Dipl.-Ing. Karima B. Hein, Bakk.techn.

# Assessing Event Detection Algorithms for Wireless Sensor Networks

**DISSERTATION**

zur Erlangung des akademischen Grades

Doktorin der technischen Wissenschaften

eingereicht an der

**Technischen Universität Graz**

Betreuer

Em.Univ.-Prof. Dipl.-Ing. Dr.techn. Reinhold Weiß

Institut für Technische Informatik
Vorstand: Univ.-Prof. Dipl.-Inform. Dr.sc.ETH Kay Uwe Römer

Fakultät für Elektrotechnik und Informationstechnik

Graz, Juli 2014

## EIDESSTATTLICHE ERKLÄRUNG

## *AFFIDAVIT*

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Dissertation identisch.

*I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral dissertation.*

Datum / Date

Unterschrift / Signature

*"If you cannot measure it, you cannot improve it."*

William Thomson, $1^{st}$ Baron Kelvin

*I dedicate this work to the memory*
*of my dear mother Christa Klamminger.*

## Abstract

Wireless sensor networks (WSNs) are distributed systems formed by an often large number of nodes that work in an autonomous fashion. A common and crucial class of applications in the domain of WSNs is event detection, where nodes detect physical phenomena - the events - in their vicinity using the data sampled by their sensors. WSN installations are often confronted with harsh or even hostile environments. Unanticipated threats originating in the nodes' environments such as forces of nature, interactions with animals or even deliberate attacks can cause the nodes to malfunction and, consequently, to disturb the event detection process, thereby causing inaccurate reactions and erratic behavior of the WSN, that may lead to service interruptions and possibly even to harmful subsequent actions.
The main contribution of this thesis is a methodology for the assessment of event detection algorithms (EDAs) by means of fault injection. The result of the assessment provides the application engineer (AE) with an additional decision criterion for choosing an EDA. The injected erroneous input data represent the effects of the environment on the nodes, so-called disturbances. By defining disturbances, the AE describes characteristics of faulty test data including e.g. contortion functions and associated fault probabilities. The AE simulates the EDAs under consideration in a scenario of predefined events and disturbances. By analyzing the resulting behavior of the EDAs using designated metrics, the AE attains the means to compare and rank the EDAs, thus incorporating a dependability aspect into the assessment. In addition, the AE can derive feedback from the evaluation results as to how an EDA can be enhanced and the evaluation results may also be employed for risk analysis. The experimental part of this thesis demonstrates the applicability of the presented methodology on two detailed case studies that compare two simple threshold-based EDAs. Additionally, the case studies illustrate how a leaky bucket counter can be used as a threshold mechanism for an EDA.

# Kurzfassung

Drahtlose Sensornetzwerke (WSNs) sind verteilte Systeme, die oft aus einer großen Menge von Knoten bestehen und autonom arbeiten. Eine verbreitete und wichtige Anwendungsklasse ist Event Detection (Ereignisdetektion). Die Knoten detektieren physikalische Phänomene - die Events - in ihrer Umgebung mit Hilfe der Messdaten ihrer Sensoren. WSNs werden oft in rauen oder auch feindseligen Umgebungen installiert. Unerwartete Bedrohungen aus der Umgebung der Knoten, wie z.B. Naturkräfte, schädigende Einwirkung durch Tiere oder auch bewusste Attacken, können dazu beitragen, dass die Knoten Fehlfunktionen aufweisen und dadurch den Detektionsprozess stören; weiters können inkorrekte Detektionsergebnisse falsche Reaktionen und erratisches Verhalten des WSNs bedingen, was zu Serviceausfällen und sogar zu schädlichen Maßnahmen führen kann.
Als Hauptbeitrag stellt dieser Dissertation eine Methodologie für die Bewertung von Event Detection Algorithms (EDAs) mittels Fehlerinjektion vor. Das Ergebnis dieses Bewertungsvorgangs liefert der testenden Person ein zusätzliches Entscheidungskriterium für die Auswahl eines passenden EDAs. Die fehlerbehafteten Inputdaten repräsentieren die Effekte der Umgebung auf die Knoten, sogenannte "Disturbances". Indem die testende Person Disturbances definiert, beschreibt sie Eigenschaften der fehlerbehafteten Testdaten, z.B. Verzerrungsfunktion und Fehlerwahrscheinlichkeiten. Die testende Person simuliert die in Betracht kommenden EDAs in einem Szenario mit vordefinierten Ereignissen und Disturbances. Mit Hilfe geeigneter Metriken analysiert die testende Person das resultierende Verhalten der EDAs und erhält so die Mittel, um die EDAs zu vergleichen und in Hinblick auf ihre Zuverlässigkeit zu reihen. Außerdem kann die testende Person Feedback aus den Evaluierungsergebnissen ableiten und dieses für die Verbesserung der EDAs zu verwenden; zusätzlich eignen sich die Ergebnisse auch für eine eventuelle Risikoanalyse. Der experimentelle Teil dieser Dissertation veranschaulicht die Anwendbarkeit der vorgestellten Methodologie an zwei detaillierten Fallstudien, die zwei einfache Schwellenwert-basierte EDAs vergleichen. Darüber hinaus zeigen die Fallstudien als weiteren Beitrag, wie ein Leaky Bucket Counter als Schwellenwert-Mechanismus für einen EDA verwendet werden kann.

## Extended Abstract

Wireless sensor networks (WSNs) are a well-established technology representing a popular approach to distributed and cooperative problem solving. An essential branch of the applications of WSNs is event detection, where the nodes, and possibly also the base station, draw conclusions about the behavior of their environment. Effects of events that take place in the nodes' surroundings, e.g. an explosion or an animal passing by, are reflected in the measurements taken by the nodes' sensors. Event detection algorithms (EDAs) are used for discovering special constellations, trends or patterns in those measurements in order to identify the occurrence of an event.

WSNs are often deployed in harsh environments where they are exposed to extreme, and often, unexpected conditions. Damaged casings or physical imperfections resulting from a bad manufacturing process contribute to the formation of weak points that can cause external influences to affect the system. Additionally, attackers can do physical harm to the system. Hence, nodes may malfunction and the nodes' activities may severely deviate from the anticipated behavior. With respect to the sensors, deviations include freezing, where the sensor continuously delivers the same value, delayed arrival of sensor values or general erratic behavior. In addition to the consequences of failures of the considered system's subsystems, long time effects of system deterioration can also be modeled. Long term effects are of interest as mission times for WSNs can stretch to several months. These circumstances motivate the analysis of EDAs by means of fault injection, a technique for the arbitrary insertion of faults in order to test a system.

Fault injection for WSNs has been used for the evaluation of routing protocols[1] and also for analyzing the reactions of different operating systems to bit flips[2]. This dissertation, however, considers fault injection that solely targets the input values processed by the node's application. Groundwork[3] to the subsequently presented methodology outlines a strategy to categorize nodes in being faulty or fault-free based on the event detection results of the nodes' neighbors and the nodes adjacent to the neighbors. The approach exploits the massive redundancy of large WSNs in combination with the knowledge of the nodes and their two-hop-neighborhood

[1] Friginal et al., "Using Performance, Energy Consumption, and Resilience Experimental Measures to Evaluate Routing Protocols for Ad Hoc Networks", *10th IEEE International Symposium on Network Computing and Applications, 2011.*

[2] Cinque et al., "Analyzing and Modeling the Failure Behavior of Wireless Sensor Networks Software under Errors", *IEEE International Wireless Communications and Mobile Computing Conference, 2012.*

[3] Hein and Weiss, "Minesweeper for Sensor Networks – Making Event Detection in Sensor Networks Dependable", *Twelfth IEEE International Conference on Computational Science and Engineering, 2009.*

to identify faulty nodes. As in the pioneering publication of this domain[4], this approach considers snapshots of one instant in time where every node produces only one sample.
The main contribution of this dissertation is a methodology for the assessment of EDAs by means of fault injection, thereby extending the range of decision criteria for the selection of an EDA[5]. Conventional criteria for the selection of an appropriate EDA encompass computational complexity along with resulting memory or power requirements and also performance related metrics such as the quality of the detection results based on anticipated data and timeliness of detection outcome. The additional criterion is derived from analyzing the EDA's performance and behavior when being confronted with erroneous data. This approach supports the design process by making EDAs comparable with respect to dependability and performance attributes, thereby easing the application engineer's task of choosing between available EDAs. The innovative core idea is to take into account the resiliency to faults and to resulting service failures that are caused by external influences. The application engineer can derive feedback from the evaluation results as to how an EDA can be enhanced and in addition, the evaluation results may also be employed for risk analysis. As an additional part of the contribution this thesis illustrates how a leaky bucket counter can be used as a threshold mechanism for an EDA[6].
Seminal work in dependable computing[7] defines that a service failure occurs when *"at least one (or more) external state of the system deviates from the correct service state"*. A fault is the *"adjudged or hypothesized cause of an error"*, the error being the above mentioned deviation of the system. The faults considered in this work exclusively include value faults in the sensor nodes' applications. The modeled behavior corresponds to that of a sensor that delivers incorrect data to the EDA. The EDAs are exposed to a set of faults, which are linked to a physical incident e.g. water that enters the casing or a local heat source. The effects of the incident are modeled as so-called *disturbances*. A disturbance is characterized by the attributes area, start time, duration, start position, direction and speed, the probability of the occurrence of a fault and the type of value contortion, i.e. the effect on the node's sensors.
The assessment of the EDAs is carried out by simulation: disturbances are modeled by a fault injection mechanism where the correct value is substituted with a contorted value. The contorted value is obtained via a contortion function. Examples for contortion functions are the substitution of the correct value with a random value or with zero. As long as a node is geographically covered by a disturbance,

[4]Krishnamachari and Iyengar, "Distributed Bayesian Algorithms for Fault-Tolerant Event Region Detection in Wireless Sensor Networks", *IEEE Transactions on Computers, 2004.*

[5]Hein et al., Analysis of Threshold–Based Event Detection Algorithms for Wireless Sensor Networks by Fault Injection, *IEEE 10th International Conference on Ubiquitous Intelligence and Computing, 2013.*

[6]Hein et al., "Using a Leaky Bucket Counter as an Advanced Threshold Mechanism for Event Detection in Wireless Sensor Networks", *Tenth Workshop on Intelligent Solutions in Embedded Systems 2012*

[7]Avižienis et al., "Basic Concepts and Taxonomy of Dependable and Secure Computing", *IEEE Transactions on Dependable and Secure Computing, 2004.*

value faults are being injected into the application. In an evaluation scenario, there can be multiple disturbances. Each disturbance is characterized by a combination of contortion function and probability of occurrence. During evaluation, vulnerabilities of a seemingly optimal EDA can be uncovered, thereby motivating the choice of a different EDA. Fault injection campaigns, i.e. sets of disturbances, are designed in a way, so that the degree of resilience to a certain kind of fault can be determined, e.g. mainly very high numbers, fluctuating numbers or only zeroes are injected. If the resilience is low, the EDA produces a detection result of poor quality for a specific kind of injected fault.
In two detailed case studies, this thesis demonstrates the applicability of the leaky bucket counter for event detection. The proposed methodology is applied to compare the suitability of the EDA based on a leaky bucket counter to that of an EDA based on a moving average. The evaluations in the two case studies that are presented are conducted using the infrastructure of the RIPLECS project[8], an installation of six sensor nodes that sample illuminance in order to detect light events. The light level of two different kinds of sources of light can be adjusted remotely. Data sampled by this installation is used for offline fault injection and the resulting data is fed to the different EDAs which are to be analyzed in order to compare the EDA's reaction to this erroneous data. A detailed analysis of both case studies illustrates the advantages and disadvantages of both EDAs.

[8]Hörmann et al., "Using a remote lab for teaching energy harvesting enhanced wireless sensor networks", *IEEE Global Engineering Education Conference, 2013.*

## Acknowledgments

First of all, I want to thank Prof. Weiß for providing me with the opportunity of conducting my thesis at the Institute for Technical Informatics (ITI) and for supervising and guiding me. In addition, I extend my gratitude to Prof. Rinner for becoming my secondary assessor and especially for the valuable feedback I received on my visit to Klagenfurt. My thanks also go to Prof. Römer for granting me access to the institute's infrastructure while completing this thesis.

It was a pleasure to work at ITI, my appreciation goes to the entire team and especially to Leander and Philipp. I also want to emphasize the excellent cooperation with Bertl and Silvia who were always ready to provide me with much appreciated support. My thanks also go to Philipp Maierl for his commitment to the project.

Finally, I want to address my family. For taking care of my sons in such a warm-hearted way and for many, many transfer activities, I am deeply grateful to my mother-in-law Sophie, father-in-law Hans as well as to my father Franz. Last, but certainly not least, thank you, Daniel, for supporting me in my work, especially late at night. Your support was essential for the culmination of this dissertation.

*Thank you all!*

Karima B. Hein Graz, July 2014

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| AE | Application Engineer |
| EDA | Event Detection Algorithm |
| GUI | Graphical User Interface |
| LBC | Leaky Bucket Counter |
| MA | Moving Average |
| RSSI | Received Signal Strength Indicator |
| WSN | Wireless Sensor Network |

# Chapter 1

# Introduction

## 1.1 Background

Wireless sensor networks (WSNs) are a well-established and widely used technology, with the first seminal work surveying WSNs [ASSC02] already dating more than a decade. WSNs are networks consisting of spatially distributed devices, so-called nodes. As the name suggests, the nodes are equipped with various kinds of sensors, usually several sensors are combined onto one single sensor board. The nodes sample their sensors periodically in order to gather information about their environment. After optionally preprocessing the gathered values, the nodes either take predefined actions or forward the data to a higher instance, e.g. a clusterhead, or even to the base station. The base station is the user's interface for communicating commands to the nodes as well as for retrieving data from the nodes.

There are a wide variety of areas of application for WSNs, virtually any location for any purpose can be monitored, ranging from birds' habitats [MCP+02] and water salinity [HDCJ12] over parameters used for precision agriculture [LBV06] to also include the activity of volcanoes [HSX+12]. In addition to these nature-oriented sensing tasks, WSNs are also well suited for practical applications including structural health monitoring [AGPB+12] of buildings and bridges and for surveillance systems [SCL08].

The actual activities of the nodes range from simple data gathering and data forwarding to much more complex tasks, e.g. autonomously detecting residential fires using a distributed approach [BMP+10]. The type of application this thesis considers is event detection. That is, the nodes discover, mostly in an autonomous way, if an event is occurring. The knowledge about the manifestations of the events of interest are either provided as prior knowledge or the nodes acquire this information by means of learning mechanisms. The application that performs the event detection uses this knowledge every time the node wakes up and samples the sensors to determine whether an event takes place or not. The combination of a wide range of mathematical operations and the application of advanced techniques like machine learning and pattern matching to the sensory data form the actual detection procedure. In other words, the event detection algorithm (EDA) is applied to the

data that the node gathers from its sensors. How the nodes process the detection results, i.e. if they simply pass it on or if they trigger any consequential actions is beyond the scope of this thesis.

In addition to any decision values a node determines, the node may also incorporate data received from neighboring nodes to improve or to refine its local detection result. If it is possible to incorporate the necessary data exchange for node cooperation into the existing messages without introducing additional traffic, large-scale WSNs are perfect candidates for such a cooperative scheme. The sheer number of nodes in a large-scale WSN also comes with the advantage of inherent redundancy.

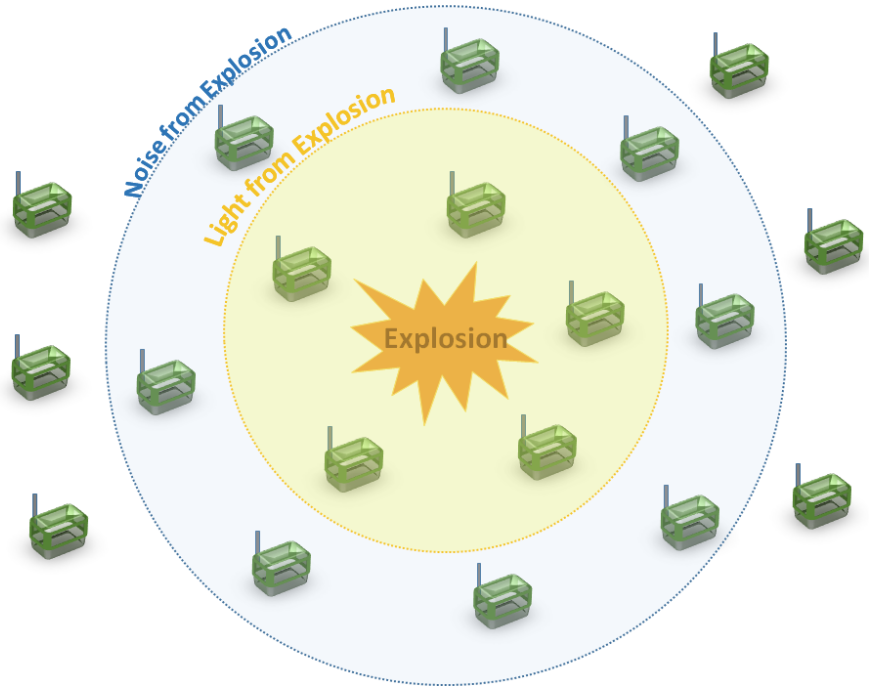



Figure 1.1: Detection of an explosion by WSN nodes. The inner nodes within the yellow area detect noise and light of the explosion, the nodes situated in the blue area only detect the noise from the explosion and the rest of the nodes do not detect anything.

Figure 1.1 illustrates how an event, an explosion in this case, can be detected by the nodes surrounding the event source. The nodes closest to the explosion, situated inside the yellow disk, sense the light from the explosion very well. The nodes situated a little farther away, located in the area with the blue background, do not sense enough of the explosion's light to detect a light event, but they are close enough to detect the explosion based on the data sampled via their microphones. The nodes detecting the light event can of course also detect the noise of the explosion. Nodes situated outside of the yellow and blue areas do not detect any effects of

the explosion. That is, the central nodes can detect the explosion by sampling two features, and the nodes some distance from the event origin can use one feature for detection.

## 1.2 Motivation

WSNs that are deployed outside often have to face harsh environments, such as an arctic environment [CB10], where the nodes are exposed to extreme and potentially even hostile conditions. Temperatures can be very high as well as extremely low and the same applies to humidity levels. Last, but not least, the nodes can be damaged by falling objects, by animals or as a result of a bad manufacturing process. Even a small crack in the node's casing may lead to corrosion of the node's circuits resulting in erratic behavior. The typical rather long mission times during which the WSN is unattended usually do not provide opportunities for repair or maintenance, resulting in deterioration of the node's electronic components. An additional drawback is that the aforementioned circumstances may arise unexpectedly. Section 2 presents a number of examples of sensor malfunction due to external influences, including nodes producing sensor readings that are out of their sensors' operating ranges [SPMC04], unanticipated node reboots and failing of sensors [BISV08], [IBS+10] and even loss of communication due to vegetation surrounding the nodes [CWJ+10], [HDCJ12].

In a similar manner, indoor installations of WSNs also face threats that occur rather unexpectedly. Examples include accidental wetting of the nodes either by the cleaning staff or due to burst pipes, damage due to collisions with large objects that are moved around, or masking of a node's sensors. A frequent and underestimated problem is the variation in link quality caused by people moving around and the use of electrical appliances [WBL12], [BKM+12].

Regardless of the WSN installation location, an additional serious threat is a physical attack by a hostile party. The resulting problem of such an attack and the scenarios mentioned earlier is that an affected node's sensor may deliver incorrect data to the application running on the node. Zhang et al. [ZMH10] argue that unattended WSNs installed in harsh environments will inevitably show some kind of malfunction, *"which may result in noisy, faulty, missing and redundant data"*. Faulty data may entail incorrect detection outcomes resulting in inappropriate or even harmful follow-up actions. As an illustrative example, Figure 1.2 demonstrates how an illegally operated water boiler causes the sensor nodes mounted on the ceiling above the water boiler to malfunction. Due to the repeated exposure to the unexpected high humidity levels and high temperature induced by the steam, the three nodes with the red cover and the red 'x' on their antennas start to malfunction. As in this setting, operation of the water boiler is not permitted in said room, this kind of disturbance was not anticipated by the engineer who installed the WSN. Due to the nodes' sensors malfunctioning, they cannot continue to detect the events they are required to detect.

The wide variety of unanticipated malfunctioning of sensor nodes, including

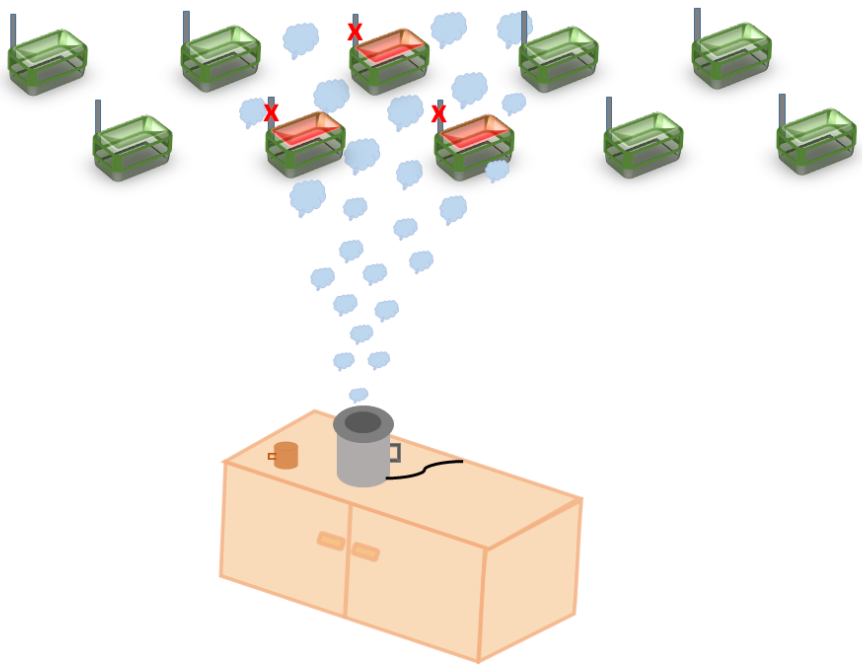


Figure 1.2: Steam from a water boiler causes three of the sensor nodes mounted on the ceiling to malfunction

any destructive influences of the anticipated events, can pose a severe threat to the objectives of any WSN installation; especially for long mission times where no repair or maintenance is undertaken. This knowledge can be used to an advantage by taking into account the behavior of the WSN and its nodes in the case of active faults when an EDA for a specific application is to be chosen. That is, beside common selection criteria that includes quality of detection results in the fault-free case, computational complexity and memory or power requirements; the EDAs' behaviors when being confronted with erroneous input should also be considered.

## 1.3 Scientific focus

This thesis focuses on enhancing the process of choosing one of several EDAs to be used as an application in a WSN. The central research question of this thesis can be broken down into two sub questions:

1. *How can a sensor node's environment affect the correctness of the data that is sampled by the node's sensors and passed on to the EDA?*
2. *How can the impact of incorrect sensor values on an EDA's service quality be quantified in order to be used in the selection process of an EDA?*

An extensive literature review addresses the first part of the research question. As a result, Chapter 2 gives an overview of reported cases of incorrect sensor data in WSN installations due to environmental influences. The chapter discusses the consequences to the affected applications as well as the consequences to the missions' objectives.

