Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 55-6 (2022) 193–198

Self-configuring BLE deep sleep network for fault tolerant WSN

C. A. Rosati*, A. Cervo*, A. Bertoli*, M. Santacaterina**, N. Battilani*, C. Fantuzzi*
DISMI, University of Modena and Reggio Emilia, 41121 Modena, Italy
**{carloalberto.rosati, andrea.cervo, annalisa.bertoli, nicola.battilani, cesare.fantuzzi} @unimore.it*
***217882@studenti.unimore.it*

Abstract: This paper is focused on Wireless Sensor Network (WSN) leveraging on Bluetooth Low Energy (BLE) connectivity for low energy applications which is fault tolerant versus communication path failures. The topic is important to create a robust sensorized environment to be applied in industrial context or smart infrastructure to enable scheduled monitoring with low power consumption applications. Currently BLE applications are mainly thought for smart home solutions, health care and positioning systems. In those applications the BLE nodes are continuously supplied by external power suppliers. Our goal is to design a self-configuring network with a synchronized deep sleep behavior, aimed to optimize the energy consumption, with an overall active time interval constraint optimized with a data-driven method. The aim is to find a tradeoff between the on time and the ability to collect all the nodes data, pursuing a low power consumption. Our research is based on BLE protocols, interaction between edge systems for data collection and cloud system for data analysis and software agent optimization system. The paper analyses different configurations and describes the possible optimization algorithm to be used for the software agent design, in order to reach a fine-tuned control to improve the fault tolerance and fault diagnosis of the system. Finally experimental results are compared with the estimates obtained via a software simulation tool implemented for this architectural pattern.

Keywords: BLE, WSN, low power network, fault tolerance, dynamic reconfiguration, virtual network simulation

1. INTRODUCTION

Thanks to the advent of industry 4.0, the Internet of Things (IoT) applications have grown in the last few decades; one application is Wireless Sensor Network (WSN). The WSN is aimed to create a network with smart devices, embedded solutions endowed with sensors capable to collect data, communicate them through the network and receive external stimuli to adapt the control policy. One of the emerging protocols in WSN application is Bluetooth Low Energy (BLE); this interest is due to many factors, including the fact that many devices are endowed with Bluetooth connectivity modules: BLE can be used to communicate with classic Bluetooth protocol, and it is energy optimized, it allows the management of multi-device communication (Bluethooth SIG, 2021). This paper has the scope to present a WSN that both optimizes energy consumption and is able to self-reconfigure when some nodes are not available for different reasons, guaranteeing at the same time the primary objective to transfer the edge collected data to a cloud cluster. Main scope of the results of the paper is to technically demonstrate the possibility to reach via BLE advertising system a self-configuring topology, robust also in case of localized node's unavailability and the possibility to optimize the active time via a software agent hosted on the cloud infrastructure, driven by data analysis principles, with the constrain to limit the on time to preserve the battery consumption. We propose a possible design of the internal policy of the software agent, that can act on the cloud server to implement fault tolerance and fault diagnosis mechanisms. Finally, the experimental results obtained in different network topologies are compared with the initial estimates computed using a software simulator designed and implemented for an initial analysis pre-installation phase.

2. STATE OF THE ART

2.1 Internet of Things and Wireless Sensor Network

The "Internet of Things" refers to the set of applications made up of a variety of devices connected directly to the Internet via cellular technologies or via a gateway in order to store data in the cloud analytics platform or using the Internet as a means of communication. In this network different kind of elements can be connected, as sensors nodes, production machines, digital enablers for production process and manufacturing lines (Vaidyaa, Ambadb, & Bhosle, 2018). The purpose to collect, aggregate and analyse the data coming from different IoT devices is to enable applications like image recognitions, object positioning, devices monitoring and tracing, and system control (Patel, Patel, Scholar, & Salazar, 2016) (Bertoli, Cervo, Rosati, & Fantuzzi, 2021). A further advantage of IoT technologies is the ability to easily manage reconfigurations and interoperability through the software, given the constraints of many industrial control systems (Vieira, Barbosa, Leitão, & Sakurada, 2020). A wireless sensor network (WSN) is a system composed by many sensors interconnected wireless (Wi-Fi or cellular technologies); the data are spread via multiple hops, and generally collected by centralized nodes with the double scope of interacting low level with the sensor network and creating a bridge towards fog-cloud applications. The purpose of creating a WSN is to store and analyse the data collected and to generate data-driven services for human operation support and improvement (Wolf, 2017), like real-time representation of a plant with automatic generation of alarms, trends and the possibility to automatically trigger actions on actuators present on the field. Generally, a common architecture adopted for the WSNs is leveraging on the choreography pattern that consists on distributed platform p2p, where a central broker acts as a service bus to interconnect the different software entities generally using message queueing protocols like AMQP, MQTT and others (MQTT and Kafka, 2019).

