Journal of
Sensor and Actuator Networks

MDPI

Article

Self-Configuration Management towards Fix-Distributed Byzantine Sensors for Clustering Schemes in Wireless Sensor Networks

Walaa M. Elsayed [1], Engy El-Shafeiy [2], Mohamed Elhoseny [3,4,*] and Mohammed K. Hassan [5]

[1] Information Technology Department, Faculty of Computing and Informatics, Damanhour University, Damanhour 22511, Egypt; walaazaid80@cis.dmu.edu.eg
[2] Computer Science Department, Faculty of Computers & Artificial Intelligence, University of Sadat City, Sadat 32897, Egypt; engy.elshafeiy@fci.usc.edu.eg
[3] College of Computing and Informatics, University of Sharjah, Sharjah 4679, United Arab Emirates
[4] Faculty of Computers and Information, Mansoura University, Mansoura 35516, Egypt
[5] Mechatronics Department, Faculty of Engineering, Horus University–Egypt (HUE), New Damietta 34517, Egypt; mkhassan@horus.edu.eg
* Correspondence: melhoseny@ieee.org

Abstract: To avoid overloading a network, it is critical to continuously monitor the natural environment and disseminate data streams in synchronization. Based on self-maintaining technology, this study presents a technique called self-configuration management (SCM). The purpose is to ensure consistency in the performance, functionality, and physical attributes of a wireless sensor network (WSN) over its lifetime. During device communication, the SCM approach delivers an operational software package for the radio board of system problematic nodes. We offered two techniques to help cluster heads manage autonomous configuration. First, we created a separate capability to determine which defective devices require the operating system (OS) replica. The software package was then delivered from the head node to the network's malfunctioning device via communication roles. Second, we built an autonomous capability to automatically install software packages and arrange the time. The simulations revealed that the suggested technique was quick in transfers and used less energy. It also provided better coverage of system fault peaks than competitors. We used the proposed SCM approach to distribute homogenous sensor networks, and it increased system fault tolerance to 93.2%.

Keywords: WSN; cluster head; sensor node; self-configuration; self-diagnosis; self-maintain

Citation: Elsayed, W.M.; El-Shafeiy, E.; Elhoseny, M.; Hassan, M.K. Self-Configuration Management towards Fix-Distributed Byzantine Sensors for Clustering Schemes in Wireless Sensor Networks. *J. Sens. Actuator Netw.* **2023**, *12*, 74. https://doi.org/10.3390/jsan12050074

Academic Editor: Lei Shu

Received: 9 August 2023
Revised: 2 October 2023
Accepted: 9 October 2023
Published: 13 October 2023

1. Introduction

Wireless sensor networks (WSNs) have a large number of sensory devices that are deployed in an area and integrated to work wirelessly together. As a result, implementing a sensor node is a key difficulty in WSNs. This has a significant impact on efficiency and costs. So, when we deploy a sensor node, we must consider factors such as the deployment's goal, as well as the topography and coverage capability of the sensor node. A WSN's primary goal is to provide broad coverage with as few sensor nodes as possible. Wireless communication has been around for a long time, dating back to the first infrared link. People have been attempting to develop wireless connections between electronic items and computers for a long time. Due to the rapid development of compact, dense radio-frequency integrated circuits (RFICs) and filters during the last five to ten years. In addition to scientific advances in spread-spectrum technologies, many wireless technology standards have been developed, allowing many manufacturers and implementers to achieve interoperability between many devices [1]. In the subject of comprehensive monitoring, habitat monitoring often entails continuous operation for many months, whereas structural monitoring (such as bridge monitoring) may take several years. A recent study has demonstrated that these

wireless networks can cause numerous system problems in dispersed networks, known as Byzantine flows. A system issue can be identified by the fact that some network nodes fail to react or respond with inaccurate information. This can be accomplished by dynamically adjusting node duty cycles in dense sensor networks. As a result, many academics have specialized in putting specific nodes into hibernation (or failure-mode coverage) while the remaining active nodes continue to provide uninterrupted service. A fundamental issue is minimizing the number of active nodes while maintaining an adequate level of service for applications. Maintaining enough sensing coverage and network connectivity with active nodes, in particular, are critical criteria in sensor networks [2].

A system error is a circumstance that infects an operating system, particularly a distributed computer system, in which a component may fail and information regarding whether or not a component has failed is insufficient. The word is derived from the story "The Problem of the Byzantine Generals" [3], which was designed to illustrate a circumstance in which the hardware components of the system have failed. A chief node can exist in both failed and working failure-detection systems, displaying various symptoms to different observers. It is difficult for the other components to announce it failed and disconnect it from the network since they must first agree on which component failed. As a result, system fault tolerance (SFT) refers to a fault-tolerant network system's resistance to such scenarios [4].

As a result, typical WSN research has focused on network configuration, network operations, and performance evaluation of WSNs that execute the primary function. The Internet of Things (IoT) is becoming a reality as sensor technology advances, allowing the installation of inexpensive, compact, and computationally efficient sensor devices. Although collecting and monitoring sensor data does not necessitate considerable local data processing for decision-making at sensor nodes, some new WSN applications necessitate fast and dependable device decision-making. Because of the large number of IoT devices that require high bandwidth and cause network connectivity challenges, on-device self-configuration eventually leads to increased computing capabilities of sensor nodes to supplement local data monitoring.

Efforts are being made in the proposed self-configuration management (SCM) to use fault tolerance mechanisms to restore the network to its original state. This study will address the programmatic platform for WSNs as follows: (1) Self-detection for infected device nodes in each cluster. After each round, each group leader performs a monitoring task for the clustering elements, then it computes the dimensions of the location of a faulty node in its hop. (2) Self-directing for conveying data from the clustering head to obtain reliable data transmission, using the synchronous and distributed fair scheduler. (3) Self-install to allow all detected nodes to complete their installations in the shortest possible time T (n2). Finally, the paper is laid out as follows: Section 2 includes articles relevant to this work. Section 3 shows the problem's preliminary stages and details the proposal (SCM) technique and suggested algorithms. Section 4 presents the findings of the evaluation and analysis.

2. Related Works

Wireless sensor networks (WSNs) are widely used across various technical fields. When coupled with Internet of Things (IoT) technologies, WSNs can collect information from different locations, ultimately improving the quality of life for people. Infrastructure WSNs, where sensor nodes and sink nodes self-configure, are suitable for most applications like building and structural health monitoring, environmental monitoring in greenhouses, water resources monitoring, and smart meters for homes and buildings. Wireless technologies are constantly evolving to cater to specific and unique markets. However, there is no single wireless technology suitable for all network systems that can be continuously deployed for long periods. When deciding on a wireless network standard, it is critical to consider the requirements and needs of the network. This section highlights relevant work

on fault tolerance for system defects in WSNs in brief, with a focus on distributed fault tolerance techniques based on cutting-edge technologies.

We have gathered more information about the subject after thorough consideration and analysis. Researchers have developed many approaches to detect errors in WSNs that do not have processors/central processing units (CPU). Due to performance constraints, these algorithms are unable to detect faults in WSN CPUs/software. Yu et al. suggested a fault-tolerant consensus algorithm based on double grouping signatures for global practical Byzantine fault tolerance (GPBFT) in July 2022. This system chooses master node and pool administrators based on creditworthiness, groups new customers, and creates signatures using a group court signature procedure. The trace phase is likewise supported by the GPBFT approach, allowing the group administrator to verify the true identity of unwanted newcomers and cancel their access. The proposed practical Byzantine fault tolerance (PBFT) consensus algorithm (GPBFT) combines the advantages of group signatures with a longer run time, greater application, and new verifiable data for administrators. The experimental results revealed a decrease in communication cost and volume, as well as an increase in communication efficacy. The GPBFT algorithm can additionally incorporate a control that extends to new malevolent users, increasing its effectiveness in detecting flaws [5].

In the period 2016–2022, many studies used performance assessment criteria to conduct a comprehensive evaluation to examine the limitations of current fault-tolerance techniques. Despite the thorough examination, there are still unanswered questions concerning the specified phrase. Adday et al. [6] presented a thorough investigation of fault tolerance structures identifying key components and categorizing faults from various angles. They also examined existing fault-tolerance strategies using eight criteria, emphasizing the significance of a network's capacity to withstand failures in wireless sensor networks (WSNs). The authors discovered that employing a fault-tolerance strategy can dramatically reduce overall WSN faults while also improving network functionality.