In response to the second part of the research question, this thesis introduces as the main contribution a methodology for assessing EDAs for WSNs [HHW13]. The methodology describes how to generate faulty input data for the EDAs under consideration, i.e. how to perform fault injection. In addition, the methodology also shows how the resulting output of the EDAs can be evaluated and compared and introduces relevant metrics for this purpose. By considering the EDAs' reactions to faulty data, the EDAs can be compared with respect to dependability-related attributes, e.g. the number of false positives or the number of faults that actually become active during the evaluation procedure. Finally, the examination of the evaluation results provides valuable feedback to the application engineer who actually choses the EDA. The evaluation might expose flaws in the implementation and reveal room for improvement. Preliminary work to this methodology [HW09] presents a strategy that exploits the massive redundancy of large WSNs, as motivated by seminal work in the domain [KI04]. The goal of this groundwork is to categorize nodes into being faulty or fault-free in a cooperative scheme based on the event detection results of the nodes and the detection results of their two-hop-neighborhood.

As an additional part of the contribution, the applicability of the proposed methodology is demonstrated in two comprehensive case studies. Moreover, the case studies illustrate how to use a leaky bucket counter as a threshold mechanism [HHW12] and compare this approach to an event detection algorithm based on a moving average.

## 1.4 Thesis outline

This thesis is organized into five main chapters. The content of each of the four chapters following this introduction is briefly described below:

- **Chapter 2** gives an introduction to the basic concepts in dependability and related schemes, along with the fundamental theory involved. In addition, this chapter surveys event detection algorithms in combination with existing approaches aiming to integrate dependability into WSNs. Furthermore, the chapter provides a detailed overview of reported malfunctions in WSNs due to environmental influences.

- **Chapter 3** presents the ideas and concepts of a methodology for assessing event detection algorithms for WSNs by fault injection. The analysis according to the elementary fault classes is reviewed and events and disturbances are described in detail along with the associated fault injection process.

- **Chapter 4** describes the infrastructure used for the case studies that were conducted. Moreover, this chapter introduces the leaky bucket counter as a threshold mechanism for event detection. Two case studies provide a detailed comparison of the leaky bucket counter and a moving average as thresholding mechanisms.
- **Chapter 5** concludes this thesis by summarizing the work done and the results achieved and gives an outlook for possible further work.

# Chapter 2

# Related Work and Problem Analysis

This chapter introduces the terminology and concepts used for describing the subsequently presented methodology for the assessment of EDAs. In particular, this chapter describes how faults evolve to become failures and discusses definitions and attributes of dependability along with the evaluation technique of fault injection. In addition, this chapter elaborates on related work. Attention is paid to reported cases of sensor malfunction as well as to existing approaches to integrating dependability related aspects and fault injection into the domain of WSNs.

## 2.1 Dependability, related concepts and fault injection

In order to define and explore the concept of *dependability* and related concepts, several basic terms have to be defined first. The following introduction to these basic concepts is based on the taxonomy presented by Avizienis et al. [ALRL04]. Moreover, this section introduces *resiliency*, a concept related to dependability and in addition, this section also discusses the principle of fault injection.

### 2.1.1 Dependability

#### Systems and services

Systems are entities that interact with each other. A system can be a piece of hardware or software or even a human, who is using another system, even an entire computer system. Systems are composed of sub-systems, which are referred to as components. Each component can be composed of further subcomponents. A component that cannot be partitioned any further is referred to as an atomic component. Each system has a boundary that separates the system from its environment, e.g. from the user or from the physical world. It is of vital importance to differentiate between a system's functional specification and the behavior of the system. The latter describes what the system actually does in order to implement the initial functional specification. The system thereby traverses a sequence of states

that constitute its behavior. Each system is a provider that supplies a service and the system's external state (or a sequence of external states) is perceivable to the user as the provided service. A system undergoes two phases that constitute the system's life cycle, the development phase and the use phase. The development phase starts with the presentation of the initial concept and ends with the system being ready to deliver its service. The use phase begins with the system starting to deliver services.

Both phases feature their own environments, each environment consists of a number of elements. Each of these elements provides openings were faults can be inserted into the system. The development environment consists of the physical world, human developers, development tools and production and test facilities. The use environment encompasses an even greater set of elements: physical world, administrators (incl. maintainers), users, providers that deliver services to the system, infrastructure and intruders (malicious entities). In the following, we will elaborate on the different kinds of faults that can be introduced by the different elements of the development environment and the use environment.

#### Threats to dependability: faults, errors and failures

The user desires a correct service, that is, the service implementing the system function. More often than not, this is not the case. In order to describe the different characteristics of malfunction, we introduce threats to dependability: faults, errors and failures along with the process of error propagation.

**Failures** If the provided service deviates from the expected, correct service, the system experiences a *service failure*, often referred to only as a failure. The form of the service's deviations can be categorized into service failure modes and can also be ranked according to failure severities. After a system has performed a transition from correct to incorrect service, the system may eventually recover. This recovery, a transition back to correct service, is referred to as service restoration. A service can be composed out of multiple functions and when the behavior of a subset of those functions deviates from correct service, the system experiences a partial failure. In many applications, it is preferable to have access to a degraded system rather than no functionality at all. Providing a degraded system instead of shutting down the entire system is referred to as graceful degradation.

**Errors** An error is the deviation from correct service itself, i.e. in the external state of at least one of the system's components. Only if the external state of this particular component becomes part of the system's external state, does the error cause a failure. Otherwise, only a failure of a component, which is not perceivable at the user interface, is present. This condition can be nicely illustrated when considering a triple modular redundancy (TMR) system: If one of the three modules yields an incorrect result, but the two other modules yield a correct result, the voter outputs the correct answer and no failure of the system occurs despite the prior existence of an error.

**Error propagation** The actual cause of an error is a fault. Depending on how a component is used, a fault is either active or dormant. Faults produce

errors as long as they are active. Dormant faults do not produce errors until they become active due to receiving appropriate input. Figure 2.1 illustrates how faults, errors and failures are related to each other. Upon activation, a fault produces an error that propagates through the component until it reaches the component's service interface where it causes a failure of the component. If the incorrect service delivered by Component A is used as input for Component B, this fault will cause an input error in Component B that, again, may evolve to be a failure if it reaches the service interface of Component B.

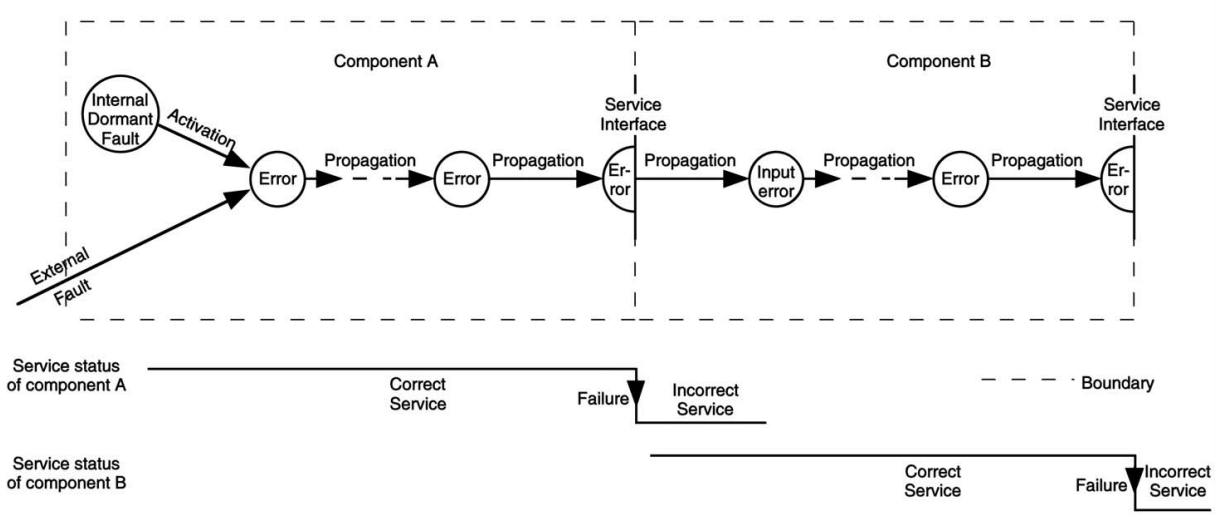



Figure 2.1: An error propagating from one component to the next, thereby evolving into a failure [ALRL04]

**Faults** Faults can be categorized according to a number of viewpoints. The taxonomy presented in [ALRL04] names eight elementary fault classes with two subclasses for each fault class, as illustrated in Figure 2.2. While most faults may be categorized according to multiple viewpoints, e.g. any development fault is either deliberate or non-deliberate, some fault classes cannot be combined. Fault classes that cannot be combined are e.g. natural faults as a result of an effect caused by natural phenomena and the objective of the fault. Avizienis et al. [ALRL04] identified 31 likely combinations of the elementary fault classes as depicted in Figure 2.3. It is noteworthy, that three main groups emerge from these likely combinations: development faults, physical faults (all faults related to hardware) and interaction faults.

### Defining dependability

Several definitions of dependability can be found in literature. A rather spartan definition is provided by the International Electrotechnical Commission, here, depend-

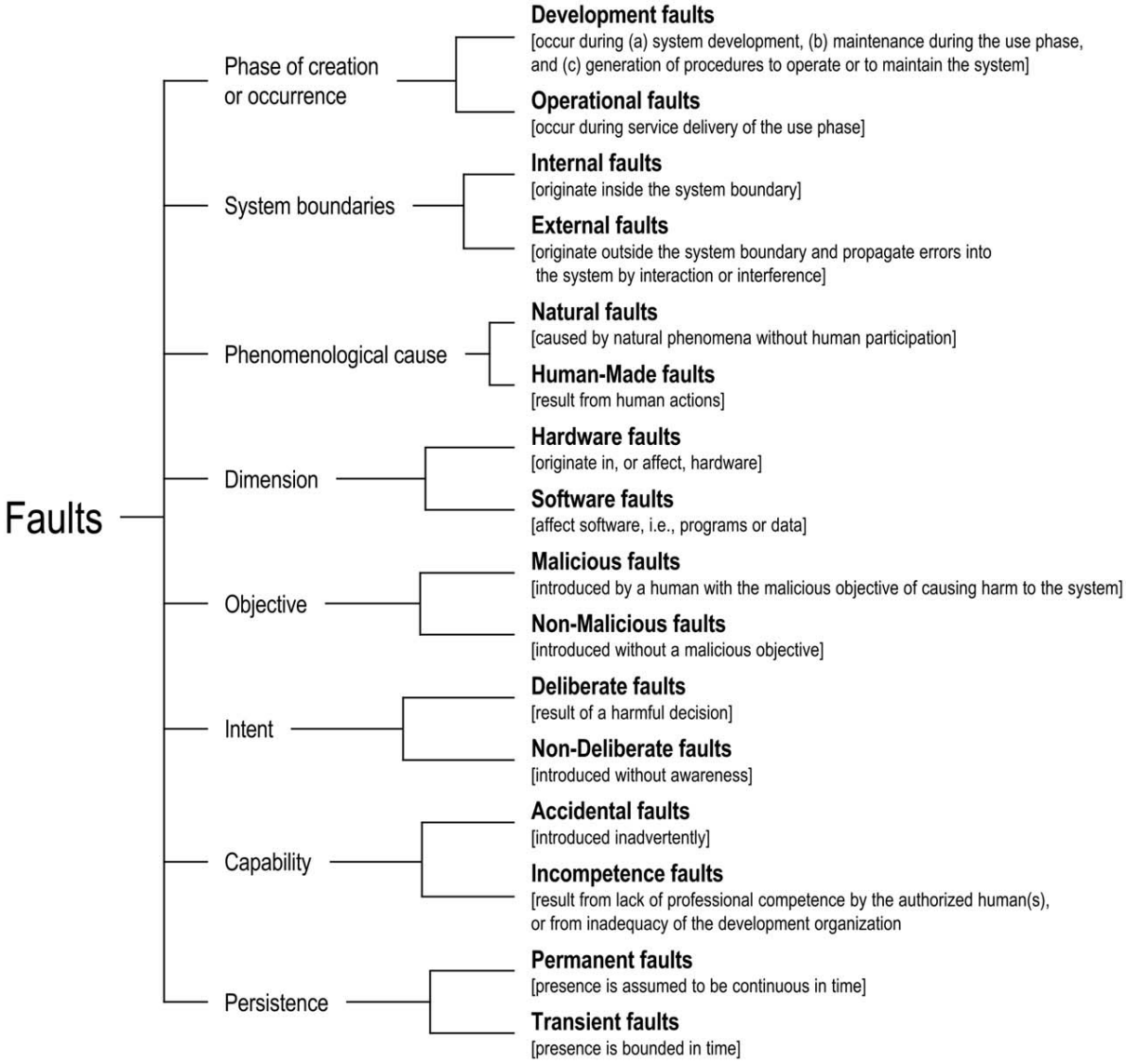



Figure 2.2: The eight elementary fault classes [ALRL04]

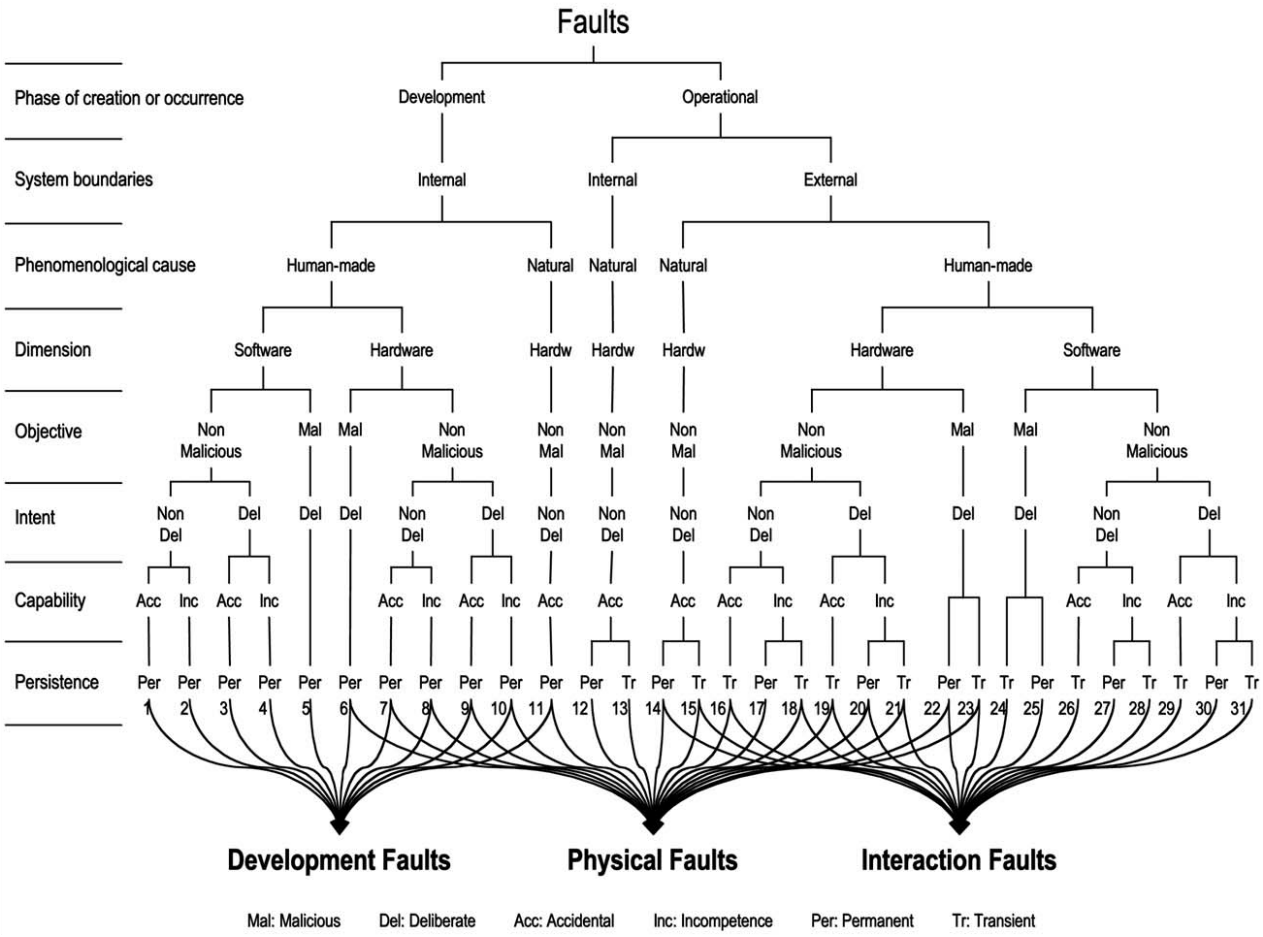



Figure 2.3: The combined fault classes [ALRL04]

ability is the *"ability to perform as and when required"* [IEC13]. A well established definition is provided in [ALRL04]:

> *"The dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable."*

Defining dependability in order to describe and assess system attributes has been further developed over the past few decades. Resulting from this long-term evolvement is the description of dependability as an integrating concept which encompasses five attributes as explained in the following.

**Availability** can be described mathematically as *"The availability of a system as a function of time, A(t), is the probability that the system is operational at the instant of time, t."* [SS92] or, in short, simply as *"Readiness for correct service"* [ALRL04].

**Reliability** also features two prominent definitions: Siewiorek et al. [SS92] defines *"The reliability of a system as a function of time, R(t), is the conditional*

*probability that the system has survived the interval [0,t], given that the system was operational at time t=0."*; Avizienis et al. [ALRL04] stick, again, to a short and precise description, namely *"Continuity of correct service"*.

**Safety** The *"absence of catastrophic consequences on the user(s) and the environment"* [ALRL04] simply refers to a friendly environment when life and limb are concerned.

**Integrity** as the *"absence of improper system alterations"* [ALRL04] expresses the demand to use the system in the intended way.

**Maintainability** provides the possibility to adapt the system to unforeseen situations and to correct development faults, it is the *"ability to undergo modifications and repairs"* [ALRL04].

Security is a term that, though often mentioned in combination with dependability, primarily is associated with secrets, cryptography or attacks, the latter being referred to as malicious faults. Security is related to dependability as illustrated in Figure 2.4. While dependability and security share the attributes availability and integrity, security also encompasses confidentiality, the *"absence of unauthorized disclosure of information"* [ALRL04]. The two concepts focus on different aspects, but as the associated attributes do not belong exclusively to one concept, the concepts are related to each other. Besides, every system has to meet different requirements and potentially places an emphasis on only a subset of the six attributes.

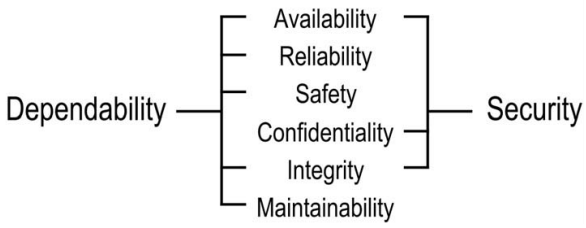



Figure 2.4: The attributes of dependability and security in comparison to each other [ALRL04], highlighting similarities and differences

The attributes of dependability and security can be expanded to include additional, so-called, secondary attributes. The purpose of these additional attributes is to refine and to specialize the primary attributes defined earlier. A representative of secondary attributes is robustness. *Robustness refers to dependability with respect to external faults* [ALRL04], this characterizes a system's reaction to a specific class of faults.

### Attaining dependability

There are four categories of means to attain dependability: fault prevention, fault tolerance, fault removal and fault forecasting. Fault prevention, obviously, is a major objective of the development phase. Fault tolerance is a composition of error detection and system recovery, for each of which there is a large pool of mechanisms

available. Fault removal takes place either during the development phase by means of verification, or during the use phase in the form of corrective maintenance.

The central means applied in the course of this dissertation is fault forecasting. Fault forecasting is an *evaluation of the system behavior with respect to fault occurrence or activation* [ALRL04]. There are two distinct aspects of evaluation, i.e. qualitative and quantitative. Quantitative evaluation is concerned with determining the probabilities to which extent certain predefined attributes, so-called measures, are satisfied. The more interesting aspect in this work is qualitative evaluation which deals with identifying, classifying and ranking the possible failure modes or with studying the respective event combinations that eventually could lead to a system failure. This last point targets the core idea of the proposed methodology in Section 3.

### 2.1.2 Dependability benchmarking

Another term related to fault forecasting and in particular to this dissertation is dependability benchmarking. Dependability benchmarking is a process where the measures of the behavior of the system under test are assessed in the presence of faults. Besides characterizing a system in terms of dependability, dependability benchmarking facilitates the comparison of different approaches to solving a given problem. When reusing software or choosing components off-the-shelf (COTS), a dependability benchmark can simplify the selection procedure.

Kanoun and Spainhower [KS08] describe a benchmark as *"an agreement (explicit or tacit) that is accepted by those who make and sell computers and those who purchase them"*, distinguishing it from existing validation and testing techniques. While emphasizing that dependability benchmarking is still *"a developing art"*, the authors mention several applications for dependability benchmarking, including to

- characterize the dependability of a component or a system, qualitatively or quantitatively
- track dependability evolution for successive versions of a product
- identify weak parts of a system, requiring more attention and perhaps needing some improvements by tuning a component to enhance its dependability, or by tuning the system architecture (e.g., adding fault tolerance) to ensure a suitable dependability level
- compare the dependability of alternative or competitive solutions according to one or several dependability attributes

Constantinesu presents an example for dependability benchmarking [Con05], where he investigates silent data corruption (SDC) in computer systems. SDC refers to undetected computational errors often due to intermittent and transient faults. SDC can be prevented or at least be reduced by using mechanisms such as error correcting codes or checksums. Although from 2005, this paper presents an interesting approach to dependability benchmarking using physical influences on

the device under test. The authors varied the environmental temperature slowly from -10°C to 70°C and also varied the operating voltage including the levels of -5%, -6%, -10%, +5%, +6% and +10% of nominal voltage. The test procedure lasted for 30 hours for each prototype. The evaluation shows that 90% of the reported errors, collected from ten prototype systems that were running a Linpack benchmark [Top14], were SDC events. Dependability benchmarking is closely related to fault injection as described in Section 2.1.4. Section 2.2.4 discusses related work on fault injection and dependability benchmarking in WSNs.

### 2.1.3 Resilience

*Resilience* [Avi13], or sometimes *resiliency*, is a well-known term in the domain of dependable computing that has been used for decades. J-C. Laprie, who is a co-author of [ALRL04] and who has been a well-known expert on dependable computing for years, defines resiliency as *"The persistence of dependability when facing changes"* [Lap08]. Laprie categorizes these changes by their nature, prospect and timing as depicted in Figure 2.5.

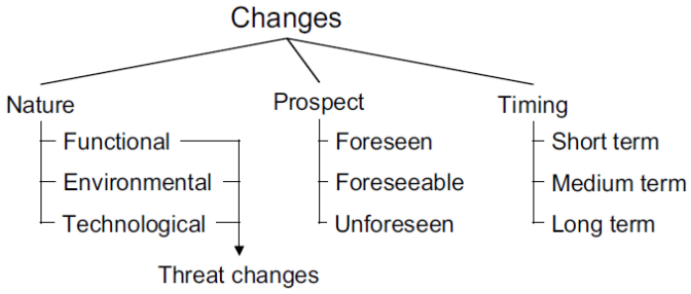



Figure 2.5: The attributes of resiliency [Lap08]

The changes considered in this work are classified as follows: Firstly, they are environmental because changes in the environment of the sensor node can have an effect on the node's behavior, i.e. when the node becomes damaged. Secondly, they are both foreseeable and unforeseen. Foreseeable, because it is expected that portions of nodes in a long-term installation of a WSN will eventually break. Some changes are unforeseen, because the complexity of a sensor node provides numerous subsystems and the effects of the individual failures of these subsystems may not be predictable. The timing this work focuses on is short and medium term, which Laprie characterizes as seconds to hours and hours to months, respectively.

