# A Survey on Fault Tolerance in Wireless Sensor Networks

Luciana Moreira Sá de Souza
SAP Research
Karlsruhe, Germany
Email: luciana.moreira.sa.de.souza@sap.com

Harald Vogt
SAP Research
Karlsruhe, Germany
Email: harald.vogt@sap.com

Michael Beigl
Technische Universität Braunschweig
Braunschweig, Germany
Email: beigl@ibr.cs.tu-bs.de

***Abstract*— The reliability of wireless sensor networks (WSN) is affected by faults that may occur due to various reasons such as malfunctioning hardware, software glitches, dislocation, or environmental hazards, e.g. fire or flood. A WSN that is not prepared to deal with such situations may suffer a reduction in overall lifetime, or lead to hazardous consequences in critical application contexts. One of the major fault recovery techniques is the exploitation of redundancy, which is often a default condition in WSNs. Another major approach is the involvement of base stations or other resourceful nodes to maintain operations after failures. In this paper we present a survey of approaches to fault tolerance and detection techniques in WSNs in both theoretical and application driven research. We provide a taxonomy of faults and classify the surveyed approaches according to their ability of detecting and recovering from faults.**

## I. Introduction

Advances in embedded systems technology have made it possible to build wireless sensor nodes, which are small devices with limited memory, processing power, and energy resources [1]. Due to the low cost associated to these devices, it is possible to conceive the deployment of large-scale wireless sensor networks (WSN) with potentially thousands of nodes [7].

Recently, the usage of WSN to monitor storage regulations of hazardous materials in chemical plants was investigated by the CoBIs project[1]. In this critical industrial environment a high degree of dependability is required. In order to be considered dependable, WSNs must offer characteristics such as: reliability, availability and maintainability.

Availability to a large extend depends on fault tolerance to keep the system working as expected. Availability on the service level means that the service delivered by a WSN (or part of it) is not affected by failures and faults in underlying components such as single nodes or node subsystems. In WSNs, the failure of such components is almost unavoidable. Most detection and recovery techniques therefore aim at reducing *MTTR* (the amount of time required for detecting and recovering from a failure) as much as possible.

We conducted an investigation on frequent faults that occur on real world WSN deployments and the techniques used to detect and overcome these faults. As indicated by deployment reports [27][41][20][43][39], the installation of large-scale sensor networks for real world applications is not a trivial task and can lead to innumerous failures. What works in theory not always performs as expected in practice.

In these WSN deployments, it is common to have a node providing functionality to its neighbors. Multi-hop routing is a simple example of such a service, where nodes forward messages on behalf of each other. Often, nodes assume dedicated roles such as *clusterhead*, which implies the responsibility for certain tasks. For example, a clusterhead could aggregate sensor data before it is forwarded to a base station, thereby saving energy. Nodes with stronger hardware capabilities can perform operations for other nodes that would either have to spend a significant amount of energy or would not be capable of performing these operations.

These services, however, may fail due to various reasons, including radio interference, de-synchronization, battery exhaustion, or dislocation. Such failures are caused by software and hardware faults, environmental conditions, malicious behavior, or bad timing of a legitimate action. In general, the consequence of such an event is that a node becomes unreachable or violates certain conditions that are essential for providing a service, for example by moving to a different location, the node can no further provide sensor data about its former location.

In some cases, a failure caused by a simple software bug can be propagated to become a massive failure of the sensor network. This results in application trials failing completely and is not acceptable in safety critical applications. Hence, our aim is to clarify the requirements for maintaining high level availability in WSNs, and to investigate the tools and mechanisms utilized in WSN research and engineering for fault detection and recovery.

In this paper, we concentrate on enhancements of service availability in WSNs through the use of fault tolerance techniques. We present a survey of approaches to fault detection and recovery techniques in WSNs. We provide a taxonomy of faults and classify the investigated approaches according to their ability to detect and tolerate faults.

This paper provides an overview of the relation of fault tolerance with other areas of research, described in section III, followed by section IV, which presents faults

[1] http://www.cobis-online.de/

reported on real world deployments and classifies the events that cause disruptions in wireless sensor networks, thus motivating the need for fault tolerance techniques. Sections V and VI surveys approaches to fault detection and recovery. In section VII, these approaches are classified and compared. Section VIII concludes this paper.

## II. The CoBIs Project

The project CoBIs develops a novel service oriented approach to support business processes that involve physical entities (goods, tools, etc.) in large-scale enterprise environments. Software systems that provide various services for enterprise businesses are usually based on highly decentralized, manual and thus often error-prone data collection and a more centralized data storage and business logic execution in so called "backend" systems. An important intention of this project is to apply recent advances in the area of sensor networks, in order to distribute business logic functionality to sensor nodes. In this way, the status of enterprises, as it is represented in business processes and in the supporting enterprise software systems, can reflect more closely what is actually happening in the real world.

In industrial scenarios where the CoBIs technology is applied, the services proviced by the WSN must be reliable to avoid major losses in the business processes caused by failures in the sensor network. Figure 1 shows one of these scenarios where CoBIs was applied in a real world deployment. There, drums containing hazardous materials were equipped with sensor nodes programmed with information on their specific content, both chemical composition and volume. These nodes were used to control storage regulations of hazardous materials in a chemical plant. This application trail was used to evaluate the usage of this technology to prevent reacting chemicals from being stored together (e.g. inflammable and oxidizing). The lessons learned from this application trial and the critical scenario where CoBIs was applied motivated the investigation on fault tolerance methods discussed in this paper.