2.3 Bluetooth Low Energy

In 2010 the Bluetooth Specification 4.0 was introduced, and it enabled the multi-hops communication between different devices thanks to BLE specifications (Todtenberg & Kraemer, 2019). This allows to develop applications requiring low power consumption, duty cycles and low data rates. Classic Bluetooth is used when a device-to-device communication is required, to favor the data streaming process, while BLE is mainly used for multi-device connectivity with low sampling frequencies and low throughput (Galeev, 2011). Those differences are due thanks to the difference in the application layer: in classic Bluetooth there is a so called "serial Port Profile", while in BLE there are the Generic Attribute (GATT) and the Generic Access Profile (GAP). The GAP takes care of implementing the advertising mechanism, the GATT works on connection-based data exchange. Today BLE is mainly used in beaconing applications, with the scope to broadcast identifiers to make possible localization and proximity (Yang, Poellabauer, Mitra, & Neubecker, 2020). Nowadays, there is not a consistent number of concrete examples of real-world implementations of multi-hops networks via Bluetooth protocols. The most common application is for indoor positioning systems, health monitoring and smart home automation. Another common application is object tracking with beaconing, where a BLE tag is applied to the device to be tracked or is needed to sense proximity (Yang, Poellabauer, Mitra, & Neubecker, 2020). The novelty of this paper is the possibility to use BLE standard to create mesh of smart nodes, without requiring some nodes active and always on as prescribed by BLE mesh standard. Moreover, this type of architecture that leverages on the concept of synchronized decrease in power consumption through the deep sleep phase and its management with a self-reconfigurable system.

2.4 Fault tolerance in WSN

Fault tolerance is a system property that defines, when a failure occurs, the ability to continue to operate without some components (Gia, Rahmani, Westerlund, P., & Tenhunen, 2015). One of the challenges that WSNs must face is the fault tolerance to a node failure or to a measured characteristic unavailability (Gupta & Kumar, 2013). The failure can be caused by different factors, like physical failure, battery failure, environmental interferences, software failure, etc. In a distributed fault tolerant WSN, the communication flow must be guaranteed at the best efficiency even if some signals are interrupted or not available, preserving the data quality and rate for the other sensors in scope (Sun, 2020). In WNS, there are many solutions to avoid the network breakdown: some strategies are deployment based, redundancy based, clustering based. (Shyama & Pillai, 2018) (Kakamanshadi, Gupta, & Singh, 2015). A Wireless Mesh Network has the possibility to be dynamically self-organized and self-configurable. (Yu, Kin-Fai, Qiu, Liu, & Ding, 2017)

With our proposed application we want to cover the fields where internet area coverage is not guarantee and the energy optimization is a requirement due to the fact that a flexible layout is needed (no wired applications).