Resiliency is further refined in [Avi13] as *"the ability of a system to sustain dependable operation in the presence of harmful changes that exceed the limits of expected threats or are not identified at all as expected threats in the system's dependability specification"*. There are two reasons indicating why a system may possess resilience, the first reason is that the system has been equipped with more defensive means than are needed to satisfy the dependability specification. The methodology proposed in this dissertation will show how the existence of such defensive means can be identified. The second reason is that resilience is provided by new features

that have been added by the designers. The proposed methodology will also illustrate how the application engineer can be motivated to enhance the EDA with suitable amendments.

### 2.1.4 Fault injection

Fault injection is a technique where faults are arbitrarily inserted into a computer system. The purpose of this procedure is to see how the system reacts to the injected faults in order to quantify the dependability of the system or device under test. Figure 2.6 [HTI97] illustrates the structure of a fault injection environment. Overseen by a controller, the fault injection system injects faults from a fault library into the target system, generates workload drawn from a workload library, monitors the target system's behavior and collects data for online or offline data analysis.

Fault injection can be divided into two main categories, hardware fault injection and software fault injection. Hardware fault injection uses additional hardware as a source of faults, examples are pin-level probes or an external source that produces heavy ion radiation. This type of fault injection has to be carried out on a physical prototype or even better on the actual system. Software fault injection on the other hand is generally cheaper because there is no need to purchase additional hardware. A major problem with software fault injection is that the additional code necessary for the injection procedure may have effects on the device under test besides those resulting from the injected faults. This is called the *probe effect* [CRS99], examples include different timing behavior or different CPU usage. Consequently, the resulting dependability evaluation may only be an approximation of the device under test's real behavior when the fault occurs. A third possibility is to combine hardware and software fault injection into one hybrid approach, combining the versatility of software fault injection and the accuracy of hardware monitoring [HTI97].

This dissertation solely deals with software fault injection. The manifestations of the considered faults are incorrect values gathered from the sensor situated on the node or received via the radio from a remote node. The term incorrect refers to the node's measurements of properties of its surroundings, for example temperature or light intensity that do not reflect the true environmental conditions of the node that houses the sensor. The reason for the node's behavior can either be internal, e.g. a loose connection, or be external, e.g. something covering the light sensor. The injection process is carried out by simulation, because it is simpler to feed the EDA contorted values than to actually cover the sensors or manipulate a node's connections. Moreover, not all kinds of faults can be created manually on an actual installation, examples include very low temperatures, rapid change between high and low temperatures or erratic sensor behavior due to loose couplings. Related work on FI in WSN will be discussed after reviewing the context of this work, i.e. WSNs and EDAs in Section 2.2.

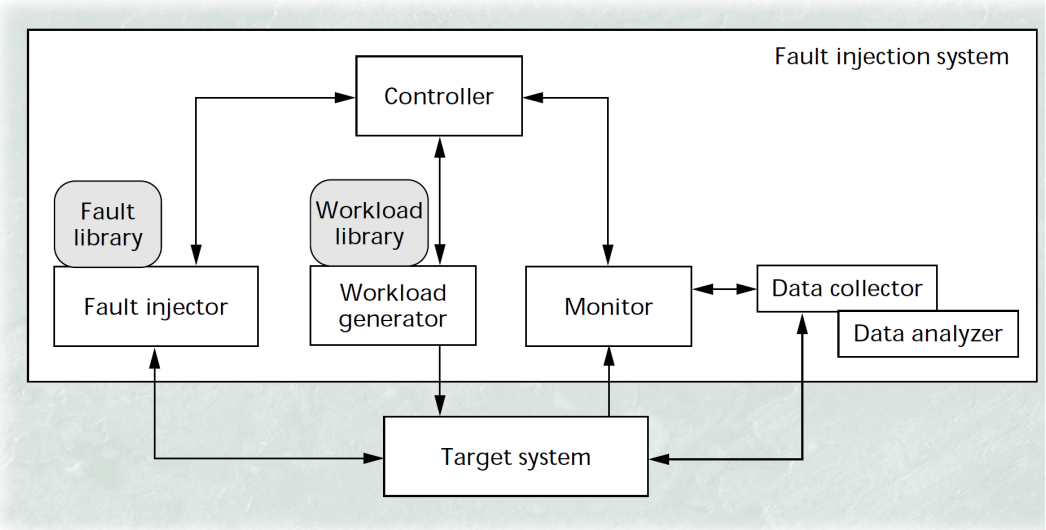



Figure 2.6: The basic components of a fault injection system [HTI97]

### 2.1.5 ISO/IEC 25045

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) published the ISO/IEC 25045 standard [ISO10a] in 2010. This standard is part of the so-called SQuaRE series, an acronym for *"Systems and software Quality Requirements and Evaluation"*. The SQuaRE series is divided into five divisions with each division consisting of a number of standards:

- Quality Management Division (2500n)
- Quality Model Division (2501n)
- Quality Measurement Division (2502n)
- Quality Requirements Division (2503n)
- Quality Evaluation Division (2504n)

One standard from the last division mentioned above is relevant for this dissertation, the Quality Evaluation Division. Although the considered standard ISO/IEC 25045 is applicable to transaction-based information systems, and systems where speedy recovery as well as ease of managing recovery are central points, it provides interesting elements that relate to the methodology presented in Section 3.

ISO/IEC 25045 is the *"Evaluation module for recoverability"*. Recoverability is a part of the product quality model defined in [ISO10b]. The product quality model consists of eight characteristics, these being functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability. Recoverability is a subcharacteristic of reliability. The standard defines two quality measures for recoverability: resiliency and automatic recovery index. Resiliency

corresponds to the already introduced term resilience. As the standard explicitly refers to resiliency, this term will be used throughout the remainder of this subsection. The contents of this standard had already, partially, been published by Coleman et al. [CLL+08] before [ISO10b] was available.

The two measures are determined by so-called *"disturbance injection"*, another name for fault injection that seems to have been chosen as the injected faults are referred to as disturbances. In fact, a disturbance is described as an *"operational fault"*, *"event"* or *"anything that could change the state of the system"*. A system's resiliency is calculated as

$$Resiliency = \frac{P_i}{P_{base}}$$

where $P_i$ is the number of transactions completed without error within a measurement interval of an injection slot where disturbance(s) were injected and $P_{base}$ is the number of transactions completed without error within the measurement interval of a baseline run. A baseline run is a golden run, which is a run where no faults, i.e. disturbances are injected. Five predefined, but not exclusive categories of disturbances are available:

- Unexpected shutdown
- Resource contention
- Loss of data
- Load resolution
- Restart failures

In order to calculate the second measure, the automatic recovery index, a set of questions is defined for each disturbance. The answers to those questions illustrate how well the system manages detecting, analyzing and resolving the disturbance. Using the answers, a score is calculated for each disturbance; the overall automatic recovery index and the overall Resiliency are then calculated by averaging the individual scores.

The procedure described in the standard consists of three phases: the baseline phase, the test phase and the check phase. The baseline phase is, as a fault free scenario, executed in order to have a performance baseline as a reference for the behavior of the system when being confronted with disturbances. During the test phase, the system operates while being exposed to disturbances. The check phase is partitioned into a number of injection slots and in every slot, exactly one disturbance takes place. The sequence of the disturbances is predefined. The check phase is used to ensure that the system is in a consistent state.

During an injection slot, not only is the disturbance injected, but also the system is expected to detect the disturbance, to initiate recovery and to optionally also apply repair actions. Finally, there is also a time interval reserved for returning to a steady state before the next disturbance is injected.

## 2.2 WSNs, EDAs and dependability in WSNs

The aim of this section is to give insight into the applicability of WSNs by describing WSN installations with reported malfunctions of sensors. In addition, a review of dependability aspects as well as related work on EDAs and fault injection complete the chapter.

### 2.2.1 Malfunctioning of sensors and WSNs

This section provides an outline of the evolution of WSNs over the past few years by describing a selection of projects that reported severe and unexpected conditions and incidents in combination with problems concerning the employed hardware.

**Habitat Monitoring on Great Duck Island** Szewcyk et al. [SPMC04] present a report from one of the first major outdoor deployments of WSNs. A total of 43 Mica sensor nodes [HC02] were used to investigate the nesting habits of the Leach's Storm Petrel (German *Wellenläufer*) for a duration of 123 days. As a result of this large scale experiment, the authors arrive at the conclusion that

> *"Sensor networks do not exist in isolation from their environment; they are embedded within it and greatly affected by it."*

Incorrect behavior observed includes erratic packet delivery, persistent sensor readings of 0°C, more than half of the nodes recording faulty temperature readings, abnormally large or very small humidity readings and even nodes producing sensor readings out of their sensors' operating ranges. The authors explain part of these unusual behaviors with permeable casings; humidity inside the casing would create a low-resistance path between the power supply terminals. The authors fail to find an explanation for the entire range of faulty behavior.

**Precision Agriculture** The authors of [LBV06] describe a multitude of problems occurring when installing a large scale WSN with about 100 nodes in the field. This particular application was about monitoring the microclimate in a potato field in order to collect data for precision agriculture. Highlights of the unanticipated problems encountered include dramatically reduced radio range when the antennas where unexpectedly covered by plant leaves, the fragile antennas came loose easily and for unknown reasons, healthy nodes would spontaneously reset once every 2-6 hours. As a consequence, during the yearlong study, a large part of the planned functionality did not come to fruition and the initial research question could not be answered.

**Self-Monitoring Sensor Nodes** The need for self-monitoring sensors is motivated by the observation that components used in low-cost sensor nodes do behave differently on different nodes as described in [FEDV08]. Reportedly, only on a subset of the identically constructed nodes, radio transmissions triggered the motion

detector. The authors managed to circumvent this problem by turning off the sensor during the times the radio was active. They did not, however, find the cause of the problem but developed an efficient self-test in order to identify the set of nodes exhibiting faulty behavior. The self-test is based on activators that trigger activities that are related to observed problems and probes that analyze the node's reaction to the activator.

**Glacier Monitoring** The importance of considering seemingly unimportant factors is underlined by the authors of [BISV08] and [IBS+10] as part of the still ongoing Sensorscope project [EPF13] at EPFL. Their work describes an exhaustive roadmap where the authors point out many potential problems on the way to successful WSN deployment. Although the authors did a very thorough job when designing their own WSN, an installation on a glacier, they still ran into several problems: due to extreme temperatures many sensors failed early, nodes rebooted frequently, remote control was limited. As a consequence, a large part of the resulting data was not usable.

**Oil Refinery Testbed: Ginseng Project** Pottner et al. [PWC+11] present an industrial testbed. This approach integrates a WSN and an existing enterprise resource management solution using a designated middleware. 16 T-Mote Sky nodes were installed in an oil refinery. The work focuses on establishing wireless communication. The authors discovered that the step-down converters for the power supply did not behave as was described in the data sheet and had to be replaced. In addition, some of the nodes would become unresponsive and would not accept new programming. These nodes were simply replaced by new ones. Similar to the step converters, the actual cause could, apparently, not be identified.

**Impact of Temperature on Signal Strength** Another article about the Ginseng project examines the influence of temperature on signal strength and link quality [BTV+10]. The authors report a considerable impact, that can lead to a loss of the SNW's connectivity, thereby showing as well as a potential to save energy during the night when, due to lower temperatures, less transmission power is needed. This work builds upon similar findings by Boano et al. [BBH+10] confirming prior experiments by Bannister et al. [BGG08].

**Sugar Farm Monitoring** The authors of [CWJ+10] and [HDCJ12], respectively, report experiences from monitoring underground water levels and water salinity at a sugar farm in Australia. The authors observed a nightly communication loss only during the time when the sugar cane was fully grown. This phenomenon is hypothesized to be caused by a layer of moist air on the cane that disrupted the channel through extreme multipathing. In addition, the authors also comment that there are complexities in radio propagation that they don't fully understand and, consequently, cannot remedy. The authors emphasize the importance of considering

*"deployment parameters such as terrain, humidity and antenna height when calculating the performance of radio links"*. Two nodes repeatedly hung and had to be reset by a watchdog timer, here, the authors suspect electrical interference from a near-by water pump motor to be the reason. Additional observations from other installations investigated in [CWJ+10] include that batteries can be overcharged due to the combination of a simplistic battery charging model and abundant solar power, that nodes deployed in a rainforest have to be revisited periodically in order to remove leaves, insect nests etc. Furthermore, vandalism is explicitly mentioned as a physical security challenge for outdoor WSNs.

**Link Properties in WSNs** With a large indoor testbed of more than 100 nodes the authors of [WBL12] investigate the suitability of the 868 MHz frequency band as an alternative to the popular 2.4GHz band. Using the 2.4GHz band comes with a crowded spectrum and the 868 MHz band is basically free from external interferences. The experiments expose a distinctive pattern in the link distribution, that, in accordance with [BKM+12], shows that the characteristics of the environment, i.e. the people moving around, are the main influence causing temporal variations in link quality.

**Badger Monitoring** The British Wildsensing project [DEM+12] monitored European badgers during an entire year in order to analyze their social colocation patterns. This installation employed a hybrid approach utilizing RFID and WSN. The casing of the sensor node housing a humidity sensor was identified to cause local condensation, thereby saturating the humidity sensor which, of course, led to incorrect measurements.

### 2.2.2 Dependability in WSNs

Work targeting dependability in WSNs focused initially on integrating fault tolerance into various aspects of WSNs. One of the first publications in this domain was presented by Koushanfar et al. [KPSV02]. This work introduces a mutual back-up approach for heterogeneous sensors in case of a sensor failure. In an illustrative example, the authors show how classification of unique objects can be performed with different sensor type configurations. Further work [KPSV03] proposes an online model-based testing technique for identifying faulty nodes with respect to the measurements delivered by the sensor. The basic concept is to analyze the impact of the readings of one sensor on the consistency of the result obtained from sensor fusion. This is done consecutively for each sensor. Any sensor, that improves the consistency of the obtained results when excluded from the sensor fusion, is considered to be faulty. Experiments were carried out on real sensor nodes, real faults were created by covering light sensors with opaque film. More work concerning dependability in WSNs with emphasis on fault injection and dependability benchmarking can be found in Section 2.2.4

### 2.2.3 Event detection in WSNs

#### Defining events and event detection

There are multiple approaches to defining events. The first of the two example definitions below emphasizes the locational aspect, i.e. the behavior of the environment, while the second definition focuses on characterization using the resulting sensor data.

Yang et al. [ZMH10] define an event as

> *"a particular phenomena [sic] that changes the real-world state"*

and Amato et al. [ACGV11] define an event as

> *"a condition expressed over a set of raw sampled data obtained by a group of sensors"*

The border between event detection and outlier detection is blurry, i.e. Yang et al. [ZMH10] argue that event detection differs from outlier detection as the former always is defined by a predefined pattern or trigger condition. In contrast to this, the authors show that outlier detection can be used as an event detection mechanism itself by identifying anomalous readings. Additionally, there are event detection mechanisms that are based on learning, e.g [BMP+10].

Bahrepour et al. present a taxonomy of event detection techniques designed for WSNs [BMH11] depicted in Figure 2.7. The authors primarily separate the set of EDAs into pattern matching for known events and pattern matching for unknown events, that is, hidden patterns. The former is accomplished primarily by supervised learning and the latter by unsupervised learning.

While the taxonomy presented in Figure 2.7 is certainly valid when focusing on learning techniques, during literature research we identified several categories of event detection mechanisms that are more suitable for describing and classifying the respective EDAs. These categories are not mutually exclusive, e.g. a statistically based algorithm can be conducted locally on the nodes and/or with the accumulated data at the base station, this categorization rather presents several aspects that are of different levels of importance with respect to the considered application. The categories include

- application of sensor fusion
- consideration of temporal and spatial correlations (neighborhood)
- based on statistical analysis
- local vs. global analysis, e.g. clustering
- employment of threshold-based mechanisms
- employment of pattern matching mechanisms
- employment of classification-based mechanisms

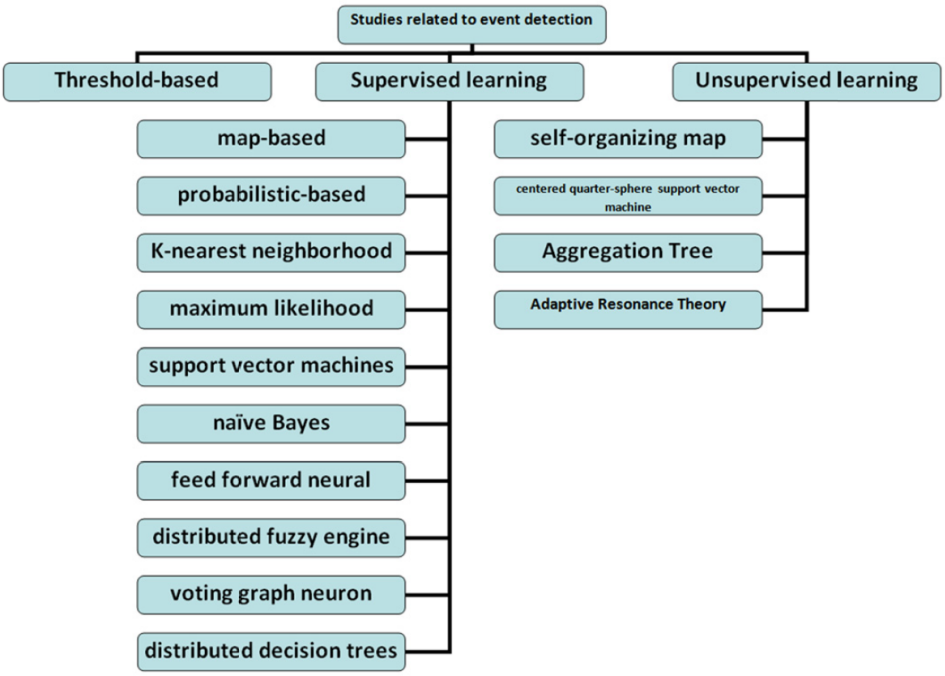



Figure 2.7: Taxonomy of event detection techniques [BMH11]

- online vs. offline detection

Special categories of events are composite events, complex events and unknown events. We will describe these special classes in the following, along with related work in these domains.

**Composite Events** Composite events are composed of several *atomic events*, these are events that are not further decomposable. The data used for detecting the individual atomic events may be provided by different types of sensors. The different sensors may even be located on different nodes, which entails overheads due to the required communication.

Amato et al. [ACGV11] introduce two approaches to composite event detection, a centralized approach based on the idea of routing paths along a spanning tree and a distributed approach which is based on local broadcast. The centralized approach requires fewer messages for the detection procedure, but the nodes closer to the root have to forward more messages than the other nodes. In the distributed approach, even though more messages have to be sent, it takes considerably longer until the first node fails due to energy depletion, as the node's energy is drained in a balanced way.

The authors of [LCZ09] describe a publish-subscribe middleware for the detection of composite events. Composite events are described by a newly proposed event definition language. Energy requirements are reduced by considering that nodes that are capable of detecting an atomic event with one or more preceding

atomic events can remain in sleep state as long as the preceding atomic event has not been detected.

A specialized case of detecting composite events is presented in [LAV+10]. Here, each atomic event is defined and detected by simple thresholding, e.g. *light > 20 cd*. The authors developed an algorithm that finds a set of $m$ disjoint sensor sets and their respective online time so that every composite event can always be detected by at least $k$ sensors at any time. Additionally, the delay for notification of the occurrence of the event is minimized and network lifetime is maximized.

**Complex Events** Hidden patterns in the sensory data may be of interest, but cannot be detected with a simple thresholding mechanism. Escalation [ZR07] is an early data mining approach, where Zoumboulakis and Roussos transform a stream of real-valued sensor readings into a symbolic representation. By using distance metrics on this symbolic representation, complex events are detected. The authors refined their approach in [ZR09], providing mechanisms for multiple pattern detection using a suffix array, unknown pattern detection by non-parametric comparison and probabilistic detection using Markov chains.

**Unknown Events** Bahrepour et al. introduce an approach for online detection of unknown events [BMH11] where no prior knowledge is provided. The approach is based on the standard K-means algorithm, an algorithm that classifies a dataset into k clusters. The original K-means algorithm is characterized by high computational and memory complexity. The authors simplified the algorithm in order to reduce the demands on memory and its computational complexity and yielded results that were comparably good compared to the original algorithm.

### Related work on EDAs

As already indicated in the previous section, the range of underlying mechanisms used for event detection is extensive. In the following, we present a selection of different approaches to event detection, thereby emphasizing the variety of existing techniques used for event detection.

**Sensor Fusion** A two-level sensor-fusion approach is presented in [BMH09]. The first level is performed locally inside the node and the second level is carried out at a higher instance, i.e. a clusterhead or gateway. The second level fuses the results from the first level to reach a consensus as to whether an event has occurred. The motivation to introduce this second level was to eliminate outliers. There are two different methods used for both levels, they are based on either feed forward neural networks or on Naïve Bayes classifiers. Both levels need a learning phase beforehand which can be performed offline. For this sensor fusion approach it is vital to use a set of sensors that is well, or better optimally, suited for the application. For the presented case study, the authors use a set of sensors (temperature, ionization, photoelectric and CO2) that has been proven to be optimal for the task of fire detection.

**Machine Learning** Further work elaborates on another approach that combines local detection and global decision making based on machine learning: [BMP+10] focuses on a combination of decision trees and reputation based voting. Each node runs a local decision tree classifier and communicates its findings to its neighbors. Upon receiving the neighbors' findings, each node judges all its neighbors' detection results based on the node's own findings and informs the voter about its opinion of the other nodes. That way, a global reputation value for each sensor is established. Based on the individual nodes' reputations and their detection results, the voter finally decides about the outcome of the event detection process.

**Signatures** Another possibility is to describe events as signatures [MS06]. The WSN is divided into clusters of equal size and each cluster is characterized by a time-stamped value which is determined by some kind of consensus about the individual nodes' sensor values, e.g. by majority vote or moving average. Each node compares the locally stored event signature that was disseminated by the base station earlier to its own characterizing value and that of its neighbors. If the signature matches, the node informs the base station that an event was detected.

**Fault-Tolerant Event Region Detection** Krishnamachari and Iyengar [KI04] argue that sensor nodes that sense an event, probably have neighbors that also sense the event, but nodes that experience any kind of sensor malfunction probably do not have any neighbors with comparable circumstances. The authors draw the conclusion that *"measurement errors due to faulty equipment are likely to be uncorrelated, while environmental conditions are spatially correlated"*. They present a distributed Bayesian algorithm for fault-tolerant event region detection in WSNs. Their work operates on binary event detection, meaning that an event is either active or not. There is no distinction in terms of magnitude of an event. The authors only consider direct neighbors of each node. In addition, the authors generalize that if one node detects an event, all neighbors (four neighbors in the experimental setting) will detect it too. The authors are aware of this being a generalization, but fail to draw the conclusion, that the proposed scheme is only suitable for events with a large event region, and is especially unsuitable for event regions that are not simply connected. This fact is underlined by the mistakes corrected by Chen et al. [CLF05]. The resulting corrected performance of the proposed approach was reduced, with the main point of reference being that for a fault probability of 10 %, approximately 60 % of the resulting faults can be corrected using the revised scheme as opposed to 75 % in the initial publication.

In this dissertation, we act on a different assumption concerning node malfunction. That is, external disturbances, that entail nodes to display some kind of faulty behavior can stem from physical circumstances, like rain, fog or sensor concealment. These physical circumstances can affect multiple neighboring nodes, which disagrees with the assumptions of Krishnamachari and Iyengar.

**Related work on threshold-based EDAS**

As the experimental part of this thesis uses two simple EDAs based on thresholding mechanisms, we also want to look at representatives of this field. In the following, we review two advanced approaches.

