Validation of Wireless Battery Management System (wBMS) - Gen2

Mr. Chetan[1], Mrs. Sujatha Hiremath[2]

[1]*Student, RV College of Engineering,Bangalore, India*

[2]*Assistant Professor, RV College of Engineering,Bangalore, India*

[1] chetan.ec17@rvce.edu.in , [2] sujathah@rvce.edu.in ,

Abstract:*Wireless Battery Management System (wBMS) is a primary enabler for the widespread adoption of electric cars, allowing auto Original Equipment Manufacturer (OEMs) to avoid having to rework complicated wiring diagrams for each new car and ensuring battery scalability. This article mainly focuses on validation of an end-to-end wBMS system by performing several tests like Packet Transfer Ratio (PTR) for different configuration files, developing and implementing a health report application which generates health report in real-time, automating the process of OTA (Over the Air) upgrade which also includes automation of configuring the front-end application using python programming language. The main intention behind developing the script to automate OTA upgrade and health report application is to reduce time consumed to test the system, reduce human errors, and perform the tests for any number of iterations.*

Keywords:wBMS, OEM, OTA, J-Link Lite, UART, PTR

1. Introduction

In order to deliver high voltage power to its electric motors, it is necessary to regularly monitor the safety and reliability of hybrid vehicles (HEVs) and electric vehicles (EVs) in order to offer huge numbers of cells. The rationale for the use of Hybrid Electric Vehicle (HEV) and Electric Vehicle (EV) gets more stronger, since vehicle pollution control regulation is becoming tougher globally. However, the EV industry remains relatively new and faces certain difficulties before it takes over. Developments continue to increase the range, endurance, safety and trustworthiness of the vehicle while reducing size, cost and weight.

Several wireless technologies that might be utilized in an electric car in the future and provides a thorough evaluation of all of them, such four different types of wireless communication modes which can be used in BMS are namely - Cloud-Based BMS, Internet of Things (IoT) based wireless BMS, Bluetooth Based Wireless BMS, Zigbee Based Wireless BMS [1]. Wireless Smart Battery Management System (WSBMS) is a wirelessly communicated cell level BMS. When compared to normal modularized BMS, this system provided high tolerance to faults and adequate scalability. The balancing algorithm which is based on the State-of-health (SOH) and State-of-charge (SOC) is capable of balancing any number battery cells, ageing condition, and capacity deviations [2]. There also exists various wireless protocols, including ZigBee, Bluetooth Low

Energy, Near Field Communication, Wi-Fi, and Wi-Fi HaLow. It is observed that Wi-Fi and Wi-Fi HaLowappears to be the best solution in certain ways, however predicts that it may be over-specified for WBMS applications, resulting in an increased cost overhead [3]. The battery devaluation measure should be diminished by molding the battery in an appropriate way, for example, controlling incessant charge and profound release cycles. To discover the connection between voyaging distance and release cycle an examination dependent on IoT with ongoing remote wBMS is need of the hour [4].

Present cyber-physical systems rely on the continual service battery backup systems. The conventional methods employed for battery authentication, particularly when they are composed of large numbers of cells, cannot be introduced to new smart cells as they lead to increased cable requirements [5]. There have also been the case where use of IBM's Hyperledger-Fabric is applied for IoT-applications as Hyperledger-Fabric is private and authorized blockchain for access control, which consumes less energy and less computational resources for consensus creating a blockchain ledger than other platforms, which leads to a significantly less latency [6]. Wireless connection and cloud support of the IoT module may eliminate wireless problems and make the use of simple on-board controllers, which increases the scalability and productivity of battery module. In addition, it is tolerant to either the wireless slave's failure or control units that the proposed IoT network using the suggested leadership election methodology fails [7]. The wireless control of the battery reduces battery failure sites to a minimum. Moreover, it enables the replacement of individual components without the whole reconstruction while reducing the influence of system changes on cumbersome cords. Possessing memory also allows the system to recover if data is lost by pushing data for processing and storage once data transmission has been restored, leading to enhanced data integrity [8].