Chang et al. [7] suggested a technique to increase the effectiveness of data aggregation in wireless sensor networks (WSNs) through node cooperation in January 2022. According to the findings of the study, utilizing a multi-cluster network design with an adequate fault tolerance value can increase the success rate of achieving a data consensus and transmitting data to the sink. They also discovered that cluster heads (CHs) can make distributed network management and consensus computation more complex, resulting in a non-deterministic polynomial issue. Furthermore, the study yielded two major discoveries. To begin, using the Byzantine fault tolerance technique may prevent malfunctioning nodes from interfering with regular network operation and increase the network life cycle. Second, employing the MC method can cut sensor node energy consumption and data aggregation time.

Cotroneo et al. [8] introduced a practical Byzantine fault tolerance (PBFT) model at the same time. Certain violations discovered by ByzzFuzz were not detected by it. These problems involve a process error that corrupts a message's sequence number with the sequence number of a future round, which a twin duplicate cannot provide. Although PBFT permits copies to fall behind by losing some process information, it does not introduce incorrect messages with protocol message field values in the future. ByzzFuzz and PBFT, we think, may be used in tandem, complementing each other by methodically evaluating Twin's collection of adversary behaviors and ByzzFuzz's sampling from a broader variety of problematic process behaviors.

In December 2022, a model for optimizing the node structure of a big consortium chain was suggested. This was accomplished by splitting the vast network nodes into various institutions to form autonomous consensus groups using clustering characteristics. A respectable rating incentive system was also presented to boost the consortium chain's consensus efficiency. This system featured a formula for calculating reputation scores, which was used to pick master nodes with high reputations. A replacement cycle was performed to improve the chain's consensus efficiency even further. This cycle replaced nodes with high reputation with nodes with poor reputation. Feature grouping and credit

optimization Byzantine fault tolerance (FCBFT) have lower latency and greater throughput of the transaction processing system (TPS) than PBFT, according to experimental data [9].

The researchers presented a solution in 2023. Algorithms for fault identification are often based on statistical approaches such as repeated testing or machine learning. These procedures, however, can be difficult and time-consuming. To solve these concerns, the researchers suggest a one-shot likelihood ratio test (LRT) for evaluating a sensor node's failure state in a wireless sensor network. The suggested approach entails computing the mean and variance of the received data over time and then comparing the probability ratio with a threshold value linked with a specific tolerance limit. As a result, they were able to detect intermittent problems in the nodes with high accuracy, without the need for repetitive testing or computationally intensive machine-learning algorithms [10].

In August 2023, Goud et al. [11] outlined the three approaches utilized by WSNs to recover from failed nodes: LDTR, FNR, and GD algorithms. They compared these techniques to the proposed system, which incorporates a path arbitrator and the RAFT blockchain consensus methodology. According to the article, the proposed failure recovery technique, which is based on the RAFT blockchain consensus mechanism, can aid in the control of Byzantine failures in wireless sensor networks as well as the recovery from malicious activities. In simulations, the suggested algorithm was assessed based on energy utilization, defect detection precision, packet delivery ratio, number of dead nodes, and time number of neighbor nodes.

3. Self-Configuration Management (SCM)

The introduction of a self-configuration management paradigm that addresses software flaws is the main goal of our suggested approach. The SCM model, which includes the network, radio, and system fault models, will be used to do this. We are currently trying to finish the suggested plan and expand on prior work [12]. In essence, the SCM method strives to develop a dual-integrated approach that combines IOT technology with our suggested network. To link and exchange software packets among selected network nodes and systems over the internet, it will be necessary to integrate sensors, software, and monitoring values from many fields. Table 1 provides the notations and their explanations.

Table 1. The notations used in modeling the proposed distributed SCM technique.

Symbol	Explanation
S	Set of sensor nodes in sensor networks
S_i	A sensor node distributed on Pi(x_i, y_i)
N	Number of deployed sensor nodes
P_R	Likelihood of faulty sensor nodes in sensor networks
S_1	Set of sensor nodes experiencing error-free deadlocks in sensor networks, $S_1 \in S$
S_2	Set of sensor nodes where deadlocks occur when sensor networks fail, $S_2 \in S$
S_3	Set of sensor nodes experiencing random failures in sensor networks. $S_3 \subset S$
$N_{CONF.}$	The number of faulty sensor nodes that were configured, $N_{CONF.} < N$
S_F	Set of faulty sensor nodes in sensor networks, $S_F \subset S$
S_H	Set of error-free sensor nodes in sensor networks, $S_H \subset S$
C_{Ti}	Table of clustering of si which contains all the information about each sensor node and its neighbors.
S_{FI}	Failure status of sensor node S_i.
D_{Tr}	Distance range will be covered by a predefined transfer rate
C_I	The cluster supervisor is in charge of the configuration mode and is usually the head of the group.
C (bps)	Channel throughput capacity (bandwidth), measured in bits per second.
B	The frequency bandwidth in Hertz, and
M	the number of levels a single signal can take on.
SNR	Desired signal-to-noise ratio at the receiving node.
NR	Receiver signal-to-noise ratio.
Gant	Transmit antenna gain.
E_T	Consumed energy in bit transfers.
E_{ncnode}	Encoder power consumption.
N_b	Total number of bits transmitted.
E_{tb}	Energy consumption when transmitting a single bit from a node.
E_{rb}	Energy consumption when a single bit is received by a node.

3.1. Network Model & Problem Sanrio Preliminaries

IRIS sensor units, which may be dispersed randomly or in a set location around the environment, have been used to design a wireless sensor network. The network employs a multi-hop method for sensor transmission. The network should be set up using existing wireless network deployment and training procedures. A representative node is chosen to serve as the cluster head (CH) for each cluster, and a predetermined number of clusters (C) are also set. A distributed method is created when the cluster leaders make local decisions based on data received by cluster members and convey these decisions to the base station (BS). Each sensor node in the sensor network has a specific identification number and is aware of its location. Using the LEACH routing protocol, the group members broadcast beacon messages across the network during the mobility phase. Aggregation nodes are inserted into the network to remedy system flaws. These nodes reduce the number of messages sent between nodes and conserve energy by receiving data from malfunctioning nodes, processing it, and then sending the filtered data to the following hop. Communication between sensor nodes inside a cluster passes through the CH until it reaches the base station. Sensor nodes are grouped into clusters with a CH acting as the leader. Through specified Mac & IP addresses of the clustering devices, the CH may identify working devices and available aggregation devices in the network. Additionally, it can track the development of ongoing processes and receive alerts when both processes are finished.

The sensor nodes can communicate with one another using the fundamental one-to-many transmission mechanism. It is assumed that each sensor node's transmission range is the same. The network was built during the implementation of the specified network assumptions and complies with the international resource identification system (IRIS) for wireless sensor networks (WSNs). Data is transferred by the IRIS sensors at a rate of 250 kb/s. Two AA batteries (3 V) with a current capacity of 2200 mAh^{-1} are required to power each sensor [13]. The initial battery capacity is calculated using the formula battery capacity = $3 \times 2.2 \times 3600$. The processor's computational capacity, when in full operation mode, can reach up to 7.37 MHz.

During deployments, the communication model generated a random network topology. In the transmission range Tr of Ci, each sensor node is referred to as a connected cluster. Any sensor node located within Tr of Ci's communication range is referred to as a linked cluster. When two sensor nodes are connected, the healthy sensor is given the job of the damaged sensor if the distance between Sensor Si and its neighboring Sensor Sj is smaller than DTr in fault status. Each node in the sensor network, if it is tightly connected, is a member of the cluster and has a set of nearby sensor nodes (SNegi). It stores the information locally in memory and sends it to its leaders for evaluation.

By defining Ci as the ratio between the total number of sensor nodes covered by the configuration in the network provided and the sum of all the sensor nodes' degrees, the average degree of adaptation of the configured sensor nodes during the operational cycles was computed as:

$$D_{CONF.} = \sum_{I=1}^{N} N_{CONF.}/\mathrm{N}_i \quad (1)$$

3.2. Data and Radio Model

We used sensor nodes equipped with wireless transceivers to enable continuous connectivity and integrated coverage. If the distance between two nodes is less than the communication area Ca the two nodes can communicate directly, then $|S1 \,\&\, S2| < Ca$. To enable data transfer without human intervention, distinctive identities were given to the sensing devices in the network. System replicas were transferred between devices with the use of the main node's services. Additionally, it chooses the main node CHi for each cluster and serves as a manager device, with the main node managing the transfer of system replicas across devices.

We created the radio system with self-configuration techniques in mind. Every cluster's N-nodes are included in the communication path, starting with n1, moving on to

n2, n3, n4, and so forth until it reaches Nc. It is crucial to remember that a package size is sent from S1 to SN within the time interval [0, t]. We may calculate the size of the sent software package in bits per configuration by specifying the configuration flow rate Pi. The transmission duration for each cycle can be written as S1 → Si + 1, and the time interval t_i can then be determined.

$$t_i =^n / P_i \times R_i \tag{1}$$

Let us say that β1 represents the energy needed by the transmitter, and β2 represents the energy required by the amplifier-receiver electronics. To calculate the total energy necessary for transmitting and receiving a volume of P_i over a distance of D, the formula for transmission energy, $(E_T(Pi, D))$ can be used as:

$$E_T(Pi, D) = P_i(\alpha\beta 1 + \alpha \times D) \tag{3}$$

α is a spacing coefficient that can vary from 2 to 4 [14].

