Comparing communication network topologies for low power microcontrollers

Dennis Matena
s2163047
d.h.matena@student.utwente.nl

***Abstract*—This research discusses the design and evaluation of two communication network topologies, in which the ESP32 microcontroller is used as a network node. These networks are designed to create a communication system across multiple Electric Vehicle (EV) charging stations. The investigated topologies are the hierarchical network and the mesh network. These network topologies are evaluated on multiple aspects: the hardware usage (RAM and Flash storage), the time needed for a node to send data, the time needed for an entire send and receive cycle and the effects of the network size on those timings.**
The hierarchical network is faster with sending data than the mesh network and the hierarchical uses less RAM as well, but the mesh network is more suitable for the communication system, because of network robustness.

***Keywords*—** Microcontroller, Hierarchical Network, Mesh Network, Smart Grid

I. Introduction

Sustainable energy is becoming the standard nowadays. Whereas 30 years ago, fossil fuels were being used for almost all of the production of electricity, more and more households are being powered by solar energy and other sustainable means. Not only the production of electricity is becoming more sustainable, cars are also switching to more environmentally friendly ways. While cars used to solely drive on diesel and petrol, numerous cars are nowadays being powered by electricity.
The main issue with the increase of electric cars, is that they need to be charged often. Many cars are being charged after a working day of the driver. As a working day ends around the same time for most people, they all plug their car into the electrical grid around the same time. This causes the problem of peak power usage around that time and barely any power usage later in the night, when the car battery has been fully charged. The current electrical grid is not able to cope with the influx of electric vehicles[4].
A solution to this problem is by performing smart measurements to control the charging behavior. This has been implemented by using energy efficient microcontrollers[1], which is able to distribute the power load over the course of the night. Now that chargers are able to calculate their individual ideal power distribution, the next step is to get multiple chargers to communicate with each other and create a (stable) network. This way, they are able to send their data to the other chargers, and thus create the best charging pattern for all the chargers. This leads to the following question:
RQ: Which network topology delivers the best network stability, while maintaining high data throughput for smart Electric Vehicle (EV) charging?

To better understand the problem and to start working towards a solution, background info is provided in Section II. After the background information, multiple suitable communication networks for this problem are described in Section III. After an explanation of the networks, their implementation of the algorithm are explained in Section IV, as well as the hardware and software utilized in Section V. The benchmark methods are described in Section V and the results of those benchmarks in Section VII. The results are followed by the conclusion in Section VIII and future work in Section IX.

II. Background

Gerards et al. [6] describe a way to perform energy managment using profile steering efficiently. They first present drawbacks of using energy prices as steering signals. They conclude that using price steering can cause problems in power quality and a loss in efficiency.
After price steering, they proposed to use flat power profiles instead of energy prices and compare the flat power profiles to price steering and uniform pricing. The flat power profiles resulted in a better power quality and lower distribution losses. When comparing to uniform pricing, it resulted in a reduction of distribution losses of 48%.

Böhmer [1] developed an algorithm which only uses a single core of the ESP32 microcontroller to determine the most efficient power planning for an EV charger. The algorithm uses measurements from set intervals, after which it will calculate when it is best to turn on the charger and to set the power level of the charger. [1] also concludes that the ESP32 microprocessor is better than the ESP32-S2 and ESP8266 counterparts, in terms of speed of the algorithm.

Hoogsteen et al. [4] present a stress test to test if the current electrical grid can cope with the expected power usage in 2025. The simulated this by stress testing the electrical grid in Lochem. They came to the conclusion that the current electrical grid cannot cope with the estimated usage in 2025 without measures, such as Demand Side Management (DSM).

III. Networks

This section covers the operation method of the communication network topologies used to discover the solution of the research question. Two different types of networks are used and evaluated. These networks are the hierarchical network and the mesh network.

A. Hierarchical Network

A hierarchical network consists of a coordinator node and one or more subnodes. These subnodes have direct contact with the coordinator node, but they do not have contact with each other. The weakest link within this network topology, is the coordinator node. The coordinator node can fail or lose a link, which results in the entire network to no longer be operational or a node which cannot be accessed anymore. A diagram of a hierarchical network is shown in Figure 1.

