dependent on some crucial parameters of the network. These limitations and dependencies require the user of Painless-Mesh WMN to exercise caution to avoid degrading the Reliability and Real-time of the WMN network performance. The Crucial Parameters are bitrate and number of sent messages per second. While theoretically, an increment in bit rate increases the number of sent messages at each second, Any non-conservative increase in bit rate and sending messages with unreasonable speed leads to an increase in data loss and makes the WMN not Reliable. The limitations in bit rate and number of sent messages mainly arise from the limited CPU frequency and RAM size of the ESP Microcontroller, which restrict the Microcontroller's ability to properly handle multiple tasks concurrently. In the Painless-Mesh library documentation, some recommendations are noted to bypass the probable problems related to these limitations. [5]. to address this limitation in the Painless-Mesh library document suggests trying to avoid using any delay in the code and instead use the task scheduler function that is described in chapter 3 and section 3.4.

**Enhancement to address limitations in PainlessMesh library** : Many of the problems that might disturb the proper functionality of Painless-Mesh are addressed by the provided tools and techniques in the Painless-Mesh library. The notable problem is high Power consumption, which can lead to loss of nodes in the network after a short time

and make routing difficult. In message routing, having fewer nodes in the network can increase data congestion, which causes overload on a microcontroller and negatively affects the Reliability and Real-time performance of the Network. This problem is addressed through dynamic power adjustment. To establish a reliable and stable network, proper synchronization and a true ad-hoc network are essential to ensuring a low data loss percentage. Synchronization and a true ad-hoc network allow synchronizing a non-real-time node during initializing a connection or in critical conditions. The provided solution is also analyzed and studied in the same chapter as power adjustment, the synchronization in the initializing of a new connection is studied in sections 5.1, and the experimental validation is presented in section 5.2. The use of Painless-Mesh a resilient and stable communication is provided, but Despite all the mentioned and tested enhancements, critical conditions that overwhelm the RAM and CPU of ESP8266 remain a concern, specially for functionality network in a vehicular network.

**Reduce overload through fog computing** : To mitigate the problem of overload, we employed fog computing. In fog computing, we use another board with a powerful CPU and enough capacity of RAM to process intensive tasks. In this work, we used Luckfox PICO Mini A will process the complex task and play the role of the fog node in fog computing. In the vehicular network, fog computing helps to reduce overload by handling intensive tasks. This reduces the RAM and CPU usage of the ESP8266 and mitigates the overload on the ESP8266 to reduce the negative impact of overload on bitrate and data loss. While fog computing significantly reduces the amount of data that an ESP Microcontroller should process, still a node in the vehicular network needs to perform some of the tasks individually. Excluded from tasks related to forming a WMN, In a vehicular network, each ESP Microcontroller is responsible for performing some essential tasks, such as utilizing an analog to digital converter (ADC) to measure the voltage level of the battery or using SPI / I2C peripherals to read GPS or IMU data, etc. for navigation to share the data in the network.

### 5.2.1 Experimental test of WMN under a simulated overload condition

: The discussed metrics (bit rate, message per second, and data loss) as explained are affected by Power supplies, synchronization, and some limitations of ESP8266 in RAM and CPU power. As previously explained, synchronization and power supply are addressed through specific solutions, but still we have some concerns caused by the CPU and RAM of the ESP8266. These limitations are significantly affected by the size of messages and the speed of sending messages. The size of each message is mainly determined by the payload size, while the speed of sending the message is defined in the code as a time interval that represents the gap time between the transmission of two messages. By testing different payload sizes and time intervals, we can identify an optimal point, which is the "Sweet spot," that the network performs at its maximum capability. Additionally, finding the sweet spot for Payload size and time interval helps to comprehend and define the payload size based on the required bit rate. Parsing a message is possible by different methods that allow reducing the payload size that can be used along with ArduinoJson its effect is explained in chapter 3, section 3.5. In terms of Reliability, it provides an understanding

of the trade-off between data loss percentage and the number of sent messages per second. Additionally, understanding the trade-off that is plotted in figure 5.12 allows us to define the Time interval based on the payload size to achieve a reasonable functionality in WMN based on the Painless Mesh.