Fig. 1. Application trial at a chemical plant where storage regulations of hazardous materials were monitored

## III. Background

The availability of the services provided by a WSN to a large extent depends on fault tolerance, since usually it cannot be assumed that all sources of error can be eliminated, even through careful engineering.

By service availability, we understand the probability with which a request will lead to a valid and useful response. (Point) availability is defined as:

$$P(A) = \frac{MTTF}{MTTF+MTTR}$$

Where $MTTF$ stands for *Mean Time To Failure* and $MTTR$ stands for *Mean Time To Repair* [3].

By analyzing this equation we can conclude that systems that constantly fail and require long repair time will result in systems with very low availability. However, systems that have a high $MTTF$ and can be quickly repaired are considered highly available systems.

Attacks on the availability of sensor nodes are analyzed in [45]. This paper considers possible attacks on all layers of the communication infrastructure. Examples include jamming, which affects potentially a large number of nodes, and sinkhole routing, where messages are diverted to a node determined by the adversary. Such attacks directly influence the availability of services.

Fault tolerant systems can overcome faults and system failures, therefore increasing the $MTTF$ and system availability. Fault tolerance techniques especially crafted for wireless sensor networks have not been extensively studied so far. One of the most important one is replication, which is well-suited for sensor networks due to their inherent node redundancy. Fault Tolerance has been discussed in detail in the literature on distributed systems (see, e.g. [40]). Although wireless sensor networks have special characteristics that distinguish them from traditional distributed systems, many of the existing techniques still apply. An introduction to the fundamental mechanisms for implementing replicated services in distributed systems is discussed in [11].

While the effects of faults and security breaches are often similar, the underlying causes are fundamentally different and often require different countermeasures. Nevertheless, there is a overlap, which is demonstrated by the concept of Byzantine failures [19], describing arbitrary behavior of parties, which may be caused by either unintentional faults in the system or malicious behavior.

## IV. Failures in Wireless Sensor Networks

To comprehend fault tolerance mechanisms, it is important to point out the difference between faults, errors, and failures. Various definitions of these terms have been used [3], [40]. This paper refers to the definition given in [40]:

- A *fault* is any kind of defect that leads to an error.
- An *error* corresponds to an incorrect (undefined) system state. Such a state may lead to a failure.

- A *failure* is the (observable) manifestation of an error, which occurs when the system deviates from its specification and cannot deliver its intended functionality.

Figure 2 illustrates the difference between fault, error, and failure. A sensor service running on *node A* is expected to periodically send the measurements of its sensors to an aggregation service running on *node B*. However, *node A* suffers an impact causing a loose connection with one of its sensors. Since the code implementing *node A*'s service is not designed to detect and overcome such situations, an erroneous state is reached when the sensor service tries to acquire data from the sensor. Due to this state, the service does not send sensor data to the aggregation service within the specified time interval. This results in a crash or omission failure of *node A* observed by *node B*.

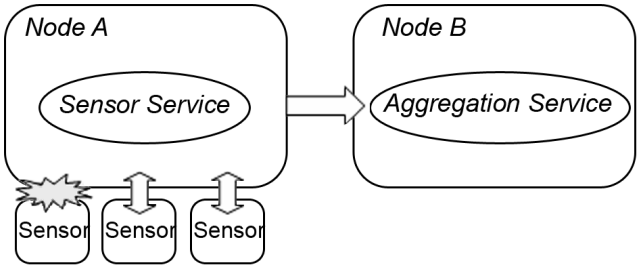


Fig. 2. Failure cause by a loosely connected sensor

In the scenario explained above, the fault is the loose connection of the sensor. The error is the state of the service after trying to read the sensor data and the failure occurs when the application does not send the sensor data within the specified time interval. To provide resilience in faulty situations two main actions must be performed:

**Fault detection**. To provide any countermeasures, the first step a system must perform is to detect that a specific functionality is or will be faulty.

**Fault Recovery**. After the system has detected a fault, the next step is to prevent or recover from it. The main technique to achieve this goal is to replicate the components of the system that are vital for its correct operation.

### A. Source of Faults in Real WSN Applications

Wireless sensor networks are commonly deployed in harsh environment and are subject to faults in several layers of the system. To analyze the faults that can occur in real application scenarios we performed a research on several application trial reports. The experience from these expeditions can be used as guidance for future application trials to avoid the same errors from happening.

Figure 3 presents a layered classification of components in a WSN that can suffer faults. A fault in each layer of the system has the possibility to propagate to above levels. For example, a power failure of a node will cause the entire node to fail. If this node is on a routing path, the messages of other nodes relying on this routing path will not be delivered making an entire region of the network silent until the routing path is restored.

Ultimately, if the application in the back-end which presents the WSN data to the users suffers a fault due to some software bug or hardware failure the entire system is considered faulty. In this paper, however, we will concentrate on faults that can happen in the sensor nodes up to the sink.

