Hai Ngoc Nguyen

Hainguyen200321@gmail.com

ELECTRIC VEHICLE CHARGING GRID: OPERATED BY A DISTRIBUTED WIRELESS SENSOR NETWORK (WSN)

Report

Table of Contents

[Introduction 3](#_Toc148483622)

[Methodology 3](#_Toc148483623)

[Architecture 3](#_Toc148483624)

[Charging Port 3](#_Toc148483625)

[EV Charging Station 3](#_Toc148483626)

[Base station 3](#_Toc148483627)

[Technical/Code section 4](#_Toc148483628)

[Setup MPI thread for base station and EV charging stations 4](#_Toc148483629)

[Setting Up EV Charging Station and Charging Port 6](#_Toc148483630)

[Results Tabulation 8](#_Toc148483631)

[Analysis & Discussion 9](#_Toc148483632)

[Message passing 9](#_Toc148483633)

[Single Computer vs cluster computing setup 9](#_Toc148483634)

[References 9](#_Toc148483635)

# Introduction

This report discusses the design and development of an ELECTRIC VEHICLE CHARGING GRID, operated by a DISTRIBUTED WIRELESS SENSOR NETWORK (WSN) [1]. WSNs can be effectively utilized in electric vehicle charging stations.

A WSN is a network consisting of EV charging stations. Its base stations serve as the main server, performing critical tasks which EV charging stations aren’t capable of doing, like processing data and communication between all the EV charging stations [1]. Each EV charging station has multiple ports, which serve as chargers. We will design a system for them to effectively communicate with each other.

# Methodology

## Architecture

In this section, the rationale behind the chosen architecture is discussed. For this WSN, the network is divided into three primary sections: EV charging stations, charging ports belonging to that EV charging station, and the Base station (centre) [2].

### Charging Port

Each EV charging station has its own set of charging ports. These charging ports are localized, so I will create a shared memory system, where ports will communicate with their EV charging station to update its availability, whether they are in use or not [2]. This way, each station can report if they are full or not to the customer, and this will be done almost instantly.

### EV Charging Station

There are lots of charging stations located all over the map. For effective communication, each station can act as its own “node” in a grid system (distributed memory) [3]. If the current EV charging station is full, it can reach out to its neighbours seeking space. But what if all of its neighbours are also full? This is where effective communication comes in. We allow the EV to communicate within their own system (Grid), but we don’t want it to be overwhelmed with existing port and neighbouring charging stations' suggestions. We can introduce a centre node to handle this critical case [3].

### Base station

The Base station will be in charge of more complex decision-making [2]. For instance, if a node sends a report to it stating that itself and all the surrounding neighbours are full, the Base station will need to find out the nearest available station and send that information back to the node. We only want the Base station to handle this, as it is located far away from each of the charging stations. Sending and receiving data can be time and cost-intensive, and effective communication means only sending data to the Base station when necessary [2].

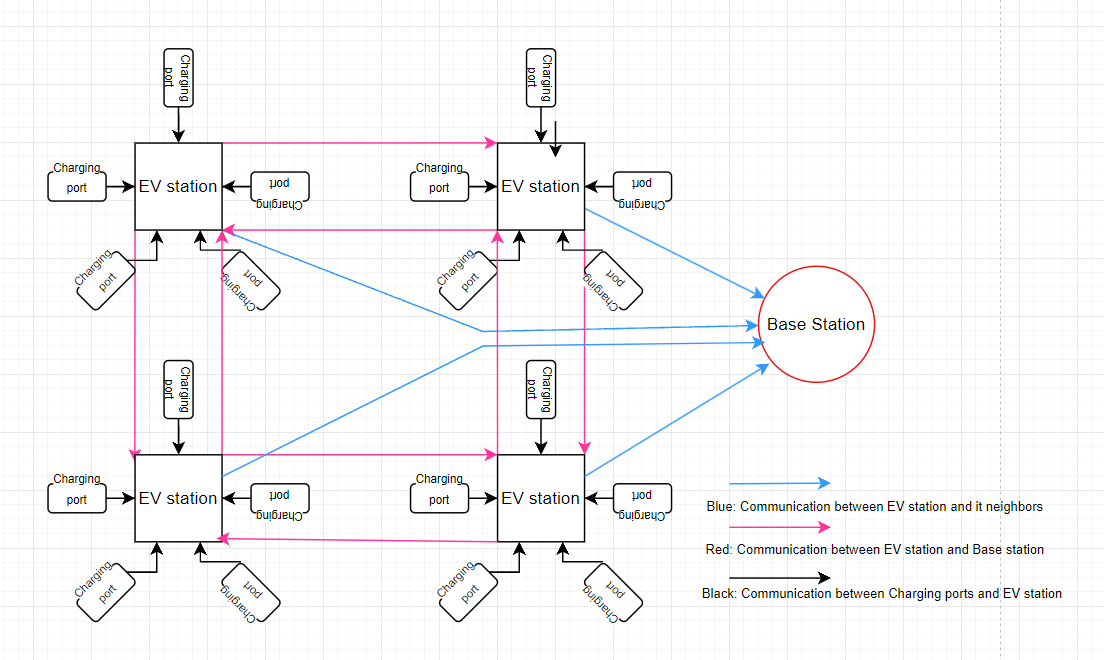


Figure 1 Overview of the wireless sensor network as the electric charging grid

## Technical/Code section

### Setup MPI thread for base station and EV charging stations

First, we aim to separate the processes of the base station node and the EV charging station node into two communicators. For convenience, I have designated the process with rank = 0 as our base station and categorized it with its own colour. Any other process with a rank not equal to 0 is given a different colour. Processes with the same colour (i.e., EV charging stations) execute their logic by calling the simulate\_ev\_node function and create an MPI Cartesian topology for themselves (as shown in figure 2.2). On the other hand, the node with rank = 0 follows the logic of the base station. It periodically listens for incoming messages from the EV charging stations, checks for the nearest available node, logs a report on this, and sends the available node's information back to the reporting node(figure 2.1).