**Test scenario** : We aim to simulate a vehicular fog computing overload ESP8266. In fog computing, the ESP8266 Microcontroller interacts with a high-frequency CPU to process complex tasks. Additionally, in a battery-powered vehicular network, voltage level monitoring along with other tasks for navigation is required. To experimentally simulate almost the same overload of fog computing, excluding the overload for the Painless-Mesh network, we expose an overload on ESP8266 by having a UART communication with Luckfox PICO Mini A. Additionally, the "data loss" evaluation is calculated in the code. As the calculation of data loss requires having each message with a sequence number, the receiver should also track the sequence of numbers to find a gap between this test to expose extra overload along with other tasks to the ESP8266 Microcontroller.

**Test setting** : Two ESP8266 are placed at a "5-meter physical distance", and no antenna is used in either of them. The Two ESP8266 are connected to "USB3 Ports" of a Laptop to provide power supply and monitor the output data. To monitor the output of UART protocol we use "Arduino" IDE and also to log the output data "VOVSOFTE" serial monitor software is used. USB3 can guarantee to provide proper power supply. As USB3 provides 900 mA continuous current. [23], which can satisfy the maximum continuous demanded current of an ESP8266 in "WiFi transmission mode". The maximum continuous current for ESP8266 is for max transmission power of ESP8266 is +20 db, which is needed in poor signal conditions. The initial current consumption of ESP8266 is the power-up surge current that is 300-400 according to the stated maximum continuous current and surge current consumption of ESP8266, USB3.0 can guarantee the proper power consumption for ESP8266 even in the worst cases. For each test, we log 100 messages with the defined payload size and time interval for transmission to achieve an acceptable and equal accuracy and the average value of the 100 tests is inserted in the table. 5.1

### 5.2.2 Evaluation and measurement of key metrics

:Here we evaluate several Key parameters of WMN, to offer a comprehensive analysis of the performance and throughput of the enhanced WMN under challenging conditions. The assessed parameters are bitrate, number of messages per second, and data loss. All the metrics are tested in an adverse condition, such that the microcontroller is overloaded. Considering and assessing these results in challenging conditions allows us to have a clear insight into recommendations to address the limitations.

**Bit rate calculation** :As explained key metrics define the performance of a networks and transmission speed of a network is fatal metric. Bit rate is represents the speed of data transmitting at each second which is indicated as bit per second. The formula to calculate bit-rate is shown below 5.12. To measure the bit rate we need to know the number of received bits per second. Each of the Transmitted messages contains a payload

size that is defined by the user in the code, and to provide a message with this amount of payload we used characters. As each character is 1 byte, while we create a message that carries 100 payload size, actually it contains 100 characters or 100 bytes.

$$\text{Bit rate (bps)} = \frac{(\text{number of received data (bytes)}) \times 8}{\text{Elapsed time(seconds)}} \quad (5.12)$$

- **Number of received data:** is the number of messages that in its payload contains the number of bytes, defined by the value of payload size in the code.
- **Elapsed time:** Elapsed time is the duration that the "Number of Received data" Value is counted. As we count the received data every 1 second the "Elapsed Time" is constant and value is 1.

**Message per second** : The "Message per second" metric provides an understanding of the network's capacity to handle the traffic load of message transmission. To calculate this metric we need the number of sent messages that are counted by the receiver at each second which is calculated by the formula that is shown below:

$$\text{Message per second} = \frac{\text{total number transmitted message}}{\text{elapsed time(second)}} \quad (5.13)$$

- **Total number transmitted messages:** It is the number of messages with a defined payload size that the Transmitter node sends.
- **Elapsed Time (Second):** it is the duration that the "Total number of transmitted message" Value is counted.