In a vehicle, based on overhearing, a dependable, multi-hop communications is designed. Both the BMS and the sensor networks is used for the technique. The ratio of packet arrival for BMS and the nearby sensor network within 20ms is around 99.48%, the packet arrival rate gets reduced if the there is an increase in the number of nearby networks [9]. The developed system comprises of hardware namely sensors, the microcontroller and the Bluetooth module and software. This has been built using a cheap Arduino UNO micro-controller. The microprocessor transfers temperature, current and voltage data, subsequently battery data is transmitted through Bluetooth connection to display. In real-time, the monitoring system has been able to present temperature, current and voltage data, and simultaneously display data on smartphones and Personal computers [10].

2.End-to-End wBMS:Overview

An end-to-end wBMS consists of a master board, slave board, cell interface board, PC with front-end application configured in it. So basically, hardware part of the wBMS goes into the electric vehicle with slaves being attached or placed along with the battery packs and these slaves are in turn connected to master board wirelessly. The figure 1. shows the brief block diagram of how the components are interconnected to each other.

The main hardware components are the master board, slave board, cell interface, a personal computer with front-end application installed in it and a USB hub to connect all the UARTs and J-Link lite debuggers an ethernet cable to connect personal computer to master board. An end-to-end wBMS can contain any number of slave cell boards based upon requirement. In this project, initially a system with one slave board and one master board was setup and the same system was later extended to a system with many slave boards and one or more master board.

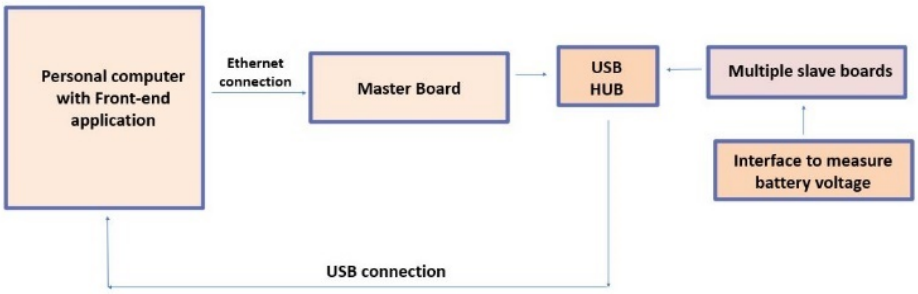


Figure 1: An end-to-end wBMS

Within the car, a wireless network is formed by a set of slaves and masters. Only slaves and network masters inside that particular vehicle, or more particularly a specific battery pack within that vehicle, are permitted to join the network. Other car's wBMS networks are entirely distinct systems. Larger vehicles, like trucks, may have numerous battery backs, however each battery back runs as a distinct wBMS and is therefore considered a distinct system.

A slave in this system has a chip, a 2.4 GHz RF transceiver to interact with the wireless network, and a microcontroller(s) to interface with the battery and the transceiver. Because the network master does not communicate with a battery module, it just has the 2.4 GHz RF transceiver and a microprocessor to communicate with the transceiver and the BMS Controller. However, one of the network masters is usually linked to a sensor, which checks the battery's total charge level.

The command/response architecture of communication from the BMS Controller to the BMS Monitors in the wireless system is limited by what every radio can accomplish - basically either send or receive in a time slot. If the wireless system attempts to implement the same command/response sequence as a cable system, the communication might be exceedingly inefficient. To mitigate this, the slaves in a wired system require greater autonomy to perform the loop operated in the BMS Controller. The BMS Controller takes on the role of a traffic controller, instructing the slaves to start and stop the loop, requesting more information, and modifying configuration parameters that allow the slave loops to run in different ways.

3.Test Methodology

The basic procedure followed to fulfil any of the framed objective includes

1. Connecting master, slave, Universal Serial Bus (USB) to Universal AsynchronousReceiver Transmitter (UART) converters, J-Link lite to form a network.
2. Configuring the Front-end application, downloading configuration and binary files to master and slaves.
3. Writing and using the corresponding python scripts to perform framed objectives.