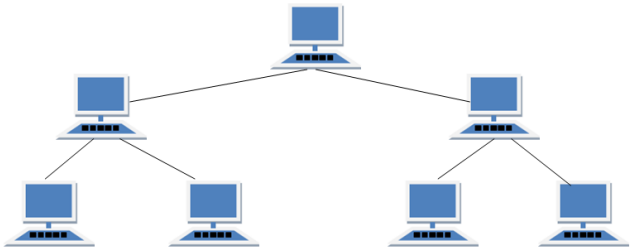

Figure 1. Diagram of a hierarchical network[5]

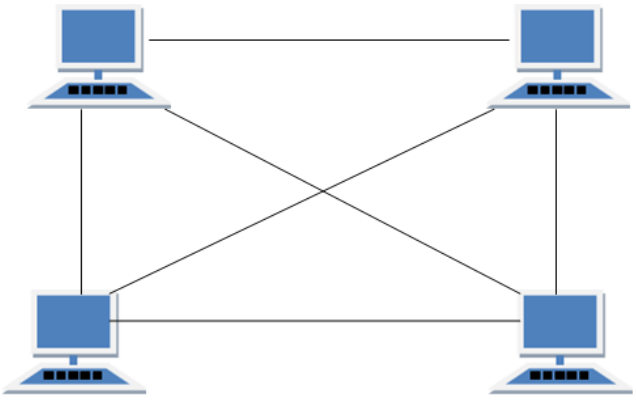

Figure 2. Diagram of a mesh network[5]

B. Mesh Network

A mesh network consists of multiple nodes which are connected to each other, instead of to a coordinator node. This allows for any node to fail, but still keeping the network fully operational, for example by re-routing the messages. This in turn results in the stability of a mesh network to be higher than the stability of a hierarchical network. A diagram of a mesh network is shown in Figure 2.

IV. Algorithms

This section describes the functionality of the algorithms for both network topologies. The main goal of the algorithm is to have one node sending a request, followed by the other nodes sending a message to the node requesting data. This message is an array, consisting of 480 float values.

A. Hierarchical network

At first, the coordinator sends a request to one of the nodes, asking for the node to send its data. Once a subnode has received the request, it will start sending fragments to the coordinator. Since the amount of data in a single message is greater than the maximum size of a single message (250 bytes, which corresponds to a maximum of 62 float values[2]), the message needs to be fragmented into smaller fragments. The layout of a fragment is shown in Figure 3. A fragments consist of the following:
- A sequence number to indicate which part of the message is being sent in that fragment.
- A variable to determine whether it is the final fragment of the entire message.
- A flag to determine if the fragment is a request or response.
- Which node is transmitting the data.
- The length of the fragment.
- The size of the entire message.
- The message.

These fragments will be sent one by one until all the fragments have been received. When the coordinator has received a fragment, it will store the content in an array, which results in the final contents of the array to be exactly the same as the contents of the total array sent by the broadcasting subnode.