**Data loss calculation** : The evaluation of data loss in the network communication required labeling each of the sent messages with a sequence number. The receiver uses this sequence of numbers to identify data loss. The received messages are two types 1-user messages and 2-control messages. Control messages are used to manage connections and are used in the background and the receiver does not capture them. When the receiver node Captures a new message it compares the sequence number of the new message with the sequence number of the last received message. In case of identifying any gap between them, the number of losses will increase by the detected gap value. In the final step, the receiver uses the value of expected data which is the same as the payload size, and the value of loss messages to calculate the data loss percentage based on the formula stated below 5.14. The percentage of data loss is printed periodically at every second along with the bit-rate.



$$\text{Data loss percentage} = \left( \frac{\text{number of lost data}}{\text{number of expected data to receive}} \right) \times 100\% \quad (5.14)$$

**Number of lost data** : number of data loss is the number of packets that are not received.

**Expected packets** : The expected number of packets is the number of data that we expected while data loss is 0%.

### Description of logged output

: In figure 5.12 a sample of logged output from the receivers is illustrated. In this sample, the test scenario that was explained in previous sections in 5.2.1 is implemented for the payload size of 800 bytes and the interval time of 20 milliseconds. The receiver also logs the received packets calculates the value for bit-rate per second and counts the number of received messages in each second. Besides, the receiver tracks and monitors the sequence of the received packets to detect any gap between them to identify data loss. Each row indicates the data of a received message and values in each row in order represent the value of "sequence number", "Node number", "Node-ID" and the last one is "Payload Size" of each message that is sent every 20 milliseconds. In the illustrated sample just part of the received messages in 1 second with the sequence number 17 to 35 is shown.

```

Received #17 from 3592407678: 800 bytes
Received #18 from 3592407678: 800 bytes
Received #19 from 3592407678: 800 bytes
Received #20 from 3592407678: 800 bytes
Received #21 from 3592407678: 800 bytes
Received #22 from 3592407678: 800 bytes
Received #23 from 3592407678: 800 bytes
Received #24 from 3592407678: 800 bytes
Received #25 from 3592407678: 800 bytes
Received #26 from 3592407678: 800 bytes
Received #27 from 3592407678: 800 bytes
Received #28 from 3592407678: 800 bytes
Received #29 from 3592407678: 800 bytes
Received #30 from 3592407678: 800 bytes
Received #31 from 3592407678: 800 bytes
Received #32 from 3592407678: 800 bytes
Received #33 from 3592407678: 800 bytes
Received #34 from 3592407678: 800 bytes
Received #35 from 3592407678: 800 bytes
Received 35 messages, 28000 bytes in 1.01 seconds
Reception rate: 222664.00 bps (222.66 Kbps)
Packet loss: 2/243 (0.8%)

```

Figure 5.12. bit-rate, message/sec, and data loss for payload 800 and time interval 20ms

**Explanation of the logged output** : In figure 5.2.2 The represented values in the first row are the values of the received message with sequence number 17. The following printed value is the ID of the transmitter node, that is *Node – 3*. The next printed value representing the Node-ID, The Node-ID for Node-3 is 592407678. The last value of the first row is payload-Size for the messages which is 800 bytes. Receiving messages In order

allows the receiver to track the sequence and identify any lost packets. Here messages with sequence numbers 0 to 35 are sent. Considering that the total Number of Messages is 37 and 35 messages are Received we have 2 data packet loss which leads to 0.85% data loss as shown in figure 5.12.the bit rate is also calculated based on the formula 5.14 that was explained before.

**Evaluation of results : Bit-rate and message/Sec In different conditions :**

Payload Size (bytes)	Send message interval(ms)	Messages per Second	Bit Rate (Kbps)	Data Loss
100	20	53	42.95	0.0%
100	15	46.82	37.373	0.0%
300	20	37.06	86.52	0.0%
300	15	32.85	78.72	0.0%
600	20	29.65	164.17	0.0%
600	15	43.2	180.64	0.85%
800	20	36.73	203.82	0.68%
800	15	22.64	134.63	5.38%
1000	20	30.69	239.31	2.38%
1000	15	16.92	99.66	19.1%
1200	15	5.67	45.77	0.0%
1200	20	5.36	44.75	12.57%
1500	20	5.35	50.71	0.5%
1500	15	5.75	55.42	0.0%