As referenced in Figure 2, initially hardware setup is done, which includes connecting all the components. Further these are connected to user's computer using USB hubs. Since there will be new versions coming now and then with fault correction, master and slave

are flashed using J-Link lite debuggers. Next step involves downloading files to master and slaves and configure them so as to make them capable of sending data to each other. Once the Front-end application is configured, the network information like PTR, number of slaves connected, type of operation can be seen in Front-end application. Front-end application is basically is Graphical User Interface (GUI) provided for user to monitor and control the network remotely.

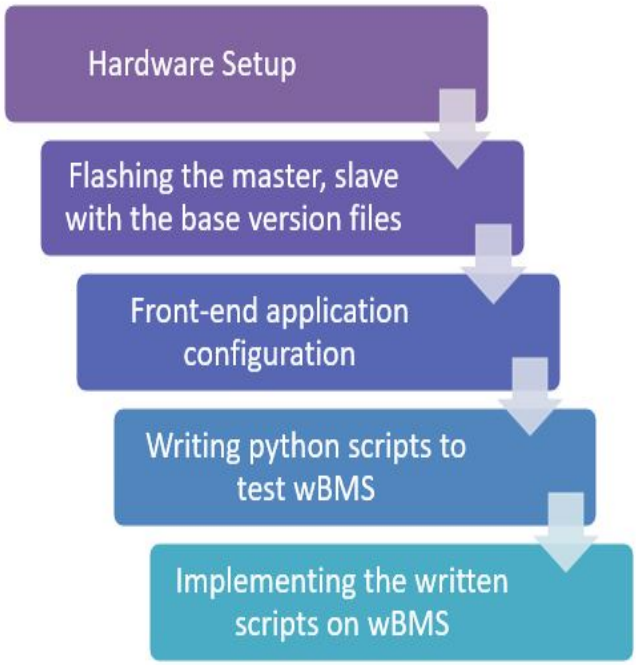


Figure 2: Flowchart representing test methodology

3.1. Health Report Application

The health report application parses the real time packets sent by master and slaves by subscribing to a back-end interface, generates few reports, displays the few fields of formatted data in several different consoles in tabular column, and simultaneously logs the data in excel sheet. This network health information in these packets can be used to calculate system-level performance metrics and to diagnose network performance issues.

This application parses the real-time packets obtained from masters and slaves, which will be in raw byte form, modifies the bytes as per the fields and display few fields of all the four below mentioned reports in table format on different consoles and simultaneously logs data into excel sheet which can be found in ”Health reports” folder. The excel sheets will start logging data when the .bat file is run and stops logging when scripts are killed. Many packets will be received every minute from masters and slaves.

The packets obtained from slaves and masters are basically of two kind one is packet-1 and the other is packet-2. So, based upon these packets it generates four health reports namely,

1. Slave report-1
2. Slave report-2
3. Master report-1

4. Master report-2

3.2. Packet Transfer Ratio (PTR)

Measuring PTR of a system is one of the very crucial process of validating an wBMS since it gives an overview of how well the components of the system are communicating with each other and also the rate at which the packets are being sent, and also specifies the number of packets lost due to interference. By downloading front-end application, by default we get few configuration files with ".cfg" extension which can be further modified as per requirement. The default configuration file is modified and then 10 more configuration files are generated from it. Each of the configuration file generated differs from each other by only one parameter i.e., BMS sampling rate. So, the requirement here is to check for 10 different configuration files namely config-1, config-2 and so on till config-10. BMS sampling rate is nothing but the rate at which the slaves samples the packet.

The expected PTR should be always be high since the setup and tests are performed in a clean lab environment. Even though if the setup or the system is kept in vicinity of another system, it shouldn't decrease the PTR of both the systems. There will be a possibility of one system getting the packets from another system causing interference, but it should be taken care by making sure that both the systems are isolated and packet transfer between them is negligible. PTR can be tested for any specified time, but it is always preferred to measure PTR by keeping the system for as long as possible to cover the worst-case scenario.