In addition, we developed a model to determine how much energy was lost throughout the package exchanges. This model accounts for the energy used by the sensors in the previous transfer mode (E_T) as well as the radio frequency transceiver power (RF transceiver), a parameter Ci that affects transmission quality in terms of bit error rate and noise.

$$\mathrm{E}_i = RF\text{–}(E_T \times C_i \times t_i) \tag{4}$$

Based on this, the receiving model (E_R) can be determined using the following equation:

$$E_R(Pi, D) = P_i \times \ \alpha\beta 2 \tag{5}$$

The radio communication (*RC*) model was used to create a network connection between sensors within each cluster in the proposed network. The relationship between tolerance and connectivity was analyzed and then the cost of RC was calculated based on configuration energy $\left(E_{conf.}\right)$, tolerance energy $(E_{to.})$, and performance. Using these calculations, the most efficient power consumption for $(E_{RC}\ )$ was evaluated.

$$\mathrm{C(bps)} = 2\mathrm{B} \times \ log_2\mathrm{M} \tag{6}$$

$$E_{to.} = \sum_{i=1}^{m} N_b \times E_{rb} \ \times E_{encod.} \tag{7}$$

$$E_{conf.} = \frac{SNR \times NR \times E_{to} \ \times C(bps)}{\mathrm{Gant} \ \times Et_b \times \mathrm{B}} \tag{8}$$

$$E_{RC} \ = \sum_{i=1}^{m} \left(E_R \ , E_{conf.}, \ E_{to.} \ , E_T \ , E_i\right) \tag{9}$$

The suggested technique for finding defects in a distributed system is divided into two phases and follows an established methodology. It seeks to make it possible for the system to recognize and fix any configuration problems on its own.

❶ *Phase 1: Distributed soft fault detection model (DFD)*

In sensor networks, we assumed that all error-free sensor nodes measure the same physical value at time t. Different physical quantities are measured by WSN sensor nodes with errors (data collected at time t by sensor node Si). The data model requires that the mean value M must be the same for all healthy sensor nodes. However, the unpredictability (sometimes referred to as misconduct or misbehaving) that various sensor nodes measure may vary. The WSN literature frequently assumes that this sensor data pattern can identify faulty sensor nodes. As a result of the sensor circuitry producing inaccurate data, we suggest the set S of sensor nodes is susceptible to inaccuracy. We presumed that all error-free sensor nodes in sensor networks measure the identical physical value at time t. With faults, WSN sensor nodes measure various physical quantities (data collected at time t

by sensor node Si). The data model stipulates that for all healthy sensor nodes, the mean value M must be the same. However, the degree of unpredictable behavior—also known as misconduct or misbehavior—that different sensor nodes measure may differ. The premise in the WSN literature is that this sensor data pattern can spot malfunctioning sensor nodes. We propose that the set S of sensor nodes is subject to inaccuracy because the sensor circuitry generates erroneous data. It deployed S_1, S_2, and S_3 to the set of randomly selected sensor nodes suffering from zero blocking, one blocking, and random failures. Thus, the set of healthy sensor nodes present in the network is $\mathbf{S_H} = S - S_F$, where $S_F \subset \{S_1 \cup S_2 \cup S_3\}$.

Each sensor node verifies its group at the start of the radio communication phase by transmitting a pulse-of-life notification message that includes information about its heartbeat intensity, energy, transmission duration, and distance. It then produces data based on the surroundings where the sensor is being used. As a result, the data from the sensors that were fixed in accordance with the clusters of the designated network (C1, C2, C3... Cn) was provided. The data generated is evaluated to classify the information of the impulses sent, according to hypothesis H1, for which the CH credits the integrity of the node-set responses. While redundant impulses are categorized as hypothesis H2 (transient unstable impulses) and are regarded as system failures. H2 is another symbol for functional system disorder over discrete timescales. The error data concluded at the sensor node (*E*) are classified based on comparing accurate information coming from sensors of each cluster, and data of redundant and unstable sensors. Therefore, partial fault state coverage will be applied based on the faulted node's neighboring nodes S_{Negi} from the received data X_j. Fault detection was based on testing the former binary hypotheses. Let us hypothesize that hij has the binary decision (0 or 1) caused by testing information s_i by S_j, where $s_i \in S$ and $S_j \in$ Negi. Accordingly, let us say that hij = 0 if the sensor node decides the error hypothesis H2, whereas hij = 1 if it comes true and hypothesis H1 and healthy data decide it. Algorithm 1 illustrated the distributed system-fault detection method at the CH level.

Algorithm 1. Distributed system fault detection (DSFD) method at the CH level.

Input: Sensor position, Dx distance, transmission rang Rt, *M*.
Output: system fault status of sensor (f_{system})
Initialization phase

Each sensor Si∈S, each sensor discovers its neighbor S_{Negi}*; and Si* is considered a free fault sensor (*Hardware physical composition is healthy*)

Radio phase
For i = 1 to S do

1. Send a live-pulse message to CH synchronically;
2. CH delivers a periodic sensor message *M* and checks it timely t_i;
3. *CH establishes vector* $S_{healthy} = S + S_{healthy}$;
4. *CH establishes vector* $S_{faulty} = S + S_{faulty}$;
5. Conduct fault detection.

Fault detection phase
For i = 1 to S
If S fails to respond within transfer time t_i
CH set H2 in its *memory*
$S_{faulty} = f_{system} + S_{faulty}$
End else
If M is incorrect information signals
CH set H2 in its *memory*
$S_{faulty} = f_{system} + S_{faulty}$
End else
CH set H1 in its *memory*
$S_{healthy} = S + S_{healthy}$
End
End

❷ *Phase 2: Self-Configuration Model*

The self-configuration model is managed through the succeeding sequences:

- Cluster head level

After the discovery phase, the CH would manage the network and should test the health of the network during its deployment. The network status includes the connection status and each node's location and battery level. To check the network status, the CH needs to receive this information from the deployed nodes to confirm its test. Then, it establishes a status table of the attached clustering devices. Through the status table, CH in each cluster settles on defective nodes that issued the system fault in its cluster, and also the distance dimensions of that node for its location. Next, CHs in each cluster cooperate with the IoT service to cover the faulty nodes by dispatching the picked package to them across the network layer. The network layer encapsulates the software packet in datagrams and handles a message at the sending and receiving nodes. This includes the time to prepare the message (adding header, trailer, and error correction information), the time to execute the routing algorithm, and the time to establish an interface between the routing node and the receiving node. The delay incurred only once for a single packet transfer. This is known as startup time (t_s) and is the time that requires the *head* to initiate configuration-communicate capability and accept to route that packet to the transport layer for delivering the packet to the decided node. Through that level, The CH could designate which device will receive the software packet, by deciding on it from the *status table*. The table comprehends:

1. Sensor ID and its distance;
2. Sensor Mac address;
3. Sensor energy level;
4. Activity mode (idle, active, sleep).

The steps of installation for the problematic node may also be selected, as well as what would happen once the software installation is complete. Additionally, it has the ability to postpone the program installation and transfer in the event that significant obstacles arise at the same moment. Low battery life, limited computational power, and ineffective communication resource use are a few of these difficulties. Figure 1 depicts the key steps of the distributed configuration. The setup dialog that was carried out under the direction of the admin node, represented as CH at that phase, is highlighted in this figure. The CH chooses which device will receive the software replica first after determining the status table of the malfunctioning sensors that were sent to the BS (base station). The BS accepts the chosen course of action and grants permission for the networking system's setup and transfer processes. Following the approval log, CH creates a record containing a replica and installation details for each SF in its cluster after planning installation procedures and assigning sensor distances. The Byzantine sensor SFI receives the installation data via intranet networking, together with executable files, and the installation process works as follows.

- Byzantine sensor level:

The sensor (S_{FI}) receives a configure notification message from CH. S_{FI} accepts the notification issued by the configuration message and starts to prepare to establish a communication interface among them, which is needed to execute its reconfiguration progress. It is sent software packet as a request packet across a virtual circuit network from the source (CH) to the destination (S_{FI}). The time taken for configuration/programming the node interface and transferring the routing information is independent of the packet length. The sequence of the self-configuration of S_{FI} was implemented as:

1. Sensor S_{FI} receives a configure notification from CH.
2. S_{FI} knows that there is a software packet going from CH that comes out through the communication port established among them.
3. S_{FI} opens the communication channels/sessions within the receiver module (R) for passing the packet, ensuring they remain open and functional while data is being transferred, and closing them when communication ends.
4. The receiver module (R) listens to the request packet through its amplifier.
5. R dispatches the setup request packet to the processor unit (P).