Table 5.1. Message Rate and Bit Rate for Different Payload Sizes

**Introduction on The table : 5.1** The table provides the results of the experimental test by implementing a Painless-Mesh network using two ESP8266 Microcontrollers to perform some tasks and also form a Painless-Mesh network under the test condition mentioned in 5.2.1. The results provided by the table are averages of 100 periodical logs. The test is performed using a variety of fixed payload sizes for each transmitted message, ranging from 100 bytes to 1500 bytes. Then Each payload size is tested by two different message transmission intervals. The second column shows the transmission interval of messages, which are 15 ms and 20 ms. The third column of the table is dedicated to the number of sent messages per second. It represents the number of transmitted messages at each second that were successful, and the related "Ack" to this message is revived. A lower interval reduces the number of messages that we want to broadcast per second. The last column of the table is the Bit Rate (Kbps), which shows the data that represents the quantity of transmitted data in KBPS. Correspondingly, larger payload sizes suggest a

higher bit rate. The rightmost column, Data Loss (%), is the percentage of messages that were lost in transmission.

**Analysis Of The Test Results** :The result provided in the Table of Message Rate and Bit Rate for Different Payload Sizes 5.1 declares the average of logged messages per second, bit rate, and data loss for each (payload size, interval )setup. In the results, We see the following noteworthy trends:

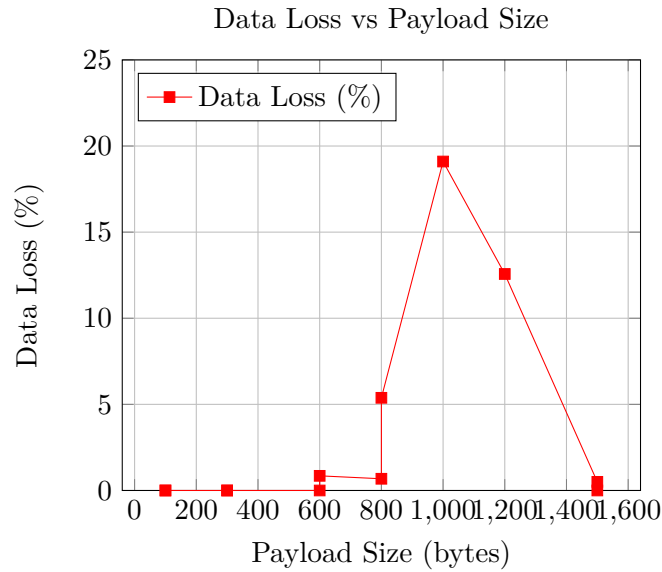
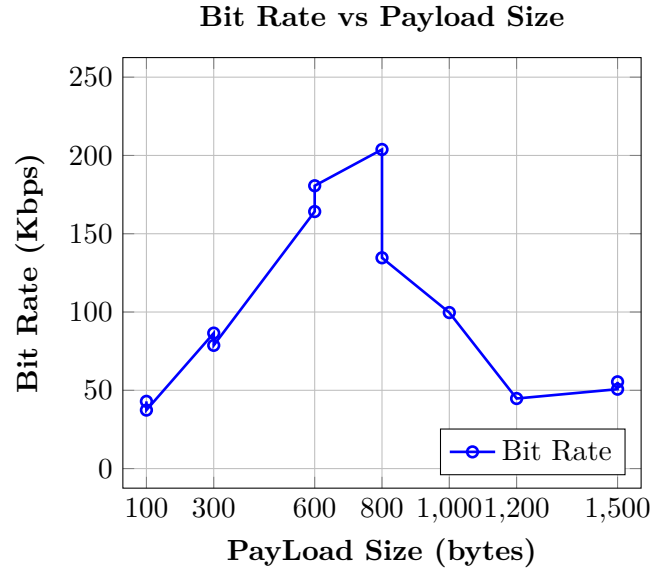
**High efficient payload range (500–600bytes)** :This range consistently provides high bit rates which is up to 164Kbps with negligible packet loss.It seems to be the “sweet spot,” where the overhead of the network is balanced with the payload size of the transmitted message.

**lower interval (15ms) vs higher interval(20-ms) impact** : Considering the shorter send interval, in theory, results in a rise in message rate, while our results show that under high processing loads on the network and the ESP8266 CPU the throughput can be degraded, or data loss can rise. In higher payload sizes (>800bytes) the the negative impact is more considerable.

**High data loss for large payload size ( $\geq 800$  bytes)** : The combination of high payload size and intervals can overwhelm the node’s internal buffers, which leads to a considerable packet loss (as high as 19.1%). As in the vehicular Network reliability is crucial, such high loss rates are unacceptable.