3.3.Automation of Front-End Application Configuration and OTA Upgrade Process

This test automation covers complete automation of front-end application, from flashing slaves and masters, to setting up the front-end application and OTA upgrade both masters and slaves. The sequence of events occurring can be seen in console and when the script is completely run, an excel file will be generated which contains the structured report with different columns indicating the version and also failures if any during the OTA upgrade process. The entire script is written in python and is integrated with front-end application using back-end APIs.

The whole script is written in python using pytest framework which is specifically developed to perform efficient testing and make the script bug free. There are two scripts, the main script which contains implementation of the sequence of events mentioned above and the other script, conftest.py where the paths to configuration, container files are specified. The main script imports the paths from the other script as and when required during the run. The second script conftest.py is maintained to easily modify the script and run quickly when there is an upgrade. So, the OTA upgrade script should be rigorous and robust and should be capable of identifying the problems in the system which cannot be found out manually. Again, it can be run for any number of iterations continuously without stopping which increases the efficiency of finding defects in the system as well as saves time and reduces manual intervention.

4. Results and Discussions

Setting up an end-to-end wBMS is common step for all proposed tests to be performed so initially a slave, a master setup, USB to UART converter, personal computer with front-

end application, USB hub and J-Link Lite was performed. Since one slave only gives an insight of how the system behaves, increasing the number of slaves gives the actual behavior of the wBMS in real time scenario. So, the number of slaves were extended from one to more than one slave in an end-to-end wBMS.

4.1. Health Report Application

As discussed earlier, type-1 packet is sent from each master per minute. It can be easily observed from the figure 3 that two packets are received per minute, which are highlighted in the output. 240 is device id of the master-1 and 241 is the device id of master-2.

DID	Time	PS	PN	NMF	MMF_0	MMF_1	MMF_2	MC
241	11:01:03.773	0	0	0	98	0	0	3
240	11:01:10.492	0	0	0	5	0	0	1
241	11:02:03.777	0	0	0	287	0	0	3
240	11:02:10.492	0	0	0	10	0	0	1
241	11:03:03.776	0	0	0	288	0	0	3
240	11:03:10.493	0	0	0	14	0	0	1
241	11:04:03.779	0	0	0	288	0	0	3
240	11:04:10.495	0	0	0	15	0	0	1
241	11:05:03.779	0	0	0	289	0	0	3
240	11:05:10.498	0	0	0	20	0	0	1
241	11:06:03.781	0	0	0	289	0	0	3
240	11:06:10.500	0	0	0	22	0	0	1
241	11:07:03.781	0	0	0	310	0	0	3
240	11:07:10.499	0	0	0	77	0	0	1
241	11:08:03.781	0	0	0	324	0	0	3
240	11:08:10.501	0	0	0	133	63	0	1

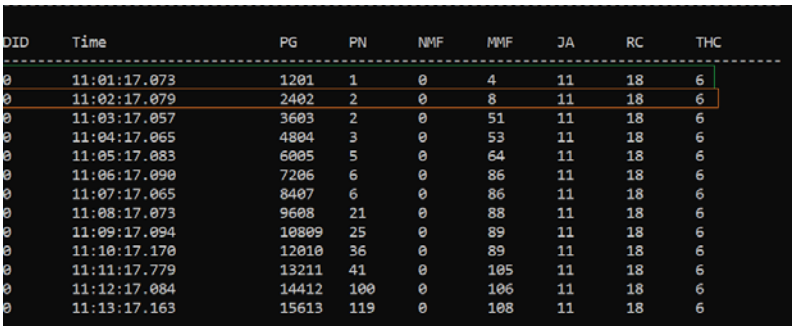
Figure 3: Real-time console output of master report-1

Figure 4 shows the console output of Master report-2. As mentioned earlier, 2 packets indicating important information of the master are sent per minute from the master board. There exists a lot of fields in this report which can be seen in the excel.