**Dual Threshold Approach** Yim and Choi [YC09] present an event detection scheme that employs two thresholds. Sensor readings are assumed to be binary (1 in case of an event, 0 otherwise) and the two thresholds refer to the number of a node's neighbors that detect an event. A node detects an event if either the node passes the high threshold or if the node passes the low threshold and in addition has a neighbor that passes the high threshold. That is, the node's own detection outcome is incorporated into all neighbors' decisions. The two prime metrics are detection accuracy and false alarm rate. Detection accuracy refers to the number of events detected divided by the number of events occurred and false alarm rate is the number of false alarm nodes divided by the total number of nodes. Additionally employed metrics are event region detection rate and boundary false alarm rate. For the evaluation, a moving average window is used to filter out transient faults. It seems that only one simulation for each parameter set was conducted. 1024 randomly deployed nodes were simulated, the considered event shape was circular and the failure probability was the same for all nodes. The quality of event detection was compared to that of simple majority voting. The approach is of great interest as it is indeed very simple, but the choice of parameters seems to involve a random element and no process for optimal parameter selection is given. In addition, for each instance of detection, communication with all neighbors is mandatory and it depends on the actual application, whether this communications are justifiable if they cannot be piggybacked.

**Grid-based Fault Detection** Lee et al. [LYC09] present a grid-based approach, where the sensor network is partitioned into a grid and each cell of the grid elects a clusterhead. Each cell is partitioned into a 2x2 subgrid and for each of the cells of the subgrid, the ratio of the nodes that display an abnormal reading to the total number of nodes in the cell is computed. Finally, these ratios are compared to a threshold $\Theta$ in order to decide whether an event was detected. The scheme provides fairly good detection accuracy while keeping the false alarm rate low by employing an additional filter. The communication overhead is described as being very low, as only four numbers have to be sent, but the authors fail to see that it is not the payload that is the criterion, but the actual fact that messages have to be sent at regular intervals.

### 2.2.4 Related Work on fault injection and dependability benchmarking in WSNs

An example worth mentioning concerning debugging of WSNs was presented by Ramanathan et al. Sympathy [RCK+05] is a network debugger for WSNs that

detects failures based on data quantity as opposed to data quality. In addition, Sympathy is able to find root-causes of failures.

**Packet Loss Pattern Generation** He and Voigt [HV11] display an approach similar to the concept of fault injection. However, they never mention the term *fault injection* and refer to *intentional interference* instead. In their work, they present a reactive interferer in order to generate different packet loss patterns in a precise manner, thereby emulating parameterized lossy links. This emulation enables testing and debugging of protocol implementations and robustness evaluation of protocols. The latter is a similar idea to the approach presented in this thesis, but focuses exclusively on link quality and is not concerned with event detection at all.

**JamLab** JamLab [BVN+11] also proposes an infrastructure for generating interference patterns, but on a lower level than in [HV11]. JamLab is used to emulate interference produced by various devices used in an office environment, e.g. a microwave oven and Wi-Fi traffic. The crucial point is that JamLab features playback capabilities in order to regenerate interference patterns that were recorded previously. That is, only a few nodes have to be brought to the environment in question to locally record the interference patterns present, minimizing the effort for acquiring them. These pattern are used to understand and evaluate the performance and behavior of sensornet protocols when being confronted with interference. These experiments can be repeated any time after the nodes that have recorded the patterns have been brought back to the lab. This idea is related to the concept of fault injection, with the interference patterns representing the considered faults. However, the authors exclusively focus on the evaluation of routing protocols and do not consider event detection.

**Sensor Validation in a Smart Home** The authors of [MR13] introduce a data-driven approach to sensor validation for a WSN in a smart home. In a relatively small network consisting of 14 sensor nodes, statistical data analysis is performed in order to detect events as well as to classify the activities occurring. The study shows how to locate failed sensors by modeling relationships between the sensors during normal operations. Permanent and transient faults are detected using principal component analysis and canonical correlation analysis. Two experiments for both kinds of analyses were run, the injection of a permanent error and of a transient error. As the sensors' detection results are only binary, e.g. the dishwasher is in use or not, the injection procedure is simple. The detailed analysis shows that a multitude of statistical considerations can stem from only two simple injections. The data used is genuine sensor data, but the fault detection was conducted offline.

**Miss-Classified Events as Faults** Gupchup et al. [GST+08] discuss an interesting side effect of fault detection techniques. The authors elaborate on how events maybe miss-classified as faults. Their main argument is that the two considered

fault detection techniques are based upon the assumption that *faulty data are inherently different from so-called normal data"*. In the conducted experiments, the parameter representing the main decision criterion for each fault detection technique is varied. The metrics employed for the evaluation are the number of false negatives and the misclassification error. The results show how a parameter variation that increases the resulting number of false negatives leads to a decrease of the misclassification error and vice versa. Two different kinds of faults were injected, SHORT and NOISE faults. SHORT faults are one-time faults and NOISE faults, on the other hand, are sets of successive errors. The variation in magnitude of the injected values was rather small in both cases. Only one run of injections was performed for both kinds of faults and the fault probabilities were low with only 1.5 % and 6.5% respectively.

**Attack Injection** In [dAFRG09] the authors evaluate the robustness of ad hoc networks. Instead of fault injection, attack injection is employed. Three different runs are performed on a network featuring seven nodes: without attack and with two variations of the grey hole attack, an attack where packets are dropped by a malicious node. This approach aims to characterize purely network specific properties such as route vulnerability, route availability and data flow goodput. Still, there is a distinct dependability related aspect as the principle of fault injection is applied in order to evaluate and compare different versions of a specific routing protocol.

**Enhanced Attack Injection** In a follow-up work [FdARG11], the authors enhance their approach to include a wide variety of so-called perturbations, as the fault sources are referred to. All eleven considered perturbations are attacks, i.e. malicious faults. The experiments conducted are of substantial range: two network types and eleven different perturbations were used to analyze four versions of olsrd, an implementation of the Optimized Link State Routing (OLSR) protocol. Four main categories of measures serve to describe the behavior of the different routing protocols: performance, energy consumption, resilience, and watchdog behavior. Watchdog behavior quantifies the percentage of correct detections, false detections, missed detections and watchdog coverage. The number of subcategories totals twelve actual measures. The authors conclude that the results of the performed evaluation could be used for comparison and an eventual selection of a routing protocol.

**Attacks and ISO/IEC 25045** The aforementioned work culminates in [FdARM11], where the authors propose a dependability benchmarking approach for the selection of commercial off the shelf (COTS) components. The paper strongly relates to the ISO/IEC 25045 standard discussed in Section 2.1.5. The authors are critical of the fact that only accidental faults and no malicious faults are taken into account and motivate the introduction of an *"attack load"*. In a case study, the same versions of the OLSR protocol as in [FdARG11] are analyzed, but with

a smaller set of disturbances. The difference to the previous work is, that the authors employ a fuzzy-logic-based technique, the Logic Score of Preference (LSP). The LSP computes, roughly said, a global score that is computed by summing up the weighted elementary scores of the considered characteristics. A total of eight subcharacteristics are used to compute three immediate scores for performance efficiency, reliability and security in order to finally arrive at a global score for each version of olsrd.

**AVR-INJECT** Cinque et al. conducted several simulation-based experiments targeting the failure behavior of operating systems [CMT12] when being exposed to low level faults. The focus of their work is the injection of transient hardware-induced errors in the form of single event upsets (SEUs), also known as bit flips. The SEUs are injected by an injection tool for Atmel's AVR controller called AVR-INJECT. AVR-INJECT, as introduced in the previous work [CCM+09], emulates hardware faults at assembly level. Three operating systems for WSNs, TinyOS, MantisOS and LiteOS, were analyzed during three intensive fault injection campaigns using AVR-INJECT. The employed workload consisted of a simple application that only periodically switched on the LEDs of the sensor node. Faults are injected into different locations: code area, memory area, stack pointer and status register. The implementation of the used application differs for each of the three operating systems due to the different structures of the operation systems, especially with regard to the concurrency models. For each operating system, the reaction of the node is categorized into one of the four categories crash, hang, unknown and not manifested. The presented results are used as input for constructing a simulation model based on stochastic activity networks, a variant of stochastic Petri nets. Further work [MCC12] introduces a framework for the assessment of WSNs with respect to performance and dependability.

## 2.3 Summary and difference to related work

As outlined by the high number of publications reviewed, the quantity of potential threats to WSNs, particularly of those from the nodes' environments, is considerable. Even with the benefit of hindsight, a number of authors failed to find explanations for some of the nodes' behaviors and could only hypothesize about the underlying causes. The consequences of the unanticipated behaviors of the nodes in most cases were grave and with regard to the studies' research questions, the corresponding contributions were reduced. In addition, it is likely that, even if the causes of malfunctioning had been anticipated by the engineers, the magnitude of the consequences would only have been foreseeable to some extent.

These circumstances motivate the analysis of eligible EDAs by means of fault injection before they come into use in order to assess their behavior when being confronted with faulty data. The outcome of such an analysis can serve as a decision criterion when choosing between EDAs and also for establishing guarantees about an EDA's minimum performance under clearly defined conditions. In addition,

choosing an EDA that performs better than others with respect to faulty input data may lead to a larger yield of usable data if the WSN installation experiences a harsh or hostile environment.

Studying the behavior of nodes when being confronted with faulty data has so far mainly be done with regard to wireless communication [BVN+11], [HV11], especially focusing on routing protocols [dAFRG09], [FdARG11], [FdARM11] and - on a very low level - with respect to the general behavior of designated operating systems for WSNs [CCM+09], [CMT12], [MCC12]. Targeting a hitherto neglected dependability-related aspect of event detection algorithms, this thesis focuses on the impact and consequences of incorrect sensor data on the performance of EDAs by means of high level fault injection. With respect to EDA selection, it is of greater practical value to see how the EDA reacts to an incorrect value than to study how a low level fault evolves to become an incorrect sensor value. Injecting value faults on a high level directly into the EDA allows for a more efficient assessment procedure than introducing low level faults by flipping bits randomly as the effects can be perceived immediately.

## 2.4 Contribution

This thesis addresses the existing gaps in related work as discussed in the previous sections. With respect to the initial research question presented in Section 1.3, the contribution of this work includes:

- **A methodology for assessing EDAs for WSNs**
  The main contribution of this dissertation is a methodology for the assessment of EDAs by means of fault injection, thereby extending the range of decision criteria for the selection of an EDA. The additional criterion is derived from analyzing the EDA's performance and behavior when being confronted with erroneous data. This approach supports the design process by making EDAs comparable with respect to dependability and performance attributes, thereby easing the application engineer's task of choosing between available EDAs. The innovative core idea is to take into account the resiliency to faults and to resulting service failures that are caused by external influences. The application engineer can derive feedback from the evaluation results as to how an EDA can be enhanced and in addition, the evaluation results may also be employed for risk analysis.

- **The leaky bucket counter (LBC) as a threshold mechanism**
  After outlining the LBC's operational principle, this thesis describes how an LBC can efficiently be used as a threshold mechanism for an EDA. A continuous input of water drops dripping into a leaky bucket symbolizes the incoming sensor values. The bucket's filling level decreases with a constant rate - the outflow through the leak - and increases with newly arriving sensor values. When the bucket's outflow can no longer compensate for the input, eventually, the bucket overflows and an event is detected.

Two detailed case studies demonstrate the applicability of the proposed methodology. The case studies compare the performance of an EDA based on an LBC to that of an EDA based on a moving average using the proposed methodology. The evaluation of two different application scenarios highlights the strengths and weaknesses of each EDA.

# Chapter 3

# A Methodology for Assessing Event Detection Algorithms for WSNs by Fault Injection

Events can be characterized by a multitude of different attributes. The challenge in detecting an event is to do so with only a subset of the event's attributes being measurable by the sensors available. Section 3.3 elaborates on event descriptions and the range of event properties. Depending on the prerequisites, an event detection algorithm (EDA) for a particular application can either be developed by the application engineer herself or be chosen from an existing pool of EDAs, i.e. from libraries already available or from previous similar projects with comparable demands and requirements.

EDAs are distinguished by a number of desired requirements and properties, among them being functional requirements. Functional requirements define the services the system should provide and how the system should behave and react to particular inputs [Som12]. Other categories of properties are resource requirements including e.g. memory requirements and storage requirements, or performance, that is, a sufficiently fast and accurate delivery of the desired result, in this case the detection of an event.

Usually, an application engineer (AE) who chooses the EDA for a specific application uses the above mentioned criteria in order to reach a decision. That is, when using algorithms that are already available, the AE identifies a group of suitable candidates, or, alternatively, designs a new EDA. Besides finding a suitable mechanism for event detection, choosing an appropriate parameter set may also represent a challenge. In order to rank the suitable candidates, the AE analyzes the quality of the various EDAs' detection results using test data. Nevertheless, only using data of the kind that the sensors are expected to deliver, only shows how the EDAs process correct data. The idea proposed in this methodology is to not only use correct sample test data, but to additionally confront the EDAs with erroneous data in order to analyze and assess the EDAs' reactions to this erroneous data. The outcome of this analyses constitutes an additional decision criterion for

the selection of an EDA, facilitating the incorporation of a dependability aspect.

Depending on the available options, the AE will be confronted with one of four situations as summarized in Table 3.1 [HHW13].

| | Parameter set(s) already fixed | Different parameter sets considered |
|---|---|---|
| One algorithm | **A** | **B** |
| Several algorithms | **C** | **D** |

Table 3.1: The four categories in the process of choosing an EDA [HHW13]

**Case A** Although all decisions concerning the EDA selection have already been made, the proposed methodology can be used to understand how this specific combination of EDA and parameters would perform when being exposed to erroneous data, i.e. how long correct service would be sustained in case of emergency. These results can be used for risk assessment.

**Case B** The AE has already decided on a certain EDA and has a number of suitable parameter sets at hand, but is not sure which parameter set to choose. Especially if multiple metrics are employed, there may not be one optimal set, i.e. one algorithm may have a very low rate of false positives, but needs several samples before the start of an event is recognized. By employing the proposed methodology the AE obtains an additional decision criterion.

**Case C** The application engineer has chosen several EDAs that are suited for the WSN's task and for each algorithm, a parameter set that is considered optimal has been provided. This situation bears a certain similarity to Case B, but comparing the individual performances in this case is more elaborate, as different EDAs show different dynamics and weak points. The result is, that the performances of the considered EDAs may differ to a large extent. That is, one EDA is exceptionally good at one half of the metrics while performing remarkably badly at the other half of the metrics. In contrast, another EDA might perform equally well at all metrics without demonstrating any particular strengths. Selecting one of the EDAs involves ranking the metrics according to their importance or, respectively, according to the importance of interesting combinations of metrics. Moreover, if only one parameter set is considered, any bad performance can either be attributed to the EDA's functionality but also to properties related to the parameter set.

**Case D** As there are several EDAs and (at least for a non-empty subset of the EDAs) several parameter sets, this case represents a large decision room. The proposed methodology can also be applied to this case, but may not be suitable for a large number of choices, as the necessary analyses may become too costly in terms of time.

## 3.1 Benefits

The approach improves the design process by making EDAs comparable with respect to dependability and performance attributes. This supports the AE in the process of choosing an EDA or parameter set, respectively. Applying the proposed methodology can reveal weaknesses in the analyzed EDAs. The AE chooses the injected faults in a way that creates an emphasis on a specific attribute of the considered data, for example, only zeroes or highly fluctuating numbers are injected. If an EDA shows exceptionally bad performance for a specific kind of data, a weakness is discovered. In addition, a profile of the EDA's reaction to different kinds or classes of faults can be created in order to easily compare the EDAs. With this knowledge about the EDAs' behaviors, the AE receives valuable feedback. During the development phase, this feedback can be used to enhance the implementation of the algorithm(s) in question, so that a weakness can be removed or can at least be reduced in its magnitude. In addition, the AE can perform risk analysis, especially if knowledge about the probability of occurrence of possible disturbances is at hand.

## 3.2 Node and network model

This section discusses the possible characteristics of the system, i.e. the considered WSN. There are two distinct perspectives, particularly with regard to evaluation and visualization: the node perspective and the network perspective. The network consists of many nodes and each node can deliver correct service or not. Depending on the employed mechanism, the input to the EDA may consist of sensor data from neighboring nodes in addition to the node's own sensory data, making the EDA not only a purely local mechanism but also a cooperative mechanism.

### 3.2.1 Node model

Sensor nodes are usually composed of four basic components [ASSC02]: a sensing unit, a processing unit, a transceiver unit and a power unit. The sensing unit consists of at least one sensor, usually the sensing unit is a sensorboard that combines several sensors. An example for a sensorboard is the MTS310 [Inc14b] for IRIS, MICAz and MICA2 nodes that is equipped with a dual-axis accelerometer, dual-axis magnetometer, light, temperature, acoustic sensor and sounder. In addition to the sensors, the sensing unit features designated analog to digital converters to provide the processing unit with a digital form of the sensory data.

The processing unit includes a microcontroller and a small memory unit. The processing unit is in charge of processing the acquired data and performing any tasks that are assigned to the sensor node.

The transceiver unit constitutes the single node's connection to the network. Compared to data processing by the microcontroller, communication is a costly operation in terms of energy. A power unit supplies the node with the required operational voltage. The power unit may only be a simple battery pack or may additionally be equipped with an energy harvesting device.

The node model provides the following information for every instant of time:

- which sensors are available on each node
- which of the sensors are operational
- operative state of the transceiver unit
- maximum transmission range
- sampling schedule
  - which features are sampled
  - sampling intervals for all sensors
- employed EDA
  - which sensors' data are needed
  - where are sensory data acquired from (local sensors versus radio)
  - how are the sensory data processed by the EDA
  - what data is expected from other nodes

The configuration outlined above may differ for every node of the network. Due to the number of node's neighbors within communication range, the data acquisition scheme may differ, e.g. nodes with only one neighbor that can provide sensory data may not be interested in data exchange. On the other hand, nodes that are surrounded by a high number of neighbors may want to exchange sensory data or detection results in order to fine tune their own findings.

### 3.2.2 Network model

The network model describes the attributes of the WSN's communication infrastructure and how the individual nodes work together if needed. Aside from neighbor to neighbor communication, there is also the possibility of semi-centralized approaches that perform only part of the data processing locally and forward some data to the base station or a clusterhead for additional analysis. The network model describes the network using the following properties:

- number of nodes
- employed topology, i.e. connections between the nodes and resulting neighbors of each node based on the nodes' transmission ranges
- employed routing protocol
- position and links to and from the clusterheads
- position and links to and from the base station

In a cooperative event detection scheme, message exchange may be an integral part of the EDA. In this case, additional parameters related to the aspect of communication are of interest, e.g. message format, number of transmission attempts or the send and receive schedule.

## 3.3 Events

### 3.3.1 Events versus incidents

An event is a happening that an observer is interested in as opposed to incidents that take place by chance in the nodes' environment and are of no concern to the observer. Throughout this thesis, we will talk about events when we refer to a phenomenon that we require awareness of its occurrence and we will talk about incidents when some other - uninteresting - change in the environment's attributes occurs. The consequences of such an incident can, however, be detectable by the considered WSN and can, by all means, be a relevant event in other WSN installations. An event is characterized by at least one measurable attribute, also referred to as a *feature*. An example for a simple event that is easily described by a threshold is

$$event = \text{'it is hot'}$$

with the description being

$$temp > 50^{o}C.$$

In many cases, it is not possible to express the attributes of an event in such simple and straightforward mathematical relations. Reasons for this are that the event itself is complex, the accompanying attributes do not manifest themselves at once or in varying intensities, or an incident superimposes the sought-after event, i.e. [GST+08] reports how rain seriously disturbed soil moisture measurements. To make assessment possible, some kind of ground truth describing when an event takes place has to be provided by the AE. In a simulated environment it is relatively easy to provide such a ground truth, as the AE knows what the input data stands for.

### 3.3.2 Detecting events

Events, as considered in this thesis, are detected locally by a node by measuring the observable characteristics of the node's environment and subsequent processing of the data, i.e. by executing an EDA. Observable refers to when the magnitude of the features that characterize the sought after event can be measured using the available set of sensors. Available can refer either to only the node's own sensors or also to sensors of neighboring nodes, depending on the employed EDA.

#### Reactive systems vs. data collection

The considered WSNs have a purpose, they may even be reactive, i.e. they may be part of a control system. In order to properly react to changes in the environment, the individual nodes need to collect data in order to deduce the occurrence of an event locally.

Circumstances are different with a data collecting application, as in this case any change in the environment's state may be of interest. The collected data is used for offline analysis. Event detection during offline analysis can be conducted on computer systems that, compared to a sensor node, are far better equipped in terms of computational power and memory. Employing such a computer system facilitates the use of complex algorithms with high demands on resources, e.g. pattern matching or intensive statistical analysis. In a hybrid approach, the node may process the data and only store or forward events that were detected autonomously by the node.

#### Event detection algorithms (EDAs) considered

As already outlined in Section 2.2.3, there is a wide variety in the domain of event detection techniques. In this thesis, we focus our considerations on *threshold based* EDAs. Presumably, this methodology is easily extendable to EDAs based on other mechanisms, but the evaluation process may become more complex and time consuming. The methodology is applicable to both, pure localized EDAs as well as to EDAs that work on a cooperative basis and exchange information about detection outcomes or sensory data with neighboring nodes.

## 3.4 Disturbances

The idea of the presented methodology is to feed erroneous data to the EDAs. This input represents data that does not reflect the current state of the monitored environment due to a malfunction, i.e. a service failure, somewhere within the WSN. The reason for this malfunction originates in a physical influence on one or more sensor nodes. We refer to this physical influence as a *disturbance*. Depending on the context, events can also be disturbances. That is, a phenomenon can at the same time be an interesting event and also exhibit attributes that act as a disturbance on some nodes, e.g. high humidity levels are of interest and they can also cause shorts or corrosion in nodes with permeable casings.

### 3.4.1 Reasons for faults - disturbances

A disturbance is the unwanted but unavoidable effect on sensor nodes that is caused by a physical phenomenon. This physical effect of the phenomenon causes faults in the sensor nodes. A disturbance can be transient or permanent, causing the resulting faults to also be transient or permanent. Note, that the attributes that describe the persistence refer to the disturbance and not to the physical phenomenon. That

is, if an ever so short physical effect on the node, i.e. a kick by an animal, ends, the resulting disturbance and fault can still persist, even permanently.

An alternative description is that disturbances also can be effects caused by behavior of the environment that deviates from the anticipated behavior, i.e. exceeding variations in the magnitude of the environment's attributes that are expected to occur. Examples are incorrect assumptions about the system's behavior, i.e. underestimating the maximum illuminance of a street light that illuminates the monitored area. What this kind of disturbance distinguishes from the description given in the previous paragraph is the motivation. In the first case, disturbance refers to damage that was caused by natural processes of the environment, including animals passing by and effects by natural powers. The second description, however, refers to circumstances that have not been anticipated by the persons who planned the installation of the WSN. Examples of physical phenomena that cause disturbances are

- the sensor node comes in contact with water, e.g. rain
- a physical impact by falling or colliding objects
- an unanticipated additional source of sensor feature, e.g. street light
- an unexpected source of radiation

Another cause for disturbing effects are attacks. The disturbances caused by attacks are, of course, also effects of physical interactions, but again, the motivation is different. An attack is a malicious and deliberate fault. An attacker can use tools of virtually any kind to damage the sensor nodes. Accordingly, consequences of attacks can include any of the types of faults discussed in Section 3.4.3. Attackers can be people who are bored, people who want to prove something or people who occasionally feel the need to vandalize.

### 3.4.2 Fault model

The fault model illustrates which faults are addressed via the disturbance abstraction. In general, we adopt a binary model when it comes to service provision; the algorithm either delivers correct service or not. Figure 3.1 visualizes the principle, see below for a detailed description.