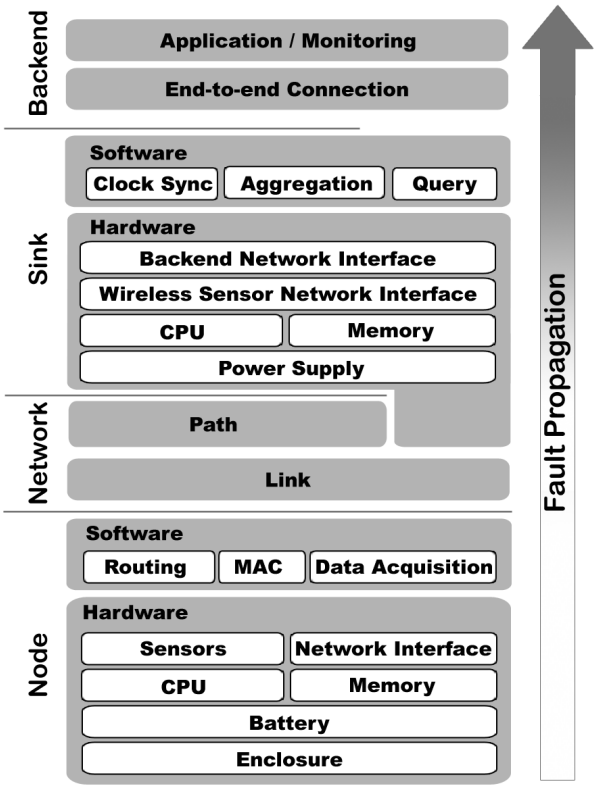


Fig. 3. Fault classification and propagation

*1) Node Faults:* Nodes have several hardware and software components that can produce malfunctions. For example, the enclosure can suffer impacts and expose the hardware of the sensor node to the extreme conditions of the environment. In [27][41][39] due to stress from the environment and inadequate enclosures, the sensor nodes were exposed to direct contact with water causing short circuits. The report of a large-scale deployment in a potatoes field [20] indicated that the antennas from the nodes were quite fragile and would become loose when inserting the node into the packaging.

When the battery of a node reaches a certain stage, sensor readings may become incorrect. This has been observed in [43] where many outlier readings were generated in the network caused by imminent battery failure. As demonstrated in Figure 3, hardware failures will generally lead to software failure. A *Data Acquisition* application will not perform properly if the underlying sensors are providing incorrect readings. Nevertheless, some hardware failures do not affect all the services in a sensor node. In the example discussed, although the node cannot be used to provide correct sensor readings it still

can be used to route packages in the sensor network.

Software bugs are a common source of errors in WSNs. In [44], the researchers reported that a software bug caused the longest continuous network outage taking the system offline for three days until the nodes could be reprogrammed manually.

Organizing a network in clusters is an approach used in many applications, for example to extend the lifetime of the network [15]. A small number of nodes are selected to become clusterheads. They are responsible for coordinating the nodes in their clusters, for instance by collecting data from them and forwarding it to the base station.

In case that a clusterhead fails, no messages of its cluster will be forwarded to the base station any longer. The clusterhead can also intentionally or due to software bugs forward incorrect information. Depending on the application case, the impact of such a failure can vary from quality degradation of measurements to alarm messages not being delivered to a back-end system.

While forwarding messages, nodes can aggregate data from multiple other nodes in order to reduce the amount of data sent to the base station. One common simple approach is to calculate the average of correlated measured values such as temperature, humidity and pressure, sending only one message to the back-end.

If a node generates incorrect data, the data aggregation results can suffer deviations from the real value. Also, if a node responsible for generating the aggregated data is subject to a value failure, the base station will receive incorrect information of an entire region of the network.

*2) Network Faults:* Routing is one of the fundamental building blocks in a WSN. It is essential for collecting sensor data, distributing software and configuration updates, and for coordination among nodes. Additionally, there may be application-specific routing protocols required, for example for tracking and "following" moving objects. Faults on the routing layer can lead to dropped or misguided messages, or unacceptable delays.

In WSNs, communication links between nodes are highly volatile. WSNs not always yield the same delivery rate of messages in field trials as in lab trials. For instance, in [38] a delivery rate of only 58% of the messages was observed, in [36] the instability of the links between nodes lead to constant changes in the routing paths.

In several scenarios of sensor networks nodes have a certain degree of mobility. In a glacial expedition [27] the experiment assumed a one hop network. The connection of the nodes to the sink was calibrated during deployment with a reliable link connection. Nevertheless, after some time the nodes moved to a different location where the node was unreachable resulting in complete loss of data from this node.

Radio interference can also cause the link between nodes to become faulty. For instance, in agricultural fields the placement of the nodes must be carefully planned to take into consideration that when plants start growing the link range is considerably reduced, as discussed in [42]. Another source of link failure is the collision of messages. In [39] researchers observed a potential for collision of messages of nodes in close proximity due to a phase change and overlap.

In other situations, however, nodes may have perfect link connections but the messages are not delivered to their destination due to path errors. A software bug in the routing layer can generate circular paths or simply deliver messages to the incorrect destination.

*3) Sink Faults:* On a higher level of the network a device (sink) that collects all the data generated in the network and propagates it to the back-end system is also subject to faults of its components. When this device fails, unless fault tolerant measurements are present, a massive failure of the network happens given that the data from the sensor nodes cannot be accessed.

The sink can be deployed in areas where no permanent power supply is present, in such applications batteries together with solar cells are commonly applied [27][20][43] to provide the amount of energy necessary. In the glacial expedition reported in [27], this traditional technique has proven to be inefficient. Although this worked perfectly for other expeditions, in this glacial environment the sink suffered a power failure due to snow covering the solar cells for several days.

Network infrastructure is usually also not available in the area where the sink is deployed, therefore alternative solutions such as a satellite connection are used, which can cause fluctuations in the back-end network interface. In [24] researchers indicated that during periods of severe thunderstorm activity the satellite connection becomes unavailable.

