An Effective Fault Clustering Management Approach Based Self-Recovery Mechanism for Decentralized Wireless Sensor Networks

Walaa M. Elsayed
Faculty of Computers and Informatics, Damanhour University, Damanhour, Egypt

Abstract: *Wireless sensor networks (WSNs) have several uses and provide endless future possibilities Wireless sensor network nodes are prone to failure because of energy depletion, communication link difficulties, and malicious attacks. As a result, Self-recovery techniques are one of the most significant issues in WSNs. Error detection is the primary strategy in the Self-recovery mechanism in wireless sensor networks (WSNs), with each cluster head frequently checking the readings of its members. According to previous research, most comparing approaches will fail if more than half of a sensor's nearby nodes are incorrect. Furthermore, these comparing approaches cannot discover common mode failures. The recommended fault Self-recovery approach functions by comparing the pulse sequence number generated by clustering nodes and distributing the choice made about each node. This paper presents an approach which can both locate and recover malfunctioning nodes in sensor networks. The proposed model integrates the capabilities of isolating the defective cluster sensors, which cause WSN malfunctions, from the cluster cycling and advertising the new path coordinates for the base station (BS). The simulation results show that the proposed Effective Fault Clustering Management (EFCM) approach is very exact in locating malfunctioning nodes and very quick in establishing a cover free of such nodes. When using the NS3 simulator.*

Keywords: *WSNs, Clustering, Node failure, Self-discovery, Self-recovery.*

1. Introduction

WSN is an autonomous organizing network comprised of thousands of low-cost, low-power sensor nodes. Each sensor node has restricted functionalities such as processing, communication, and sensing. These devices can be employed for particular purposes, such as detection and reporting the occurrence of intriguing occurrences. The precision of individual node data is crucial in many applications. For instance, in a surveillance network, sensor readings must be accurate to minimize missed detections. As a result, All WSNs must meet the energy competence, scalability, and error tolerance specifications. Particular concerns must be addressed for WSN to continue to operate. 1) WSNs consisting of sensor nodes may be installed in unmanaged and possibly hostile locations, increasing the likelihood of node failure. 2 In contrast to wireless local area networks, the journey from source to destination in wireless sensor networks frequently includes many wireless links (hops). Wireless links between nodes are susceptible to wireless channel fading, resulting in channel errors. 3) Data from each sensor node is routed to the sink node [1]. To maintain effective bandwidth consumption, erroneous data generated by faulty sensor nodes must be removed from the network.

Sensor nodes are powered by batteries, which means they have limited power sources. Additionally, these nodes are placed in hard and hazardous locations, and the sensors are prone to failure. Faulty sensor nodes can result in faulty data sensing, inaccurate data processing, and inappropriate data communications [2]. Faults in WSN nodes arise when one or more of their hardware components fail. WSN node status is classified into two groups based on numerous faults: healthy and defective. A node is considered broken if its battery power exceeds the threshold limit, its microcontroller fails, or the transceiver's transmitter circuit malfunctions. If the transmitter circuit of a node fails, even if all other hardware components are in good working order, the node is classified as defective. The faulty node must be replaced with a fresh node. Otherwise, its responsibilities must be handled by another healthy node. The healthy node can be divided into three groups: traffic node, normal node, and end node. In a healthy node, the transceiver is operational but the sensing device is malfunctioning, we can utilize it as a traffic node. A traffic node can be used as a router in multi-hop wireless data transfer. The usual healthy node, which has all of the sensor nodes' components in good working order, can be used for any form of WSN job. The transceiver's receiver circuit malfunctions at the end node. As a result, it may sense monitoring field parameters and communicate data to the base station via another node. However, the end node cannot receive the data from any other node. Therefore, it cannot be used as a router in WSN.

We can broadly classify faults (shown in Fig. 1) that can impair sensor network performance into two types: hard faults (permanent or occupation defects) and systematic faults (transitory, devious, and recurrent faults) [3-8]. In the event of a function failure, the malfunctioning sensor nodes do not respond or send any reading to the other nodes. However, in a temporary glitch, the sensors are unable to conduct their expected operations for a short period, making Self-recovery problematic. Whereas intermittent

malfunctioning sensor nodes offer sometimes fault-free information, making it difficult for the fusion centre to conclude the rank of a sensor. Devious faulty sensors behave in unpredictable ways, sending different data at different times.

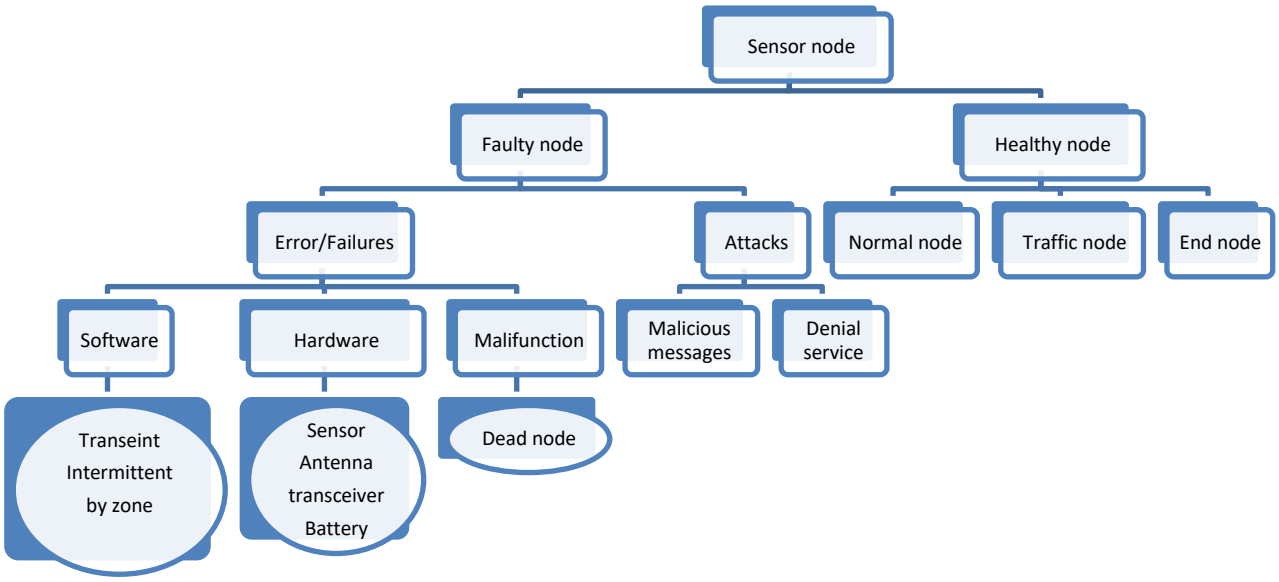

Fig.1. Fault classification in the clustering sensor nodes.

Failures in wireless sensor networks can be caused by a variety of factors, including node failures, link failures, design flaws, and implementation errors. Locating the causes of such failures is critical to ensuring the network's reliability, but it is typically a difficult task due to several factors, including the distributed nature of most protocols and applications, the energy constraints imposed on any technique, and the wide range of faults in such networks, which range from node crashes to bugs in the code running on the nodes. Furthermore, the complexity of software develops dramatically for its size. Large-scale software systems are particularly error-prone and fail frequently, especially for sensor network applications, which are intrinsically distributed. As a result, Self-recovery must be used to ensure that such systems provide the necessary level of functioning even when there are errors present. Because sensor nodes are susceptible to failure, Self-recovery should be actively addressed in many sensor network applications [9]. This paper presents a self-healing system that consists of three major stages: fault detection, Self-recovery and recovery. During the detection phase, each node generates a heartbeat message, which is then broadcast to its neighbours. Each fault-free and soft-faulty node responds to the heartbeat request within a specific timeframe. By the end of this stage, each node has a local view of the fault status of its one-hop neighbours. During the Self-recovery phase, each node broadcasts a heartbeat message regularly, and the job assigned to nodes is to increment the heartbeat sequence number. Finally, a covering phase covers each node that does not broadcast the heartbeat message and does not receive the heartbeat message from 1-hop neighbours; the coverage job is given to cover no mal-functioning nodes and isolate. We are listed as follows.