**Systems** The core of this methodology is to analyze the EDA's behavior in terms of service provision and we strongly focus on this aspect. There are, however, multiple relevant viewpoints as to what a system is. *System* may refer to either the entire WSN as an event detection infrastructure with all the hardware, routing tables, clusterheads, etc., or it may refer to the individual node that performs event detection, including the node's hardware and connections to the node's neighbors (for cooperative detection schemes), or, finally, it may refer to only the EDA itself. In the course of this section, we will point out to which of these viewpoints we refer, i.e. which constraints and considerations apply.

### Dependability aspects - implementing the system function

In Section 2.1.1, availability, an attribute of dependability was defined as *"Readiness for correct service"* [ALRL04], i.e. the service implements the system function. The system function of the considered EDAs is to detect events as soon as and for as long as they take place. This is a rather stringent demand, as in some case it might not be possible to detect the event within only one sampling period. Similarly, there may be a delay when detecting the end of an event. Nevertheless, defining the system function this way allows a clear distinction between correct and incorrect service. It is crucial to point out that the detection outcome is expected to be correct with respect to environmental conditions of the node that performs the event detection and not based on the node's sensory data.

**Availability** The considered EDAs will, in most instances, possess a memory-like property, i.e. prior input values will influence the current detection outcome. When erroneous values are injected into the value stream, future detection results might not be correct. As an example, consider a simple EDA that, for every sample, computes the arithmetic mean of the current and the three previous samples and compares the mean to a predefined threshold. If the threshold is crossed, an event is detected. A single injected fault with a value that is very high compared to the correct value and the following correct values, can falsify the outcome of subsequent detection operations. Consequently, a single injected fault can prevent the system from being available for several units of time. An example for a corresponding real-life situation is if a sensor malfunctions due to an internal short and delivers an incorrect value. The sensor as a subsystem delivers incorrect service to the EDA, i.e. the subsystem fails and provides another system with an erroneous input.

**Service states** Using a finite-state machine, Figure 3.1 depicts the possible states an EDA may traverse during the fault injection process. In each state, there are only two possible stimuli. The EDA is either provided with a value reflecting the correct state of the environment, in the figure referred to as Input, or alternatively, a fault is injected, which is referred to as FI. Besides the initial state *Algorithm initialization* and the accepting state *Experiment termination*, there are two states associated with correct service, *Correct service without prior FI* and *Correct service with prior FI*. The former state can only be reached as long as no fault injection has taken place and the latter if and only if a fault has already been injected. The property also applies to the state *Incorrect service - algorithm fails*, this state also can only be reached if at least one fault has been injected. There are three possible input sequences to reach this state, firstly, due to the first input being an injected fault that becomes active immediately after the injection, secondly due to the input sequence FI(Input/FI)(Input/FI)* or thirdly due to the input sequence InputFI(Input/FI)(Input/FI)*. Note that we explicitly assume a correctly coded implementation of the considered EDAs, e.g. that the algorithm implementation checks whether the inputs are within the correct value range, i.e. if a positive integer is expected and a negative real value is provided, the application does not crash,

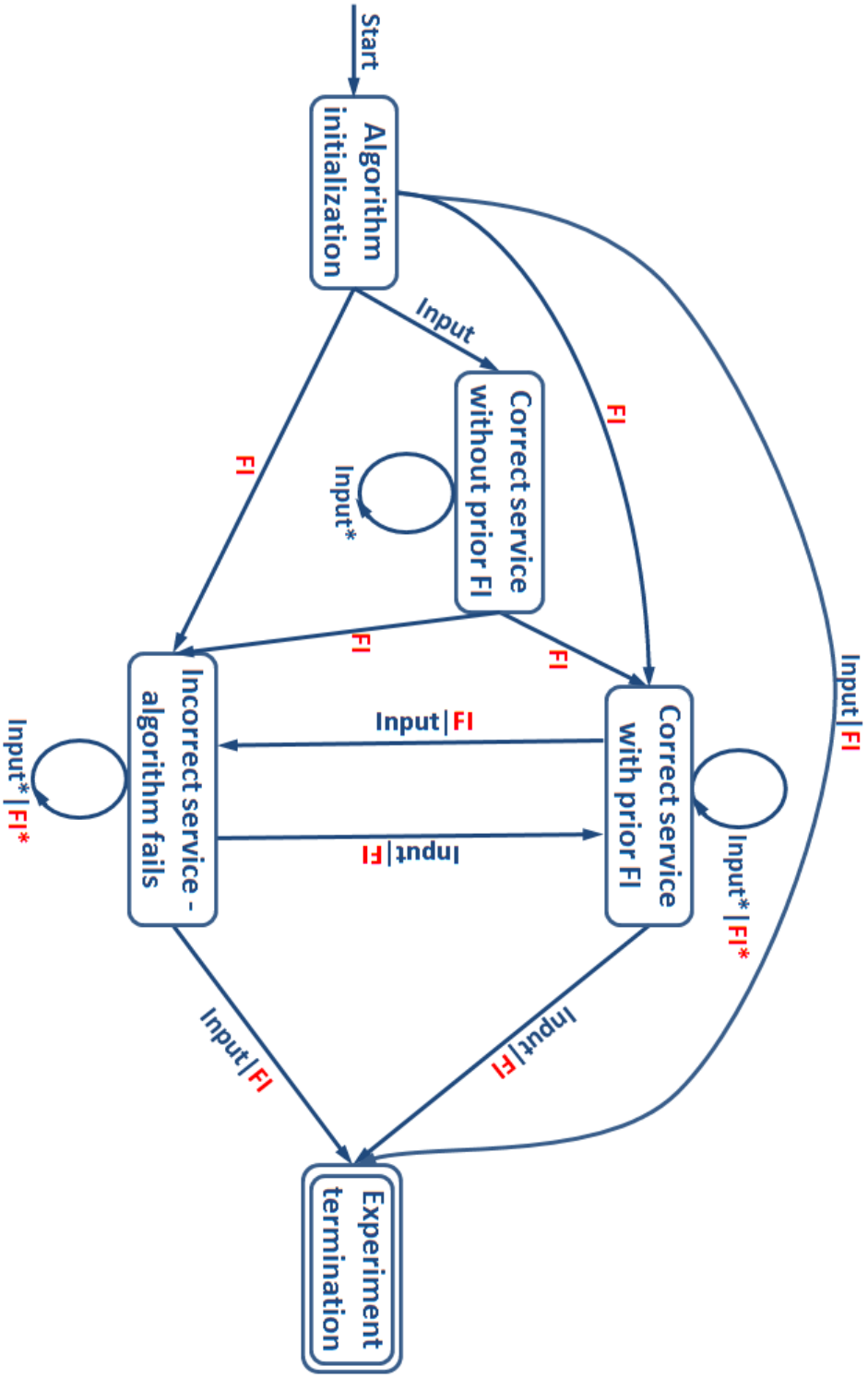



Figure 3.1: An illustration of an EDA's behavior using a finite-state machine, Input represents an input value that adequately reflects the state of the sensor node's environment and FI is an injected, faulty value.

but handles the matter in a pre-defined way. Consequently, the EDA will provide incorrect service only after a fault has been injected. As it may take some time until an injected fault manifests itself as a failure, it is possible that incorrect service is only provided after several correct inputs that follow the injected fault.

### 3.4.3 Faults as failures of subsystems

Faults are the consequences of disturbances and any fault may eventually lead to one or more failures. The considered service is that the EDA provides correct detection results with respect to the current state of the node's environment. A failure of the EDA occurs if the EDA's output is incorrect. The fault that is the reason for the failure is the prior failure of a subsystem of the node. These faults, with respect to the output of the EDA, resulting from the disturbances mentioned earlier are summed up as

- the node does not react anymore and has to be restarted (node black out)
- the sensor does not deliver any data (sensor black out)
- sensor delivers data that does not accurately reflect the actual situation, e.g.
  - always the same value in "stuck-at-fashion"
  - fluctuating, erratic values
  - offset values due to a calibration fault or due to longterm drift
  - delay of sensor value (the value received by the EDA is outdated)
- no communication to other nodes possible, therefore no data exchange with neighbors is feasible

This methodology centers on analyzing an EDA's behavior when being confronted with erroneous data. Hence, the reasons for the faults are not of interest, only the consequences, i.e. the actual failures of subsystems are considered.

#### Fault classification according to the elementary fault classes

The considered faults can be described and categorized according to the elementary fault classes shown in Figure 2.2 in the previous section. In the following, we will characterize the considered faults in detail.

**Phase of creation or occurrence** We strongly focus on *operational faults*. However, *development faults*, that occur during the development phase or during maintenance within the use phase, could also be identified using this methodology, e.g. when the attempt to enhance the EDA would reveal a programming mistake.

**System boundaries** Depending on which system is considered, faults can be either *internal* or *external*. Considering an individual node that provides the service of event detection, an internal fault would be an incorrect value delivered by the node's own sensor. An example for an external fault in this context is an incorrect value received from a neighboring node, as this neighbor represents another system that provides input for the considered node.

**Phenomenological cause** Examples for *natural faults* include malfunctions due to rain or snow, corrosion, damage caused by animals, etc. *Human-made faults* on the other hand encompass similar malfunctions, but the origin of those malfunctions is human-made, that is, water is poured on the node (for whatever reason), the node is physically damaged or removed from its position.

**Dimension** *Hardware faults* are considered as well as *software faults*. An active hardware fault, e.g. within the sensor, produces incorrect data, that in turn, is a software fault. Analyzing the impact of incorrect data on a running program - the EDA - is an objective of this work.

**Objective** Any attack causes *malicious faults*, because they are introduced with a malicious objective. While *non-malicious faults* lack the malicious objective, the person causing the fault is fully aware that she is introducing a fault to the system, e.g. a software backdoor for easy access to carry out maintenance.

**Intent** A fault being *deliberate* or *non-deliberate* mainly refers to the motivation of the human-made physical phenomenon that causes the fault.

**Capability** *Accidental faults* result from physical phenomenon caused by an accidental action, whereas *incompetence faults* stem from the lack of professional competence.

**Persistence** *Permanent faults* stem from disturbances that are here to stay, that is, physical effects that cause a permanent defect on the hardware. e.g. deterioration. *Transient faults* are not permanent, they disappear after some time, i.e. when the disturbance ends.

## 3.5 Assessment process

For the assessment process, we focus our attention on the consequences of active faults rather than on the physical reasons for the faults. Our main concern is the case where the EDA is fed incorrect data from local sensors or from neighboring nodes. The assessment is carried out purely as a simulation in order to avoid problems with the probe effect and because the disturbances can be modeled exactly as needed. Creating a disturbance for a real-life WSN installation might prove to be impossible. Sensory data can be obtained from a WSN installation or be created manually, but

the actual fault injection and assessment is done offline using a computer system more powerful than a sensor node.

The assessment procedure is depicted in Figure 3.2. The AE defines a number of

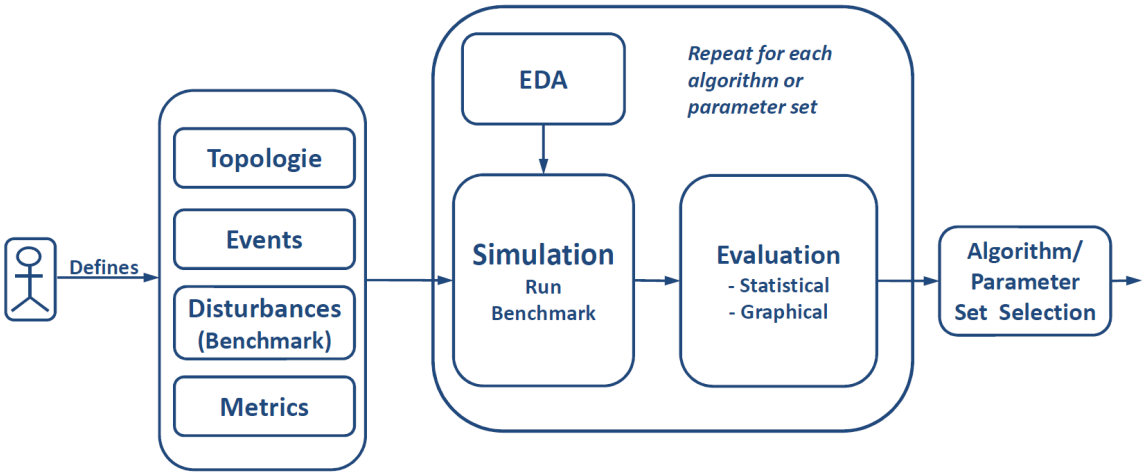



Figure 3.2: The assessment process, adapted from [HHW13]

parameters necessary for conducting the assessment procedure. These parameters include the topology (number of nodes, their spatial distribution and communication links), the events that will take place, the disturbances (i.e. benchmarks, as explained in the following section) that will affect the detection process and, finally, the metrics used for evaluating the EDA and their parameter sets, respectively. Section 3.5.5 gives illustrated examples of how these parameters can be conveniently set using a prototype of a graphical user interface. In the next step, each EDA is simulated with the parameters defined earlier. Subsequently, the behavior is evaluated with respect to the metrics provided in order to choose an EDA with a suitable parameter set.

### 3.5.1 Defining events and disturbances

Events and disturbances are characterized by a number of properties. The following list itemizes the properties that apply to both of them:

- start point in time with respect to the beginning of the simulation
- duration
- geographic spread as a function of time
- movement direction as a function of time
- speed as a function of time, accordingly zero if stationary

When describing events, there are two additional properties concerning the manifestation of the events:

- which type(s) of sensor can sense effects, i.e. the features of the event
- for each feature: the feature's intensity as a function of time

Not every sensor can sense each event, but some events might be perceptible by more than one sensor. As an example, consider an explosion: providing that the features are sufficiently intense, that the nodes sample their sensors at the right time and that the nodes are equipped with the appropriate sensors, the nodes can sense the light of the explosion, the loud noise and the vibration of the detonation. A node only equipped with a temperature sensor, on the other hand, may not detect anything.

Finally, disturbances are characterized by three additional attributes:

- a list of sensor types that are affected by the disturbance
- the associated contortion function(s) for each type of sensor
- the fault probability $p_i(t)$ for each contortion function $f_i$

The contortion function describes the effect of the disturbance on the sensor values by a mathematical expression. Examples include adding a constant offset to model drift, substituting the correct value with a constant to model stuck-at-faults or substituting the correct value with a random value to model erratic behavior.

Using a time dependent fault probability allows the modeling of permanent faults as well as transient faults. An example of a permanent fault is a calibration error or a known measuring inaccuracy. Both correspond to a ground disturbance, affecting all nodes to the same extent that can be modeled as a permanent disturbance that covers the entire WSN. Long term effects on the system, e.g. system deterioration can be modeled by a ground disturbance with a slowly increasing fault probability.

As indicated, it is possible to associate one disturbance with multiple contortion functions and, accordingly, with multiple fault probabilities. However, depending on the application, it may be more practical to define multiple disturbances with the same attributes except for the contortion function and associated fault probabilities and to superimpose these disturbances.

#### Benchmarks

Within this methodology, a benchmark refers to a set of disturbances. The AE defines these before considering any EDAs, in order not to be biased by any knowledge about an EDA's strengths or weaknesses. A benchmark is a scenario providing a test to show if an EDA is suitable for at least one situation that the EDA will potentially have to face. In addition to the disturbances, the AE also defines the expected outcome. That is, the goals the EDA has to reach, e.g. the maximum number of failures, the maximum time to recovery or the minimum percentage of correct detections.

The results of a benchmark-based test can be used for issuing guarantees. This aspect is interesting as not all possible scenarios that the considered WSN can

possibly face can be covered by simulation. It is a challenge for the AE to adequately choose the disturbances and their parameters. It is, naturally, not possible to test for every eventuality, and the AE has to decide, which effects, and especially, which fault probabilities, are to be used. With regard to feedback from the evaluation results, the AE also has to choose appropriate metrics.

### 3.5.2 Fault injection

Before the fault injection procedure begins, the AE performs a so-called golden run, which is a simulation run with fault-free data. The results of this golden run serve as a ground truth later on in the evaluation process.

The fault injection process consists of two main tasks. First, the samples that are to be contorted have to be determined; a straightforward way to implement this selection is to use a random number generator in combination with the fault probability provided as a parameter. The second task is to apply the contortion function to the selected samples. The contortion function can be either a function of the correct values, e.g. adding a constant value or a percentage of the correct value, or the correct value is simply replaced by another value. This value can be constant, thereby modeling a stuck-at fault or it can be a random value from a predefined interval.

This procedure has to be repeated many times and the results from applying the EDAs to each test vector that contains the faulty data, have to be averaged in order to provide meaningful results. The necessary number of repetitions depends on the number of samples, i.e. the duration of the considered scenario, on the distribution of the fault probability and also on the kind of fault injected. That is, when random values are injected, there is a greater degree of variability than when injecting zeroes. In addition, the fault probability of a disturbance is a time dependent function. That is, for the assessment process, the fault probability may be gradually increased or randomly selected from a given interval.

### 3.5.3 Metrics

The function of the metrics is to quantify the performance of the EDAs in a way so that the EDAs become comparable with respect to a dependability-related perspective. In addition, the behavior of the considered EDAs shall be explainable using the metrics, e.g. one of the EDA detects many more events than the other ones, because a single contorted sample already causes the EDA to detect the start of a new event. In addition, the AE shall be able to draw conclusions about the general reaction to a certain kind of faulty data, e.g. to the injection of zeroes only or to values injected from a specific interval.

In the following, we will give a list of suitable metrics for comparing EDAs. Note that the importance and explanatory power of each of the metrics depends on the actual scenario and application, e.g. in some applications it is of high importance to keep the false alarm rate as low as possible, while in others, it is more important to detect each and every event as fast as possible, with a few false alarms being no

trouble. Recommended metrics include:

- number of detected events
- number of false positives
- number of false negatives
- number of failures
- cumulative duration of (correctly) detected events
- percentage of event region that was correctly detected
- rate of faults that become active during the experiment
- average fault latency
- average time to recovery
- the mean time to failure
- minimum number of successive faults that are necessary to cause a failure
- error propagation in number of hops

It may be of interest to run the simulation using more than one scenario, each addressing a special aspect. As an example, the AE arranges the events in a way so that there are a large number of events with only short intervals between them in order to see if the EDAs detect the pauses between pairs of events. An additional possibility is to analyze, to what extent the individual performances change with the total duration of events.

The task of choosing the right metrics rests with the AE. For an exhaustive assessment of the considered EDAs, it is advisable to calculate as many metrics as possible. This recommendation is motivated by the purpose of the assessment, which is to discover weaknesses and strengths of the EDAs that are not known beforehand as opposed to confirm properties of the EDAs that have already been identified before running the experiments. In the course of choosing the metrics of interest, the AE also ranks the metrics according to their importance.

### 3.5.4 Evaluation and expected results

For evaluating the behavior of the EDAs in the presence of faulty data, the AE has to provide a ground truth. Depending on the metric and the interests of the AE, this ground truth is either the detection outcome of the golden run or a vector that indicates when an event actually took place. After running each EDA on the vector containing the faulty data, for every detection outcome, the evaluation application compares the result from the run with the faulty data to the ground truth. From these outcomes, the desired metrics are calculated.

Evaluating the results of the fault injection process includes gathering all results for all runs, calculating the respective means of the metrics and presenting the resulting numbers in a comprehensible way. A convenient approach to a qualified way of presenting the numbers is to export the results to a spreadsheet application where the data can be properly displayed by appropriate diagrams. In addition, employing a spreadsheet application provides opportunities for optional further analysis of the data.

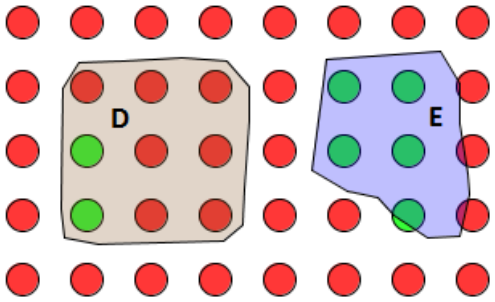
Using the resulting diagrams, the AE compares the individual behaviors of the EDAs. With the help of the predefined ranking of the metrics, the AE can also rank the considered EDAs with respect to their performance. This ranking already yields the most suitable candidate. In addition to the ranking, the AE can study the reasons for the individual behaviors of the EDAs by analyzing the provided diagrams. While obtaining the ranking and consequentially choosing the EDA can be automated, this further comprehensive analysis is manual work. The benefit from performing such an analysis albeit a suitable candidate has already been elected, lies in a potential enhancement of one or more of the EDAs. Using the feedback of the evaluation, the AE can redesign parts of the EDAs' implementations. After removing a conceptual fault in the implementation or optimizing parameters, re-evaluating the EDAs might yield a different ranking and the selected EDA might exhibit superior performance compared to the EDA chosen in the first place.

### 3.5.5 Visualization aspects

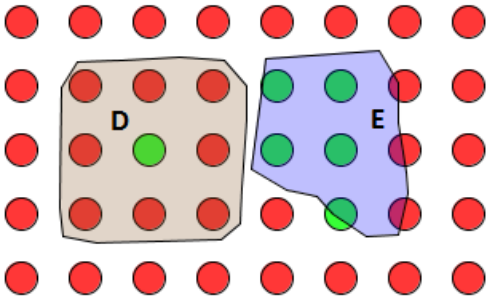
Visualization is an important aspect in the assessment process. First of all, there is a need for a graphical user interface that allows convenient data entry, i.e. the network topology along with the events and disturbances to be drawn as regions directly onto the defined topology in a click-by-click-fashion. For each of those elements, the AE enters the parameters describing the properties of the element as specified in Section 3.5.1, e.g. duration or movement direction. Secondly, in addition to the input, the course of action shall also be visible to the AE, e.g. the movements and periods of validity of the events and disturbances in combination with the correctness of the EDA's output for each node.

Figures 3.3a to 3.3f present an illustrative example showing the approach to visualizing the parameters of interest which was created by a prototype application [Mai10]. In the figures, green nodes detect an event and red nodes do not detect an event. Nodes only detect an event or are affected by a disturbance if the corresponding region covers the center point of the node.

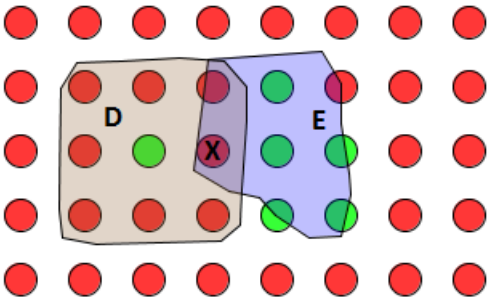
During the simulation, the blue event region which is annotated with 'E' moves towards the west and the grey disturbance region which is annotated with 'D' is stationary and remains present for the first 75% of the simulation's duration. Note that the descriptive letters have been manually added to the figures in order to make the figures self-explanatory. With every step of the simulation, the application updates the nodes' detection states, positions of the events and disturbances and calculates for every node if it is affected by the disturbance. To keep it simple, the effect of the disturbance in this example is that nodes have a 20% chance of



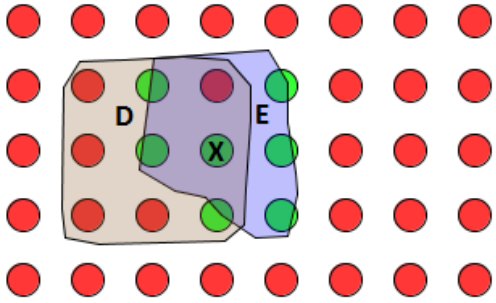

(a) In the beginning of the simulation, the event and the disturbance do not overlap. The event covers the center of five nodes, these nodes - colored in green - detect the event.