3. PROPOSED ARCHITECTURE

3.1 BLE network with synch deep-sleep: the concept

The architecture proposed (Figure 1) leverages on a BLE-connected customized mesh, where the standard BLE mesh (introduced with the release of July 2017) is not applied because it works only if there are nodes always active (relay nodes). Our goal is to avoid such behaviour optimizing the overall energy consumption when the data transfer of the network is not needed. Every BLE node is equipped with a sensor and a battery: the sensor data and the battery status are shared with other nodes in the network in each sampling time, with the aim to reach the gateway node to transmit the updated values to the cloud platform. We define gateway node, the BLE device being part of the BLE network but with two characteristics: it is always active (no deep sleep phase), and it is capable to establish a connection over standard internet protocols to interact with a MQTT broker hosted on cloud. The data are then stored and visible, via a web application implemented for dashboarding purposes, to the final user. We started analysing the needs for certain applications of having wireless communication, low power consumption requests for data capturing with the concurrent constraint of limited Internet access in difficult environments, like could happen in large buildings where the intranet is not covering the entire area (for monitoring consumables like paper, soap or disinfectant gels) or in major infrastructure under construction for monitoring purposes where Internet access points are not available. Recent examples are the modern Temporary hospitals that are built to relieve conventional hospitals during the pandemic periods, where it is important to monitor the hygienic condition, to maintain the safety of patients and staff. (Campos, dos Santos, Gabriel, & Montevechi, 2022)

3.2 BLE network with synch deep-sleep: how it works

To avoid battery consumption, during the cycle time there are two phases: the active phase and the deep sleep phase. During the active phase every node collects the requested data and makes them available to the network, by means of the advertising mechanism notifying the data availability in the advertising message payload. When the active phase finishes, the nodes start the deep sleep phase, turning off the RF modules to save the battery energy. We decided not to implement a Time Slotted Channel Hopping (TSCH) because it is difficult to keep the overall synchronisation of the network (Chang, Watteyne, Pister, & Wang, 2015). According to TSCH approach, when a node finishes its transmission and it is not required by other nodes for a fixed amount of time, it starts the deep sleep phase. Every node knows its hop in the network; during active phase a node advertises the nodes in the neighbourhood and adject nodes with lower hop level establish a temporary connectivity to collect the updated values. The hop hierarchy is important because the updated information is gradually migrated towards the gateway. This last node is the only device that never goes in deep sleep phase and when the active phase arrives, it spreads to the network the current time used to keep the overall synchronization and the updated active time threshold, if changed by the cloud platform. The cloud server knows how many nodes are configured in the network, since during deployment phase the metadata needed for configuration are added via the web application.

3.3 BLE network with synch deep-sleep: the achievements

There are some advantages in using this architectural strategy. First, if a node is added, the network will be able to adapt to the new dataset provided by the additional characteristic. Another important thing is that multiple routes can be created in a mesh network, and this increases the resilience when a connection failure occurs.

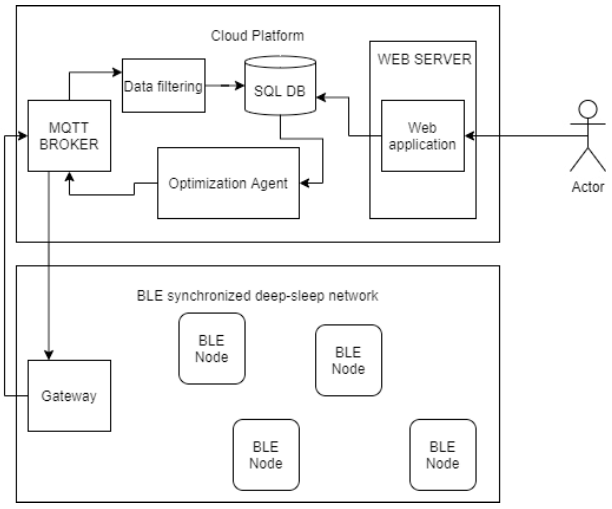

Figure 1: System architecture

3.4 Fault tolerance

In the presented architecture, the system has fault tolerance when a BLE node fails, but the others can be reached by the network and the other data can be collected by the gateway anyhow. In this kind of applications, the fault tolerance property depends on some characteristics, including the network topology and the ability of the system to self-configure. The first way that we used to reach the fault tolerance is to prefer nodes in parallel rather than in line when they are on the same hop level. With this topology we increase the fault tolerance of the system because we have the possibility to use different paths to reach the gateway if a node is missing: this is redundancy-based fault tolerance. But the novelty in the proposed architecture is that the cloud server has the possibility to compare the amount of data received every cycle time with the expected amount of data configured every time a new node is added to the network, as previously described. Leveraging on the experimental tests we performed in laboratory, we defined a pattern to be automatized in a software agent that can be hosted in the same cloud cluster as the MQTT broker and the other services for the web application. The approach we used is iterative research of the optimal threshold time for the network sampling time, using the bisection method to find the optimal sampling time (the lowest to reach global energy optimization) guaranteeing no samples lost in average. The mechanism is based on the possibility to store in a SQL DB the historical data representing the Data capturing accuracy, the Datetime of the synchronization, the time threshold for the activation time of the data chain and the number of lost samples. The algorithm starts from the two extremes of the search range and takes the intermediate threshold between extreme with no lost samples and the extreme causing lost samples.