1. Investigate fault detection using a distributed technique.
2. Propose a Self-recovery technique that detects hardware failures with high accuracy while requiring little time, heartbeat messages, and energy overhead.
3. Our method is based on the assumption that a faulty node can be switched off if it is covered, causing the network to behave normally.

2. Related works

Fault detection techniques were utilized to detect prospective problems and locate the source of faults, which is useful for fault recovery operations. There have been numerous research accomplishments in recent years. In 2021, Khilar [10] proposed a revolutionary method for detecting nodes with both types of problems without relying on a specific sensor model. The suggested fault model is more accurate and involves less communication than the existing

methods. In the same year, Jurdak and Rosalind Wang presented anomaly detection in WSNs [11], which focused on data anomalies caused by security assaults and the statistical methodologies used to identify them. Because of their close connection to often hostile physical environments, WSNs and other networks utilized in extreme situations (such as space) are more likely than normal networks to have connectivity or hardware breakdown anomalies. The research also focuses on developing detection algorithms that target network and data-level anomalies. Banerjee et al. [12] suggested a strategy for distributed defect detection and sensor management in WSNs using cellular automata. Node failures are recognized in a distributed manner, and cellular automata are used for network management.

In 2022, Mahapatro [13] could be studied an online fault Self-recovery algorithm for wireless sensor networks. The researcher explicitly takes into account the possibility of faults in different sections of sensor networks and communication channels. The diagnostic local view is obtained by exploiting the spatially correlated sensor measurements. These local views are then disseminated using a spanning tree of cluster heads. In the same year, Camilo *et. al.* [14] provided an overview of existing post-deployment WSN diagnostic tools, by briefly presenting their functionality, architecture and constraints, to enable a basic understating of each tool. The survey also includes a multi-dimensional comparative analysis of the various tools, based on a proposed classification scheme and evaluation criteria, as well as an identification of the main open research issues. Mishra et al. [15] attempted to address mistakes and faults that arise for a variety of reasons, including hardware malfunction, software problems, environmental dangers, and so on. Thus, a sensor network should be fault-resistant to properly cope with these erroneous conditions. Duche *et. al.* [16] presented a new method to detect sensor node failure or malfunctioning in such an environment. The proposed method used the round trip delay (RTD) time to estimate the confidence factor of the RTD path. Based on the confidence factor the failed or malfunctioning sensor node is detected.

In 2023, Babaie *et. al.* [17] proposed the behaviours of the components of a sensor are independently analyzed using the proposed model based on Petri nets and the links of the sensor's components are investigated through the correlation graph. Saurabh et al. [18] considered this paper's major purpose to provide a comparative examination of fault detection strategies employing various approaches. Sensor nodes face varied energy and computational restrictions. To deliver excellent service using coverage standards, there is a need to design procedures for fault tolerance, event reporting, and maintaining energy efficiency. In 2024, Lau *et. al.* [19] suggested a revolutionary centralized hardware defect detection solution for a structured Wireless Sensor Network (WSN) using the Naïve Bayes paradigm. Various defects are widely classified in the following section; they infect the behaviour of wireless sensor networks, causing them to misbehave and affecting the network's functional performance.

Failure Classification

To understand the Self-recovery mechanism, we should distinguish between faults, errors, and failures. A fault is any type of imperfection that causes a mistake. An error indicates an improper (undefined) system state. Such a situation may fail. Failure is the (observable) manifestation of an error, which occurs when the system deviates from its specification and cannot perform its intended functionality. Several problems could cause faults in WSNs: a node could be moved to a different region, resulting in a link failure; nodes could lose power and stop responding to requests; or they could start sending arbitrary values, either intentionally (after a security breach) or due to a malfunction. The errors that a WSN may encounter will be classed as follows: crash, omission, timing, value, and arbitrary. These failures are observable manifestations of the underlying problems listed below: 1) Crash or omission: A failure by omission occurs when a service does not reply to requests consistently. For example, this could be due to radio interference, which causes periodic communication loss. A crash failure happens when the service stops responding to any requests. An omission degree f can be defined as a limit on the number of omission flaws a node can have before being classed as crashed. 2) Timing: Services may fail owing to a timeout in processing a request or providing data too early. Such timing faults occur when a node answers a request with the right value, but the response is received outside of the timing interval set by the application. Time failures will only happen if the application specifies time limitations. 3) Value: A service is regarded to have failed due to an inaccurate value if it sends a timely response but fails to deliver the value accurately. For example, a service that aggregates data from other nodes may transmit a result value to the base station that does not precisely represent the original data. Such issues could be caused by faulty software, hardware, corrupt communications, or even hostile nodes that generate erroneous data. 4) Arbitrary failures are those that cannot be grouped into the previously specified categories. 5) Byzantine: failures describe a type of

arbitrary failure that is in general caused by a malicious service that not only behaves erroneously but also fails to behave consistently when interacting with other services and applications. In sensor networks, an aggregation service could start sending both incorrect and correct values to the sink, or a node routing messages could not forward a message despite sending an acknowledgement back to the sender [20].

Self-recovery on Different Levels

Self-recovery is a new area of research that focuses on fault tolerance in dynamic systems. It deals with imperfect specifications, uncontrollable environments, and system reconfiguration based on their dynamics. The term "Self-recovery" refers to a software system's ability to study, identify, diagnose, and respond to system problems. Self-recovery components or programs must be able to detect system failures, evaluate external restrictions, and make appropriate modifications. Self-recovery categories of elements include fault model or fault hypothesis, system reaction, system completeness, and design context. A fault model of a Self-recovery system is to state what faults or injuries to be self-healed including fault duration, fault source such as operational errors, defective system requirements implementation errors etc. System response includes the aspects of fault detection, degree of degradation, fault response and an attempt to recover action or compensation for a fault. Fault detection approaches involved in a self-recovery system include the application system's semantics-driven assertions, supervisory checks, examining the computing answers, comparison of replicated components, online self-testing etc. The system completeness aspect deals with the reality of knowledge limits, and incompleteness in specifications and designs thereof. It also deals with the problem of system self-knowledge, system evolution etc. Handling the architectural incompleteness for example, of third-party components or various patches during or after system deployment is a challenging issue in developing a Self-recovery system. A fault is an anomaly that causes a node to malfunction and is caused by hardware or software issues at a single node and has nothing to do with connectivity with surrounding nodes. Anomalies are noise-related measurements resulting from a malfunctioning sensor. This condition could emerge as a result of weak or broken hardware components, or bad software integration of the components, but an event is defined as a specific thing that alters the real-world state, such as a forest fire, air pollution, etc. Thus, five levels of fault tolerance were discussed in [18]. The five levels are: physical, hardware, system software, middleware, and application. More specifically, we distinguish self-recovery in WSNs into four system levels. A self-recovery in a WSN system can occur at the hardware, software, network communication, and application levels.