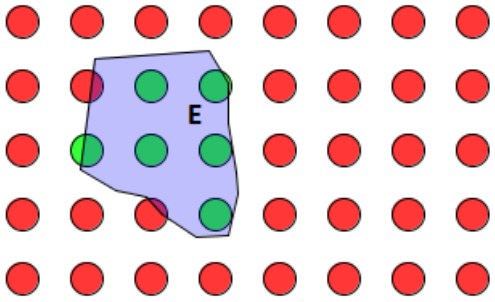

(b) After having moved west, again five nodes detect the event. The node situated at the bottom right of the event region does not detect the event, because its center is not covered by the event region.



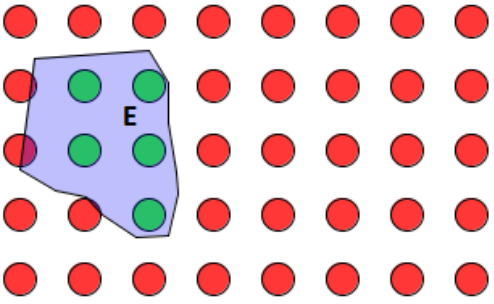

(c) The intersection of the event and the disturbance covers two nodes and both nodes deliver incorrect results due to the disturbance. The node marked with an 'X' delivers an incorrect detection result while the opposite is true in the next figure.




(d) The intersection of the event and the disturbance covers four nodes, only three of them detect the event. As the center of the node at the bottom right of the disturbance is not covered by the event region, the delivered result is incorrect.




(e) The disturbance's period of validity has ended and all nodes that have their center covered by the event detect the event.




(f) The simulation ends with the event having reached its final destination and all nodes reporting correct results.

Figure 3.3: An event (the blue area, annotated with 'E') that is moving west, eventually covers a stationary disturbance (the grey area, annotated with 'D'). The green nodes detect an event and the red nodes do not detect an event. The effect of the disturbance is that affected nodes produce an incorrect detection result with a probability of 20%.

malfunctioning. As long as a node malfunctions, it does exactly the opposite of what is correct, i.e. if an event is present, the detection outcome will be negative and vice versa. Figures 3.3c and 3.3d illustrate this situation: due to being affected by the disturbance, the node marked with an 'X' does not detect an event in Figure 3.3c although its center is covered by the event, a false negative. Subsequently, Figure 3.3d shows that the node detects an event in the next simulation step, because the result of the new calculation whether this node was affected by the disturbance returned a result different from the one in the previous simulation step. Figures 3.3a to 3.3d show that nodes that are part of a disturbance region can show incorrect detection results, independent from them being overlapped by an event region or not. As an example, consider the bottom right node of the disturbance region in Figure 3.3d: Although its center is not covered by the event region, this node incorrectly detects an event, a false positive. In the final two subfigures, the disturbance has ended and accordingly, all nodes exhibit correct detection results.

Although the model of the considered disturbance in this example is very simple and this precise behavior would be difficult to find in reality, this scenario was chosen deliberately to illustrate that a disturbance can also affect nodes that are not part of an event region. In addition, as the disturbance's effect on the nodes is recalculated with every simulation step, this example beautifully illustrates the nature of transient faults. Moreover, this frequent update of the effects of the disturbances allows one to observe visually how injected faults propagate through the network when employing a cooperative detection scheme.

# Chapter 4

# Experimental Evaluation - Two Case Studies

This chapter describes an exemplary application of the methodology discussed in the previous chapter. The RIPLECS project, which provides the infrastructure for the two presented case studies, is introduced in the next section. In addition, this chapter describes how to use the leaky bucket counter as a threshold algorithm. This mechanism serves as a subject of study in the two presented case studies. The chapter concludes with a summarizing comparison of the conducted experiments.

## 4.1 The RIPLECS project

RIPLECS is an acronym for *Remote-labs Access in Internet-based Performance-centered Learning Environment for Curriculum Support* [Con13]. Within the RIPLECS project, educational remote labs are provided, enabling students to complete lab work from their home. The focus of the part of the RIPLECS project provided by Graz University of Technology is the domain of WSNs with special emphasis on energy harvesting [HSKK12], [HSKK13].

Energy harvesting is a process, where ambient energy is converted into electrical energy using energy harvesting devices, for instance piezoelectric crystals or solar cells. The collected energy is stored in dedicated devices like capacitors or super capacitors. Energy harvesting is an established principle in WSNs in order to solve the *battery replacement problem* [MMA05]. Several factors constitute to the battery replacement problem, including the fact that nodes are often placed at remote locations, that the lifetime of batteries is limited, the existence of permanent leakage current and the often high number of nodes. The battery replacement problem is usually tackled by using energy harvesting in combination with power-aware software, that is, the node's software is designed in a way that makes it highly efficient in terms of power usage. A prominent example for the importance of this approach is the wide variety of available power-aware routing protocols [SM14].

The educational aim of the RIPLECS project is to provide students with a convenient means to observe how the WSN reacts to changes in the WSN's con-

figuration and to external stimuli whilst making it possible for students to learn about the structures of sensor nodes and WSNs. In addition, students shall perceive how communication in WSNs is organized and also become acquainted with low power techniques and the mode of operation of WSNs that are enhanced with energy harvesting devices.

The installation used for the conducted case studies features six MICAz nodes [Inc14a] equipped with an additional MTS420/400CC sensor board [Inc07]. For the network communication the IEEE 802,15.4 protocol is used. The installation of the WSN is situated in a darkened room, where all windows are entirely covered by dark cloths. The function of the installation is to detect light events and for this purpose, only the light sensor is used. There are two types of light source, two large-area LEDs measuring 50 x 50 cm situated directly above each of the solar panels and one central LED spot. The former provide the ambient light for energy harvesting and the led spot serves as the event light. Both sources of light can be controlled via the web interface depicted in Figure 4.2. An event detection feature is already built in, but this functionality is not used in the case study, it is only the measurement data that is captured.

### 4.1.1 The remote laboratory concept

Figure 4.1 shows a block diagram of the educational remote lab that is used for the case studies. A base station manages the communication between the WSN and the web server, including data and status messages arriving from the network and control messages addressed to the network. Four of the six sensor nodes (Nodes 1 to 4) receive their power from a continuous power supply unit, while the other two sensor nodes (Nodes 5 and 6) are equipped with solar cells only. Nodes can be switched on and off remotely via the power supply control. The power measurement unit and measurement and control unit are used to determine the power consumptions of Nodes 1 to 4. In addition, the measurement and control unit is used to drive the ambient lighting and the event light. There is also a web server providing remote access to control the experiment, including selecting, starting and ending experiments, reconfiguration of the nodes and the WSN, control of the ambient light and the event light and access to the logging files that contain the measurements. Finally, there is a webcam that delivers a constant live video stream of the installation.

### 4.1.2 The web interface

The web interface consists of three different views, the live laboratory, a tool for the creation and maintenance of configurations and the results display. It is possible to change between the views by using the buttons in the upper left in Figure 4.2, next to the logo of Graz University of Technology.

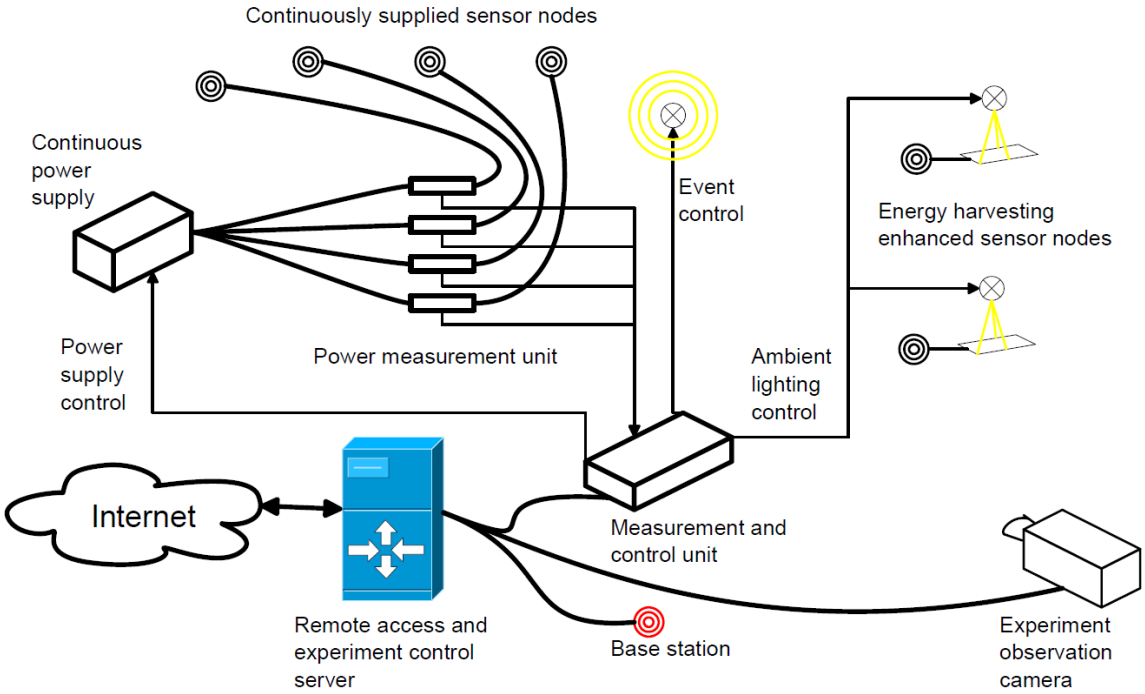



Figure 4.1: Block diagram of the educational remote lab setup [HSKK12]

### Creating and maintaining setups

This view is selected via the button *setup* and allows the creation and amendment of setups, which are configurations of the WSN's parameters. There are a number of parameters and options to set that determine the behavior of the WSN, the parameters relevant for the considered case studies include:

- type of topology, there are four options available:
  1. line topology, where the nodes form a (virtual) line and each node can only communicate with its right and left neighbor
  2. star topology, where each node can directly communicate with the base station
  3. mesh topology, where each sensor can directly communicate with a non-empty subset of all sensor nodes
  4. mesh topology with energy harvesting
- communication interval in seconds, which is the time interval in which the voltage levels of the nodes enhanced with energy harvesting devices and the sensor data are transmitted to the base station
- measurements per interval is the number of measurements performed during one communication interval, the average of the measured data is transmitted to the base station

- the routing protocol can be either shortest path, direct to base station, flooding or energy aware

There are also options to enable the nodes' status LEDs and to enable package traces. Both options are useful for debugging purposes. In addition, there are parameters concerning transmission options and energy harvesting specific settings which are of no concern in the context of the case studies. It is possible to maintain several different setups and to select a suitable configuration for each new experiment.

### The live laboratory

Figure 4.2 shows the live laboratory part of the web interface, accessible via the button *live lab*. There are four distinct sections visible, beginning in the upper right corner we will describe them in a clockwise order.

**The live video feed** allows one to see the entire installation with the four nodes that use the continuous power supply at the bottom and the two nodes powered by the solar cells at the top. Each node is clearly labeled with a number. The spots for the ambient light are situated just above the solar cells and the event light is situated at the top between the solar cells. The base station is not visible in the figure, it is mounted next to the setup so that all nodes can directly communicate with the base station.

**The live controls** are divided into three tabs. The tab visible in the figure provides buttons to switch on and off Nodes 1 to 4 as well as two sliders to control the two sources of light, the ambient light and the event light at 256 levels. The second tab allows one to choose a setup previously created and to start predefined exercises. Starting an exercise initiates recording and logging of measured data and messages. In the last tab, the details of the current setup are displayed in a read-only fashion.

**The live message log** displays status messages from the base station as well as confirmation of the actions initiated by the user. Examples include package traces, measured value of the illuminance, information about whether a node was switched on or off, error messages or information about the start and end of exercises.

**The live current drain** of the continuously supplied sensor nodes shown in the upper left corner is updated every second. There are several zooming options available, ranging from ten seconds to the duration of the entire experiment.

### The result display

In addition to the configurations view and the live laboratory, the web interface offers the result view, accessible via the button *my results*. Plots of the measured

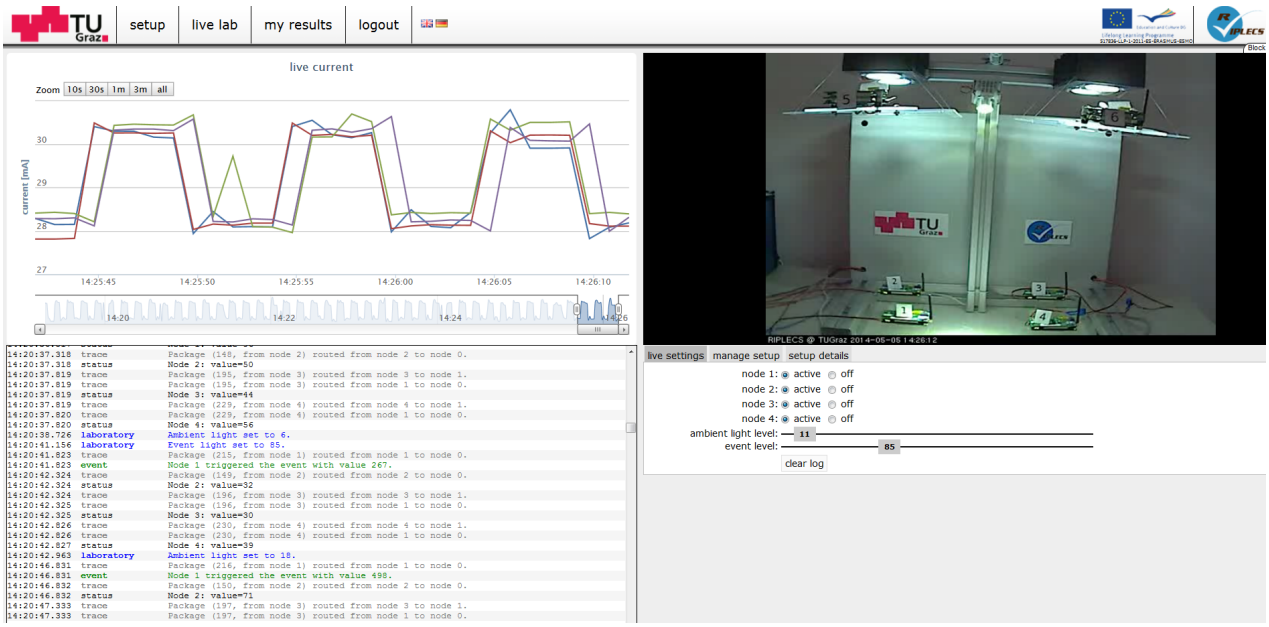



Figure 4.2: A screenshot of the live laboratory

current drain and files containing the measured data as well as log files, both in different formats, are available for download. Users have access to the results of all of their conducted experiments.

## 4.2 The leaky bucket counter

The leaky bucket counter is a mechanism that is based upon the metaphor of a bucket with a leak in the bottom. Water drips into the bucket at a varying rate and the water runs out through the leak in the bottom at a constant flow rate $\rho$ as long as there is water in the bucket. When the bucket is empty, then the outflow is zero. As soon as the bucket is full and more water drops into the bucket, the excess water simply spills over.

Figure 4.3 illustrates the fundamental principle of the leaky bucket counter. Starting in Figure 4.3a the bucket is empty and several drops of water are about to drip into the bucket one by one. Figure 4.3b shows a bucket that is half-full with the outflow clearly visible below the bucket's leak. Figure 4.3c depicts how the bucket overflows. Here, the magnitude of the constant outflow was exceeded by the input. The final Figure 4.3d illustrates how the filling level of the bucket has decreased due to the outflow. Judging from the input that is about to drip into the bucket, another overflow is to be expected soon.

The leaky bucket counter is the basis for a simple algorithm - the *leaky bucket algorithm* - that is used for packet management in networks [Tan96]. Here, the leaky bucket counter is basically employed as a finite queue. Packets arriving at a full queue are discarded , i.e. the bucket overflows, otherwise they are added to the

queue. If the queue is not empty, one packet is transmitted at every clock tick, thus implementing a traffic shaping function. The leaky bucket algorithm was originally introduced by Turner [Tur86].

#### The leaky bucket counter as a threshold mechanism

The leaky bucket counter can also be used as a mechanism for event detection [HHW12]. The drops represent the sampled values from the sensors. Depending on the application, there are two ways to model this relation: either the size of the drop is mapped to the magnitude of the sampled value or the frequency of arrival of equally sized drops is mapped to the magnitude of the sampled value. In this dissertation, we will apply the first variant. The capacity of the bucket is the threshold that, if crossed, indicates that an event has taken place.

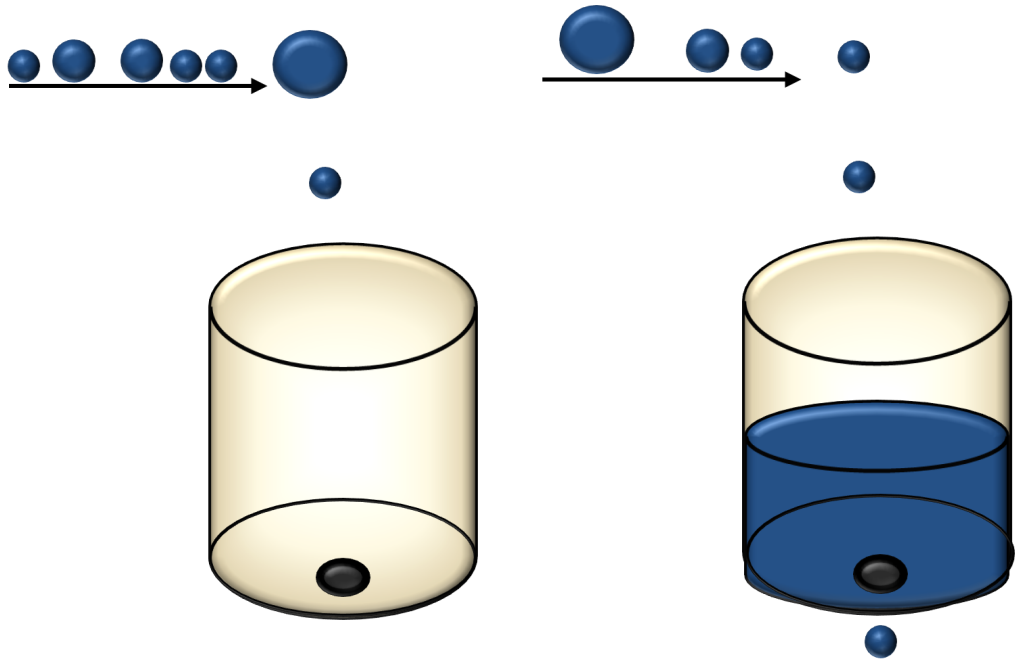
Using the leaky bucket counter as a threshold mechanism comes with the benefit of a certain filtering characteristic. If the node samples spikes in the monitored feature, the bucket will not overflow if the sum of those spikes do not surpass the capacity of the bucket. That way, one-time spikes with a magnitude up to the bucket's capacity due to faults or noise are elegantly filtered out. However, multiple consecutive spikes will, at some point, cause an overflow. For the sake of simplicity, we will only consider positive sampling values for the event detection procedure. If necessary, however, negative sampling values can be incorporated by using appropriate offsets.

## 4.3 Event detection: leaky bucket counter vs. moving average

The case studies described in the following sections compare an EDA based on a leaky bucket counter with an EDA based on a moving average. Henceforth, when referring to the EDA using a leaky bucket counter, we will use the term *LBC*. The EDA using the principle of a moving average will be denoted as the *MA*. These two algorithms were chosen for the case studies because they are simple enough to allow for an illustrative demonstration of the proposed methodology and in addition, they have comparable resource requirements.

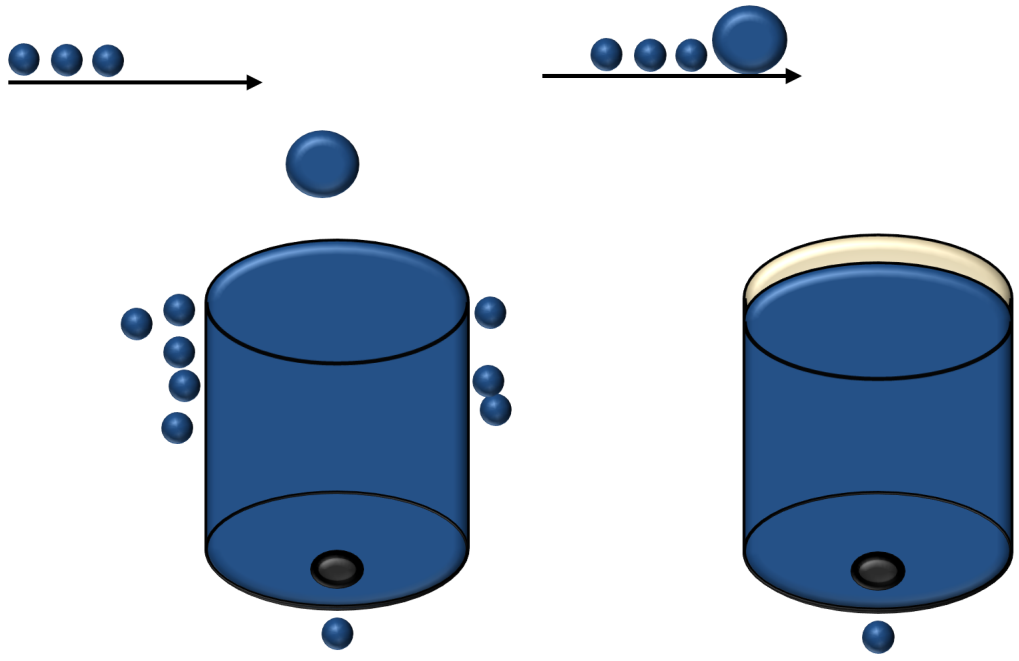
There are two parameters that define the moving average, the window size $k$ and the threshold $\Theta$. If the average of the last $k$ samples exceeds the threshold $\Theta$, an event is detected. The average is recalculated with every arriving sample. This scheme can be refined by assigning weights to the individual values. Choosing a large window size results in low sensitivity. That is, more high-valued samples have to arrive in order to raise the average than for a smaller window size.

The LBC is also characterized by two parameters, the flow rate $\rho$ which is proportional to the LBC's leak and the capacity $\Gamma$. These two parameters roughly correspond to the MA's $\Theta$ and $k$. Both algorithms are of comparable complexity.



(a) The leaky bucket is empty, a few drops are about to drip into the bucket. The leak in the bottom is visible, but there is no outflow, because the bucket is empty.

(b) The leaky bucket's filling level has increased to half-full. The outflow is visible as a drop beneath the bucket and more input is already coming up.



(c) The leaky bucket has overflowed because the input exceeded the outflow and an event was detected.

(d) Right after the overflow, the contents of the leaky bucket has already decreased due to the outflow.

Figure 4.3: The functional principle of an LBC as a threshold mechanism adapted from [HHW12]

There is even a basic case, where the algorithms produce the same outputs:

$$\Theta = \Gamma$$
$$k = 1$$
$$\rho = \Gamma$$

Although the two algorithms work in a related way, their output follows different dynamics. The main differences are listed below.

**Algorithm's memory** In the case of the MA, the maximum time that a sample influences the detection outcome is limited to exactly $k$ instants of time. In the case of the LBC, this time depends upon the magnitude of the recent inputs. Inputs of small magnitudes will leave the bucket faster than inputs of large magnitudes. An input that has already left the bucket has no influence on the detection outcome. Vice versa, an input of high magnitude will stay for some time in the bucket and influence the future detection outcomes for at most $\frac{\Gamma}{\rho}$ units of time.