6. P takes delivery of the setup request packet and then installs the software packet based on the forthcoming basic configuration.

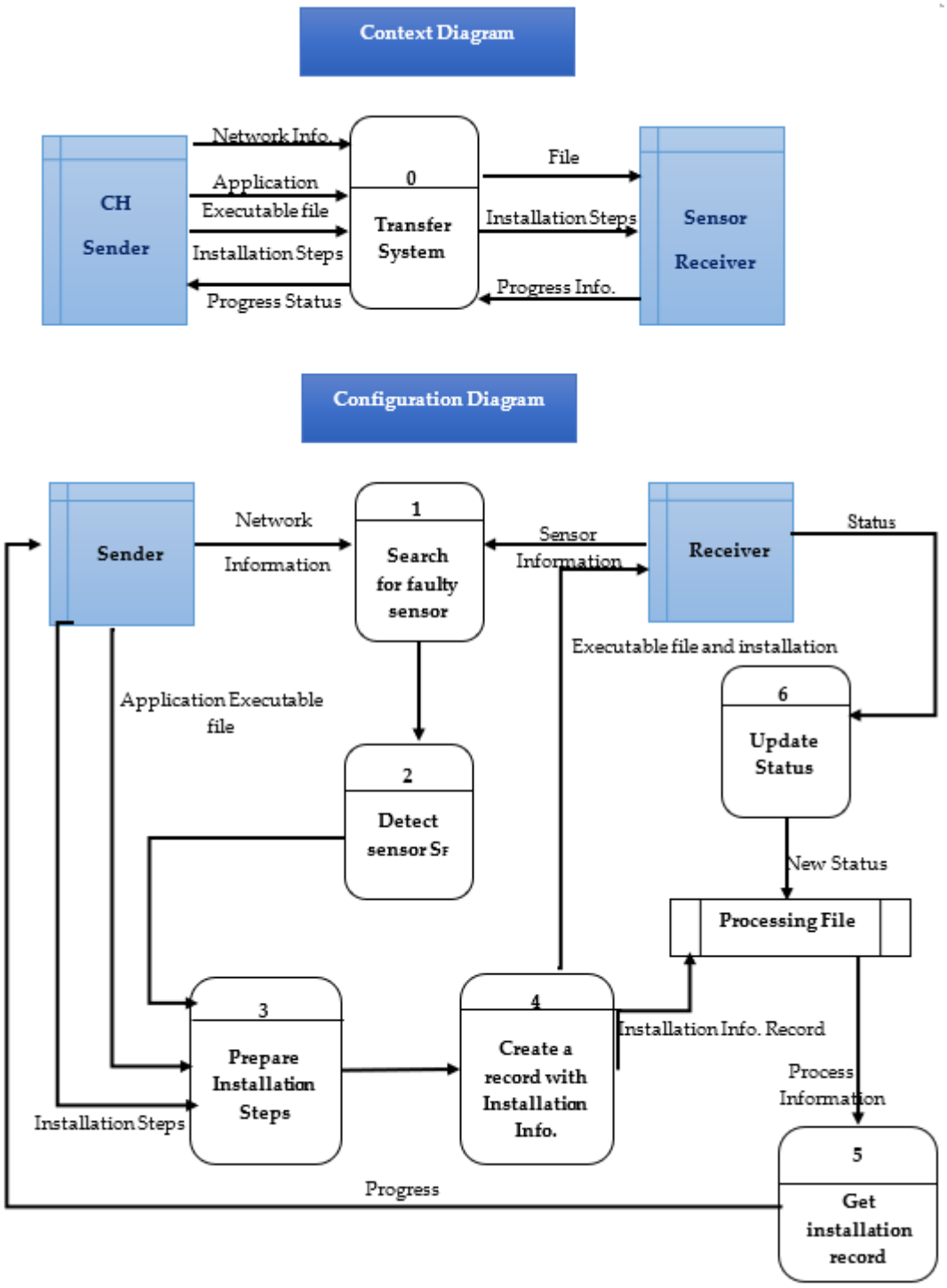


Figure 1. Serial steps of the distributed configuration model for the proposed SCM technique.

The major computational module, which is a programmable component that provides calculation and storage for the conveyor packets in the system, is formed by the processor of a sensor node in the arrangement. After receiving the sent signal and processing it using straightforward translations based on calibration data and sensory filters, the computational module sends the data to the application. The correlation techniques used in data fusion, which combine the software packets received from the chief sensor (CH)

with its fundamental operational software to build a single software package within the processing module (data fusion) for fault tolerance, are examples of correlation techniques that can also be used in sensor processing. With the use of data fusion, a network can be autonomously maintained and configured to offer coverage based on energy needs.

Since the emitter software packet requires configuration, it was assumed that IE is the input emitter packet at the instant, and any changes to the input packet will create a corresponding change in the collector configure packet in the period, T_C. Thus, the current transfer rate gain, R_i, is given as T_{OUT}/T_{IN} and determined by the same formula T_C/T_E. Alpha (α), a configuration factor in the process, was initiated to acquire the initial configuration period of the package (CPi), with typical values of alpha falling between 0.980 and 0.995. The formula for calculating CPi is CPi = α (TC + TE). After the configuration process, the packages configured inside the processing unit may change, necessitating the stabilization and installation operations. To obtain the created stable configuration period (CP_d), CPi has therefore evolved across the stabilization factor (RC), making it a function of RC = α * TC. Therefore, the value of CP_d is equal to CPi + R_C. Therefore, any change in T_E will also result in a change in Tc.

The common configuration model is adopted on analysis of these bases. First, the alpha factor is inferred by:

$$\propto = \frac{T_C}{T_E}, \text{ also } \quad \alpha = \frac{\beta}{1-\beta} \tag{10}$$

Thus

$$\beta = \frac{\alpha}{1-\alpha} \tag{11}$$

Then,

$$T_C = \propto \times T_E \tag{12}$$

Hence,

$$R_i = \frac{T_{OUT}}{T_{IN}} = \frac{T_C}{T_E} \cong \frac{T_C \times RC}{T_E \times RC} \cong \frac{\beta}{\beta+1} \cong 1 \tag{13}$$

By substituting, we obtain

$$R_i = \propto \frac{RC}{RC} \cong 1 \tag{14}$$

4. Simulation and Analytical Results

Based on the IRIS (international resource identifier system) specifications for wireless sensor networks (WSNs), we conducted a simulation to test our proposed network hypotheses. The simulation was executed using the IRIS motes board (radio board) values described in [14] during deployments. Our proposed work is an extension of our previous studies [15,16]. The analytic simulation involved several phases, which are detailed below.

4.1. Communication-Initialization Phase

It is expected that the sensor network has been properly configured. Each sensor node is aware of its group head (CH) and nearby Negi nodes, as well as their IDs. CH stores this information in the status table (STi). During the initialization step, it is also assumed that all sensor nodes are error-free. During this phase, the sensor node sends the locally measured physical value Xi to the group leader (CH), who collects data from the group members for defect diagnostics. It was broken into deployments according to the number of rounds. Each round has three cycles: the authentication cycle, the dispatch cycle, and the diagnosis cycle. Each CH in the planned network receives welcoming joined messages and monitoring information from the elements of its cluster regularly. Following that, CH begins analyzing the incoming data packets and is capable of detecting the state of each sensor within its cluster. CH can establish the status table (STi) for the cluster individuals through analyses, and then declare STi to base station (BS) to prepare the maintenance process in which faulty deployed nodes are configured by automatic installation procedures to obtain the basic configuration for system operation. In our simulation, we used relational

datasets from IoT devices to collect temperature values throughout data radio transfer. In our simulation, sensor nodes are randomly inserted and distributed over a square area of 220 m × 220 m. Table 2 illustrates the measures of concluded parameters from this simulation. At the beginning of each round, the CH collects heartbeat messages from all nodes of the cluster to confirm joining its cluster.

Table 2. The results were revealed by applying SCM deployments.