A. Hardware-level

Faults at the hardware layer can arise when any hardware component of a sensor node fails, including memory, battery, CPU, sensing unit, and network interface (wireless radio). Sensor node hardware failure can be attributed to three major causes. The first is that sensor networks are typically used commercially, and sensor nodes are costly. As a result, the components used to create a sensor node may not necessarily be of the best quality. The second is that severe energy limits limit sensor nodes' long-term and dependable functioning. For instance, when a sensor's battery exceeds a particular value. Sensor measurements may become erroneous [21]. The third reason is that sensor networks are commonly implemented in harsh and dangerous environments, impacting the regular operation of sensor nodes. These environmental conditions significantly influence the wireless radios of sensor nodes.

B. Software level

A sensor node's software consists of two parts: system software, such as the operating system, and middleware, which comprises communication, routing, and aggregation. The ability to execute localized algorithms dispersed and simultaneously is an important feature of system software. Software faults are a common source of mistakes in WSNs. One feasible option is software variety, which involves delivering each program in several versions. Because it is difficult to provide fault tolerance affordably at the hardware level of a sensor node, numerous fault-tolerant solutions are anticipated at the middleware layer. The great majority of existing WSN applications are easy. To adapt to real-life applications, it is necessary to design considerably more complex middleware for WSNs.

C. Communication level

Errors at the network layer affect wireless communication links. Link faults in WSNs are often caused by the surrounding environment, assuming no hardware errors exist. Radio interference between sensor nodes can also cause links to fail. For example, if sensor A is in the interference range of other sensors transmitting messages simultaneously, sensor A will be unable to properly receive a message from sensor B. The conventional approach to improving wireless communication performance is to use aggressive error-

correcting algorithms and retransmissions. These two methods may cause additional delays in operation. It is worth noting that there is always a trade-off between fault tolerance and performance.

D. Application level

Fault tolerance can also be handled at the application level. For example, identifying many node-disjoint paths enhances routing fault tolerance. The system can transition from an unreachable path with broken links to an accessible candidate path. However, a fault tolerance strategy in one application cannot be used directly in another. It is necessary to address fault tolerance in various applications on an individual basis. In contrast, Fault tolerance at the level of an application can be utilized for tackling faults in nearly any type of resource [22].

3. The proposed Effective Fault Clustering Management (EFCM)

The Self-recovery mechanism acts primarily through a series of cycles. The first cycle is known as the monitoring cycle. During the monitoring cycle, the systems monitor will examine the computer environment for any inappropriate behaviour. After the monitor's inspections are completed, it will forward the data acquired from current observations to the next level. The second step of the cycle is known as error detection and Self-recovery; if Self-recovery reports that there is no defect in the system, it will return to the monitor for further observations. If the monitor detects an error, it will report it to the next stage of the cycle. The third stage of the cycle is known as the analysis and selection of a repair operation. At this stage, the fault is analyzed and a method of recovering is determined at this part of the cycle. After the repair recovery operation is determined, the report is passed onto the final phase of the cycle called execute repair and operation (self-regular). Any repairs that are needed are completed at this phase in the cycle. Once, the faulty areas are self-repaired the cycle begins all over again. Since this cycle is a closed loop, the process of Self-recovery environments will continuously heal itself as depicted in Fig. 2.

Fig. 2: The proposed Self-recovery System Process.

3.1. The Problem Definition

This section dealt with the designated network description and problem formulation, following certain necessary definitions.
Definition 1: A normal node is a healthy node that has all of its sensor node components in good working order and may be used for any type of job in WSN .
Definition 2: If a traffic node is healthy, the transceiver is functioning, but the sensing device fails, we can use it as a network traffic node. A traffic node can be used as a router in multi-hop wireless data transfer.
Definition 3: The end node is healthy, but the transceiver's reception circuit is malfunctioning. It may identify the parameters of the monitoring field and transfer data to the base station through another node. However, the end node cannot accept data from any other nodes. As a result, it cannot be used.
Definition 4: A bad node has hardware or software components that can cause faults within the network; we may remove this node and cover for it using neighbours.
Definition 5: A passing cover has no malfunctioning nodes. In contrast, a failed cover contains at least one defective node.

3.2. Network Scenario

The wireless sensor network includes several clusters managed by a sink node (fusion centre). The sink node is a higher level of the network and collects all the data generated in the network and propagates it to the back-end system. A cluster within a wireless sensor network contains *N* number of sensor nodes and one head cluster node, which are randomly distributed and deployed in a cluster located in two dimensional. Each sensor node in the cluster has an initial power source, a processing unit, memory, radio unit and sensors. The sensor nodes interact wirelessly and use the one-to-many broadcast primitive in their basic transmission mode. The transmission range of each sensor node is fixed. The transmission average degree of the network is determined by its transmission range. The data perceived by the sensor node is stored locally on its memory and distributes its detected data to its neighbours as well as the cluster head, which checks transferred data from cluster nodes and sends valuable data to the fusion centre regularly within a fixed time interval. Each node knows the distances separating it from each of its 1-hop neighbours and thus can know if it is covered or not when the failure occurs. Our proposed EFCM schema aims to detect the failure in any node within the cluster, and the Self-recovery whereabouts of failure exactly in any of the nodes in the cluster, according to the above parameters.

3.3. Problem formulation

The EFCM schema in our proposed model is based on the following assumptions :
1) As shown in Figure 2, we consider a WSN with N nodes distributed using a clustering model. Assume N = {X1, X2,... XN}
2) All sensor nodes are identical in structure and function, and their status is normal. This signifies that every node in the cluster is active; it has a heartbeat message and location information in memory.
3) A good sensor node can accurately relay the data it gathers to its neighbouring nodes and the cluster head. In contrast, a malfunctioning sensor node might receive correct data from its neighbours and the cluster head but may send out random or incorrect values due to issues with its functional or processing unit.
4) A synchronous mode of communication is used to deliver data from all sensor nodes to the cluster head at a set period.
5) There is only one cluster head with the properties of receiving data from all cluster nodes, checking received data, and delivering data to the sink node. It is deployed outside the network and has unlimited energy.
6) If a faulty transmitter node is represented as N1 = {x1, x2,... xa}, a faulty receiver node or traffic node is selected as N3 = {x1, x2,..., xh}, the active nodes in the network. are $N_{active} = \{N\text{-}N_D\}$. The dead vertices.
7) The energy (E) represents each node's starting energy, Tt bit data transmitted time, and Rt bit data received time, all approximated in microseconds. The energy consumption models are dependent on preceding parameters, as demonstrated by equations (1) and (2).

Consumption Transmit Energy (*CTE*)

$$CTE = E - (E * Tt) \qquad t1 \le t \le t2 \qquad (1)$$

Remaining energy after transmitting *(RET)*

$$RET = E\text{-}Tt \qquad t1 \le t \le t2 \qquad (2)$$

Consumption Receive Energy (*CRE*)

$$CRE = RET - (RET * Rt) \qquad t1 \le t \le t2 \qquad (3)$$

Loss of emission energy $E_d = 0$ (4)

Where we denote *CTE* as the energy consumption by transforming *l* bit data crossing a certain time, *RET* denotes the remaining energy after transmitting, and *CRE* as the energy consumption by receiving *l* bit data. *Ed* is denoted as the emission circuit's loss of energy. The power amplification loss is calculated by comparing the energy of the node after receiving data with the threshold. If its energy is less than a threshold, the node needs to be covered by neighbours. Previous fault tolerance techniques did not address the reduction of emission energy loss. Therefore, we proposed an

EFCM schema for the development of fault tolerance techniques, to include both solutions for fault detection and self-recovery problems and solutions of recovery for dead nodes or nodes that exceed threshold problems. Our proposed EFCM-based self-recovery mechanism has been able to increase the significant performance of WSNs.

3.4. EFCM methodology

The EFCM schema manages the activities of the proposed defective node detection model and the Self-recovery model, which is based on the Self-recovery mechanism in fault tolerance.