DID	Time	RS_C1_B1	RS_C1_B2	RS_C1_B3	RS_C1_B4	RS_C1_B5	RS_C2_B1	MAFC
241	11:01:03.774	14	0	0	0	108	14	0
240	11:01:10.493	14	0	0	0	109	14	0
241	11:02:03.779	14	0	0	0	108	14	0
240	11:02:10.494	14	0	0	0	109	14	0
241	11:03:03.778	14	0	0	0	109	14	0
240	11:03:10.496	14	0	0	0	109	14	0
241	11:04:03.789	14	0	0	0	109	14	0
240	11:04:10.503	14	1	0	0	109	14	0
241	11:05:03.781	14	0	0	0	108	14	0
240	11:05:10.504	14	0	0	0	108	14	0
241	11:06:03.783	14	0	0	0	108	14	0
240	11:06:10.502	14	0	0	0	108	14	0
241	11:07:03.785	14	0	0	0	109	13	0
240	11:07:10.506	14	0	0	0	109	14	0

Figure 4: Real-time console output of master report-2

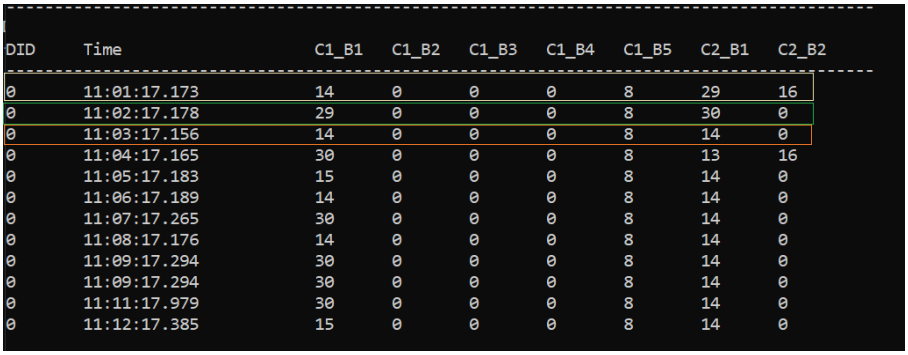
Figure 5 shows the console output of the slave report-1 which is formatted from the raw packet data obtained from slaves present in the network. Since only packet is sent from each device present in the system, so only one packet from each slave is captured per minute and displayed in the table format.



DID	Time	PG	PN	NMF	MMF	JA	RC	THC
0	11:01:17.073	1201	1	0	4	11	18	6
0	11:02:17.079	2402	2	0	8	11	18	6
0	11:03:17.057	3603	2	0	51	11	18	6
0	11:04:17.065	4804	3	0	53	11	18	6
0	11:05:17.083	6005	5	0	64	11	18	6
0	11:06:17.090	7206	6	0	86	11	18	6
0	11:07:17.065	8407	6	0	86	11	18	6
0	11:08:17.073	9608	21	0	88	11	18	6
0	11:09:17.094	10809	25	0	89	11	18	6
0	11:10:17.170	12010	36	0	89	11	18	6
0	11:11:17.779	13211	41	0	105	11	18	6
0	11:12:17.084	14412	100	0	106	11	18	6
0	11:13:17.163	15613	119	0	108	11	18	6

Figure 5: Real-time console output of slave report-1

Figure 6 shows the console output of the slave report-2 of each slave which is formulated by capturing the raw data packet obtained from each slave every minute. Since we are measuring interference obtained from other slaves in the system on specific channel, so again there exists many fields in this report each representing the amount of interference at a particular channel.



DID	Time	C1_B1	C1_B2	C1_B3	C1_B4	C1_B5	C2_B1	C2_B2
0	11:01:17.173	14	0	0	0	8	29	16
0	11:02:17.178	29	0	0	0	8	30	0
0	11:03:17.156	14	0	0	0	8	14	0
0	11:04:17.165	30	0	0	0	8	13	16
0	11:05:17.183	15	0	0	0	8	14	0
0	11:06:17.189	14	0	0	0	8	14	0
0	11:07:17.265	30	0	0	0	8	14	0
0	11:08:17.176	14	0	0	0	8	14	0
0	11:09:17.294	30	0	0	0	8	14	0
0	11:09:17.294	30	0	0	0	8	14	0
0	11:11:17.979	30	0	0	0	8	14	0
0	11:12:17.385	15	0	0	0	8	14	0

Figure 5: Real-time console output of slave report-2

4.2. Packet Transfer Ratio (PTR)

The table 1 shows the PTR of each slave and also the number of packets generated and received. It can be observed that PTR of each slave is greater than 99.7% for config-1. Similarly, the PTR is calculated for other config files which were generated and the average PTR of all slaves per configuration file is as shown in the table 1.