Finally, the software that stores the data collected from the network, processes it and sends it to the back-end system, is subject to bugs that when present can lead to loss of data within the period where the fault occurred. In the first application trial realized by the CoBIs project, the software of the gateway presented malfunctions that prevented the back-end application from receiving the data generated in the network during several periods.

### B. Failure Classification

As discussed in Section IV-A, several faults could lead to failures in wireless sensor networks: a node could be moved to a different region, causing a link failure; nodes can suffer power failure and stop responding to requests, or they can start sending arbitrary values either intentionally (after a security breach) or due to a malfunction.

Here, we classify the failures that a WSN is susceptible to, which are: crash, omission, timing, value and arbitrary. These failures are the observable manifestation of underlying faults presented on Section IV-A.

*1) Crash or omission:* A failure by omission is determined by a service sporadically not responding to requests. For instance, this could be caused by radio interference that leads to occasional message loss. A crash failure occurs when the service at some point stops responding to any request. An omission degree $f$ can be defined which imposes a limit to the amount of omission

failures a node might have before being classified as crashed.

*2) Timing:* Services might fail due to a timeout in processing a request or by providing data too early. Such timing failures occur when a node responds to a request with the correct value, but the response is received out of the time interval specified by the application. Timing failures will only occur when the application specifies timing constraints.

*3) Value:* A service is considered having failed due to an incorrect value when the service sends a timely response, however with lack in accuracy of the delivered value. For instance, a service performing aggregation of data generated by other nodes could forward a result value to the base station that does not correctly reflect the input data. Such situations could be caused by malfunctioning software, hardware, corrupt messages, or even malicious nodes generating incorrect data.

*4) Arbitrary:* Arbitrary failures include all the types of failures that cannot be classified in previously described categories. In [19], Lamport has introduced the Byzantine Generals Problem in the context of distributed systems. Recent work shows how to deal with this problem in the domain of wireless sensor networks [16]. Byzantine failures describe a type of arbitrary failures that are 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 acknowledge back to the sender.

## V. Fault Detection Techniques

The goal of fault detection is to verify that the services being provided are functioning properly, and in some cases to predict if they will continue to function properly in the near future. The simplest way to perform such a task is through visual observation and manual removal of incorrect values. This technique has obvious drawbacks: human interaction leads to errors, it has a high cost and it is not efficient. Hence, we investigated automatic fault detection techniques for WSN.

We classified the techniques we investigated according to the parties involved in the process. Through *self-diagnosis* the node itself can identify faults in its components. With *group detection*, several nodes monitor the behavior of another node. Finally, in *hierarchical detection* the fault detection is performed using a detection tree where a hierarchy is defined for the identification of failed nodes. Often in a *hierarchical detection* the detection is shifted to a more powerful device such as the sink.

### A. Self-Diagnosis

In many cases, nodes can identify possible failures by performing self-diagnosis. In [14], the authors propose an approach where a node would perform a self-diagnosis based on the measurements of accelerometers to determine if the node suffers from an impact that could lead to hardware malfunctions.

Using a similar approach, nodes could detect when they are being moved to a different location. Another approach would be to keep track of the identities of the nodes in the neighborhood. A considerable change in the neighborhood could indicate that either the node itself or some of its previous neighbors have been moved.

Faults caused by battery exhaustion can be predicted when the hardware allows the measurement of the current battery voltage [2][29]. By analyzing the battery discharge curve and the current discharge rate, an algorithm can determine an estimation of the time to death of the battery.

Nodes can also identify that their current connection to surrounding nodes is unreliable by probing the link connection therefore identifying that it is isolated.

### B. Group Detection

The detection of services failing due to incorrectly generated values is only possible if a reference value is available. In [18] and [6], detection mechanisms are prosed to identify faulty sensor nodes. Both algorithms are based on the idea that sensors from the same region should have similar values unless a node is at the boundary of the event-region. The algorithms start by taking measurements of all neighbors of a node and uses the results to calculate the probability of the node being faulty.

Another approach proposed in the literature is to let consumer nodes observe whether the service provider is in fact performing the operations that it is supposed to. In [26], a misbehavior detection algorithm to aide the routing layer is proposed. The misbehavior detection mechanism is based on the idea of monitoring the communication of the service provider to verify whether messages are forwarded correctly.

Focusing on providing a fault-tolerant approach for clusters in WSNs, in [12] it is proposed to support the dynamic recovery of failed gateways (high-energy devices that act as clusterheads). The proposed protocol assumes that a gateway has failed only when no other gateways can communicate with it. The fault detection mechanism is based on constant status updates being exchanged between gateways and further use of a consensus algorithm.

### C. Hierarchical Detection

The definition of a detection tree enables a scalable fault detection algorithm in WSN. Memento [33] proposes the usage of the network topology to forward the fault detection results of child nodes to the parent nodes and up to the sink. Each node forwards the status of the child nodes that it is monitoring to its parent node. The parent performs an aggregation (bitwise OR) operation on the results of the child nodes together with its on results and forwards it to the next level.

The approach proposed by Memento scales well with the network size, however it consume resources of the network. Shifting the fault detection task to a more

powerful device is an alternative that can help to increase the lifetime of the WSN. In [37], the authors propose an algorithm that puts the burden of detecting and tracing failed nodes to the base station. At first the nodes learn the network topology and send their portion of the topology information to the base station. With this information the base station learns the complete network topology which is used to send route updates as soon as it detects that nodes become silent.