A. Detection Phase.

To provide any countermeasures, a system must first determine whether a specific functionality is or will be faulty. As a result, the cluster head estimates the network architecture that will be required during the fault self-recovery phase, to boost efficiency and improve performance statistics .

a) The initialization phase. The cluster head sends fixed data to all sensors in the cluster's network. Each sensor is supposed to transmit a heartbeat message to the cluster head, which verifies the node's condition to determine whether it is healthy or malfunctioning. Sensor nodes ($S_i \in S$) transmit and receive data from their neighbours (Negt (i)). Each sensor node has its fault status set to fault-free.
b) Computational phase. Each sensor receives information from its neighbouring sensor nodes. At time t = 300 μs [23], which is the interval between successive packets in the sensor node, it applies the mean operation to both the incoming data and its own sensed data. The computed mean is then sent back to the cluster head.

c) Detection phase. During this phase, the cluster head gets both the calculated mean and the heartbeat messages from each sensor node. Following that, it analyzes the acquired data to determine the function and identify malfunctioning sensor nodes. As a result, the cluster head extracts the sender Identifier from the received data to determine which sensor nodes are capable of sending data to the cluster. Sensor nodes that cannot send a heartbeat message to the cluster head within an estimated time t are considered to function faulty. Second, after identifying faulty sensor nodes, the nodes are diagnosed based on their status: alive or dead. The cluster head compares the sensor node data with its data for placing the data fault sensor nodes. If it matches, the cluster head determines that all sensor nodes in the cluster are alive and healthy, as determined by the mean of nearby sensor data. In contrast, if the cluster head does not receive any data from any sensor node inside the cluster within the predicted duration, the node might be considered dead. If the node's data is not matched then the cluster head believes that the sensor node present in the cluster is considered a defective node. Finally, the cluster head verifies heartbeat messages for all nodes throughout the predicted period t to ensure their operational state by determining which sensor node is unable to send heartbeat messages and so is considered a bad node. The cluster head then conveys information about the status of the node, whether it is alive or dead, as well as information about the operational status of the node, whether it is healthy or faulty, to all nodes in the cluster, so that the remaining nodes in the cluster do not send or receive data from this node. This allows for the maintenance of network quality. The cluster head sends a job to each broken sensor node for further Self-recovery.

B. Recovery Phase

The suggested Self-recovery model is based on sequence packets (transmit, receive) within a time interval estimated to be 300 μs for each sensor node in the cluster. Each node is responsible for detecting its operational status. Each node detects sensor node failure using the previously defined fault detection methodology. The node takes data from neighbouring nodes in the same cluster and compares it to its sensing data. The Self-recovery model is used to detect the operational status of the node's hardware components, which include the battery, microcontroller, sensor, transmitter circuit, and receiver circuit. The proposed Self-recovery model investigates failures of the aforesaid circuits based on the presence of their heartbeat messages for the cluster head node, Algorithm (1).

The primary objective of sensor utilization in applications is to avoid disasters and crimes by detecting anomalies. When an abnormality is detected, the user should receive a warning in real-time, which needs an event-driven data processing mechanism.

Algorithm (1): The clustering self-recover mechanism

Initialization phase

Input :- Negt (*i*) : set of single hop the neighbours of Si ;

- X: measures of *S values* by the transmission from sensor S*i* to its neighbour S*j* ;
- Y : measures of *S values* by the transmission from S*j* from sensor S*i* ;
- dij: difference between X and Y at time(t) ;
- dij=X-Y
- dij : measure difference rate between X and Y at a new time (t+1), from time t to t+1 :α
- dij =(X' –X) –(Y'-Y)α
- cij : test results between X and Y , cij € {0, 1}, cij = cji ;
- ɵ: predefined threshold value ;
- ɵ1=0.2, θ2=0.1 (suppose two values for comparing differences rate in data for every time within the transmission range between sensors);
- *Ti:* tendency value of a sensor, *Ti* € {LG, LF, GD, FT};
- *Ti* € "likely goody, likely fail, good, faulty"

Computation phase

Step 1: Each sensor (*Si*) tests every member of Sj € Negt (*i*) to generate test value cij {0, 1}

Each sensor *Si* , set cij = 0 and compute dij=X-Y

IF dij > 0 THEN
dij =(X' –X) –(Y' -Y) ;
IF α dij > *ɵ*2 THEN
Set cij = 1;

IF dij > *ɵ*1 THEN
α dij =(X' –X) –(Y' -Y); α
IF α dij > dij THEN
Set cij = 1;

Detection phase

Step 2: *Si* generates a tendency value *Ti* based on its neighbouring sensors' test value through transmissic range:

$IF \sum_{k=1}^{n} Sj€\ Negt\ (\ i)(\ cij) < \left[Negt\ \frac{i}{2}\right]$ THEN

Ti = LG;
ELSE
Ti = LF;
END

Confirm Phase

Step 3: Determine the state and accuracy of the test by comparing the number of neighbouring nodes with different test results.

IF $\sum_{k=1}^{n} Sj€\ Negt\ (\ i)(1 - 2\ cij) \geq \left[Negt\ \frac{i}{2}\right]$ THEN

Tj = GD;
ELSE *Tj* = LF;
END

Step 4: For the remaining undecided sensors, perform the following steps:
FOR i = 1 to S_i
If Ti = LG, THEN
Tj = Tk = GD.
If cji=cki=0, THEN
Ti = GD.
ELSE
Ti = FT;
Repeat;
END
END

Step 5: for cover malfunction or faulty sensor nodes, do the following steps
FOR k = 1 to n
TEMP = Negt (*i*)
S=Select (k,n)
END

Another scenario requires applications to continuously gather and integrate data provided by a large number of physically distant sensor nodes. There are numerous devices exchanging data, however some information sources and sinks may not be available in the network at the time. Therefore, request/response communication is insufficient. For example, a client that wants instantaneous updates of information would have to continuously poll the information providers, resulting in network overload and congestion. Furthermore, because energy is a limited resource, excessive information demands should be avoided. If the projected frequency of fundamental event occurrence is low, consider constructing simultaneous systems. The alternative is to use event-driven communication, which is an asynchronous architecture with independent senders and receivers. Its clients are event publishers and event subscribers, with message transmission and notification services allowing both one-to-many and many-to-many communication. (as depicted in Figure 3).

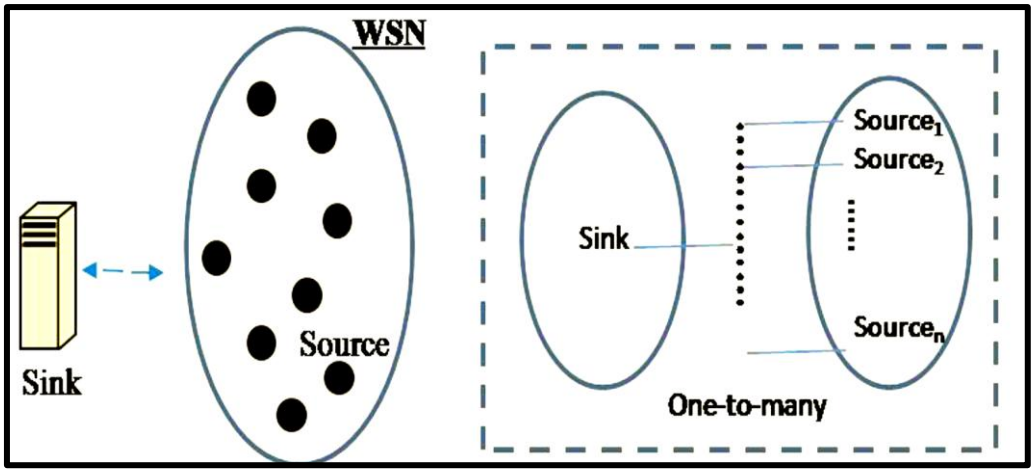

Fig. 3. Event-driven communication in WSNs

- Faulty transmitter nodes: Each sensor node computes the data comparison results and delivers them to the cluster head, who checks the node's condition over an estimated period of sequence packets; we can use this duration as a threshold. If it exceeds the threshold value and does not relay its results to the CH, the sensor is reported to fail; otherwise, the transceiver circuit is deemed disconnected. As a result, a sensor sends a heartbeat message to the cluster head at regular intervals, to indicate its operational health. As a result, the cluster head sends a message to the remaining cluster nodes, announcing that those nodes are faulty. If it crosses the threshold value and sends its results to the cluster head, the sensor node is declared as connected.