Parameter	Value
Deploy Area	220 × 220 m^2
Initial Energy	2.0 J
Broadcast Time (Between Succession Packets)	300 µs.
Packet Size	256 byte
Size of Tiny OS replica	400 bytes
The Ratio of Dispatch Energy Cost E_T	≅60% of the battery size
The Ratio of Receipt Energy Cost E_R	≅30% of battery size
The Ratio of Dissipated Energy Cost E_i	≅10% of battery size
Heartbeat Message	"00000000"

The table illustrates the estimated deployment costs from our constructed WSN using the proposed SCM approach. Through experiments, we eventually incorporated 100 s to 1000 s of autonomous sensor nodes that were spatially distributed through our planned WSN. It used an NS3 simulator to plan twenty clusters. Temperatures can be detected, treated, and sent via an RF emitter on the network by a sensor node within each cluster. The sensor readings chosen for implementation, as well as the temperatures recorded in various rooms of residences in one of Germany's geographical districts. Due to a lack of additional power, the sensors relied on the limited battery and approximated the peak voltage by 2 joules. The functioning radio, the status of the sensor, the distance between the nodes, and the elapsed diffusion time all have a substantial impact on the energy consumption of the sensor node as a result of the implementation. During communication, the most power is consumed. It is worth noting that the Tiny operating system (OS) automates the proposed network pieces. Tiny OS is a prominent platform for sensor network research due to its ability to successfully address system difficulties. It is used by over a hundred organizations throughout the world and supports a wide range of research topics and applications. Tiny OS was chosen because it is a versatile, application-specific operating system for sensor networks. It also allows for the execution of concurrent programs with reduced memory needs.

In practice, we considered Tiny OS to be a tiny system with fewer than 400 bytes that fits within 16 KB of memory space. Our fundamental operating system is under 400 bytes in size, making it highly efficient and low-power in all duty cycles. Our experiences with Tiny OS have been positive, and we have optimized sensor network apps and offered advances in coverage speed and repair. In our implementation, the Tiny OS component method worked well, and we reflect on our practical experiences. Table 3 records the selected samples from STi responses of various infected clusters with system faults throughout numerous deployment rounds.

Table 3 shows the two types of system errors that are taken into account. The first is the fault-stop (in which the node fails and stops running) induced in the previous rounds to return a result. S4 demonstrated this in R15. During deployments, we noted that S4 did not react with the result from the previous round R14, and this occurred again in R15. Another sort of failure is an arbitrary-node failure, which includes a variety of failure types. S4 in R10 made one response with an incorrect outcome. The second option is for S2 in R15 to respond with a deliberately deceptive response. Another option is to respond differently to distinct members of the cluster, as seen in S4 in R5. In R10, S2 is afflicted by a failure to synchronously respond.

Two approaches can be used to detect a system problem caused in the transmitted data during transmission. The first is a request from the suspicious sensor to resend the data within a preset time (t = 300 µs.) (head request query, HRQ). The second procedure

involves comparing some nearby pieces of data to the original data from the sensor's transmitter. Comparing bits of data will detect the functional condition of the suspected sensor, provided that the distance parameter was obtained at short distances (D) < 50 m, and assuming that it is more effective in the reorganization of system malfunction. Throughout the communication mode, we computed relative connectivity (RC) via each operational round to ensure the network's communication strength; RC was determined using the following Equation:

$$RC = \frac{number\ of\ suspected\ nodes\ in\ the\ round\ (Offbeat)}{total\ number\ of\ lively\ nodes\ in\ the\ round} - 1(\%). \quad (15)$$

The details of fault-detect confirmation will be taken in the next system fault tolerance section.

Table 3. Illustrated practical results recorded through communication for different clusters at three diverse rounds.

Round #	Cluster #	Mac Address	Heartbeat Data	Elapsed Diffusion Time (μ)	Neighboring Data	D (m^2)	Status	E *
R5	C3	192.168.3.1	"10000000"(29)	300	(24,41,31)	25	Traffic	1.0
		192.168.3.2	"11000000"		(29,24,41,31)		CH	1.66
		192.168.3.3	"11100000"(24)	260	(29,41,31)	30	Traffic	1.2
		192.168.3.4	"11110000"(41)	300	(29,24,31)	40	Offbeat	1.0
		192.168.3.5	"11111000"(31)	99	(29,24,41)	15	Traffic	1.48
RC				80%				
R10	C12	192.168.12.1	"10000000"(29)	298	(32,30,18)	50	Traffic	0.77
		192.168.12.2	"11000000"(32)	430	(29,30,18)	35	offbeat	0.90
		192.168.12.3	"11100000"(30)	287	(29,32,18)	45	Traffic	0.84
		192.168.12.4	"11110000"(18)	296	(29,32,30)	10	offbeat	0.92
		192.168.12.5	"11111000"		(29,32,30,18)		CH	1.0
RC				60%				
R15	C20	192.168.20.1	"10000000"(35)		(35,29,41,38,37)		CH	0.73
		192.168.20.2	"11000000"(29,41)	264, 299	(35,38,37)	40	Offbeat	0.52
		192.168.20.3	"11100000"(38)	274	(35,29,41,37)	30	Traffic	0.55
		192.168.20.4	"11110000"(-)		(35,29,41,38,37)	20	Fail-Stop	0.62
		192.168.20.5	"11111000"(37)	182	(35,29,41,38)	40	Traffic	0.67
RC				60%				

* Energy Level.

4.2. System Fault Correction-Management

System fault correction management (SFCM) is a self-alert algorithm that is divided into two parts: autonomous detection and configuration. The SFCM operated well in asynchronous systems (where there is no upper limit on when a response to a request will be established). We optimized it to cut down on overhead. Its goal was to fix several difficulties linked with system malfunctions and to correct existing solutions.

4.2.1. Fault Detection Method

We used the defect detection method based on alert theories during this period. Initially, each head node Si essentially discovers the partial defect (responding with an inaccurate result was rank1) by comparing the received data from the clustering nodes Ci with the data from its neighbor sensors S_{Negi}. The detection of errors is based on binary hypothesis testing. Let us consider the binary decision (0 or 1) made by the sensor node Si

on the Sj adjacent node. If high = 0, the head node determines that the sensor produced an inaccurate result and declares a system error. Otherwise, the hypothesis hij = 1 is decided on. Following that, it is applied to =Algorithm 1 to determine if the problematic node is present in the cluster or not, and then each head node CH can estimate the mean of the healthy nodes SH received from them.

We divided the preceding phase into two practical parts to optimize the computation involved in faultfinding. The first stage was to use the neighbors to detect malfunctioning sensor nodes. If the node did not locate a defect, it proceeds to the second stage of searching for an identical faulty node in the planned network using a detection alarm. This was discovered alarmingly during the suggested network programming known as a system error detection alarm (SEDA). We employed the alarm system to monitor network traffic for sent packets and send a quick alert to CH if it exceeded a specified period. This diagnosed rank 2 is the response with two different timings of the special broadcast round.

In addition, the system monitored a network on a regular basis for repeated dispatch operations within the same time slot in distribution. This was classified as a forbidden activity with a ranking of 3 or produced an incorrect result. We documented each forbidden behavior centrally to the CH by warning it twice with alert beats using the SEDA system. Otherwise, the sensor node gives all clustering nodes Ci, which are most likely fault-free. Table 4 examined the observed problems in the proposed network utilizing the alarm system during dissemination cycle 22.

Table 4. The detection-chain mechanism of the number alarm: alert status; heartbeat status is sent for the cluster head; node status.

Alarm	Incoming Reading	Inward Passed Time	Mac Address	Status	*CH Alert*	Heartbeat Rank
Alarm ON	29	257	192.168.16.1	Normal		
Alarm ON	29	253	192.168.3.2	Normal		
Alarm ON	41	255	192.168.5.3	Ab Normal	0	11100000-5
Alarm ON	38	255	192.168.3.2	Normal	000-000	
Alarm ON	29	253	192.168.3.5	Normal		
Alarm ON	42	252	192.168.10.1	Ab Normal	0	10000000-10
Alarm ON	30	317	192.168.12.2	Ab Normal	000	11000000-12
Alarm ON	30	316	192.168.11.3	Ab Normal	000	11100000-11
Alarm ON	29	316	192.168.9.4	Ab Normal	000	11110000-9
Alarm ON	29	214	192.168.2.2	Normal		
Alarm ON	42	314	192.168.20.1	Ab Normal	000	10000000-20
Alarm ON	42	312	192.168.2.2	Ab Normal	000-000	11000000-2
Alarm ON	29	246	192.168.16.3	Normal		
Alarm ON	29	257	192.168.20.4	Normal		
Alarm ON	29	255	192.168.14.5	Normal		

Table 4 lists a sample of the alarm system's reactions during the implementation of deployment round No. 22. The data clarify that the system transferred alert responses to the cluster head, to avoid receiving such misleading readings/system faults from the sensor elements in the cluster. The alert system acts as a check circuit that checks all outgoing readings from the network elements throughout the duty cycle in every round. It organizes the traffic of the readings to each CH in the designated network. Additionally, the alarm catches the temperature-encapsulated data packets corresponding to a predefined default time (t = 300 μ) coming from all network elements in that round. Then, it sorts normal nodes and abnormal nodes according to the difference formula:

$$S_i - S_j = \{N \mid N \in S_i \text{ and } S_j \notin s\} \quad (16)$$

where S_i and S_j must be compatible and have the same cluster-parity. Behind that, the alert system operates as a clock measuring the elapsed access time compared to t. If the elapsed inward time exceeds t and has happened at elapsed times 317, 316, 316, 314; it considers the system fault and codded it as rank 2. Whereas, if the inward time repeats in the sending

period or other epochs, it logs the fault in rank 3 and sends an alert with code "000-000". The head node receives the S_F alert status and heartbeat status to prepare the configuration model for these detected sensors and complete the software fault tolerance measurements. In this phase, all alert messages were realized in SCM and tolerated up to 98.86% of the injected system faults.