Figure 2.1 pseudocode initialising MPI process for base station and EV charge Stations

### Setting Up EV Charging Station and Charging Port

In each EV charging station, there are 5 charging ports. We simulate the communication process between the charging port and the charging station using parallelism on shared memory. A total of 5 OpenMP threads are spawned, with each being responsible for randomly writing down either 0 or 1, representing whether the port is in use. After each cycle, the system calculates the availability based on reports from each port. Once this calculation is complete, the EV charging station determines whether to seek assistance from its neighbours. If the neighbours are also fully occupied, the EV charging station sends an alert with relevant details to the base station, asking for help. By communicating in this manner, the EV station only reaches out to the base station when genuinely in need. This layered communication ensures that if a node wants to communicate with another distant node, it must do so via the base station. Such an approach makes the system easier to maintain and expand.

# Results Tabulation

Information about the simulation:   
-These simulations define that a node needs to reach out to its neighbours only when and where its own availability is lower than 2. And it will only send a report when their neighbours has availability as 0. This means that a node will only send an alert to the base station when it has low capabilities, and its neighbours has no capabilities.  
-A node will reach out to its neighbours (generating an event) if and only if it has its capabilities lower than 2. A node will send and receive a reply with the information from its neighbours. This action will be counted as 2 events.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | SYSTEM | NUMBER OF LOG | RUNNING TIME (second) | EV CHARGING STATION CYCLE (second) | GRID SIZE | EVENT GENERATED |
| 1 | CASS | 9 | 20 | 1 | 3\*3 | 480 |
| 2 | SINGLE COMPUTER | 9 | 20 | 1 | 3\*3 | 480 |
| 3 | CASS | 9 | 10 | 1 | 3\*3 | 130 |
| 4 | CASS | 9 | 5 | 1 | 3\*3 | 31 |
| 5 | CASS | 9 | 3 | 1 | 3\*3 | 31 |
| 6 | CASS | 9 | 2 | 1 | 3\*3 | 31 |

A white background with colorful text

Description automatically generated

Figure 3.1 Log screen shoot from first report

A white background with colorful text

Description automatically generated

Figure 4 Log screen shot from second report

Figure 5 Relation between time and event

# Analysis & Discussion

## Message passing

Based on the results covered above, we can observe that over time, the number of events increases rapidly as more time elapses. This could be due to the fact that as more time passes, unexpected situations arise, closely mirroring real-life scenarios (fig5). This implies that as more events occur, there's a higher likelihood of more reports being sent to the base station. In my system's implementation, the base station doesn’t have a buffer responsible for receiving messages. As illustrated in fig 3 and fig 4, the time taken for the alerts to be processed increases as more alerts enter the system from 18380 microseconds -> 29235 microseconds. This happens because I utilized blocking message-passing functions: MPI SEND and RECV. This means all the EV charging stations have to wait for their turn to pass on the alert to the system.

This creates two problematic scenarios:

1. When all the stations have to wait to ascertain the nearest availability, the delay could be so extensive that real-time updates become unfeasible.
2. Given that it employs blocking message passing, if there's a long queue of messages waiting to be executed, the stations can't continue with their ongoing processes. This could potentially crash the entire system if too many alerts are triggered simultaneously.

## Single Computer vs cluster computing setup

In a single computer setup, there are limitations on the number of tasks/processes that can be spawned for each station, as well as on the number of threads that can be spawned for the ports. This impacts performance. As the implementation demonstrates, most of the states are parallelizable. This implies that on a single computer with fewer CPUs, fewer parallel tasks can operate simultaneously. Take, for instance, the ports for each station: on a cluster computer, you can spawn as many threads as you want for each port. Information is received instantly and written onto the shared memory. Conversely, on a single computer, fewer tasks can be parallelized. In the worst-case scenario, only a single CPU handles all the ports, which could potentially result in a performance decline of roughly 80%.

# References

[1] W. Wang, X. Liu, and H. Zhang, "A survey on wireless sensor network infrastructures for emerging smart grid systems," IEEE Transactions on Industrial Informatics, vol. 14, no. 1, pp. 422-431, 2018.

[2] Y. Zhang, S. He, and J. Chen, "Data-driven adaptive optimal control for plug-in hybrid electric buses," Applied Energy, vol. 154, pp. 686-696, 2015.

[3] Z. Liu, F. Wu, and X. Zhou, "Powering transportation with wireless electric vehicle charging technology: State of the art and the path forward," IEEE Vehicular Technology Magazine, vol. 8, no. 3, pp. 28-38, 2013.