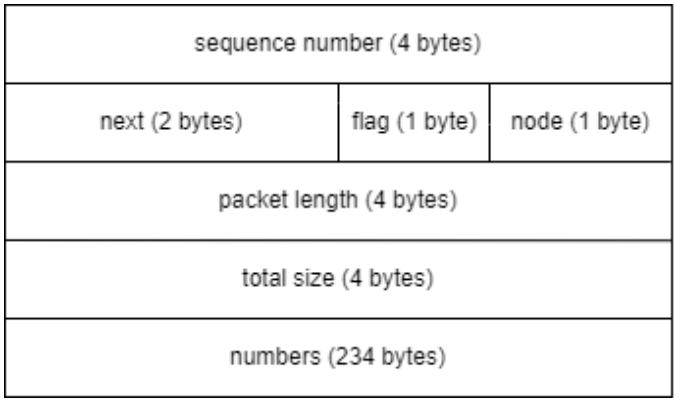

Figure 3. fragment layout of a hierarchical network

B. Mesh network

The start for sending messages works different for a mesh network in comparison to a hierarchical network. Since there is no coordinator node, a mechanism needs to be designed to avoid congestion and possible loss of data. This was not mentioned in the library used for this protocol, thus it had to be implemented manually. This is done by having one node sending a small request. This request is the same as a request in a hierarchical network. Sending this request is achieved by having every node setting a random timeout. This timeout exists to prevent a node from direct sending of data and to allow nodes to join the network before a node sends out a request. This results in the minimum timeout to be set to 10 seconds. The maximum timeout depends on the total amount of nodes in the network, where 2 seconds are added to the maximum timeout for each node in the network. For example in a network with 4 nodes, the minimum timeout is 10 seconds and the maximum timeout is $10 + 4 \cdot 2 = 18$ seconds. A change in the network size (a node joining or disconnecting from the network) resets this timeout. For instance if the node was accidentally disconnected, the node has time to reconnect within that timeout period.
When the other nodes have received the request. They will set a random timeout to avoid congestion. This timeout will be elaborated further in Section VII-C2.
Once this timeout has passed for one of the nodes, it will broadcast a single message to the other nodes, which does not yet contain any information from the broadcasted array. This allows that specific node to broadcast the actual message and block other nodes from simultaneously broadcasting a message, and hence avoiding collision.
After the initial fragment, the entire message will be fragmented into smaller fragments. The layout of one of those fragments can be seen in Figure 4. The only difference between a fragment in a Mesh network versus a fragment in the hierarchical network is the size of the array in the fragment. The motivation behind this difference is that more bytes can be transmitted with the Painlessmesh library than with the ESP_NOW library[7].

The PainlessMesh protocol utilizes Strings to broadcast messages, which requires the message to be transformed to a String before broadcasting. This is done by converting the fragment into the JSON format. After this conversion, the fragment is broadcasted to all the other nodes in the network. At the receiving end, all nodes which received the fragments will combine every fragment back into one single array. If this step has been completed, one node has

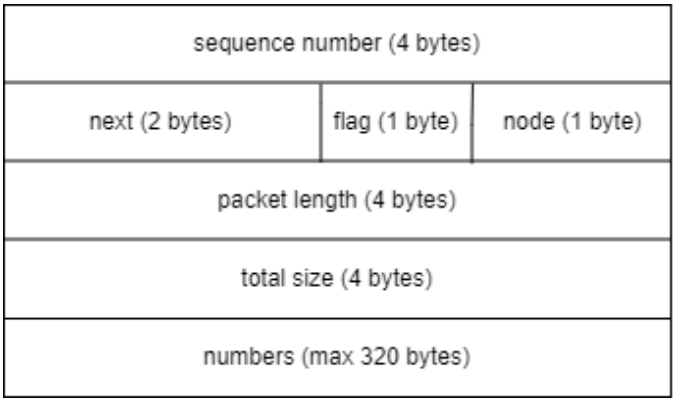

Figure 4. fragment layout of a mesh network

Table I
SPECIFICATIONS OF THE ESP32 MICROCONTROLLER

RAM	520 KB
Flash storage	4 MB
WiFi	WiFi 4(up to 150 Mbps)

successfully sent all its data and the other nodes are allowed to send next.
Once the entire message has arrived, the nodes who have not yet broadcasted their message will set a timeout. This is the same random timeout as the timeout set when the nodes receive a request. The node which just sent its message will set its timeout to the starting timeout (between 10 seconds and 10 seconds with 2 extra seconds for each node in the network). This is to ensure every node to be able to send the message and refrain a single node from constantly sending the messages.

V. TEST PLATFORM

The microcontroller used for the creation of the communication network is the ESP32 microcontroller by Espressif [3]. The specifications which are of interest for this research are shown in Table I. Even though the maximum amount of RAM is 520 KB and the maximum amount of Flash storage is 4 MB, the operable range for the ESP32 is 320 KB RAM and 1 MB Flash storage.

For the programming environment, Visual Studio Code in collaboration with PlatformIO [8] has been used.

VI. EVALUATION METHOD

This section explains how both network topologies will be evaluated to discover the best performing network. In total the topologies will be benchmarked regarding three separate topics.

The first benchmarking method is hardware performance. This benchmark assesses the resources used of the microcontrollers and compares them to the maximum resources which can be allocated to the algorithm. The focus of hardware usage monitoring is the RAM usage and the Flash storage being used by the algorithms. Both of these can be evaluated by the Project Inspection tool from PlatformIO.