4.2.2. Fault Configuration Method

The CH attempts to supply software replication for system failure states infecting the surprise operation of active nodes in the system during this period. In a configuration-enabled distributed system, nodes are arranged sequentially, with one node serving as the head (or leader node) and the rest serving as secondary (or clustering nodes). It is important to note that any eligible node in the system can become primary by switching from secondary to primary (often when a primary node fails). The goal is for the leading node to use the emitter packet's gain to assist all honest nodes in achieving state-aware coverage. A practical system configuration model was used, as long as the maximum number of failed nodes is less than or equal to one-third of the total number of nodes in the system. The system gains confidence as the number of nodes grows. We divided the configuration model's rounds into five phases.

1. The alarm system sends an alert message M1 enclosed by heartbeat status within each round to the cluster head (primary node); one message exchange (M1).
2. The primary (leading) node transmits the heartbeat status report M1 and a software delivery request (replicas) M2 to the receiving node (Sink node); two message exchange (M1 + M2).
3. The sink node accepts the sent service request and then sends replicas in a prepared message to each other, including the primary nodes (cluster head); and message exchange (M1 × M2).
4. The primary nodes forward all replicas and direct commit messages M3 to each faulty node; message exchange ((M1 × M2) + M3).
5. All nodes send a reply message M4 to their cluster head node to confirm their configuration of them. The head sends a memo to the sink after diffusion and fixes rounds periodically for message exchange (M4 + 1).

We completed the request under the condition that each cluster head returns a "k + 1" answer, confirming that system sick nodes are at least one-third of the system's nodes, where k is the maximum allowable number of unhealthy nodes.

$$Total_{\text{ message exchange}} = \text{M1} + (\text{M1} + \text{M2}) + (\text{M1} \times \text{M2}) + ((\text{M1} \times \text{M2}) + \text{M3}) + (\text{M4} + 1) \quad (17)$$

In practice, it was testing the aforementioned condition, and the software package transfer process began to execute the required configuration for the detected system fault sensors, as indicated in the previous step. At this point, we attempted to provide the reader with a computation of the elapsed configuration period, which includes the emitter of the software, removal, and change times, as well as periods of initial configuration and installation of the software package, via epochs. Table 5 summarizes the head's memo in one of the duty cycles and the time spent (in seconds) in the configuration process, which evaluates the complementation of the fault tolerance technique to configure the discovered Byzantine sensors in the previously mentioned round.

Table 5 shows how successful each cluster's head is at covering the amount of detected defective nodes and reconfiguring during each duty cycle in a short time. When the head received "k + 1" replies from faulty nodes in the network, it satisfied the CH request. Similarly, we discovered that it survived numerous defective nodes over the service cycle. The head node is changed at each cycle of the practical implementation. If a preset period has passed without the head node broadcasting the memo message, it can be replaced with another, as seen in Figure 2. If necessary, a majority of clustering nodes can vote on the legitimacy of the current leading node and replace it with the cluster's next leading node.

Table 5. Summary of primary nodes' memos through the duty cycle for a number of the participating nodes.

Address	S_f Heartbeat	T_C	TE	CP_i	CP_d	Memo	Status
192.168.16.1						Talk	Normal
192.168.3.2						Talk	Normal
192.168.5.3	11100000	170	84	249	415.60	*M4*	Configured
192.168.3.5						Talk	Normal
192.168.10.1	10000000	101	98	195	293.98	*M4*	Configured
192.168.12.2	11000000	257	79	329	580.86	*M4*	Configured
192.168.11.3	11100000	182	56	233	411.36	*M4*	Configured
192.168.9.4	11110000	162	39	250	408.71	*M4*	Configured
192.168.17.2						Talk	Normal
192.168.20.1	10000000	200	66	261	457	*M4*	Configured
192.168.2.2	11000000	128	95	219	344.44	*M4*	Configured
192.168.16.3						Talk	Normal
192.168.20.4						Talk	Normal
192.168.14.5						Talk	Normal

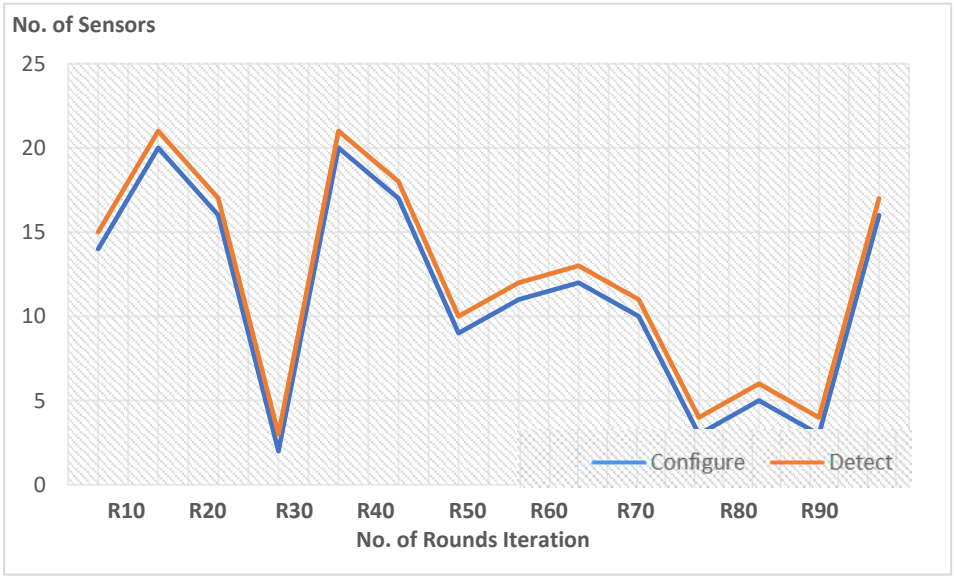


Figure 2. The SCM fault tolerance mechanism paralleling in actions of detection and configuring for the infected sensors.

During that phase, we attempted to calculate an efficient fault tolerance ratio in the operational round by estimating the response haste of each configured sensor Si in the message exchange in all models, based on the number of sensor nodes N present in the network and the number of times a sensor node needs to broadcast the message. $RT_i = \frac{S_{fi}}{N}.(S_f \sum_{i=1}^{Sf} Cp_d)$. In the proposed configuration model, we achieved efficient fault tolerance, which was estimated at 93.2%.

Figure 2 demonstrated that the results of experimental trials of detection and configuration operations were equalizing. This demonstrates that the progress timeline is parallel. During the defect detection phase, we counted the number of false alarm messages generated by the implementation. Detection accuracy (DA) was calculated during deployments by dividing the total number of messages sent by the average degree of malfunctioning sensors that emitted the fraudulent data (Xi). Throughout numerous deployments, we achieved a detection accuracy (DA) of 98.8%. DAi was given by,

$$DA = \frac{\text{number of the alarm messages outgoing nodes that were injected in the round}}{\text{participated nodes in the network through the round}}\% \quad (18)$$

Following simulation experiments, MATLAB R2019b was utilized to evaluate configuration parameters implemented by experiments. We evaluated the proposed network's efficiency in delivering data and verifying the network for defects. So, it was giving the specified networks a vast volume of temperature measurements from inner and outside sensors with varying ranges, examining whether or not alert messages appeared, and its ability to correct them. The study included configuration characteristics such as the distance between the sender head node and the receiving node, packet elapsed arrival time, packet size, transmitted node power level, detection latency, received signal strength, and successful/failed packet delivery. As shown in Figure 3, we implemented many datasets ranging in size from 16 to 1024 bits.