Table 1. PTR for 10 configuration files

Configuration file	PTR in percentage (%)
Config-1	99.64989
Config-2	99.58932
Config-3	99.45097
Config-4	99.22723
Config-5	98.93412
Config-6	98.94131
Config-7	95.12438

Config-8	96.39085
Config-9	98.37578
Config-10	98.36865

4.3. Automation of Automation of OTA upgrade process

The table 2 shows the output observed after running the script. It can be observed that all the slaves and managers were upgrade to version 2.0 from version 1.0.It can be observed that time taken to OTA upgrade all slaves parallelly is around 2 min. The time taken for every iteration varies since the OTA upgrade is wireless and may be delayed due to interference in some cases. All of the iterations were successful with all the files being retained after OTA upgrade and even the slaves getting upgraded to the OTA upgraded version.

Table 2. Observations from 10 iterations of OTA upgrade process

Iteration count	Previous version	Upgraded version	Success/Fail
1	1.0	2.0	Success
2	1.0	2.0	Success
3	1.0	2.0	Success
4	1.0	2.0	Success
5	1.0	2.0	Success
6	1.0	2.0	Success
7	1.0	2.0	Success
8	1.0	2.0	Success
9	1.0	2.0	Success
10	1.0	2.0	Success

Table 3 shows the time taken per iteration. It is around 9 minutes approximately. So it proves that automation is very less time consuming and saves manual intervention too. Average time taken to perform OTA upgrade and Explorer configurations is around 13 minutes and it also tedious. Automating will not only reduce the time consumed per iteration but also will find the bugs which cannot be found manually. And it is also hard to perform 100 iterations of OTA upgrade manually, but it can be easily achieved using OTA upgrade automation and moreover, it can be run for any number of iterations, be it 100 or 1000 or more.

Table 3. Comparison of time taken for manual and automated OTA upgrade process

Iteration count	Time taken by automation script	Time taken manually
1	9 minutes	13 minutes
10	1.5 hours	2 hours
50	7.5 hours	10.8 hours
100	15 hours	21 hours
1000	150 hours	216 hours

It is to be noted that manual time for iteration count other than 1 is calculated using the time taken to OTA upgrade manually once. The time taken per iteration is multiplied by number of iterations to differentiate between two methods.

5. Conclusion

Setting up the more than one slave, more than one master network being common procedure for implementing the framed tests, so it is basically a prerequisite. Health report application was not existent previously and development of this application has allowed the user to examine the packets from the slaves and masters effectively from the excel report generated. PTR calculation for different configuration files gave a PTR of above 95% for all configurations, maximum being 99.9431%. Front-end application and OTA upgrade automation has made it a lot easier not only by reducing time consumption per iteration to less than 10 minutes from manual process which would take around 13 minutes per iteration but also giving the flexibility to perform OTA upgrade for any number of iteration be it 10, 100 or 1000 or anything more. Assuming 100 iterations are performed manually and by the proposed automation, the time saved due to automation would be 6-7 hours and also, it's very tedious to perform 100 iterations of OTA upgrade manually. So, OTA upgrade automation is a big boost for the implementation of wBMS.

7. Future Scope

One of the major constraints of this project is that it experiences interference from the other nearby networks if any, and it may lead to few slave disconnections or slaves not joining the network within specified time. Exploring the best ways to isolate a system from the other nearby system will be the challenging step and this should come at the least cost. Also enhancing the features so that remote access of the system becomes feasible is another key challenge to make the system robust and flexible.

REFERENCES

8.1. Journal Articles

[1]. V. K, R. K. Nema, and A. Ojha, "Various types of wireless battery management system in ev," in 2020 IEEE International Students' Conference on Electrical,Electronics and Computer Science (SCEECS), 2020, pp. 1–5. doi: 10.1109/ SCEECS48394.2020.115.