At the beginning of the process anyhow both the deep sleep duration and the two initial interval extremes for the active time thresholds are set manually from the user. The optimized time is iteratively refined starting from the two initial values using the bisection method to find the trade-off between nodes' samples preservation and a threshold time decrease, as described in Figure 2. In the system is saved only the last value of lost samples for a given threshold, in this way the system is capable not to overfit in case of changed configuration if there are local failures in the graph. If a failure occurs and some data are missing, the agent will send to the gateway a new active time that is higher than the previous one to compensate the higher computational effort to adapt to the new topology. Worth to mention is the fact that the overall cycle time is a property of the system that is not adaptive: what instead can dynamically vary is the percentage of time in active or in deep sleep phase. The gateway will send the new active time duration to the network when the new active phase will occur. In this way if some nodes are missing because the network is not able to reach all the nodes in the active phase, the network will be reconfigured automatically, increasing the system fault tolerance. A maximus active time is present, to avoid it from growing too much and to reach a low power application. The sampling time is set by the user according to the system use.

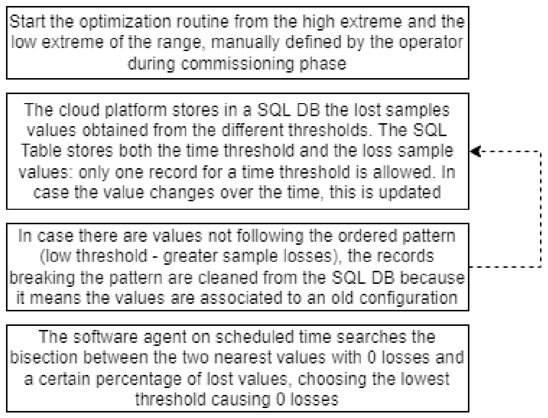
Start the optimization routine from the high extreme and the low extreme of the range, manually defined by the operator during commissioning phase
The cloud platform stores in a SQL DB the lost samples values obtained from the different thresholds. The SQL Table stores both the time threshold and the loss sample values: only one record for a time threshold is allowed. In case the value changes over the time, this is updated
In case there are values not following the ordered pattern (low threshold - greater sample losses), the records breaking the pattern are cleaned from the SQL DB because it means the values are associated to an old configuration
The software agent on scheduled time searches the bisection between the two nearest values with 0 losses and a certain percentage of lost values, choosing the lowest threshold causing 0 losses

Figure 2: Agent behavior

3.5 Fault diagnosis

Fault diagnosis is the ability to understand which fault occurred in the network. To determine that, the system must know the trend over the time of the system, in order to detect anomalies. If the agent that is working on the server does not receive the data from some nodes, it will increase the active phase. If after some interactions some nodes are still missing, it means that the nodes have different problem that cannot be fixed without the intervention of a human operator. The causes can be many: the battery may have run out, it could be a hardware problem, or the network may be modified, and some nodes are no more visible to the network. The agent knows the past status of the nodes, and if the nodes is still missing after some interactions, it will send a notification to the user, e.g. an email. In general, we recommend leaving the number of interactions as a configurable parameter, since it depends on the topology and the application. When the notification is sent to the user, the deep sleep phase is not decrease anymore; otherwise, the system will continue to keep the device active for no reason. After the notification, if the node is found in next iteration by the cloud software agent, the active time is reduced, restarting the optimization process. Other predictive alerts can be sent by the software agents monitoring the battery status of each device, setting a alarm threshold to notify human operator that a battery change is needed.