**Relation between bit rate and data loss vs. payload size** :The illustrated Plot in figure 5.2.2 displays the relation between Payload size vs bit rate and in figure 5.2.2 fluctuation of data loss percentage vs payload size is presented. The bit rate initially ascends as the payload grows, and peaks around 600 bytes which is the maximum achievable bit-rate in the test while data loss as shown in figure 5.12 is below 1%.Beyond the payload size 800 bytes the bit-rate declined due to an increase in data loss and network congestion.While the lowest bit-rate was achieved at 1000 bytes due to some issues in message transmission, surprisingly a slight recovery is shown in bit-rate at 1500 bytes. The increase of bit-rate in 1500-byte payload size in comparison to 1000-byte payload size is because of the overhead that Painless-Mesh adds to each message. The reason for this slight increase can be found in the overhead that the network added to each data frame. The payload is encapsulated in a data frame to be transmitted, and each data frame excluded payload contains overhead of IEEE802.11 and also Painless-Mesh that is evaluated in the section 5.1. So any increase in payload size reduces the required number of transmitted data frames which accordingly reduces the amount of overheads that we have at each second. These overheads are data that the receiver needs to process to control data transition and by reducing the amount of needed process the receiver can handle the connection As shown in figure 5.2.2, it leads to a better bit rate at 1500, additionally it less overload on RAM of ESP8266 can enhance the data loss percentage which reaches to less than 0% at 1500. Beyond, the payload size of 1000 by reducing

the amount of overhead, the data loss gradually drops to its initial percentage that was evaluated in the small payload size. The fluctuation of Data loss shows the importance of fast processing of the Microcontroller to send all the messages and handle any retry in case of failure in message transition.



**Evaluation of overload effect on PainlessMesh network:** Compared to state-of-the-art (SOTA) measurements in "Performance Evaluation of ESP8266 Mesh Networks" [10], which avoids any overload on the ESP8266 to evaluate the maximum capability of the network, our Test imposes overloads that result in a change in the under-test network

efficiency. The results represent a reduced margin before packet loss, which expresses the importance of the experiment. The results of the test in realistic situations provide a clear and realistic overview that assists in considering the Hardware limitation or implementing solutions suggested in chapter 6, section 6.1 to address the envisioned limitation to keep the network on its efficient, reliable performance.

**Comparative performance of lightweight vs. overloaded PainlessMesh** : In previous research [10], the lightweight performance of the Painless Mesh network was analyzed in a minimal processing overload to ESP8266 CPU. In the research two ESP8266 are used as a sender and a receiver along with forming a WMN the two ESP8266, toggle an LED and also serial print the outputs by UART peripheral as user tasks which are basic user tasks. Minimizing user tasks allows ESP8266 Microcontroller to mainly focus on forming a Painless-Mesh-based WMN and the network almost performs at maximum achievable performance of a Painless-Mesh network. The provided results observed a maximum reception rate of 461 messages/sec with a 10-byte payload decreasing to 28 messages/sec for a 4400-byte payload. The research declares that the payloads further than 4400 bytes will result in data loss and wrong messages. On the contrary, our experiments can reflect the Painless-Mesh-based WMN that involves various tasks to simulate the overload that we may experience while implementing a Painless-Mesh network using ESP8266 to have a vehicular network using fog computing. In a vehicular network, each ESP8266 may involve additional tasks like communicating with fog node and essential local data processing such as analog to digital convert (ADC) to read voltage level or reading the data of IMU and GPS via SPI or I2C peripherals to be used in navigation. All these essential tasks inherently add more computational overload to the CPU of the ESP8266 Microcontroller. so the different situations of the test result in different outputs and analyses that indicate that in applications that raise overload for ESP8266, there is a trade-off between throughput and payload size. The baseline study [10] achieved higher message rates by low overhead, our measurements indicate that, a balance between Time interval and payload size, the Painless-Mesh-based WMN peaks at approximately 500 to 600 bytes payload with approximately 164 Kbps at a 20ms interval for message broadcast. This sweet spot leads to a 0.85% packet loss. Besides this “sweet spot, ” while the payload is increased due to the overload of ESP8266 the throughput decreases and the packet loss has a significant increase.