**Zero-valued inputs** Adding a zero-valued sample to the moving average can lead to the detection of an event. This happens when just before the arrival of the zero, an event was detected and the moving average was sufficiently high, so that substituting the oldest of the $k$ values with zero still yields a sufficiently high average. The leaky bucket, on the other hand, can never overflow when nothing is added to it. Accordingly, sampling a zero can never lead to an event.

**Minimal stimulus for detecting the second of two successive events** For the MA, the newly added sample has an upper bound corresponding to the magnitude of sample $k+1$ that has just been discarded. In the case of the LBC, the new sample simply has to exceed the leakage rate and does not depend on prior input as long as an event has just been detected.

## 4.4 Introduction to the case studies

The infrastructure of the RIPLECS project is exclusively used to capture sensor data. The actual event detection and fault injection procedures are performed by a Java application that uses the sensor data provided by the RIPLECS project's log files. The event data for the case studies was manually created using the two sliders provided by the web interface. The sliders act like dimmers, with one slider controlling the event light and the second slider controlling all four of the large-area LEDs.

The event light is used to represent an actual event, that is, if the slider position exceeds a certain position, an event should be detected. The ambient light is used to model changing weather conditions, e.g. the transition between direct sunlight and cloudiness. The actual determination of what constitutes an event was conducted by personal experience. That is, we tried out different settings of the ambient and

the event light and decided what illuminance could be perceived as an event. In the conducted case studies, the purpose is to detect when the event light switched on to a sufficiently high degree. Solely turning on the ambient light should - per definition - never constitute an event. In order to provide a realistic scenario, neither of the two sliders ever exceeds position 100.

The two case studies consider different behaviors of the environment's illuminance. The first case study concerns itself with detecting several different types of events, that is, the slider of the event light is not always at the same position, when an event takes place. In addition, multiple peaks in the ambient light, that symbolize sudden increases in illuminance due to e.g. intense sunshine, must not be detected as events. The challenge is to parameterize the EDAs in a way so that only real events are identified as events and peaks in the ambient light are not detected as events. The peaks are kept deliberately brief, as a long period of intense sunshine would not be distinguishable from a real event and could also be moderated by inducing countermeasures from other systems, like closing shutters. The second case study considers a less dynamic setting; there is only one event and no peaks in the ambient light. Some of the considered metrics are considered for both case studies, while some are only applicable for the second case study.

### 4.4.1 Configuration and parameters for the case studies

For all experiments conducted in the scope of the case studies, the same configuration, as outlined below, was used:

- topology: *mesh*
- communication interval: *5*
- measurements per interval: *5*
- routing protocol: *direct to base staion*
- enable status LEDs: *yes*

All other available options were set to zero or no, respectively, as any additional features, like energy harvesting or package traces, are of no concern for the conducted experiments. Enabling the status LED does not provide any additional benefit apart from the visual feedback to the person carrying out the experiment, this option was enabled purely out of convenience.

As can be seen from the log files in Appendix A, the communication interval is slightly longer than 5 seconds. An additional length of five to seven milliseconds can be observed in most communication intervals. According to the settings above, a node performs five sampling operations within five seconds and sends the average of those five sampled values to the base station every five seconds. This average is then passed on to the server to finally be displayed in the live laboratory's live message log in near to real time. As the unit of abscissa, the charts presenting the results of the case studies display *Time [#samples]*, where, due to the choice of communication interval, each sample represents five seconds.

### 4.4.2 From one node to many - generalization of the base case

In both case studies, only one node of the installation is considered for the evaluation, which is Node 1, situated in the front row on the left hand side. There was no specific intention in choosing this particular node, it could have been any other of the four nodes with permanent current supply. Nodes 5 and 6 are not candidates for the case studies because they are only used for energy and energy harvesting related considerations and do not sample their sensors and, consequently, do not pass on any sensory data.

Considering only one node corresponds to a basic case, where a disturbance covers only one node. Deciding upon suitable metrics either for a single node or for a cluster or even the entire network is, again, the task of the application or test engineer. In the following, we will give some guidelines and discuss a number of strategies for expanding to the general case. Determining metrics for single nodes can in many cases be used as a preliminary for determining further metrics.

#### Neighborhood level

The neighborhood of a node can be defined in different ways. A good example is the one-hop-neighborhood, which includes all nodes that are situated within the chosen transmission range. However, depending on the physical topology, the actual distance to these one-hop neighbors can be quite diverse ranging from nearly zero to the maximum transmission range.

When using a grid topology, there are two popular types of neighborhoods, the Neumann neighborhood and the Moore neighborhood. The Neumann neighborhood *"applies on two-dimensional lattices, and it comprises all cells orthogonally surrounding one given cell"* [dQMS10], totaling in four neighbors. The Moore neighborhood includes eight neighbors, that is, in a two-dimensional lattice, all eight cells that surround the center cell. Integrating the single nodes' evaluation results into a combined evaluation for the neighborhood can be done in several ways, examples include

- the number of nodes in the neighborhood that deliver correct service
  - at every instant in time or
  - for a minimum fraction $t$ of time or
  - on average during the experiment
- the number of nodes in the neighborhood that detect an event within $t$ instances of time after the event started
- the time until at least $k$ out of the $n$ nodes with $k \leq n$ detect the event

In addition to determining these numbers, it may also be of interest to analyze whether the neighborhood fulfills predefined requirements. That is, for each of the aforementioned criteria, an upper or lower limit is defined and the evaluation yields a simple yes or no. The analysis is not limited to purely considering correct service, any metric, as demonstrated in the case studies is applicable.

### Cluster level

Analyzing the nodes' behaviors on a cluster-based level bears similarities to the neighborhood-based approach. But while the neighborhood-based approach is more localized, in general, a cluster covers more nodes. The definition of the clusters results from the demands on the application, e.g. there may be naturally formed clusters anyway or requirements may be expressed using topological information. Metrics can be calculated for each node and subsequently for each cluster, executing an averaging strategy for subsets of the network's nodes. As the number of nodes within each cluster may not be known beforehand or, in many cases, may differ from cluster to cluster, desired performance should (at least additionally) be expressed in percentage rates.

### Network level

Extending the concept of cluster consideration further, looking at all nodes of the entire network at once yields one large cluster. Evaluating the entire WSN's behavior can be done in multiple ways. One way is to consider the set of all nodes' characteristics using statistical evaluation of the individual characteristics. For a large WSN, simple averaging over all nodes might not be meaningful, as the average of any characteristic of thousands of nodes may locally be of no significance. A good solution is to combine the statistical evaluation of the set of nodes with the findings from a previously conducted cluster-level analysis. That way, it can be seen if the individual clusters deviate from the average over all nodes, which indicates local anomalies.

### Considering event regions and non-event regions separately

There is a potential benefit in evaluating the nodes that are part of an event region separate from those that are not. The benefit is that the analysis can uncover any tendencies that are linked to the presence of an event, e.g. faults manifest themselves much quicker as failures when an event is present. Any of the three previous approaches can be enhanced with this extension.

### Cooperative schemes

Cooperative schemes are detection procedures, where nodes incorporate findings, e.g. detection results, of their neighbors into their own detection process. Beside the aforementioned approaches, an essential characteristic of these cooperative schemes is the possibility of error propagation. That is, if a node arrives at a wrong conclusion and passes this conclusion on to its neighbors, the erroneous data spreads. The passed on error becomes a fault in the neighbor's system and, consequently, can also become a failure. Only when there is no cooperation between the nodes, can there be no error propagation. Error propagation can be quantified according to a number of points of view, including

- number of hops per unit of time that the error propagates

- maximum number of hops an error can propagate
- time until maximum degree of propagation is reached
- time until all consequences of the propagated error have ceased again

## 4.5 Case study 1 - single injections

In this case study, we will look at the behavior of the MA and LBC when being confronted with data containing independent single faults. Different fault probabilities will be considered. A related experiment, where different parameter sets of an MA were compared, was described in [HHW13]. The parameters for the considered MA and LBC are specified in Table 4.1

Table 4.1: Parameter sets for the EDAs in case study 1

| **MA** | |
|---|---|
| $k$ | 2 |
| $\Theta$ | 310.0 |

(a) Parameter set for the EDA based on a moving average

| **LBC** | |
|---|---|
| $\rho$ | 300 drops/t |
| $\Gamma$ | 430 drops |

(b) Parameter set for the EDA based on a leaky bucket counter

### 4.5.1 Input data

Figure 4.4 depicts the samples captured by the RIPLECS infrastructure. There are 123 values that Node 1 forwarded to the base station during the experiment. Figure 4.5 shows in a binary fashion for the same period of time, when an actual event took place. It is easy to see that the beginning of an event as indicated in Figure 4.5 causes the sampled value to increase almost immediately. Complementary, Figure 4.6, shows the respective positions of the sliders for the ambient light and for the event light. This figure explains the first three peaks in Figure 4.4, these stem from an elevated level of ambient light. For all three peaks the ambient light slider was brought to position 99. For the first peak, this position was held for one sampling interval, for the second peak for two sampling intervals and, finally, for the last peak, the duration was maintained for three sampling intervals. After these peaks, real events take place and the ambient light input stays between 19 and 26 in terms of slider positions.

**Event definition** In this case study, an event takes place when the slider position of the event light exceeds 67. The input to the RIPLECS infrastructure was chosen in a way so that the EDAs are confronted with several special cases that deviate from the simple case where an event starts, maintains a constant magnitude and abruptly ends. There is a total of twelve events that take place during the experiment. Only one of the peaks of the event light does not qualify for an event: sample 70

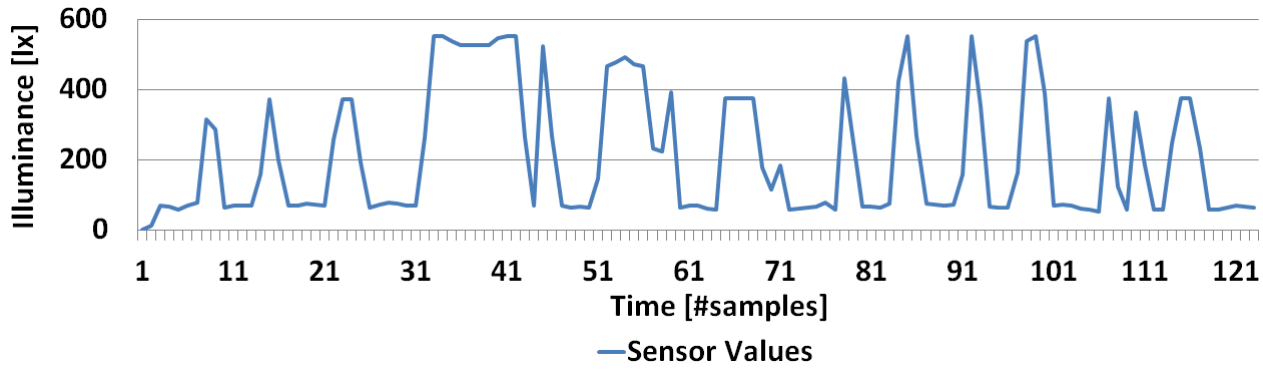



Figure 4.4: Input samples for case study 1 as captured by the RIPLECS infrastructure

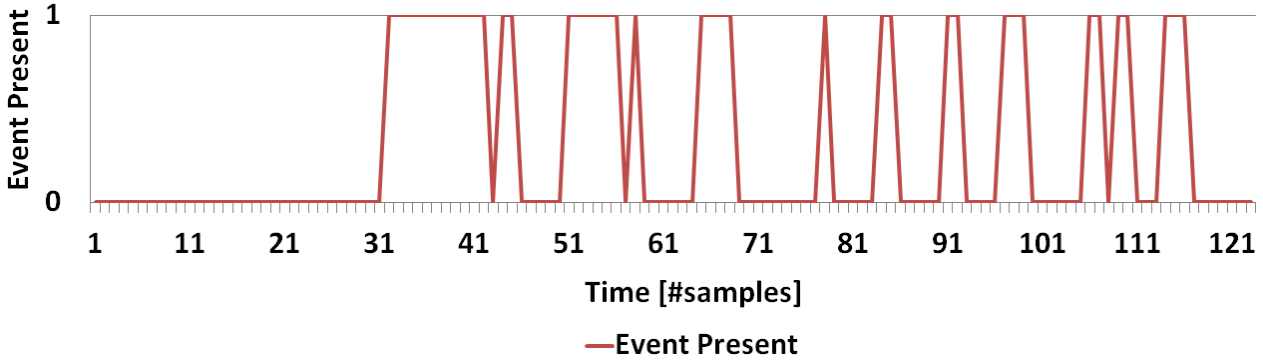



Figure 4.5: Event presence with respect to the input samples depicted in Figure 4.4

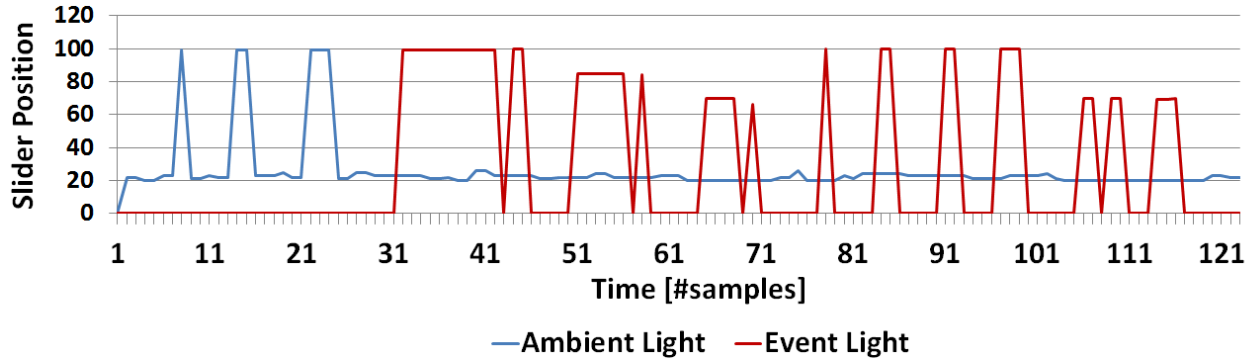



Figure 4.6: Positions for the sliders for the control of ambient light and event light in case study 1

corresponds to slider position 66. This is why Figure 4.5 does not indicate an event for sample 70.

The magnitudes with respect to the slider positions of the events is different with every instance: after the first event with 99, soon a very brief event with 100 follows. The next pair of events are of magnitude 85 and 84. The third pair features 70 and 66, with the last instant not qualifying as an event. The pattern of a long event followed by a brief one with only a brief time interval between them was chosen in order to see, if either EDA would identify two close events as one. In addition, if the only sample that separates the two events is randomly chosen to be targeted by the fault injection mechanisms, the detection outcome might change. The next four events are brief, lasting one, two or three samples. The slider for the event light was always brought up to 100. The reason for introducing these extremely brief events was to see whether the EDAs are capable of detecting such brief events. The last three events were chosen to just qualify as events. The respective slider positions are 70 and 69. The challenge for the EDAs is to correctly identify these last three events as events, but to not identify the first three peaks of ambient light as events. Appendix A.1 provides the original console output of the RIPLECS infrastructure, which was used for the evaluation of this case study.

### 4.5.2 Injected faults and fault probabilities

The faults that are injected in this case study are single faults, that is, for each fault that is injected, only one value is modified. It is, however, possible, that two or more successive values are modified because all of them were randomly chosen in separate operations, but even in this case, these modifications are independent from each other.

We employ two different contortion functions. In the first experiment of this case study, the correct value is substituted with zero. We refer to this experiment as Experiment A. In the second experiment random values $\in [0;1000]$ are injected. These limits were chosen because illuminance is by definition never a negative number and 1000 lux is roughly the maximum illuminance measurable by the considered node that can be produced using the event light and the ambient light set to their maximum. We refer to this second experiment as Experiment B.

For both experiments, we consider several different fault probabilities. That is, after a golden run with a fault probability of 0%, the fault probability is stepwise increased until a fault probability of 50% is reached, totaling in 26 different probabilities. The relevance of an EDA's performance when being confronted with a faulty data level of 50% is questionable. We chose this high upper limit of 50% in order to show the trend of the respective metrics.

### 4.5.3 Metrics

Both experiments are evaluated using the same metrics. These considered metrics include:

- number of detected events

- number of false positives
- number of false negatives
- number of failures
- cumulative duration of detected events
- cumulative duration of correctly detected events

The number of detected events can decrease if an event that consists of only one sample is canceled by an injected zero or an injected value that is very low. On the other hand, injecting high random values during a period of samples that feature a low magnitude can lead to an increase of the overall number of events. False negatives occur if no event was detected, when there was one present. False positives represent the opposite; here, an event was detected when there were none present. Both of these metrics describe a failure, because the EDA did not provide its correct service, i.e. detecting an event only for as long as it is present. Accordingly, the number of failures is the sum of false negatives and false positives.

The cumulative duration of detected events shows the duration of all detected events. Complementing the number of detected events, this metric allows one to see whether the EDAs tend to detect events. Finally, the cumulative duration of correctly detected events indicates how many of the samples, where an event was detected, actually correspond to detected events. The last two metrics were chosen in order to give an impression of the EDA's disposition to easily detecting events, whether or not there were any present.

It is notable that all of the aforementioned metrics are not necessarily optimal in the golden run, i.e. the fault-free case. The combination of the described metrics give valuable feedback about the behavior of the considered EDAs.

### 4.5.4 Fault injection and evaluation process

The first step in the evaluation process is to run the EDAs on the fault-free data depicted in Figure 4.4, where the fault probability is zero. Next, fault injection on the original fault-free data is performed. Starting with a fault probability of 2%, a vector containing the modified data is created. For Experiment A, this means that some of the data points have been changed to zero. In Experiment B, some of the data points have been substituted with a random value $\in$ [0;1000]. The vector containing the faulty data is then used as input for the LBC and the MA. That is, both EDAs use exactly the same input. Finally, the output of the EDAs is evaluated under the aspects of the aforementioned metrics. The fault probability is raised by two percentage points and the process is repeated. This procedure repeats until the maximum fault probability of 50% is reached.

For each fault probability $> 0$, the results are averaged over 500.000 runs. The ground truth, i.e. knowing for every instant in time whether an event takes place or not, is also provided via a vector that contains the data depicted in Figure 4.5.

The number of false positives and false negatives is determined via a comparison with this actual event data.

### 4.5.5 Results of case study 1

This section provides a detailed analysis of both injection campaigns. First, the results for Experiment A, where zeroes are injected are reviewed, followed by the results for Experiment B. For both experiments, diagrams of all metrics are presented, in order to compare the reactions to the different injected faults. A concise discussion of all results will be given at the end of the case study.

Figure 4.7 depicts the detection outcomes for both EDAs and fault-free input data, the golden run. In addition, the green line shows when an event actually took place.

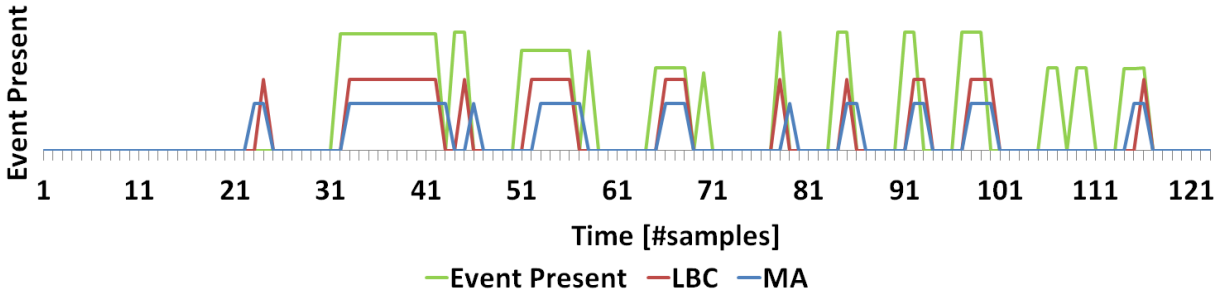



Figure 4.7: Detection results for both EDAs in the fault-free case, the golden run, in contrast to the actual events

Out of the 13 events caused by the event light, nine are correctly detected by both EDAs. Four events are not detected and both EDAs identify one of the peaks in the ambient light as an event. Making the EDAs more sensitive would lead to an increased detection rate, but not only with regard to actual events, but also with regard to peaks in the ambient light.

#### Experiment A - injecting zeroes

**Number of detected events - Figure 4.8** shows the number of detected events for the different fault probabilities. The true number of events is 13, as indicated by the green line. Both EDAs tend to detect fewer events as the fault probability rises. The LBC's detection rate, however, decreases at a smaller rate to the MA's, ending up with a total of eight detected events when the fault probability is as high as 50%.

**Cumulative duration of detected events - Figure 4.9** depicts the entire duration of all events detected by the respective EDA. This includes the correct and incorrect detection results. Naturally, this number decreases for both EDAs, because zeroes are injected. What is interesting about this figure is that after

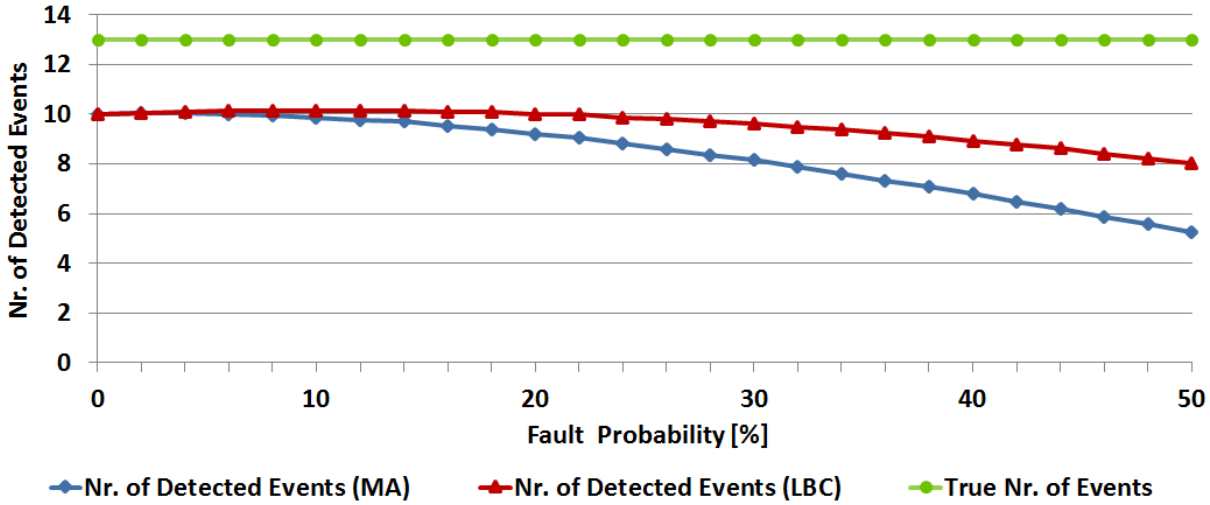



Figure 4.8: The number of detected events in contrast to the actual number of events when zeroes are injected

starting with a higher number than the LBC, the MA falls below the level of the LBC's duration of detected events.