- Faulty Receiver Nodes: If sensor circuits are within the transmission range, they are considered the primary advice for the sensor node in the proposed architecture. Sensor circuits in sensor nodes must periodically sense required events, such as temperature. Each node broadcasts its observed temperature value to all neighbours in the cluster regularly. It collects the measurement, compares it to the measured value of nearby sensors in the same cluster, and calculates the measurement difference. If a sensor senses an event as y(t) during the timing interval t1 and a neighbour node senses information as x(t) during the timing interval t2, and the component of x(t) along y(t) is cy(t), then the information difference vector e(t) is represented by applying the following formula 5:

$$e(t) = \begin{cases} x(t) - \propto \times k\, y(t), & t1 \le t \le t2 \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

where *k* is the regularity coefficient

- Traffic nodes: If the sensor detects a flaw, it identifies itself as a traffic node. In our proposed EFCM, the sensor regularly sends a heartbeat message to the cluster head to inform it of its operational status. However, if the cluster head does not receive this heartbeat message within a specified timeframe, it may consider the corresponding node to be defective. In addition, each node gets data from neighbour nodes in the cluster and compares it to its sensing, sending valuable data to the cluster head during an expected period that is deemed a threshold value. If that value

exceeds the threshold and the cluster head node does not receive any data during that period, the cluster head node may deem the node's sensor circuit broken.

- Omission Nodes. The node requires power to broadcast and receive messages to other sensor nodes in the cluster located at varying distances; thus, communication is heavily reliant on the node's energy. As a result, the node spent the majority of its energy communicating with its next-hop neighbour. In our EFCM, the cluster head (CH) periodically checks the battery status of its clustering members. The receiver node sends a heartbeat message to the cluster head informing it of the battery's condition, which is either active or sleeping. The sensor node can also detect its own battery/power failure status through periodic energy level checks on the receiver circuit. A battery error happens when a sensor node's battery energy level goes below a certain threshold. As a result, if the node's battery level exceeds the threshold amount, it can proclaim itself as a malfunctioning node by sending a message to its neighbour and a heartbeat message to the cluster head. In the proposed EFCM, the sensor node may identify its energy level by performing the following mathematical calculations, which aid in detecting either battery continuity or battery recovery.

1. If Node is needed recovered, thus (*CRE* < *Ethr*)
2. If Node is continued, thus (*CRE* > *Ethr*)
3. If the Node is removed, thus it is dead

Where *CRE* is the energy consumption after receiving *l* bit data that is estimated by joule, and *Ethr* is a threshold value of battery that is estimated by joule. After detecting and diagnosing a fault, the system proceeds to prevent or recover from it. The primary strategy for achieving this goal is to re-recover the system's components that are critical to the system's proper operation. In our recovery model, after the packet reception phase, the sensor node checks to see if it is covered; if so, it waits for a random time (to reduce competition between cluster neighbours). Hence, the sensor node that needs to be covered, we may consider as the sleeping node; the sleeping node first executes a self-search theorem, where this node sends a message to both the previous node and the next node in cluster topology; to ask them "Can they cover its task during my sleeping period?", If the head selects another set of neighbours to cover this node, he sends an invitation message. The sleeping node then waits for a set amount of time (referred to as the sleeping period) to receive feedback from the network before entering active mode. If this node does not get feedback and the sleeping duration expires, It will stay in sleep mode. Following that, The CH chooses a new pathway for the cluster's active nodes and eliminates sleeping nodes from the cluster topology. The proposed fault recovery paradigm is summarized in Figure [4].

C. Coordinates Adjustment

Adjusting coordinates in a Self-recovery system for monitoring sensors involves several key steps. Here's a structured approach:

1. Define the Sensor Network Layout
   a) Map the Environment: Create a layout of the area where sensors are deployed.
   b) Identify Sensor Positions: Record the initial coordinates of each sensor in the network.
2. Determine the Recovery Parameters
   a) Define the parameters that need adjustment (e.g., position, orientation).
   b) Establish the reference points for recovery.
3. Calculate Adjustments
   Use mathematical formulas to compute the required adjustments. For example, in a Cartesian system:

$$\text{New } X = \text{Old } X + \Delta X \quad (6)$$

$$\text{New } Y = \text{Old } Y + \Delta Y \quad (7)$$

   Ensure to account for any transformations (rotations, translations).
4. Calculate New Coordinates
   When a sensor fails or needs recovery, calculate its new coordinates based on:
   a) Distance from Other Sensors: Use the average position of neighbouring sensors.
   b) Coverage Area: Ensure the new position maintains adequate coverage.
5. Implement Recovery Protocols
   Deploy Recovery Mechanisms: If a sensor is non-functional, initiate recovery protocols to:
   a) Reposition the sensor (if movable).
   b) Activate backup sensors in nearby locations.
   c) Update the monitoring system with the new coordinates.
6. Monitor and Validate Adjustments
   a) Real-time Monitoring: Continuously monitor the sensor data after adjustments.
   b) Validation Checks: Ensure the new coordinates provide accurate and reliable readings.
7. Iterate Based on Feedback
   a) Use feedback from the monitoring system to refine the recovery process.
   b) Adjust the algorithms and thresholds based on performance data.

4. Simulation & Experimental Results

The Simulator NS3 utility is used for simulations. The simulation scenario involves a cluster of 5 sensor nodes coupled to the cluster head node; they are active nodes that are structured and deployed at random using a distributed approach. The measurement parameter xi is believed to represent atmosphere monitoring readings. We employ the EFCM approach to discover and recover malfunctioning nodes in clustering by self-adjusting new coordinates in the defective clusters. In the proposed EFCM technique, we measure energy consumption after each completed transmission of data bits to determine which node to cover next. Typically, transmission (or reception) is not allowed between a sender (and receiver) and its neighbouring nodes, even if a transmission is currently taking place between the sender and receiver. We utilized the energy model from Eq. (8) to obtain the number of nodes that will be covered.