This approach is not applicable to event-driven WSN because in such a network sensors only send messages when there is an event that should be reported, for instance when the temperature goes above a certain limit. In [35], the authors focus on providing a solution in this context. The proposed mechanism uses a hierarchical network topology where cluster heads monitor ordinary nodes, and the base station monitors the cluster heads. To perform the monitoring, the base station and the cluster-heads constantly ping those nodes that still have battery power left and that are under their direct supervision. If a node does not respond, it is marked as failed.

Sympathy [30] is a debuging tool that also utilizes the hierarchical detection approach. This tool instruments the WSN with monitoring software on the sensor nodes that generates metrics data that is forwarded to a centralized sink location for analysis. With this information Sympathy is able to detect crash, timeout and omission failures and identify the fault that generated the failure.

SNIF [31] is another example of a debuging tool that identifies the source of problems in WSN. Contrary to Sympathy, this tool does not modify the software of the sensor nodes nor requires additional traffic to be transported through the WSN. To automatically identify network failures this tool proposes a decision binary tree based on the research performed by the authors on failures in real world deployments.

## VI. Fault Recovery Techniques

Fault recovery techniques enable systems to continue operating according to their specifications even if faults of a certain type are present. As discussed in section IV, there are many potential sources for faults in WSNs. Fault tolerance techniques have been proposed in various contexts that increase the reliability of the functionality of sensor nodes in their specific domain. We attempt at giving an overview of this scattered work.

The most common of these techniques is the replication of components. Although redundancy has several advantages in terms of high reliability and availability, it also increases the costs of a deployment. As an alternative, according to the specification of the project, the quality required from the WSN can be downgraded to an acceptable level.

In this paper we classify the recovery techniques for WSN into two major approaches: *Active* and *Passive* replication. Active replication means that all requests are processed by all replicas, while with passive replication, a request is processed by a single instance and only when this instance fails, another instance takes over.

### A. Active replication in WSN

Active replication in wireless sensor networks is naturally applied in scenarios where all, or many, nodes provide the same functionality. One example is a service that periodically provides sensor data. Nodes that run this service activate their sensors and forward their readings to an aggregation service or to a base station. When some nodes fail to provide that information, the recipient still gets the results from other nodes, which is often sufficient. Fault recovery in the presence of active replicas is relatively straightforward. Nevertheless, for a consistent survey we present some of these approaches here:

*1) Multipath routing:* Usually, it is desirable to avoid that a single failing node causes the partitioning of a sensor network. Thus, a network should be $k$-connected, which allows $k-1$ nodes to fail while the network would still be connected [22]. Multipath routing [10] can be used to actively replicate routing paths. In [4], Bredin et al. proposes an algorithm that calculates the minimum amount of additional nodes and their positions to guarantee $k$-connectivity between nodes.

*2) Sensor value aggregation:* Sensor value fusion [28][5] is a reasearch area that seeks to provides high-level information derived from a number of low-level sensor inputs. There, the inherent redundancy of sensor nodes can be used to provide fault-tolerant data aggregation.This is achieved through a tradeoff between the precision (the length) of the resulting sensor reading interval and the number of faulty sensors. This ensures that despite of node failures, the resulting reading interval will contain the correct sensor reading of a region.

*3) Ignore values from faulty nodes:* A simple but efficient solution to not propagate a failure of one specific node to the entire network is to ignore the data that it is generating, as applied in [14]. The major challenge in this case is the identification of the malfunctioning nodes.

### B. Passive replication in WSN

When passive replication is applied, the primary replica receives all requests and processes them. In order to maintain consistency between replicas, the state of the primary replica and the request information are transferred to the backup replicas. Given the constraints of WSNs, applications should be designed to have only little or no state at all, which minimizes the overhead for transferring state information between nodes or eliminates it altogether.

The process of recovering from a fault when using passive replication is illustrated in Figure 4 and consists of three main steps: fault detection (discussed in Section V, primary selection and service distribution.

*1) Node selection:* After it has been established that a certain functionality is not available any longer due to a failure in the primary replica, a new service provider must be selected. After this selection phase, one or several nodes become service providers. Several approaches to how the selection is performed have been proposed. We differentiate them according to who makes the decision on which party should become a service provider.

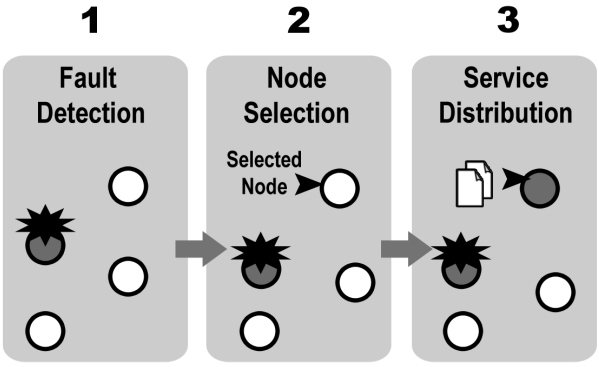


Fig. 4. Steps to recover from a failure in a passive replication mode

Self-organization techniques have proven to increase the reliability and fault-tolerance of distributed systems (cf. [46]). In the extreme case, each node makes an individual decision (possibly taking information from its neighbors into account). Or, local groups of nodes work together, coordinating their actions. On the other end, hierarchical systems assign tasks in a top-down manner. We discuss these options in turn.

*a) Self election:* In LEACH [15], nodes periodically execute a probabilistic algorithm to establish whether they should serve as clusterhead to their neighbors. In this probabilistic rotation system, nodes keep changing their role in the network. When a clusterhead node fails, it will take only one rotation period until another node starts providing the functionality of the failed or absent node.