4. TEST AND RESULTS

4.1 Materials used

To test the system, we have set up a BLE network in the research laboratory using 5 ESP32 (ESP32 datasheet, 2021). The BLE-MQTT gateway was embedded with a SIM800, that is a Quad-band GSM/GPRS solution. The cloud server is used to host the MQTT broker, the data analytics module for data capturing, filtering and aggregation and finally to host a web application based on a NodeJS webserver.

4.2 Goals and Results

In this section we present the internal goal for a technical success of the experiments and the collected.

The internal objectives are:

1. Demonstrating that the concept is suitable for different network topologies, collecting reference data with experimental performance results, capable to self-configure the path leveraging on RSSI (received signal strength indication) value of the advertising signals.
2. Demonstrating the hypothesis on inverse proportion between the low sampling time aimed to optimize energy consumption and the samples availability for each cycle on the cloud infrastructure.
3. Demonstrating the Fault-tolerance optimization in case of killed nodes in the network.
4. Demonstrating the possibility to virtually simulate the network topologies, to estimate the performances of a real network while designing the topology. We use 3 different topologies reported in Figure3, Figure 4 and Figure 5. The node named G is the BLE-MQTT gateway.

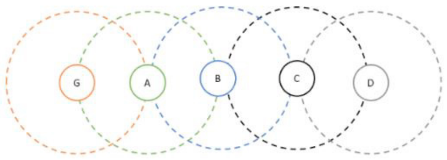

Figure 3: Line configuration

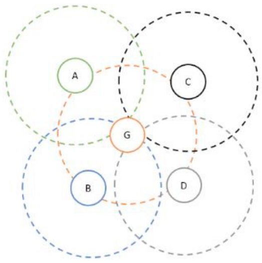

Figure 4: Star configuration

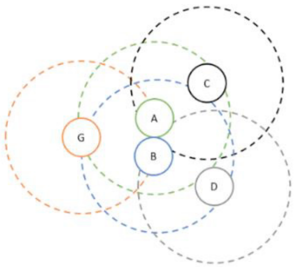

Figure 5: Mixed configuration

For all the tests the cycle time is 1 minutes, when the active time is indicated in the respective table. We run some tests reported in Table 1. We used the three presented configurations changing the threshold time. The threshold time is the active time for every node. We report the average time (in seconds) of the active time for each configuration. The average is obtained running five test repetitions for every configuration. When a node has sent its data and no nodes in its neighbourhood are available, it goes in deep sleep phase.

config	Active time [s]	failure	A [s]	B [s]	C [s]	D [s]
Line	30	0%	22.2	18.5	15.8	6.3
Line	20	66.7%	17.5	14.5	11.5	5.5
Line	25	0%	16.8	13.2	9.3	6.2
Star	30	0%	3.8	10.5	12.5	4.8
Star	20	0%	4.5	8.3	7.5	4.5
Mixed	30	0%	10.3	12.7	5.2	6.8
Mixed	20	0%	8.5	10.2	3.7	4.8

Table 1: Test results

With the Line configuration (Figure 3) we run the test with a threshold time of 30 seconds, and we reach all the nodes in the network. We decrease the active time to 20 seconds, but it happened that 66.7% of time that the network was not able to complete the task in the given time. We try with a threshold of 25 seconds and the result is that the network is able to finish the task and the nodes can save some battery energy. With the star configuration (Figure 4) we run the same test. We start with a threshold of 30 seconds, and all the nodes were reached. We try with 20 seconds and no data are missing. In a future, when the agent will be implemented, it could be interesting to try to decrease another time the threshold time. Watching the network average time, it could be set to 15 or 10 seconds.

The last configuration is mixed (Figure 5). The node C and D can communicate both with node A or B, depending on their availability. As in star configuration we run a test with a threshold of 30 seconds, then we decrease the value to 20 and in both situations all the data were collected.