$$CRE = REt - (REt * Rt) \qquad t1 \le t \le t2 \qquad (8)$$

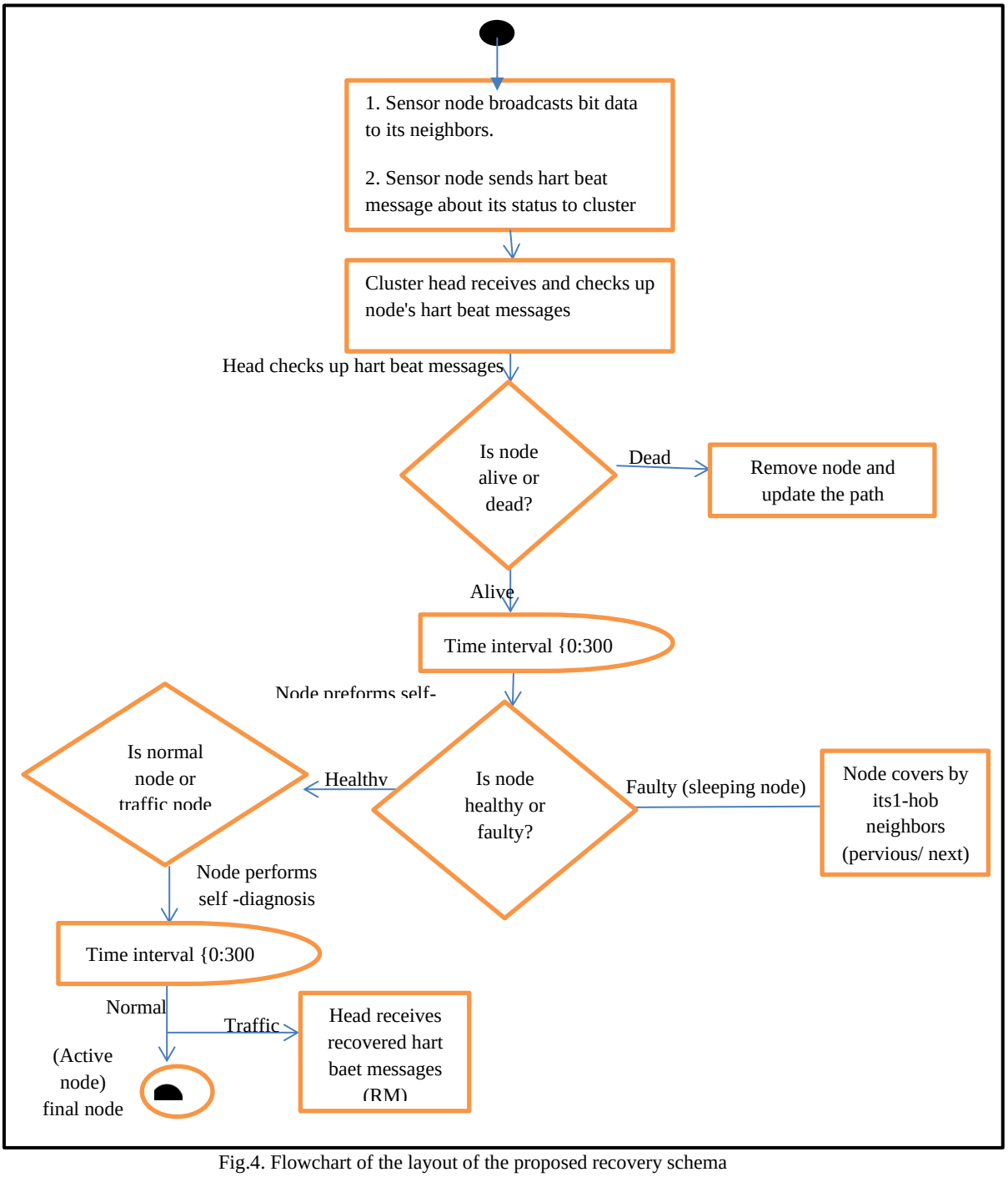

Fig.4. Flowchart of the layout of the proposed recovery schema

Experimental trials show that if the energy expenditure by receiving as CRE is less than the energy threshold (Ethr=0.03 of battery size), a node in the system will stop transmitting data and be considered malfunctioning practically, therefore this node needs to recover through the cluster. When implementing the proposed approach, the symmetrical network conditions were listed below. For a symmetrical WSN,

1) All sensor nodes must be equally spaced apart.
2) Each sensor node has the same sensitivity .
3) The operating speeds of all sensor nodes are equivalent .
4) All sensor nodes use the same wireless communication module.
5) All sensors have fixed locations.

The simulation employed normal distributions to randomly install cluster sensor nodes in a 270m×270m square terrine. Active nodes are pre-structured and deployed in advance, whilst inactive nodes are deployed randomly. During simulation, the cluster head sends a heartbeat message to all nodes in the cluster after each transmission to check for bit data that crosses an estimated time (t=300 μs). The simulation found that nodes had an 80% hardware failure rate. Table 1 illustrates the experimental outcomes. The simulation results of our suggested EFCM revealed the efficiency of the proposed model, using five criteria that have been evaluated:

1. Detection Accuracy (DA) refers to the proportion of disconnected sensor nodes identified as defective to the total number of sensors in the network.
2. Elapsed Time: is the time spent transmitting and receiving l bits to determine the network issue, as determined by MS .
3. Diagnosed heartbeat message (DM): The cluster head sends messages over time to detect whether a parameter is linked or detached.
4. Covered flag message: The cluster head sends this message to signal that a node will be covered if its battery is less than the threshold.
5. The number of healthy nodes: is the total number of nodes in working order. Normal, end and traffic nodes are considered healthy nodes.
6. The NS3 simulator tools simulated the suggested schema and evaluated its accuracy in carrying out self-detection and self-recovery models, as well as calculating the number of problematic nodes that can cause poor wireless sensor network performance. The performance of DA is very precise and quick in determining the number of faulty sensors that occurred during the detection process during the specified period. It is worth mentioning, that PR is extremely fast in recovering and is free of malfunctioning nodes. It also has great accuracy in detecting error clustering sensors via outgoing heartbeat messages used to locate and isolate failing sensor nodes during recovery phases.

Table 1. The simulation yielded experimental results.

PARAMETER	VALUES
Number of nodes in the network	1800 nodes
Number of clusters in the network	300 cluster
Number of nodes in each cluster	6 nodes
Data packet size	800 bit
Initial energy	0.5 J
The time between consecutive packets	300ms.
Detection Accuracy (DA) in duty cycle	~ 2 nodes for the round
Consumed time for Detection Accuracy (DA)	0 ms.
The number of diagnosed heartbeat messages represented faulty transmitter nodes in the duty cycle	~ 2 nodes for the round
Consumed time to occur detection messages in cluster head represented faulty transmitter node	62ms.
The number of diagnosed heartbeat messages represented faulty receiver nodes in the duty cycle	~ 1 *node* for the round
Consumed time to occur diagnosed messages in cluster head represented faulty receiver	125ms.
Ratio of lost energy in deployments	52% of battery size
Precision of Recovery (PR)	100%

The proposed design enhanced the fault recovery method, as seen in Figure (5). In the proposed model, we adopted a distributed clustering strategy to improve network lifetime, the most essential parameter for evaluating sensor network performance. Although there is no one definition of "network lifetime," our schema notion is established by the application's purpose, which is based on common heartbeat signals relating to the length until the first/last node in the network depletes its energy and disconnects from the base station. As a result, the network's lifetime can increase by more than 80 %.

a) Test Byte Examination:

In the proposed methodology, the sender node was capable of transmitting byte packets to its cluster neighbours during a specific time, and the receiver node received these packets within a specified duration. Following the receiving action, the receiver responds to the cluster head with a byte arrival notice, indicating the validity of this node (non-fail) to the head; this procedure is known as the good response of byte (GRB). Unless the receiver node responds to the cluster head and exceeds the specified time for receiving a byte packet, this is known as a poor response of byte (BRB). In a poor response, the cluster head sends a Test Byte message to the sender to check the node's transmission

integrity. If the sender delivers the heartbeat message to the head, the head will authenticate the sender's validity. Furthermore, the head node declares that the receiving node is likely to fail into the rest of the cluster. Otherwise, the sender was regarded as a defective node and reported to the cluster nodes. Then, it selects the new path of monitoring and adjusts the coordinates of the monitoring sensors within the cluster. E.g., if a temperature sensor at coordinates (10, 15) is malfunctioning, and the average position of nearby sensors is (12, 18), it might adjust the coordinates, as:

New X = (10 + 12) / 2 = 11

New Y = (15 + 18) / 2 = 16.5

Thus, the adjusted coordinates for the malfunctioning sensor would be approximately (11, 16.5).