The second benchmarking method is algorithm performance. With algorithm performance, the timings of the algorithm will be evaluated. These timings consist of the time it takes for a single fragment of data to be created and sent to the node which requested the data as well as the time taken for all the fragments to be created and sent.

Table II
HARDWARE BENCHMARKS

	RAM Usage [KB]	**Flash storage [KB]**
Hierarchical Network Coordinator	225.3 (70%)	854.6 (67%)
Hierarchical Network Subnode	233.2 (73%)	857.7 (67%)
Mesh Network	253.8 (79%)	816.3 (64%)

The third and final benchmarking method is network size. This benchmarking method is utilized to evaluate the increment of extra time taken to get a message from every single network as the size of the network increases. There are a total of four nodes in the test setup, so the benchmarks were evaluated for one, two and three nodes to send data to the requesting node.

VII. RESULTS

This section will contain all results achieved by performing the benchmark methods, which are described in section VI.

A. Hardware performance benchmarks

The broadcasted arrays used to evaluate the hardware performance of the algorithm are the same arrays described in section IV. In other words, the arrays consist of 480 float values, which corresponds to 1920 bytes. During these benchmarks, the programs are being evaluated on both the RAM usage and the Flash storage usage. The maximum amount of RAM available is 320 KB and the maximum amount of Flash storage available is 1 MB. The results can be seen in Table II. The percentages indicate how much of the maximum available RAM/Flash storage is being utilized.

B. Algorithm performance benchmarks

The second benchmarking approach investigates the time needed for a node to start creating a fragment of the original message and broadcast that fragment to the other nodes (a planning iteration).

1) Hierarchical network: While testing, it came to light that the microcontrollers could not cope with using the maximum size of a fragment (250 bytes). Trying to utilize the full size resulted in a lack of memory. Because of this, the amount of float values sent in one fragment was adjusted to 40 values (160 bytes, 176 bytes in total for one fragment). This resulted in twelve fragments being created for every message. This benchmark was performed ten times. The timing was the same for every benchmark and concluded to an average of 100 ms per fragment including sending and the total time for the message came down to 1162 ms.

2) Mesh network: While testing, the same problem as with the hierarchical network came to light, which was the limited RAM. It was necessary to create multiple fragments of the original message. However, since more data can be sent with the mesh network in contrast to the hierarchical network (as is shown in Figures 3 and 4), only six fragments need to be created for every message. This test was performed ten times and the timing for these planning iterations are shown in Table III.

C. Network size benchmarks

1) Hierarchical network: The third and final approach was to test the time it takes from sending a request to all nodes in the network one at a time, to receiving all the fragments from the nodes. The message that needed to be sent to the coordinator is the same message as the ones used for the other benchmarks. This test was executed with a network size of two, three and four nodes (of which one node

Table III
TIMING OF FRAGMENT SENDING

Test number	Average per fragment [ms]	Total time [ms]
1	101.6	728
2	101.0	722
3	101.1	724
4	101.0	722
5	101.0	725
6	101.0	722
7	101.8	728
8	101.6	727
9	101.8	727
10	101.1	724
Average	**101.4**	**725**

Table IV
TIMING OF THE MESSAGES FOR A HIERARCHICAL NETWORK

Test Number	Time for one node [ms]	Time for two nodes [ms]	Time for three nodes [ms]	Total time [ms]
1	1171	2351	3534	3541
2	1171	2341	3509	3511
3	1181	2351	3523	3531
4	1171	2341	3509	3511
5	1171	2341	3511	3521
6	1171	2341	3516	3521
7	1171	2341	3511	3511
8	1171	2341	3511	3511
9	1171	2351	3519	3521
10	1171	2341	3509	3511
Average	**1172**	**2344**	**3515.2**	**3519**

is the coordinator, so one, two and three nodes sending data). This test was performed ten times and the results are in table IV.

Table IV shows that the addition of multiple nodes cause a linear increase in time. Every extra node results approximately in an extra 1.17 seconds.