In order to understand the network behaviour when a node is missed, we run some test killing the node A in every configuration. The results are reported in Table 2. In the Line configuration if node A is missing, the network will not be reached by the gateway and no data will be collected. In the star configuration, only node A data are missing. The most interesting network is the mixed configuration. In this case if node A is missing, the node C and D communicate only with node B. The communication that is balanced when all the nodes are available, becomes unbalanced, but all the data, except node A, are available.

config	**Killed node**	**Data lost**
Line	A	100%
Star	A	25%
Mixed	A	25%

Table 2: Test results when the node A is killed

To understand the network behaviour in a different configuration, we developed a simulation in python to estimates in advance a complex system. The simulation works as a state machine for a fixed iteration. Every state has a duration in terms of number of iterations, as an example the scan state takes more iterations than the connect state. First, we test the simulation with the three presented layouts. The data are reported in Table 3, with the average time of active time per each node expressed in iterations.

Config	Active time	Failure	A	B	C	D
Line	20	0%	15	12	9	6
Star	20	0%	3	7	11	15
Mixed	20	0%	10	8	4	6

Table 3: Simulation test results

Comparing the simulation with the data obtained with the real tests on the ESP32 nodes, the system behaviour is comparable, so we simulate a complex network with 50 nodes. In Figure 6 there is the simulated network with 50 nodes randomly distributed in a physical environment. The screenshot reported from the simulation in Figure 6 represents the network working. The nodes are here described:

- the red node is the gateway, the light blue nodes are during the scanning process
- the blue nodes are connected with other nodes to collect their data and the arch represents the connection
- the grey nodes have finished the communication and are already in deep sleep phase
- the light green nodes have finished to receive information from upper level and are waiting to be read by lower nodes
- the purple nodes are waiting to be configured with system date time and active time

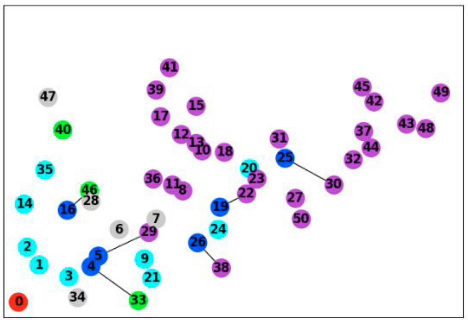

Figure 6: Simulated network

Time used to collect all information	57 iterations
Node average aweke time	28.64 iterations
Node minimum active time (nodes 7, 34)	6.0 iterations
Node maximum active time (node 3)	56.0 iterations

Table 4: Simulation results

This simulation verifies that complex topology can be managed with the proposed system. In Table 4 are shown the results obtained at the end of the procedure. With the mixed topology simulated, the network uses 57 iterations to configure the system and to obtain all the data. The average iteration per node is 28.64 seconds, but the nodes that are close to the gateway use more than 50 iterations to finish their task and to go in deep sleep, while two nodes are active for only 6 iterations. As explained, our goal is to create a network under the constraint of low energy consumption. The energy used depends on four factors:

- the active time of the nodes
- the node positioning in the network
- the capacity of the battery used to power each node
- the sampling time.

In order to reduce the active time of each node, a solution is to add gateways to the network, with objective to reduce the hops in the network (Figure 7 and Table 5). The objective is to move from line configuration to a star configuration because the network needs less time to reach all the nodes and fault tolerance increases where nodes are in parallel rather than in line.

Time used to collect all information	32 iterations
Node average aweke time	15.02 iterations
Node minimum active time (nodes 7, 34)	6.0 iterations
Node maximum active time (node 30)	31.0 iterations