[2]. X. Huang, A. B. Acharya, J. Meng, X. Sui, D.-I. Stroe, and R. Teodorescu, “Wireless smart battery management system for electric vehicles,” in 2020 IEEE Energy Conversion Congress and Exposition (ECCE), 2020, pp. 5620–5625. doi: 10.1109/ ECCE44975.2020.9236279.

[3]. P. Bansal and P. Nagaraj, “Wireless battery management system for electric vehicles,” in 2019 IEEE Transportation Electrification Conference (ITEC-India), 2019, pp. 1–5. doi: 10.1109/ITEC-India48457.2019.ITECINDIA2019-83.

[4]. S. Haldar, S. Mondal, A. Mondal, and R. Banerjee, “Battery management system using state of charge estimation: An iot based approach,” in 2020 National Conference on Emerging Trends on Sustainable Technology and Engineering Applications (NCETSTEA), 2020, pp. 1–5. doi: 10.1109/NCETSTEA48365.2020.9119945.

[5]. A. Al Khas and I. Cicek, “Sha-512 based wireless authentication scheme for smart battery management systems,” in 2019 8th International Conference on Renewable Energy Research and Applications (ICRERA), 2019, pp. 968–972. doi: 10.1109/ ICRERA47325.2019.8996531.

[6]. T. Faika, T. Kim, J. Ochoa, M. Khan, S.-W. Park, and C. S. Leung, “A blockchainbased internet of things (iot) network for security-enhanced wireless battery management systems,” in 2019 IEEE Industry Applications Society Annual Meeting, 2019, pp. 1–6. doi: 10.1109/IAS.2019.8912024.

[7]. T. Faika, T. Kim, and M. Khan, “An internet of things (iot)-based network for dispersed and decentralized wireless battery management systems,” in 2018 IEEE Transportation Electrification Conference and Expo (ITEC), 2018, pp. 1060–1064. doi: 10.1109/ITEC.2018.8450161.

[8]. C. Shell, J. Henderson, H. Verra, and J. Dyer, “Implementation of a wireless battery management system (wbms),” in 2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings, 2015, pp. 1954–1959. doi: 10.1109/I2MTC.2015.7151581.

[9]. T. Kumtachi, K. Kinoshita, and T. Watanabe, “Reliable wireless communications in battery management system of electric vehicles,” in 2017 Tenth International Conference on Mobile Computing and Ubiquitous Network (ICMU), 2017, pp. 1–6. doi: 10.23919/ICMU.2017.8330099.

[10]. A. Jamaluddin, F. A. Perdana, A. Supriyanto, A. Purwanto, Inayati, and M. Nizam, “Development of wireless battery monitoring for electric vehicle,” in 2014 International Conference on Electrical Engineering and Computer Science (ICEECS), 2014, pp. 147–151. doi: 10.1109/ICEECS.2014.7045235.

[11]. D. Alonso, O. Opalko, M. Sigle, and K. Dostert, “Towards a wireless battery management system: Evaluation of antennas and radio channel measurements inside a battery emulator,” in 2014 IEEE 80th Vehicular Technology Conference (VTC2014- Fall), 2014, pp. 1–5. doi: 10.1109/VTCFall.2014.6966212.

[12]. M. Lee, J. Lee, I. Lee, J. Lee, and A. Chon, “Wireless battery management system,” in 2013 World Electric Vehicle Symposium and Exhibition (EVS27), 2013, pp. 1–5. doi: 10.1109/EVS.2013.6914889.

[13]. S. A. Mathew, R. Prakash, and P. C. John, “A smart wireless battery monitoring system for electric vehicles,” in 2012 12th International Conference on Intelligent Systems Design and Applications (ISDA), 2012, pp. 189–193. doi: 10.1109/ISDA. 2012.6416535.

[14]. Y. Wu, X. Liao, W. Chen, and D. Chen, “A battery management system for electric vehicle based on zigbee and can,” in 2011 4th International Congress on Image and Signal Processing, vol. 5, 2011, pp. 2517–2521. doi: 10.1109/CISP.2011.6100781.