2) Mesh network: As for the effects the size of the network has on the timings of the algorithm, it behaves differently from a hierarchical network. Since adding more nodes creates a higher probability of at least two nodes trying to send data at the exact same time, thus causing congestion. Congestion can lead to collision and a loss of data. To avoid congestion, the range of the timeout needs to be increased along the network size. These benchmarks start without any timeout (the nodes start sending when a request has been received). This resulted in constant congestion and loss of data. After this initial test, multiple intervals were being benchmarked. When a node received a request, it will set a random interval in between 0 seconds and a variable maximum timeout. The maximum timeout was being evaluated at 1 second, 1.5 seconds and 2 seconds extra for each node in the network. For each of these different timeouts, 20 benchmarks were performed. During these benchmarks, both the amount of congestion cases and the time taken were analysed. Results for timeouts for these three maximum timeouts are shown in Figures 5, 6 and 7.

For a timeout in range of 0 and 1 second, a total of 6 congestion cases occurred during 20 tests, which results in a probability of 0.3 for congestion to occur during transmission.
For a timeout in range of 0 and 1.5 seconds, a total of 4 congestion cases occurred during 20 tests, which results in a probability of 0.2 for congestion to occur during transmission.
As for a timeout in range of 0 and 2 seconds, a total of 1 congestion case occurred during 20 tests, which resulted in a probability of 0.05 for congestion to occur.
Based on these results, the best timeout to avoid congestion as much as possible and still keep the time it takes for the nodes to send their messages at a minimum, comes down to a timeout in range of 0 and 2 seconds added for every node in the network.

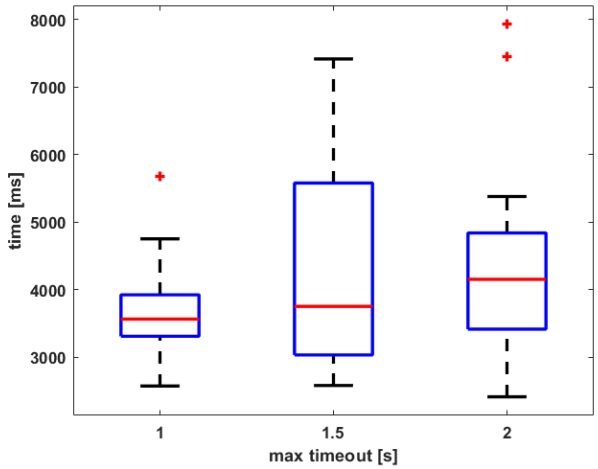

Figure 5. Timing of the messages for the mesh network for one node.

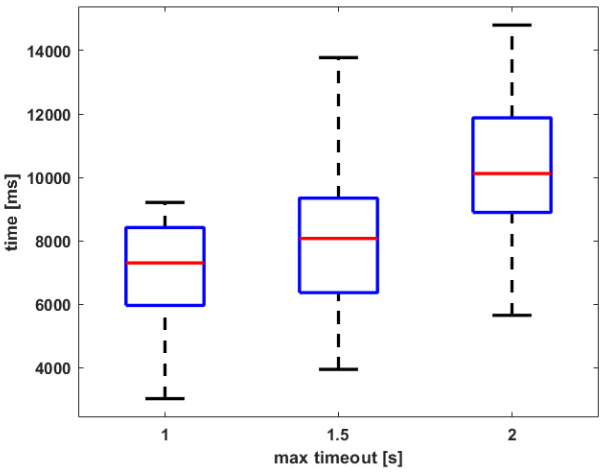

Figure 6. Timing of the messages for the mesh network for two nodes.

During the benchmarking of the network size, benchmarks were also performed regarding a node malfunction. This resulted in the rest of the network to still be fully operational. The node which failed did not have any effect on this result.

With these benchmarks, the minimum, maximum and average values have been calculated. They are shown in Table V.

Table V shows that on average, the time added for each new node in the network consists of the amount of nodes in a network, multiplied with the maximum timeout value.