Role assignment algorithms determine which of a certain role, such as coverage, clustering, and in-network aggregation, should be assumed by a node. In [9], a deterministic algorithm for autonomous role assignment is proposed that takes into account properties of the node such as battery status and location, but also its neighborhood and the roles chosen by neighboring nodes. This facilitates the localized self-configuration of a sensor network and can reestablish service provisioning if executed after some nodes have failed.

*b) Group election:* In [12], a reallocation of nodes that were part of a cluster that suffered a clusterhead failure is proposed. The clusterhead, called gateway, is considered to be a resourceful node. The solution presented considers that all the gateways in the network maintain a list of the nodes that are currently in their cluster and another backup list of nodes that could become part of their cluster. When a gateway fails, the nodes from its cluster are reallocated to the other gateways that have the nodes in their backup lists. If more than one gateway has a specific node in its backup list the node is assigned to the clusterhead that has the smallest communication cost.

*c) Hierarchical election:* In a hierarchical election, a coordinator selects the new primary node. This applies to the rebuilding of routing paths [37] as well as the selection of a new clusterhead [13]. The former describes an algorithm to select the node that is closest to the base station. The latter approach applies fuzzy logic in the base station to select which node will become a clusterhead. This algorithm makes use of a fuzzy descriptor, the node concentration, energy level in each node and its centrality with respect to the entire cluster.

Although these centralized algorithms could perform a better selection of the nodes than a local algorithm due to its global view of the network, such approaches require that nodes send periodically messages to the base station.

*2) Service Distribution:* During this phase, nodes elected to become service providers must activate the service. In some cases the service is already available on the nodes and a simple configuration change to inform the node that this service should be activated is required. However in some cases, for instance when nodes do not have enough memory to store the code of all potential services, it is necessary to inject code into the node through some technique. There are different techniques that can be used for service distribution: completely reprogramming the node, sending entire blocks of executable code, or sending small pieces of code such as scripts.

*a) Pre-Copy:* Pre-copying as described in [9], consists in making the code of all services available on all nodes before deployment. This allows nodes to change their behavior according to the role that they are assigned to.

*b) Code distribution:* Several approaches have been proposed for disseminating code throughout the network. Maté [21] is an example for a bytecode interpreter for TinyOS where code is broken into capsules of 24 instructions. These capsules can be distributed through the network and installed on nodes, which start to execute the new code. Agilla [8] is a Maté-based mobile agent middleware for programming wireless sensor networks. These mobile agents can be programmed to move through the network or replicate themselves to other nodes according to changes in the environment.

Impala [23] is a middleware for sensor networks that supports software updates and on-the-fly application adaptation. Unlike Maté, the focus of Impala is networks that have a high degree of mobility, which can lead to long delays until an update is finished. While Maté stops the execution of an application until the update is finished, Impala processes ongoing software updates in parallel.

*c) Remote Execution:* On the one hand code migration is an approach that reduces the amount of memory required in the entire network since not all nodes need to have the application pre-installed, on the other hand it consumes energy on the nodes exchanging the code and is susceptible to link failures, which could cause long delays until the code update is completed. Remote Execution [34], [32] is an alternative approach where low-power devices transfer tasks to more powerful devices without transferring the entire application code. Instead, only the required state information is transmitted. Such an approach is especially suited for heterogeneous sensor networks with at least some resourceful nodes.

A hybrid approach between code migration and remote

execution is proposed in [25], where the application code is copied to another node when the battery level reaches a first threshold. As soon as the battery reaches a critical level, the execution state is transferred and control is handed to the remote node. This allows for the full usage of the available energy resources, since control is handed over right before a node fails.

## VII. Classification and Evaluation

As presented in sections IV, V and VI, the work that deals with the unreliable nature of sensor networks is quite scattered. For the successful application of such techniques, it is necessary to understand in which cases they can be applied and what their shortcomings are. We therefore give an overview of the existing techniques in Tables I and II.

We classify them according to the types of faults they are able to detect and to recover from. This classification is intended to provide a lookup table for future applications that search for detection and recovery techniques to specific types of faults in WSN.

Each of the detection methods has its advantages and drawbacks: on the one hand self-detection mechanisms, as proposed in [25], involve no communication costs except for announcing that a fault has been detected. On the other hand, sudden crash failures cannot be detected in this way. Group detection mechanisms, where nodes monitor each other, allow the detection of crash failures. Such mechanisms impose higher costs due to the coordination of nodes. Additionally, the use of encryption is often impracticable, since this would hamper other nodes observing the contents of messages [26]. Finally, in a hierarchical detection approach, such as [37], most of the communication and coordination costs can be shifted to a more powerful device, nevertheless this approach requires a heterogeneous network which is not a valid assumption in all applications.

Table I shows that most of the solutions discussed in the literature focus on individual types of faults. To our knowledge, the approach proposed in [26] is the only one that currently detects the five types of faults used in our classification, with the restriction that the detection of value faults is limited to forwarded messages. Content analysis, as described in [18], could extend its applicability, e.g. to aggregation.

Table II presents the classification of the solutions with respect to the fault recovery mechanism applied. The fault recovery mechanisms analyzed can be divided into two main branches: passive and active replication. In the case of active replication the common approach is to remove the node from the route path or ignore its data as in [26][25][14][37]. Nevertheless, this implies that nodes must run the same applications, which imposes higher energy consumption on the network. Passive replication on the other hand only initiates a backup copy when nodes suffer failures, thereby reducing the energy costs during normal operation but with potentially high cost if a code is updated or if many messages have to be exchanged to select a new primary node.