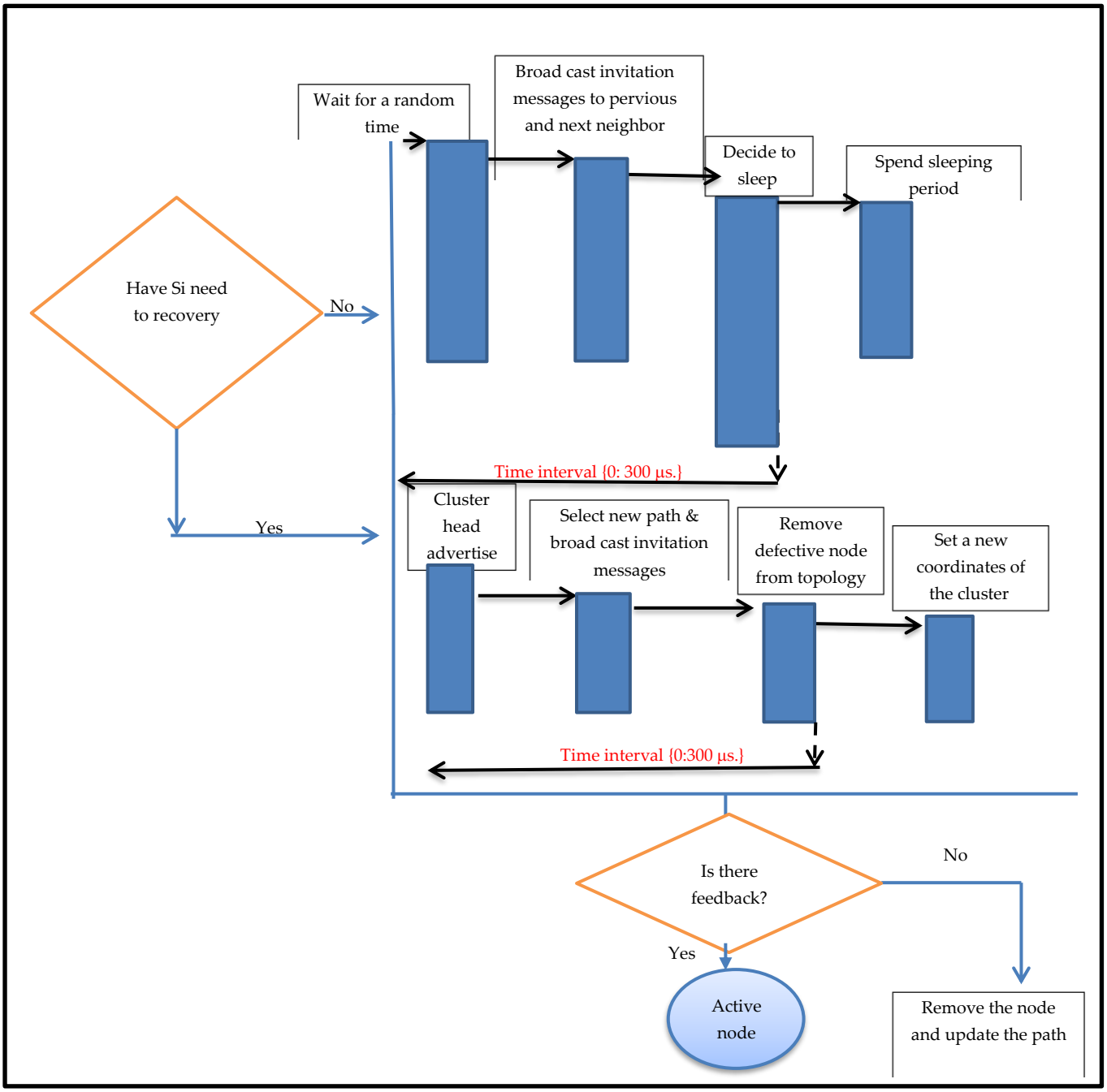

Fig. 5: Sequences of the suggested fault recovery mechanism in simulation

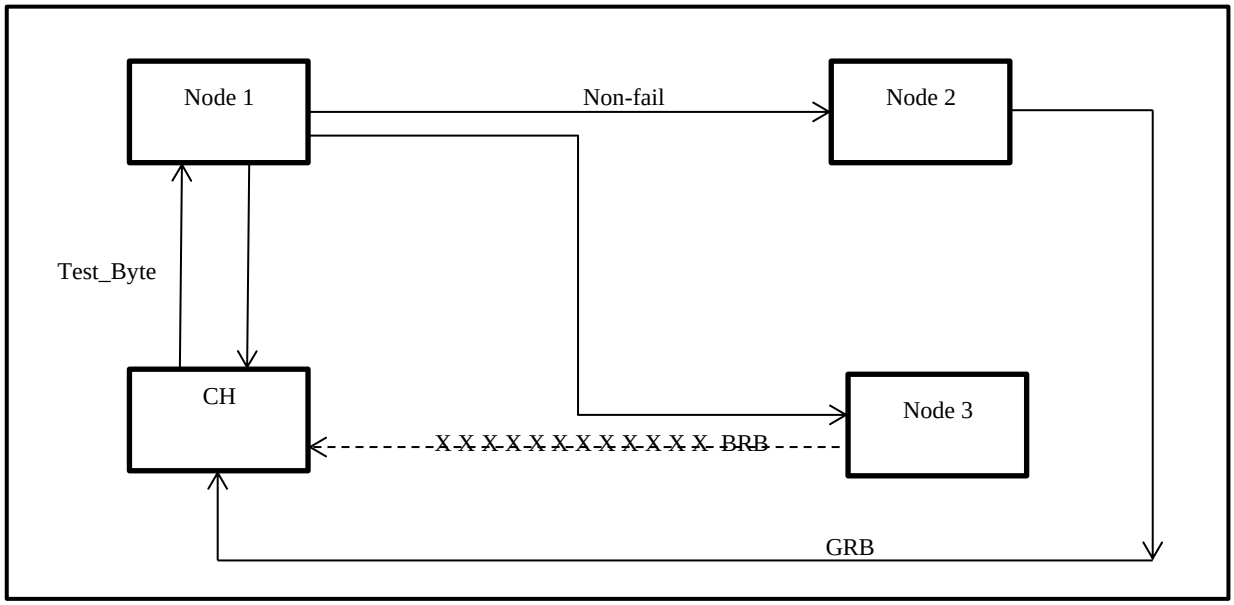

Fig. 6: Graph illustrates the implementation of the Test Byte examination

b) Bit Error detection

When an odd number of bits (including the parity bit) are sent incorrectly, the parity bit will also be incorrect, indicating a parity error during transmission. It's important to note that the parity bit can only detect errors; it cannot correct them because it cannot identify which specific bit is erroneous. This means that the entire data must be erased and re-transmitted from the beginning. Successful transmission over a noisy medium can sometimes take a long time or may not occur at all. However, the advantage of using a parity bit is that it requires only one bit and a small number of XOR gates for its implementation. For example, if we communicate the four-bit value 1001, the similarity bit can be calculated as follows.