VIII. CONCLUSION

Both the hierarchical network and the mesh network can be utilized to create a stable communication network for smart EV charging. However, the communication network topology with the highest stability, is the mesh network. Since there is no coordinator node, any node in the network can fail, but it will still be able to successfully broadcast messages in between other nodes. This adds an extra layer of robustness to the network in contrary to a hierarchical network. If the coordinator node fails in the hierarchical network, there is no method to get the other nodes to communicate with each other, thus

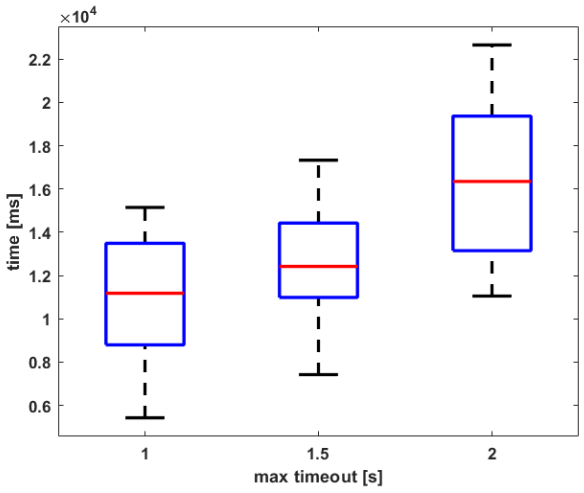

Figure 7. Timing of the messages for the mesh network for three nodes.

Table V
MINIMUM, MAXIMUM AND AVERAGE TIMINGS FOR THE MESH NETWORK

	MIN [ms]	MAX [ms]	AVG [ms]
1 node			
1 sec	2572	5672	3698
1.5 sec	2580	7413	4285
2 sec	2413	7930	4344
2 nodes			
1 sec	3016	9208	6980
1.5 sec	3940	13778	8207
2 sec	5646	14804	10335
3 nodes			
1 sec	5428	15151	10919
1.5 sec	7422	17335	12615
2 sec	11054	22659	16783

rendering the network unusable. There are two drawbacks of the mesh network. First of all, the time it takes for all the nodes to send data is larger than the time it takes for all the nodes to send its data in a hierarchical network. Furthermore, it also uses more RAM. Despite these shortcomings, the mesh network is more suited for a stable communication network between multiple EV chargers, because of the extra robustness.

IX. Future Work

Even though the mesh network topology outperforms the hierarchical network topology in robustness, every node added to the network adds more time to a communication cycle. Some further research can be done to evaluate a mesh network with a more nodes and observe the amount of congestion in the network.
Furthermore, both algorithms can still be improved. For example with loss of data. If a packet gets delivered to the node which sent the request, it can send an acknowledgment back to show that it has received the data.
Lastly, the network topologies have only been implemented and evaluated in a theoretical setup. Therefore, the algorithm still needs to be evaluated at actual EV chargers.

References

[1] Kevin Böhmer. "Model predictive control for electric vehicle charging using low power microcontrollers". University of Twente, 2021.

[2] *ESP-NOW User Guide*. Available at https://www.espressif.com/en/products/software/esp-now/resources. Espressif Systems. 2016.

[3] *ESP32 Series Datasheet*. Available at https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. Espressif Systems. 2021.

[4] G. Hoogsteen, A. Molderink, J.L. Hurink, G.J.M. Smit, B. Kootstra and F. Schuring. "Charging electric vehicles, baking pizzas, and melting a fuse in Lochem". In: *CIRED: Open Access Proceedings Journal*. 2017, pp. 1629–1633. DOI: 10.1049/oap-cired.2017.0340.

[5] Abhishek Jain. *Network Topologies (Its types, Advantages and Disadvantages)*. 2017. URL: https://www.includehelp.com/computer-networks/network-topologies-its-types-advantages-and-disadvantages.aspx (visited on 28/01/2022).

[6] M.E.T. Gerards, H.A. Toersche, G. Hoogsteen, T. van der Klauw, J.L. Hurink and G.J.M. Smit. "Demand side management using profile steering". In: *2015 IEEE Eindhoven PowerTech*. 2015, pp. 1–6. DOI: 10.1109/PTC.2015.7232328.

[7] *PainlessMesh library*. https://gitlab.com/painlessMesh/painlessMesh.

[8] PlatformIO. *Professional collaborative platform for embedded development*. 2014. URL: https://docs.platformio.org/en/latest/ (visited on 26/01/2022).