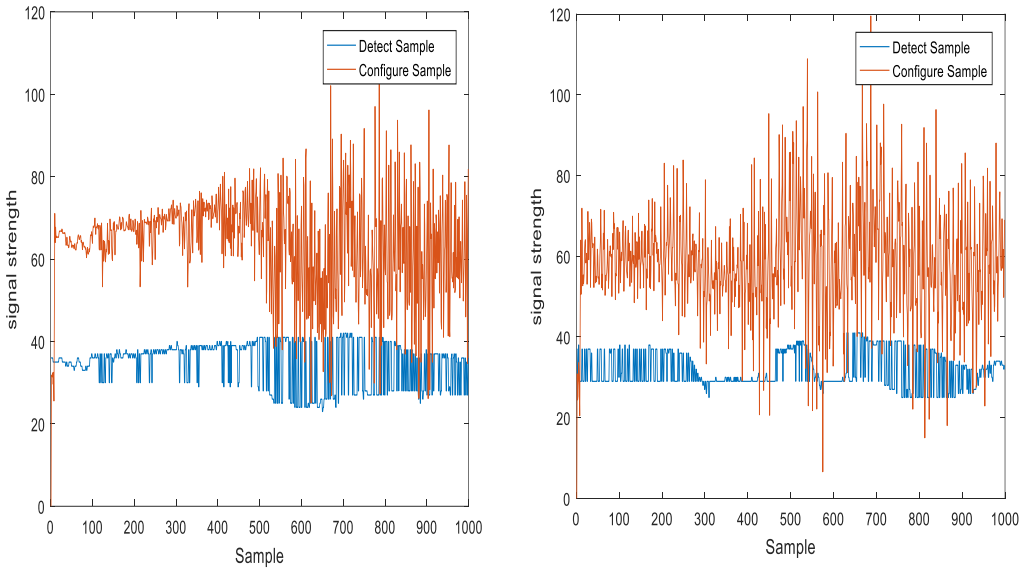


Figure 3. Illustrated are two plots of the received signal strength picking through different dataset samples to detect and configure, they are listed as (**a**) demonstration of the gained signal strength yielded after fault-configuration progress for various data samples injected in the radio and detection phase; (**b**) another plot is showed the synchronized coverage, and the produced signal strength of the faulty sensors by collaborating among the two phases throughout the synchronic implementation of the serial SCM model.

Figure 3 depicts two charts describing signal strength in a large number of trials. Through the configuration technique, we obtained a high-bandwidth signal strength in the domain of the reaction speed term, which surpassed the signal strength generated by the phase of communication and error identification. We obtained excellent results for the values of the given configuration network parameters when compared to a comparable technique that investigated the configuration parameters on a similar sensor network [17].

We have calculated the costs of the transmission of the message through implementation, which is defined as the total time required to transfer a message over a network to the target destination comprising:

- Startup time (t): the time spent sending and getting nodes;
- Transfer time per bit (t_b): This period includes all overhead costs that are determined by message length, error checking, and correction;
- Transit time per hop (t_h): This period includes parameters like detection latency and network transmission delays.

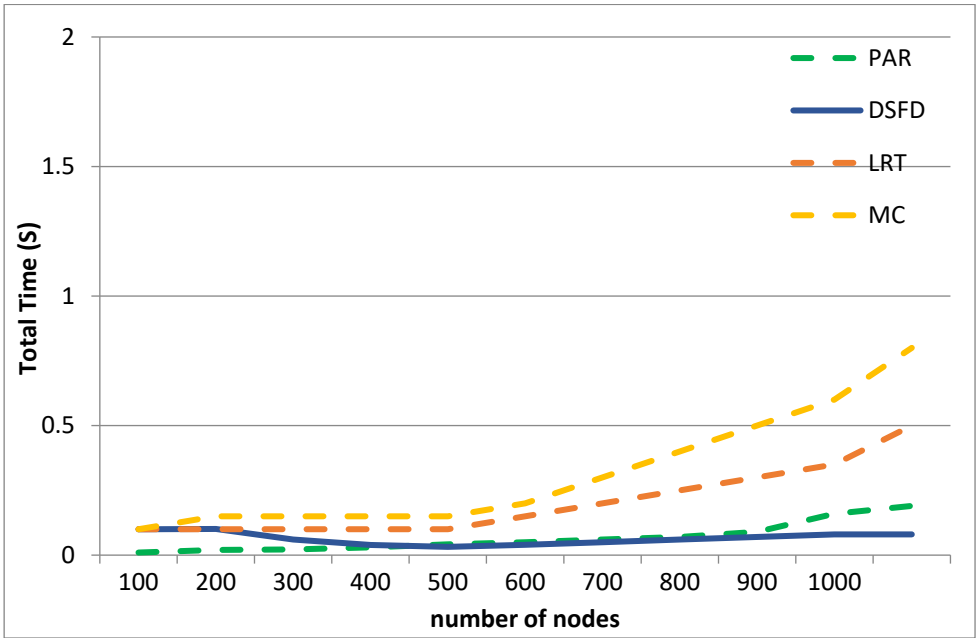
In terms of scalability and time to finish the algorithms, our suggested DSFD algorithm was determined to be more efficient than existing algorithms. The suggested DSFD technique needed fewer messages to be transferred among nodes than current algorithms that rely on heavy communication. The time required to finish the algorithm, on the other hand, varied, possibly due to network connections. The experimental results demonstrated that the position of the sink deployment could alter the overall network's data aggregation time. The placements of the sensor nodes, CHs, and sink, as well as the distances between them, will all result in variable data aggregation and transmission durations in a clustered network. Figure 4 demonstrates that the overall network environment's deployment strategy of CHs and sensor nodes produces the best results in terms of messages exchanged synchronously, known as the packet delivery ratio. This metric compares the number of transmitted packets to the number of received packets, as well as the node energy consumption and data transmission time. The research revealed that as the number of participating nodes increased, the amount of messages exchanged in DSFD decreased. When compared to the PAR and LRT approaches, the overall time required to execute the DSFD algorithm decreased with increasing node count, with an average reduction of 0.24 µs [10,11]. By expanding the number of sensors in the network, LRT and PAR increased exchange time/delay. DSFD began with 100 messages and gradually increased to 2000 messages for 150 nodes in terms of message exchange. The findings also reveal that when the injected error (E) is more than or equal to 0.25, the test rule achieves 100% accuracy, as shown in Figure 4a. We calculated the overall cost of communicating across the network to send m-bit messages.

$$t_{comm} = \sum_{i=1}^{m} t_s,\ (mt_b + t_h)l \tag{19}$$

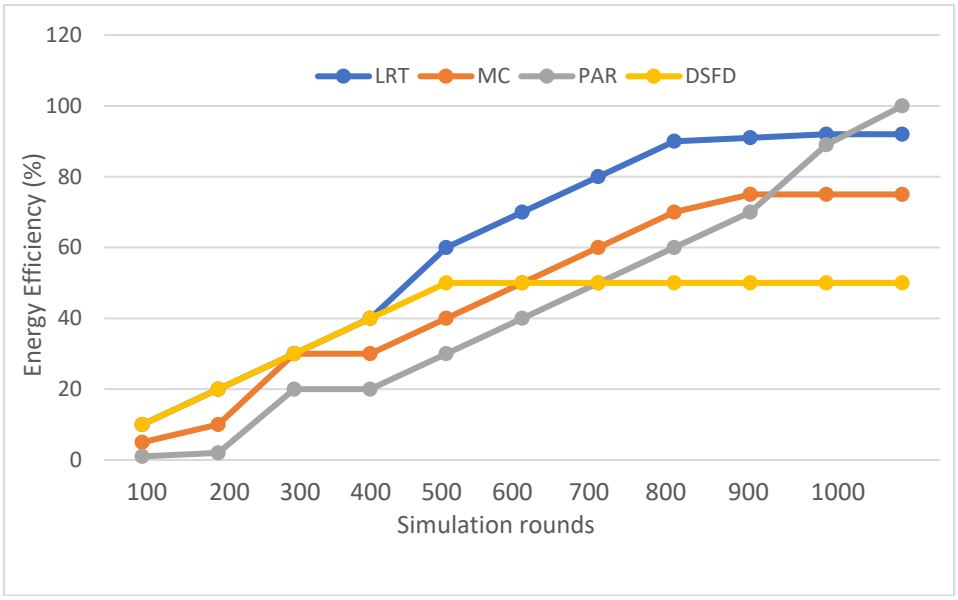
Figure 4 depicts two charts that demonstrate how, when a sensor network is operational, the overall traffic of exchange messages increases as the number of sensors inside the network increases. As a result, the delay time grows dramatically with the number of rounds. The study also discovered that the number of messages transferred between nodes had a quick effect on the total energy consumption of the nodes. Otherwise, the study discovered that the amount of messages exchanged between nodes falls when the number of participants' sensors diminishes. In practice, the proposed method's overall energy usage tends to be steady after 1000 rounds. It was discovered through studies that changes in the number of surviving nodes, after 1000 rounds of experiments, have no meaningful impact on how much energy is consumed. Where it is noticed that the transfer energy climbs just slightly in the proposed method, energy efficiency reaches 50% of the network's total energy. In contrast, it increased greatly when compared to other algorithms LRT [10] and PAR [11], as shown in Figure 4b. This means that, in the simulated experiment, the proposed technique can maintain a higher node survival rate. The results obtained in this section of the study demonstrate that the message exchange rates of other algorithms increased exponentially, resulting in significant energy loss. The DSFD method, on the other hand, has a fixed linear growth rate. The proposed method has much-improved performance in terms of node energy consumption and survival time, which can effectively extend the overall network's working performance. This is owing to fact that adopting a fault-tolerance technique and deploying the sensed data in a distributed manner is an adequate and well-organized mechanism for transfers and message exchange. This points out that the proposed fault detection methodology has significantly lower computational complexity and processing time than other methods. Furthermore, the study established a relationship between message exchange parameter efficiency and lower network energy consumption, which improves network performance by 96.4% and increases network lifetime.