## Chapter 6

# Conclusion and recommendation

In the conclusion chapter, the summary of the work provides a review of the implemented methodology to enhance the standard WMN and also reviews the experimental validation to present the achieved improvements and enhancements in this work. In this work initially, the limitations of WMN are discussed and to enhance the limitations, The WMN is implemented based on a Painless-Mesh library and ESP8266 for a vehicular network. In the following, we analyzed and studied both ESP8266 and the painless mesh library to implement an ad-hoc network to address the problems regarding complex connection and routing. Moreover, the Painless mesh library is used to implement the enhanced network and improve its reliability and real-time functioning through the tools that the library provides. In the following Fog commuting by using the Luck Fox board is utilized to address the limitation of the ESP8266 in terms of processing. Finally, the true ad-hoc nature of the network is validated by some experimental tests, the tests validate the self-organizing and self-configuration of the implemented network.

### 6.1 other recommended solutions

The enhanced network that is implemented in this work provides a real-time, reliable, and scalable network, and the limitations of the standard WMN are resolved, but as discussed and presented in the tests the limitation of the RAM and CPU of ESP8266 imposes a negative impact and makes this network vulnerable for intensive tasks. In other to improve the bit rate while the power supply and processing power are not affected inactive antennas can be used. The antenna increases the transmission and receiving power while the transfer power of the microcontroller that has a significant direct impact on power consumption is not increased. Additionally, as shown in the experimental test any overload on the CPU of the ESP8266 microcontroller can lead to latency and data loss, implementing DMA(Direct memory access) can improve the functionality of the network. While the ESP8266 interacts with the fog node, direct messaging allows the fog node to access the memory of ESP8266 and read and write the data directly while the microcontroller focuses on controlling tasks related to WMN.





# Bibliography

- [1] Yi Xu and Wenye Wang. “Wireless Mesh Network in Smart Grid: Modeling and Analysis for Time Critical Communications”. In: *IEEE Transactions on Wireless Communications* 12.7 (2013), pp. 3360–3371. DOI: [10.1109/TWC.2013.061713.121545](https://doi.org/10.1109/TWC.2013.061713.121545).
- [2] Ian Akyildiz, Xudong Wang, and Weilin Wang. “Wireless Mesh Networks: a survey”. In: *Computer Networks* 47 (Mar. 1, 2005), pp. 445–487. DOI: [10.1016/j.comnet.2004.12.001](https://doi.org/10.1016/j.comnet.2004.12.001).
- [3] Mojtaba Seyedzadegan et al. “Wireless mesh networks: WMN overview, WMN architecture”. In: *International conference on communication engineering and networks IPCSIT*. Vol. 19. Citeseer. 2011, p. 2.
- [4] Yan Zhang, Jijun Luo, and Honglin Hu. *Wireless mesh networking: architectures, protocols and standards*. CRC Press, 2006.
- [5] *painlessMesh / painlessMesh · GitLab*. GitLab. Oct. 30, 2024. URL: <https://gitlab.com/painlessMesh/painlessMesh> (visited on 02/22/2025).
- [6] Eiman Alotaibi and Biswanath Mukherjee. “A survey on routing algorithms for wireless Ad-Hoc and mesh networks”. In: *Computer Networks* 56.2 (2012), pp. 940–965. ISSN: 1389-1286. DOI: <https://doi.org/10.1016/j.comnet.2011.10.011>. URL: <https://www.sciencedirect.com/science/article/pii/S138912861100377X>.
- [7] Ian F. Akyildiz, Xudong Wang, and Weilin Wang. “Wireless mesh networks: a survey”. In: *Computer Networks* 47.4 (2005), pp. 445–487. ISSN: 1389-1286. DOI: <https://doi.org/10.1016/j.comnet.2004.12.001>. URL: <https://www.sciencedirect.com/science/article/pii/S1389128604003457>.
- [8] Anatoli Arkhipenko. *arkhipenko/TaskScheduler*. original-date: 2015-09-05T01:21:38Z. Feb. 21, 2025. URL: <https://github.com/arkhipenko/TaskScheduler> (visited on 02/22/2025).
- [9] *ESP32Async/ESPAsyncTCP*. original-date: 2025-01-20T16:37:10Z. Feb. 19, 2025. URL: <https://github.com/ESP32Async/ESPAsyncTCP> (visited on 02/22/2025).
- [10] Yoppy Chia et al. “Performance Evaluation of ESP8266 Mesh Networks”. In: *Journal of Physics: Conference Series* 1230 (July 1, 2019), p. 012023. DOI: [10.1088/1742-6596/1230/1/012023](https://doi.org/10.1088/1742-6596/1230/1/012023).