## VIII. Conclusions

In this paper we provided a thorough investigation of faults that occured in real WSN deployments. This concise investigation provides a valuable knowledge input for future application to prevent the same kind of issues from happening. By focusing only on the faults, the lessons learned from the different deployments can be used by any application even if the investigated trial had a different research focus.

We proposed a taxonomy to classify faults and failures that occur in WSN. We also studied the problem of fault detection and recovery, surveying the different techniques currently applied in WSN research. A classification of the available fault tolerance techniques for wireless sensor networks has been proposed considering the various mechanisms adopted by the solutions. To our knowledge this is the first work that provides a concise survey and classification in this area.

Through the classification proposed it is possible to compare the different solutions identifying the strong and weak points of each of them. This allows for a correct selection of the techniques that are more suitable to specific applications. By applying our classification we were able to verify that current approaches provide mechanisms for overcoming faults in sensor networks only in specific scenarios and applications. However, no approach provides extensive fault tolerance support covering all types of faults that a node is exposed to.

We identified that link and path faults are a common problem in real WSN deployments. Hence, a routing mechanism must be able to react quickly to changes in the local network topology in order to efficiently forward messages. Nevertheless, most routing algorithms for sensor networks assume a *success rate* for message delivery (under "normal" operating conditions, i.e. not under heavy load), acknowledging that "perfect" reliability is not achievable. Therefore applications must analyze the tradeoff between reliability and battery consumption.

We also identified that a common problem in real WSN deployment is the selection of packaging. Although the packaging does not play an active role in data gathering and processing future applications should consider investing more in the selection of the cases to guarantee a successful trial.

In order to apply sensor networks in safety critical applications, security threats must be addressed during all operational phases of a fault tolerant system. Most current approaches do not include security measures, which opens an opportunity for further research in this field.

## References


[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A Survey on Sensor Networks. *IEEE Communications Magazine*, 40:102–114, August 2002.

[2] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi. A Discrete-Time Battery Model for High-Level Power Estimation. In *Proceeding of the Design, Automation and Test in Europe Conference and Exhibition 2000*, pages 35–39, 2000.

[3] A. Birolini. *Quality and Reliability of Technical Systems:theory, practice, management*. Springer, 1997.

[4] J. L. Bredin, E. D. Demaine, M. Hajiaghayi, and D. Rus. Deploying Sensor Networks with Guaranteed Capacity and Fault Tolerance. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 309–319, 2005.

[5] D. Desovski, Y. Liu, and B. Cukic. Linear randomized voting algorithm for fault tolerant sensor fusion and the corresponding reliability model. In *IEEE International Symposium on Systems Engineering*, pages 153–162, October 2005.

[6] M. Ding, D. Chen, K. Xing, and X. Cheng. Localized fault-tolerant event boundary detection in sensor networks. In *INFOCOM*, 2005.

[7] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: scalable coordination in sensor networks. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 263–270, New York, NY, USA, 1999. ACM Press.

[8] C.-L. Fok, G.-C. Roman, and C. Lu. Mobile Agent Middleware for Sensor Networks: an Application Case Study. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN)*, 2005.

[9] C. Frank and K. Römer. Algorithms for Generic Role Assignment in Wireless Sensor Networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 230–242, 2005.

[10] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-Resilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks. *Mobile Computing and Communications Review*, 1(2), 1997.

[11] R. Guerraoui and A. Schiper. Fault-Tolerance by Replication in Distributed Systems. In *Proceedings of the 1996 Ada-Europe International Conference on Reliable Software Technologies*, pages 38–57, 1996.

[12] G. Gupta and M. Younis. Fault-Tolerant Clustering of Wireless Sensor Networks. *Wireless Communications and Networking*, 3:1579–1584, 2003.

[13] I. Gupta, D. Riordan, and S. Sampalli. Cluster-Head Election Using Fuzzy Logic for Wireless Sensor Networks. In *Proceedings of the 3rd Annual Communication Networks and Services Research Conference*, pages 255–260, 2005.

[14] S. Harte and A. Rahman. Fault Tolerance in Sensor Networks Using Self-Diagnosing Sensor Nodes. In *The IEE International Workshop on Intelligent Enviroment*, pages 7–12, June 2005.

[15] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, volume 8, page 8020, 2000.

[16] C.-Y. Koo. Broadcast in Radio Networks Tolerating Byzantine Adversarial Behavior. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 275–282, 2004.

[17] F. Koushanfar, M. Potkonjak, and A. Sangiovanni-Vincentell. Fault Tolerance Techniques for Wireless Ad hoc Sensor Networks. In *Proceedings of IEEE Sensors*, volume 2, pages 1491–1496, 2002.

[18] B. Krishnamachari and S. Iyengar. Distributed Bayesian Algorithms for Fault-Tolerant Event Region Detection in Wireless Sensor Networks. *IEEE Transactions on Computers*, 53:241–250, March 2004.

[19] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4:382–401, 1982.

[20] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In *IPDPS 20th International Parallel and Distributed Processing Symposium*, 2006.

[21] P. Levis and D. Culler. Maté: A Tiny Virtual Machine for Sensor Networks. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 85–95, New York, NY, USA, 2002. ACM Press.

[22] N. Li and J. C. Hou. FLSS: A Fault-Tolerant Topology Control Algorithm for Wireless Networks. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, pages 275–286, 2004.

[23] T. Liu and M. Martonosi. Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems. In *PPoPP '03: Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 107–118, New York, NY, USA, 2003. ACM Press.