The suggested SCM was put through its paces on a variety of system executions with an infinite number of network problems. During testing, the proposed DSFD reported dropped messages, increased Byzantine process problems, and omitted and damaged messages. The average degree of sensor nodes in the network influenced detection accuracy, while the total number of messages required varied based on the methods used. The overall

number of messages exchanged by all of the proposed methods was influenced by the network's number of sensor nodes as well as the energy needed by the sensors to send messages. To improve the system's efficiency, DSFD uses a feedback mechanism and tasks supervisory nodes (CHs) within each cluster to ensure the reliability of the primary node.



(**a**)



(**b**)

Figure 4. Comparison of total time and message exchange by varying average degrees for the DSFD, LART, PAR, and MC algorithms. (**a**) Total time; (**b**) energy consumed by transfer.

5. Conclusions

In this study, we proposed self-configuration management for the correction of system faults that are influencing the effectiveness of performance in a WSN's operating system platform. To account for poor detector faults, we suggested a self-sensible distributed fault identification technique. It put its abilities to the test by injecting random spots into the chosen sensor networks in order to detect the aforementioned aberrant circumstances. In each round, the cluster head collects data from the clustering sensors and then diagnoses

its functional condition using the suggested DSFD algorithm. The research work in this phase resulted in practice characteristics such as detection accuracy and false alarm rate. The suggested DSFD technique detected the malfunctioning software sensors with over 98.8% detection accuracy, converging the proportion of passing-alarm message accuracy, while maintaining a minimal (at most 2%) false alarm rate. There was a 2% improvement in detection accuracy. Following that, we created configuration policies in response to the system answers. These policies were used to configure recognized defective nodes that are part of the planned network. The IoT service was used to transfer all software replicas by primary nodes and direct commit messages M3 to each defective node for error configuration.

The reaction assessments observed efficiently and automatically respond to a wide range of system problems on the network over brief epochs. The fault configuration management achieved a 93.2% efficiency in correcting the software within the limited time frame estimated in seconds. To achieve optimal network setup and improve WSN performance, configuration parameters were created, to refine the proposed network performance and increase indications of WSN lifespan prolongation. Based on the above parameters, the actual network lifetime was calculated by subtracting the entire configuration length from the operational communication duration. The overall network lifetime is estimated by adding the period after delivering coverage messages to the continuous running time of the network prior to the warning message export, which indicates the data delivery ratio going below its predefined threshold. To ensure the success of the trials in this part, we set both criteria at 90.

Author Contributions: Conceptualization W.M.E. and E.E.-s.; methodology, M.E.; validation, M.K.H.; formal analysis, W.M.E.; writing—original draft preparation, E.E.-s.; writing—review and editing, M.E. and W.M.E. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data sharing is not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are mentioned in the paper:

SCM	Self-Configuration Management
CPU	Central Processing Unit
WSN	Wireless Sensor Network
BFT	Byzantine Fault Tolerance
GPBFT	Global Practical Byzantine Fault Tolerance
IoT	Internet of Things
RT	Radio Transferring
PBFT	Practical Byzantine Fault Tolerance
APBFT	Adaptive Practical Byzantine Fault Tolerance
FCBFT	Feature Clustering Byzantine Fault Tolerance
MCS	Mobile Crowd Detection
RF	Radio Frequency
DSFD	Distributed System Fault Detection model
BS	Base Station
CH	Cluster Head
IRIS	International Resource Identifier System
HRQ	Head Request Query
SFCM	System Fault Correction-Management
SEDA	System Error Detection Alarm
DA	Detection Accuracy

References

1. Antony, S.N.F.M.A.; Bahari, M.F.A. Implementation of Elliptic Curves in the Polynomial Blom Key Pre-Distribution Scheme for Wireless Sensor Networks and Distributed Ledger Technology. *J. Sens. Actuator Netw.* **2023**, *12*, 15. [CrossRef]
2. Heinzelman, W.C.; Balakrishnan, H. An application-specific protocol architecture for wireless microsensor networks. *J. IEEE Trans. Wirel. Commun.* **2022**, *4*, 660–700. [CrossRef]
3. Lamport, L.; Shostak, R. The Byzantine Generals Problem. *J. ACM Trans. Program. Lang. Syst.* **2021**, *4*, 382–401. [CrossRef]
4. Li, X.Y.; Wan, P.J.; Frieder, O. Coverage in wireless ad hoc sensor networks. *J. IEEE Trans. Comput.* **2003**, *52*, 753–763. [CrossRef]
5. Xiaosheng, Y.J.Q.; Peng, C. GPBFT: A Practical Byzantine Fault-Tolerant Consensus Algorithm Based on Dual Administrator Short Group Signatures. *J. Secur. Commun. Netw.* **2022**, *2022*, 270–285.
6. Adday, S.; Ghaihab, H.; Subramaniam, K.; Zuriati, A.Z.; Normalia, S. Fault Tolerance Structures in Wireless Sensor Networks (WSNs): Survey, Classification, and Future Directions. *J. Sens.* **2022**, *22*, 6041. [CrossRef] [PubMed]
7. Chang, J.; Liu, F. A Byzantine Sensing Network Based on Majority-Consensus Data Aggregation Mechanism. *J. Sens.* **2022**, *21*, 248. [CrossRef] [PubMed]
8. Cotroneo, D.; Simone, L.D.; Natella, R. ThorFI: A Novel Approach for Network Fault Injection as a Service. *J. Netw. Comput. Appl. Bp. Hung.* **2022**, *201*, 10334. [CrossRef]
9. Jin, H.; Nengroo, S.H.; Kim, I.; Har, D. Special Issue on Advanced Wireless Sensor Networks for Emerging Applications. *Appl. Sci.* **2022**, *12*, 7315. [CrossRef]
10. Gouda, B.S.; Panda, M.; Panigrahi, T. Distributed Intermittent Fault Diagnosis in Wireless Sensor Network Using Likelihood Ratio Test. *IEEE Access* **2023**, *11*, 6958–6972. [CrossRef]
11. Goud, B.H.; Anitha, R. RAFT to Improve Failure Recovery in Wireless Sensor Networks. *Int. J. Intell. Syst. Appl. Eng.* **2023**, *12*, 222–227.
12. Sariga, A.; Uthayakumar, J. Type 2 Fuzzy Logic based Unequal Clustering algorithm for multi-hop wireless sensor networks. *Int. J. Wirel. Ad Hoc Commun.* **2020**, *1*, 33–46. [CrossRef]
13. Taurshia, A.; Kathrine, J.W.; Venkatesan, V. Software Defined Network aided cluster key management system for secure fusion multicast communication in Internet of Vehicles. *Fusion Pract. Appl.* **2023**, *12*, 38–52. [CrossRef]
14. Crossbow Technology, Inc. IRIS-Wireless Measurement System. 2001. Available online: http://www.nr2.ufpr.br/~adc/documentos/iris_datasheet.pdf (accessed on 1 May 2023).
15. Ossama, H. Embarak, Raed Abu Zitar, Securing Wireless Sensor Networks Against DoS attacks in Industrial 4.0. *J. Intell. Syst. Internet Things* **2023**, *87*, 66–74. [CrossRef]
16. Mazloomi, N.; Gholipour, M.; Zaretalab, A. Efficient configuration for multi-objective QoS optimization in wireless sensor network. *J. Ad Hoc Netw.* **2022**, *125*, 399–423. [CrossRef]
17. Xing, X.W.G.; Zhang, Y.; Lu, C.; Pless, R.; Gill, C. Integrated Coverage and Connectivity Configuration in Wireless Sensor Networks. In Proceedings of the SenSys '03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, New York, NY, USA, 4–7 November 2018; pp. 28–39.