Table 5: Simulation results with two gateways

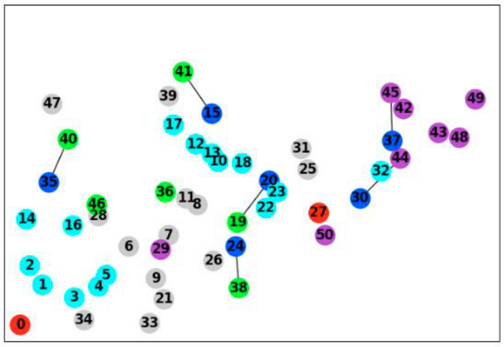

Figure 7: Simulated network with two gateways

5. CONCLUSIONS AND FUTURE RESEARCH

With this paper we show a design pattern for WSN with BLE protocol, aimed to optimize the energy consumption and improve the overall fault-tolerance of the system. In general, one complexity of such systems is the predictability of the behaviour before the installation: for this reason, one deliverable tested during the research is a simulation tool that, from the comparison with real data, is performing quite well. In future research, we will focus on the deviations between the simulation tool and real test results. The objective is to leverage on the identified deviations in order to develop an Artificial Neural Network, capable to perform a compensation on the results obtained with the simulation tool. When the network and the simulation will be finished, it will be interesting to find a field of application for this network and verify the proposed solution. A possible application can be in the health care: using some wearable device, it is possible to monitor some parameters in diabetic patients to measure the blood insulin and act if the level in not optimized. The smartphone could be the gateway system, and the wearable device becomes the BLE node.

REFERENCES

Bertoli, A., Cervo, A., Rosati, C. A., & Fantuzzi, C. (2021). Smart Node Networks Orchestration: A New E2E Approach for Analysis and Design for Agile 4.0 Implementation. *Sensors*.

Bluethooth SIG. (2021, 11 12). Retrieved from https://www.bluetooth.com/

Campos, A. T., dos Santos, C. H., Gabriel, G. T., & Montevechi, J. A. (2022). Safety assessment for temporary hospitals during the COVID-19 pandemic: A simulation approach. *Safety Science*.

Chang, T., Watteyne, T., Pister, K., & Wang, Q. (2015). Adaptive synchronization in multi-hop TSCH networks. *Computer networks*, 165-176.

ESP32 datasheet. (2021, 10 17). Retrieved from [24] “ESP32 datasheet,” [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. [Accessed 17 10 2021].

Galeev, V. (2011). Bluetooth 4.0: an introduction to bluetooth low energy-part II. *EE Times, Design*.

Gia, T. N., Rahmani, A., Westerlund, T., P., L., & Tenhunen, H. (2015). Fault tolerant and scalable IoT-based architecture for health monitoring. *IEEE Sensors Applications Symposium (SAS)*, 1-6.

Gupta, C., & Kumar, A. (2013). Wireless Sensor Networks: A Review. *International Journal of Sensors Wireless Communications and Control.*

Kakamanshadi, G., Gupta, S., & Singh, S. (2015). A survey on fault tolerance techniques in Wireless Sensor Networks. *International Conference on Green Computing and Internet of Things (ICGCIoT)*, 168-173.

MQTT and Kafka. (2019). Retrieved from medium: https://medium.com/@emqtt/mqtt-and-kafkaf20d79d9dea4

Patel, K., Patel, S., Scholar, P., & Salazar, C. (2016). Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges.

Shyama, M., & Pillai, A. S. (2018). Fault Tolerance strategies for Wireless Sensor Networks – A Comprehensive Survey. *3rd International Conference on Inventive Computation Technologies (ICICT)*, 707-711.

Sun, J. (2020). Fault Tolerance Mechanism of a Distributed System for Marine Communication Network. *Journal of Coastal Research*, 605–608.

Todtenberg, N., & Kraemer, R. (2019). A survey on Bluetooth multi-hop networks. *Ad Hoc Networks*.

Vaidyaa, S., Ambadb, P., & Bhosle, S. (2018). Industry 4.0 – A Glimpse. *2nd International Conference on Materials Manufacturing and Design Engineering* .

Vieira, G., Barbosa, J., Leitão, P., & Sakurada, L. (2020). Low-Cost Industrial Controller based on the Raspberry Pi Platform. *IEEE International Conference on Industrial Technology (ICIT)*, 292-297.

Wolf, M. (2017). Chapter 8 - Internet-of-Things Systems. *Computer as Components (Fouurth Edition)*.

Yang, J., Poellabauer, C., Mitra, P., & Neubecker, C. (2020). Beyond beaconing: Emerging applications and challenges of BLE. *Ad Hoc Networks*.

Yu, L., Kin-Fai, T., Qiu, X., Liu, Y., & Ding, X. (2017). Wireless Mesh Networks in IoT networks. *International Workshop on Electromagnetics: Applications and Student Innovation Competition*, -.