- 
- [11] Yanjun Xiao et al. “Decentralized Self-Organizing Networks for Poor Reconfigurability of Loom Networks”. In: *IEEE Access* 10 (2022), pp. 50487–50499. DOI: [10.1109/ACCESS.2022.3173415](https://doi.org/10.1109/ACCESS.2022.3173415).
- [12] Shabir Ali, Mayank Pandey, and Neeraj Tyagi. “SDFog-Mesh: A software-defined fog computing architecture over wireless mesh networks for semi-permanent smart environments”. In: *Computer Networks* 211 (July 1, 2022), p. 108985. DOI: [10.1016/j.comnet.2022.108985](https://doi.org/10.1016/j.comnet.2022.108985).
- [13] Microcontrollers Lab. *ESP8266 pinout reference and how to use GPIO pins*. Microcontrollers Lab. May 19, 2019. URL: <https://microcontrollerslab.com/esp8266-pinout-reference-gpio-pins/> (visited on 02/24/2025).
- [14] Espressif Systems. *ESP8266 Technical Reference*. 2023. URL: [https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf).
- [15] BenoitBlanchon. *ArduinoJson Assistant 7*. ArduinoJson. URL: <https://arduinojson.org/v7/assistant/> (visited on 02/26/2025).
- [16] *Luckfox Pico Mini A RV1103 Linux Micro Development Board, Integrates ARM Cortex-A7/RISC-V MCU/NPU/ISP Processors, onboard 64MB memory*. URL: <http://www.luckfox.com/Luckfox-Pico-Mini-A> (visited on 02/25/2025).
- [17] *Arduino Integrated Development Environment (IDE) v1 | Arduino Documentation*. URL: <https://docs.arduino.cc/software/ide-v1/tutorials/arduino-ide-v1-basics/> (visited on 03/29/2025).
- [18] *Software | Arduino*. URL: <https://www.arduino.cc/en/software/> (visited on 03/29/2025).
- [19] Javiera Valenzuela et al. “Water crisis in agriculture in the Casablanca Valley, low-cost monitoring of a fog catcher with ESP8266”. In: *2024 IEEE International Conference on Automation/XXVI Congress of the Chilean Association of Automatic Control (ICA-ACCA)*. 2024, pp. 1–6. DOI: [10.1109/ICA-ACCA62622.2024.10766791](https://doi.org/10.1109/ICA-ACCA62622.2024.10766791).
- [20] *LuckfoxTECH/luckfox\_pico\_rkmpi\_example*. original-date: 2024-08-20T07:05:20Z. Feb. 27, 2025. URL: [https://github.com/LuckfoxTECH/luckfox\\_pico\\_rkmpi\\_example](https://github.com/LuckfoxTECH/luckfox_pico_rkmpi_example) (visited on 03/04/2025).
- [21] *mesh protocol · Wiki · painlessMesh / painlessMesh · GitLab*. GitLab. Dec. 7, 2018. URL: <https://gitlab.com/painlessMesh/painlessMesh/-/wikis/mesh-protocol> (visited on 03/08/2025).
- [22] “IEEE Standard for Local and metropolitan area networks—Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)”. In: *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)* (2011), pp. 1–314. DOI: [10.1109/IEEESTD.2011.6012487](https://doi.org/10.1109/IEEESTD.2011.6012487).
- [23] *What are the Maximum Power Output and Data Transfer Rates for the USB Standards?* July 17, 2024. URL: <https://resources.pcb.cadence.com/blog/2020-what-are-the-maximum-power-output-and-data-transfer-rates-for-the-usb-standards> (visited on 03/15/2025).