[24] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, New York, NY, USA, 2002. ACM Press.

[25] D. Marculescu, N. H. Zamora, P. Stanley-Marbell, and R. Marculescu. Fault-Tolerant Techniques for Ambient Intelligent Distributed Systems. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 348, 2003.

[26] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating Routing Misbehavior in Mobile Ad hoc Networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 255–265, 2000.

[27] K. Martinez, P. Padhy, A. Riddoch, H. Ong, and J. Hart. Glacial environment monitoring using sensor networks. In *REALWSN' 05*, 2005.

[28] K. Marzullo. Tolerating failures of continuous-valued sensors. *ACM Transactions on Computer Systems*, 8(4):284–304, 1990.

[29] D. Rakhmatov and S. B. Vrudhula. Time-to-Failure Estimation for Batteries in Portable Electronic Systems. In *Proceedings of the 2001 international symposium on Low power electronics and design*, pages 88–91, 2001.

[30] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy: a debugging system for sensor networks. In *IEEE International Conference on Local Computer Networks*, 2004.

[31] M. Ringwald, K. Römer, and A. Vitaletti. Snif: Sensor network inspection framework. Technical Report 535, ETH Zürich, Institute for Pervasive Computing, October 2006.

[32] P. Rong and M. Pedram. Extending the lifetime of a network of battery-powered mobile devices by remote processing: A markovian decision-based approach. In *DAC '03: Proceedings of the 40th conference on Design automation*, pages 906–911, New York, NY, USA, 2003. ACM Press.

[33] S. Rost and H. Balakrishnan. Memento: A health monitoring system for wireless sensor networks. In *SECON*, 2006.

[34] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning. The Remote Processing Framework for Portable Computer Power Saving. In *SAC '99: Proceedings of the 1999 ACM symposium on Applied computing*, pages 365–372, New York, NY, USA, 1999. ACM Press.

[35] L. B. Ruiz, I. G. Siqueira, L. B. e Oliveira, H. C. Wong, J. M. S. Nogueira, and A. A. F. Loureiro. Fault Management in Event-driven Wireless Sensor Networks. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 149–156, June 2004.

[36] T. Schmid, H. Dubois-Ferrière, and M. Vetterli. Sensorscope: Experiences with a wireless building monitoring sensor network. In *REALWSN' 05*, 2005.

[37] J. Staddon, D. Balfanz, and G. Durfee. Efficient Tracing of Failed nodes in Sensor Networks. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 122–130, 2002.

[38] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 214–226, New York, NY, USA, 2004. ACM Press.

[39] R. Szewczyk, J. Polastre, A. M. Mainwaring, and D. E. Culler. Lessons from a sensor network expedition. In *EWSN*, pages 307–322, 2004.

[40] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[41] J. Tateson, C. Roadknight, A. Gonzalez, T. Khan, S. Fitz, I. Henning, N. Boyd, and C. Vincent. Real world issues in deploying a wireless sensor network for oceanography. In *RealWSN' 05*, 2005.

[42] J. Thelen, D. Goense, and K. Langendoen. Radio wave propagation in potato fields. In *First Workshop on Wireless Network Measurement*, 2005.

[43] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 51–63, New York, NY, USA, 2005. ACM Press.

[44] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, 10:18–25, 2006.

[45] A. D. Wood and J. A. Stankovic. Denial of Service in Sensor Networks. *IEEE Computer*, 35:54–62, 2002.

[46] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, 2001.

TABLE I

CLASSIFICATION OF FAULT DETECTION MECHANISMS

| | Detect Faults | | | | | Fault Detection Mechanism | | |
|---|---|---|---|---|---|---|---|---|
| | Crash | Omission | Timeout | Value | Arbitrary | Self-Detection | Group Detection | Hierarchical Detection |
| [26] | X | X | X Forwarded packages | X Forwarded packages | X | | X Eavesdropping | |
| [25] | X Battery exhaustion | | | | | X Battery lifetime estimation | | |
| [14] | | | | | X Impacts | X Detects impacts | | |
| [37] | X | X | X | | | | | X Base station detection |
| [8] | X High temperature | | | | | X Detects fire reaction | | |
| [18] | | | | X | | | | X Bayesian algorithm |
| [35] | X | X | X | | | | | X |
| [12] | X Gateway | X Gateway | X Gateway | | | | X Consensus between Gateways | |
| [33] | X | X | X | | | | | X In network detection |
| [30] | X | X | X | | | | | X Base station detection |
| [31] | X | X | X | | | | | X Base station detection |

TABLE II

CLASSIFICATION OF FAULT RECOVERY MECHANISMS

| | Fault Recovery | | | | | | |
|---|---|---|---|---|---|---|---|
| | Active Replication | Passive Replication | | | | | |
| | | Node Selection | | | Service Distribution | | |
| | Mechanism | Self Election | Group Election | Hierarchical Election | Pre-Copy | Code Distribution | Remote Execution |
| [26] | | | | X | X | | |
| [25] | | | | | | X | X |
| [14] | X Discard information from failed node | | | | X | | |
| [37] | | | | X | X | | |
| [8] | | | | X Node with mobile code makes the selection | | X | |
| [17] | X Heterogeneous backup scheme | | | | | | |
| [9] | | X Set of rules | | | X | | |
| [18] | - | - | - | - | - | - | - |
| [35] | - | - | - | - | - | - | - |
| [12] | | | X Allocation of nodes to the gateway that has the minimum communication cost | X | X | | |