Table 2. The scenario of Bit Error detection in a broadcast event

Type of Bit	Successful Broadcast Scenario
EVEN	1. S*i* wishes to transmit 1001 2. S*i* computes the parity bit value as 1+1+1+1 (mod 4) = 0. 3. S*i* then adds the parity bit and sends 11110: 4. CH gets 11110, 5. CH computes parity (1+1+1+1+0 (mod 4) = 0) 6. CH reports correct transmission based on expected even results.
ODD	1. S*i* wishes to transmit: 1001. 2. S*i* computer parity bit value is 1 + 1 + 1 + 1 (mod 4) = 1. 3. S*i* inserts the parity bit and sends: 11111. 4. CH receives 11111. 5. CH calculates overall parity as 1+1+1+1+1 (mod 4) = 1. 6. CH reports accurate transmission after noticing expected unusual results.

This technique detects single-bit faults because if one bit is flipped owing to noise, the received data will have an incorrect number of ones. In the two preceding examples, CH's estimated similarity value corresponds to the parity bit in its received value, suggesting that there are no single-bit errors. Consider the following example of a transmission fault in the second bit using XOR:

Table 3. Scenario of Bit Error detection in a failed broadcast event

Type of Bit Error	Failed Broadcast Scenario
Even Errors in the Second Bit	1. S*i* wants to transmit: 1001. 2. S*i* computes the parity bit value (1^0^0^1 = 0). 3. S*i* adds the parity bit and sends: 10010. Transmission error. 4. CH Receives 11010. 5. CH calculates overall parity as 1^1^0^1^0 = 1. 6. CH reports incorrect transmission after observing unexpected results.
Even Error in the Similarity Bit	1. S*i* wants to transmit: 1001. 2. S*i* computes the even parity value as 1^0^0^1 = 0. 3 S*i* Sends: 10010. Transmission error. 4. CH Receives 10011. 5. CH calculates overall parity as 1^0^0^1^1 = 1. 6. CH reports incorrect transmission after observing unexpected results.

5. Conclusion

Self-recovery mechanisms in sensors enhance the reliability and efficiency of systems, making them essential in modern technology applications. By integrating robust fault detection and recovery strategies, systems can maintain optimal performance even in the face of failures. Adjusting coordinates in self-recovery for sensors is crucial for maintaining accuracy and reliability. By following a structured approach that includes fault detection, data validation,

recalibration, and continuous monitoring, systems can effectively recover from sensor faults and continue to operate optimally. Fault clustering is an effective management strategy for network topology aiming to reduce communication overhead and increase data aggregation in sensor networks. We presented a new distributed clustering approach for detecting malfunctioning nodes and recovering the working domain by selecting alternate pathways in sensor networks. The proposed schema is based on a distributed measure that assesses a node's state and broadcasts it to neighbouring sensor nodes. In the future, we will investigate systematic flaws in wireless sensor networks and seek to determine the best strategy to prevent such defects from occurring in sensor networks. We tested our schema's performance with the NS3 simulator, and the results reveal that the proposed schema, based on the self-recovery mechanism, is quite efficient and can reap significant performance benefits in terms of reduced communication costs as well as extended network lifetime. On the other hand, increased network longevity refers to a network's capacity to maintain functionality and performance over time. This is frequently accomplished through the use of energy-efficient hardware, strong system architecture, and optimal resource management. A longer-lasting network eliminates the need for periodic replacements or upgrades.

Reference

1. Arunanshu Mahapatro and Pabitra Mohan Khilar, Scalable Distributed Self-recovery Protocol for Wireless Sensor Networks,2021
2. Indrajit Banerjee, Prasenjit Chanak, Hafizur Rahaman, Tuhina Samanta, Effective fault detection and routing scheme for wireless sensor networks, Bengal Engineering and Science University,2023.
3. Chessa, S., Santi, P.: Comparison Based System-Level Fault Self-recovery in Ad-hocNetworks. In: Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems, OCT (2021).
4. Ding, M., Chen, D., Xing, K., Cheng, X.: Localized fault-tolerant event boundary detection in sensor networks. In: IEEE Infocom, pp. 902–913 (2015).
5. Rangarajan, S., Fussell, D.: A Probabillistic Method for Fault Self-recovery of Multiprocessor Systems. In: Proceedings of the 18th International Symposium on Fault-Tolerant Computing, pp. 278–283, Tokyo, Japan (1988).
6. Chessa, S., Santi, P.: Crash Fault Identification for wireless sensor networks, Jour. of Computer Communications 25(14), 1273–1282 (2002)
7. Preparata, F.P., Metze, G., Chien, R.T.: On the Connection Assignment Problem of Diagnosable Systems. IEEE Trans. on Computers EC-16, 848–854 (1967).
8. Barborak, M., Malek, M., Dahbura, A.T.: The Consensus Problem in Fault Tolerant Computing ACM computing surveys, vol. 25, pp. 171–220 (1993).
9. Mohamed K. Watfa1 and Rawad Abu Assi2, A Distributed Algorithm for Isolating Malfunctioning Nodes in Wireless Sensor Networks, American University of Beirut, Computer Science Department, Beirut, Lebanon,2021.
10. Meenakshi Panda, P.M. Khilar; Efficient Fault Detection Algorithm in Wireless Sensor Network; Department of Computer Science and Engineering; 2021.
11. Raja Jurdak, X. Rosalind Wang, Oliver Obst; Wireless Sensor Network Anomalies; CSIRO ICT Centre, Australia; 2021.
12. Indrajit Banerjee, Prasenjit Chanak3, Biplab Kumar Sikdar, and Hafizur; DFDNM: Distributed Fault Detection and Node Management Scheme for Wireless Sensor Network; Department of Information Technology;2021.
13. Arunanshu Mahapatro, Pabitra Mohan Khilar; Online Distributed Fault Self-recovery in Wireless Sensor Networks; National Institute of Technology, Rourkela, India; 2022.
14. Andre´ Rodrigues, Tiago Camilo, Jorge Sa´ Silva; Diagnostic Tools for Wireless Sensor Networks; e-mail: arod@dei.uc.pt; 2022.
15. Sushruta Mishra, Lambodar Jena; Fault Tolerance in Wireless Sensor Networks; Computer Science and Software Engineering; 2022.
16. Ravindra N Duche, N.P.Sarwade; Sensor Node Failure or Malfunctioning Detection in Wireless Sensor Network; 1Department of Electrical Engineering, VJTI, Mumbai, India; 2022.
17. Shahram Babaie, Afsaneh Khosrohosseini; A new self-diagnosing approach based on Petri nets and correlation graphs for fault management in wireless sensor networks; Department of Computer Engineering, Tabriz Branch, Islamic Azad University, Tabriz, Iran;2023.
18. Er. Saurabh, Dr. Rinkle Rani Aggarwal; A Review of Fault Detection Techniques for Wireless Sensor Networks; Department of Computer Science & Engineering, Thapar University, Patiala;2023
19. Bill C.P. Lau a, Eden W.M. Maa; Probabilistic fault detector for Wireless Sensor Network; City

University of Hong Kong, Hong Kong, China;2024.
20. Sushruta Mishra, Lambodar Jena, Aarti Pradhan; Fault Tolerance in Wireless Sensor Networks, Dept. of CSE, GEC, BBSR India;2022.
21. Hai Liu1, Amiya Nayak1, Ivan Stojmenović; Fault Tolerant Algorithms/Protocols in Wireless Sensor Networks; School of Information Technology & Engineering, University of Ottawa, K1N 6N5, Canada. 2022.
22. Mohamed K. Watfa and Sesh Commuri, An Energy Efficient and Self-recovery 3-Dimensional Sensor Cover, School of Electrical and Computer Engineering University of Oklahoma, Norman,2023.
23. Abayomi M. Ajofoyinbo; Energy Efficient Packet-Duration-Value Based MAC Protocol for Wireless Sensor Networks; Department of Systems Engineering, Faculty of Engineering, University of Lagos, Lagos, Nigeria;2023