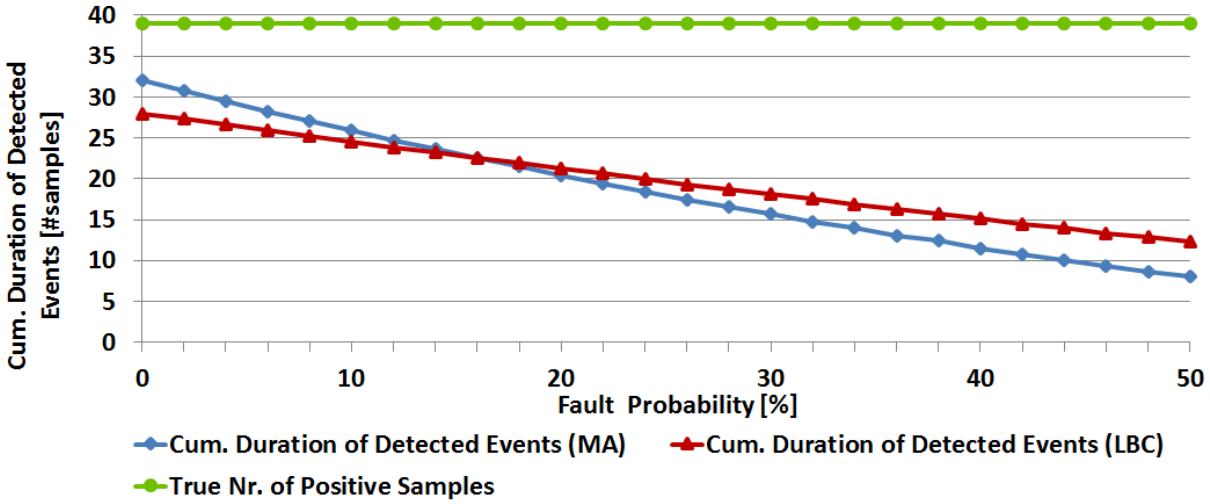



Figure 4.9: The cumulative duration of detected events when zeroes are injected, the green line indicates the correct cumulative event duration of 39 samples

**Cumulative duration of correctly detected events - Figure 4.10** In contrast to the previous figure, Figure 4.10 only takes into account the number of samples where an event was correctly detected. This number is of course lower than in the previous figure, as the false positives are not taken into account. Here, both graphs decrease with a comparable degree. While in the golden run the differ-

ence between the two corresponding data points is only two, for a fault probability of 50% this difference increases to six.

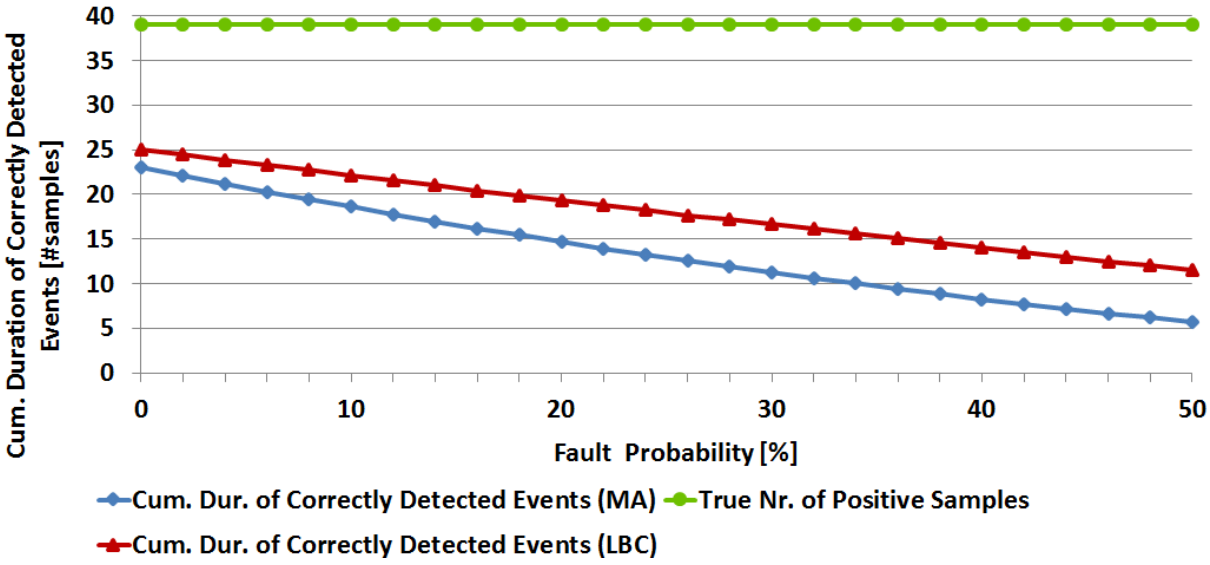



Figure 4.10: The cumulative duration of correctly detected events when zeroes are injected, again, the green line indicates the correct cumulative event duration of 39 samples

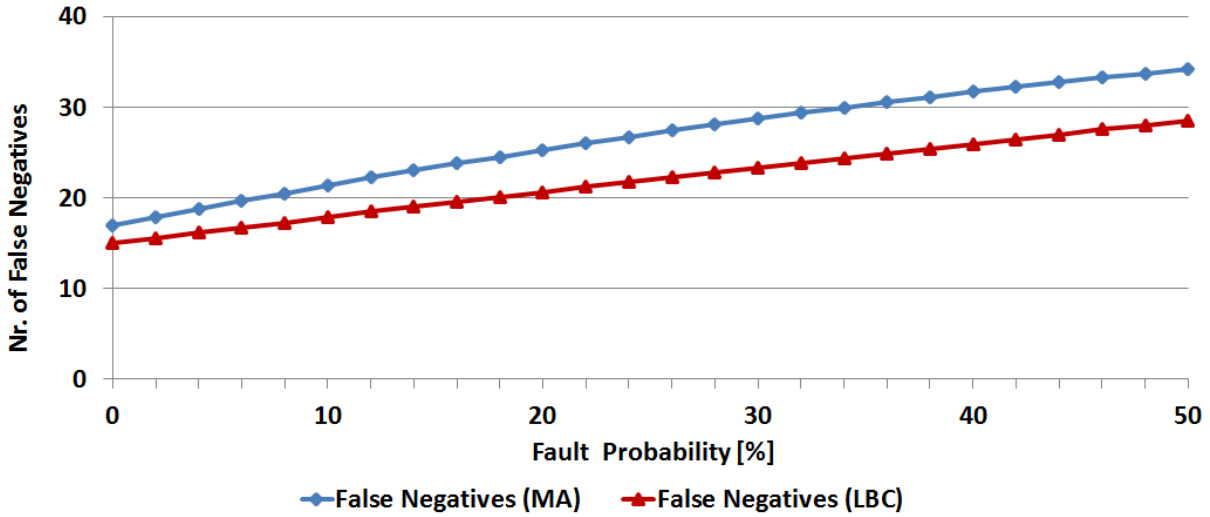



Figure 4.11: The number of false negatives when zeroes are injected, this number is always lower if the LBC is employed

**Number of false negatives - Figure 4.11** The number of false negatives is distinctly always higher when using the MA. The gap between the two lines widens

from two to six.

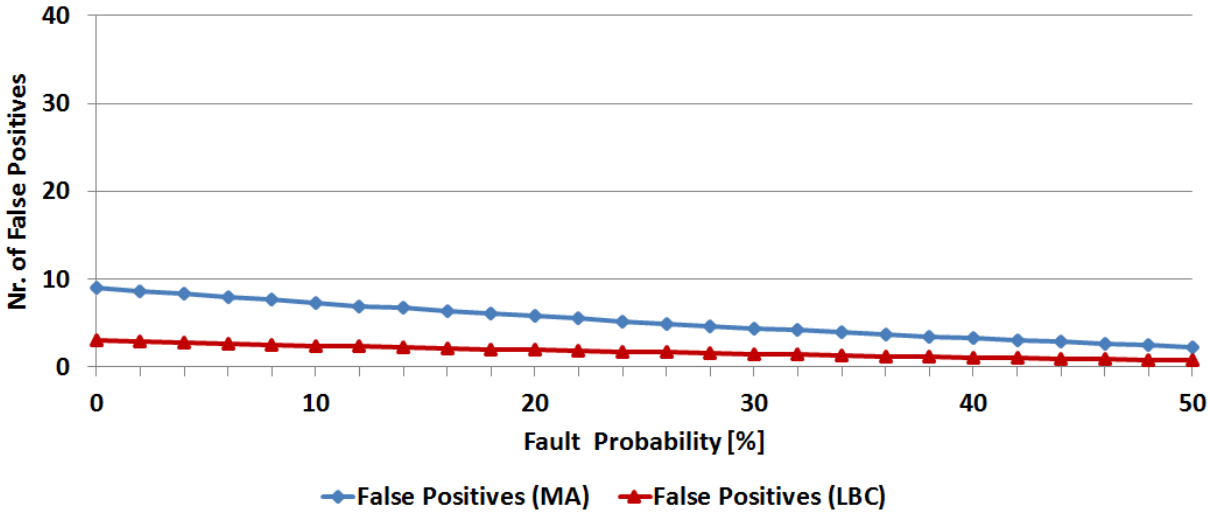



Figure 4.12: The number of false positives when zeroes are injected, this number is particularly low for the LBC, as an injected zero can never cause the bucket to overflow

**Number of false positives - Figure 4.12** The number of false positives is low for both EDAs, decreasing from nine to just over two for the MA and from three to close to zero for the LBC. For low fault probabilities, there is a distinct difference between the two EDAs. The MA's higher initial number of false positives stems from the fact that the MA is slower in detecting the end of an event. Every sample that is wrongly identified as the continuation of the detected event is a false positive. Due to the outflow with every instant of time, an injected zero can never cause the LBC to detect an event.

**Number of failures - Figure 4.13** The number of failures is the sum of the two metrics that were discussed previously, false negatives and false positives. This characteristic increases for both EDAs with comparable gain. According to this metric, the MA performs consistently worse than the LBC. This is consistent with the trend shown in the previous two figures.

### Experiment B - injecting random values

The second experiment is concerned with injecting random values $\in$ [0;1000]. In the following, the same metrics as in Experiment A are reviewed.

**Number of detected events - Figure 4.14** In contrast to the previous experiment, the number of detected events rises constantly for both EDAs. This is

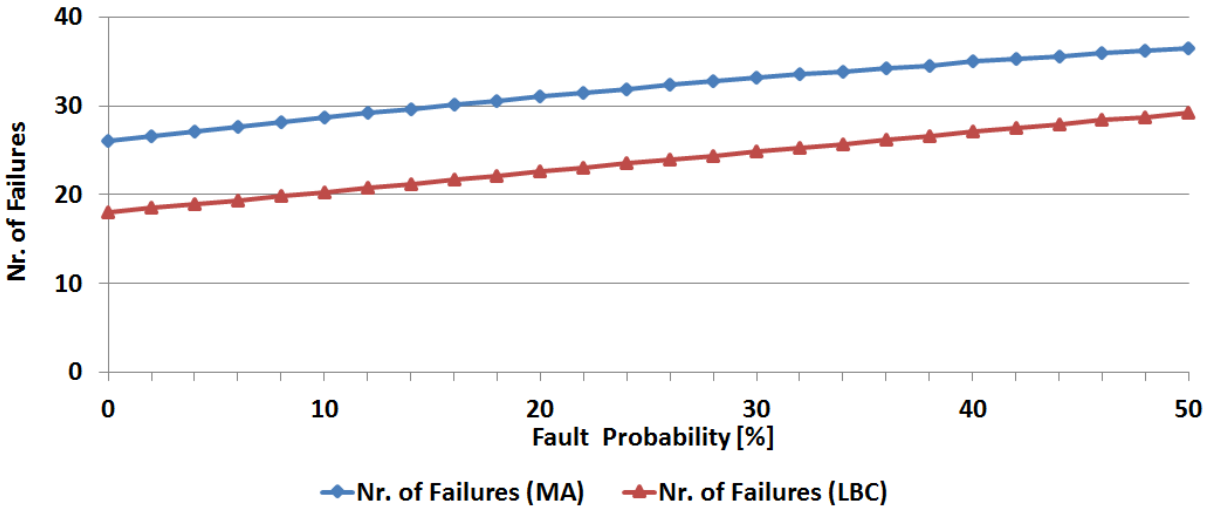



Figure 4.13: The number of failures, the sum of false positives and false negatives when zeroes are injected, in accordance with the previous two figures, the MA consistently shows a higher failure rate than the LBC

because a high-valued injected fault can cause either EDA to detect an event. The LBC, more than doubling the number of detected events, is more affected by this than the MA.

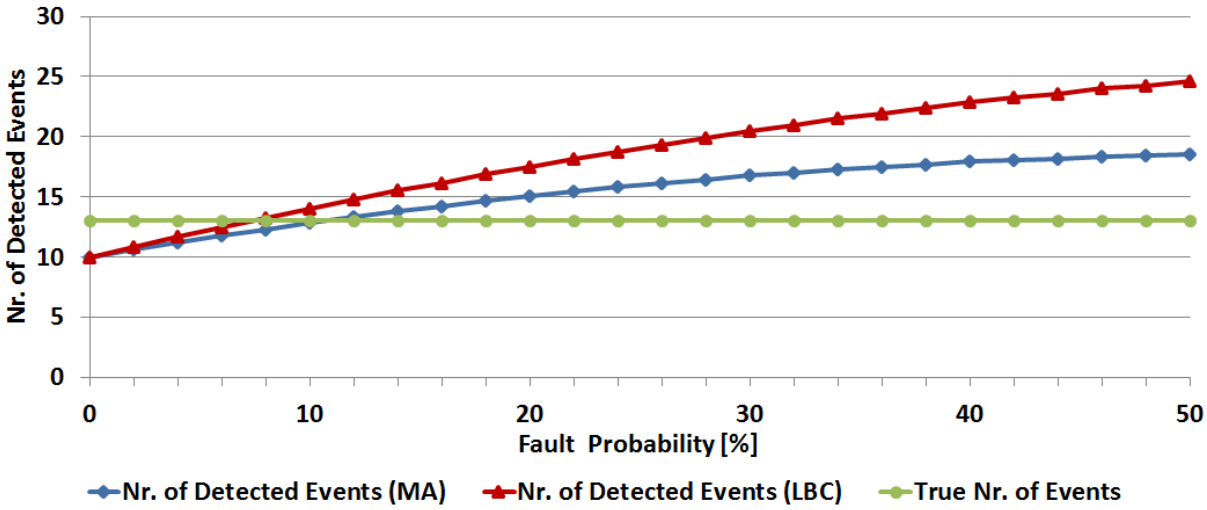



Figure 4.14: Number of detected events when random values are injected

**Cumulative duration of detected events - Figure 4.15** Altogether, the LBC tends to detect events for shorter periods of time than the MA does. The EDAs cross the true cumulative event duration of 39 samples at 10% and 22%, respectively. For a fault probability of 50%, the two final data points differ by roughly 14.

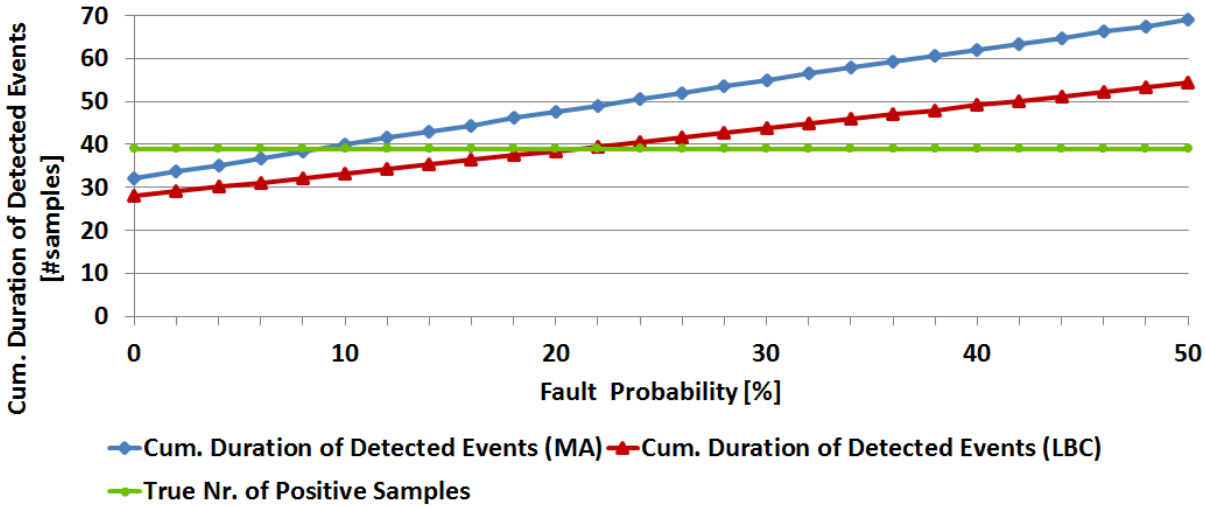



Figure 4.15: Cumulative duration of detected events when random values are injected

**Cumulative duration of correctly detected events - Figure 4.16** indicates that the duration of the correct detection outcome is nearly constant for the LBC with 25 samples at the beginning and 26.13 samples for a fault probability of 50%. The MA increases this duration from 23 samples to 28.34 samples during the same interval.

**Number of false negatives - Figure 4.17** is close to constant in the case of the LBC, with an initial number of 15, that decreases to 13.87 for the maximum fault probability. The MA starts with a higher number of false negatives than the LBC, 17, that decreases to 11.66. When the fault probability hits roughly 22%, the MA's number of false negatives falls below that of the LBC.

**Number of false positives - Figure 4.18** For all fault probabilities, the LBC is less affected by the injected random faults. From the initial three false positives, the number rises to just over 28, while the MA starts with a total of nine false positives and finishes with more than 40.

**Number of failures - Figure 4.19** The trend concerning the number of failures is dominated by the trend exhibited by the number of false positives in the previous figure. The LBC clearly shows a lower number of failures throughout all fault probabilities. The gap between the two graphs increases steadily from eight to twelve failures.

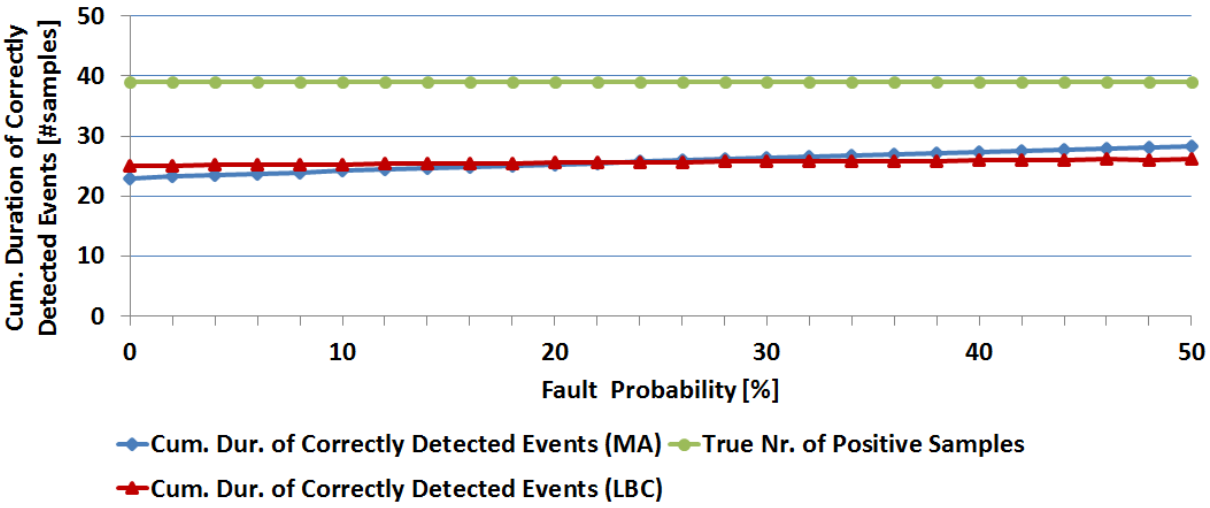



Figure 4.16: Cumulative duration of correctly detected events when random values are injected

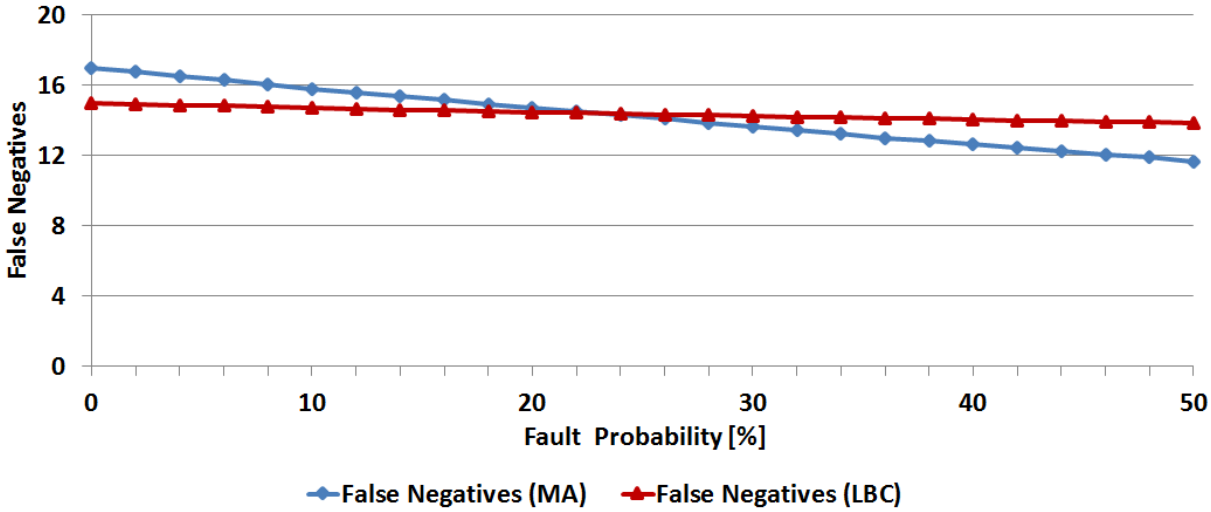



Figure 4.17: Number of false negatives when random values are injected

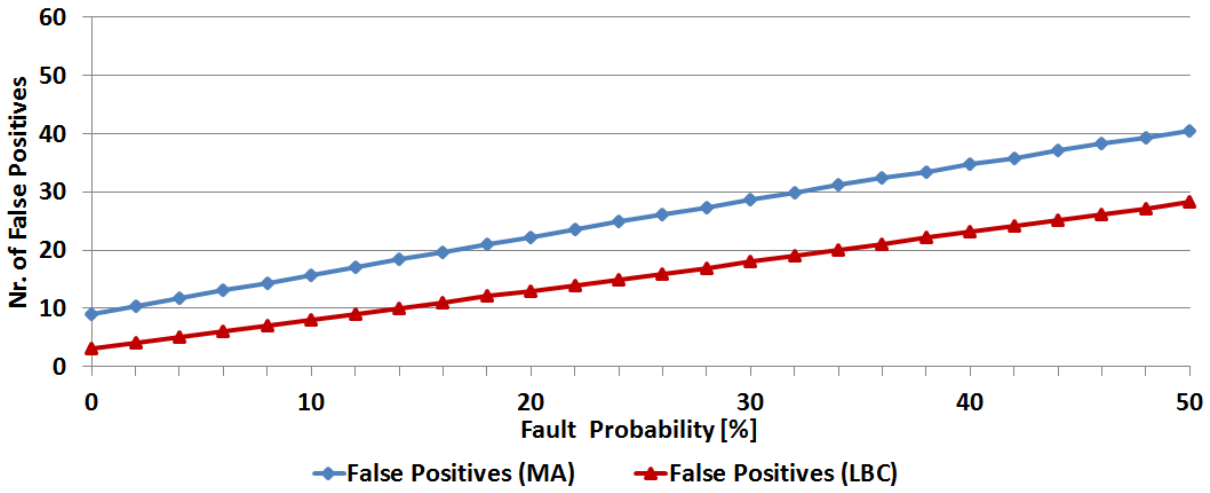



Figure 4.18: Number of false positives when random values are injected

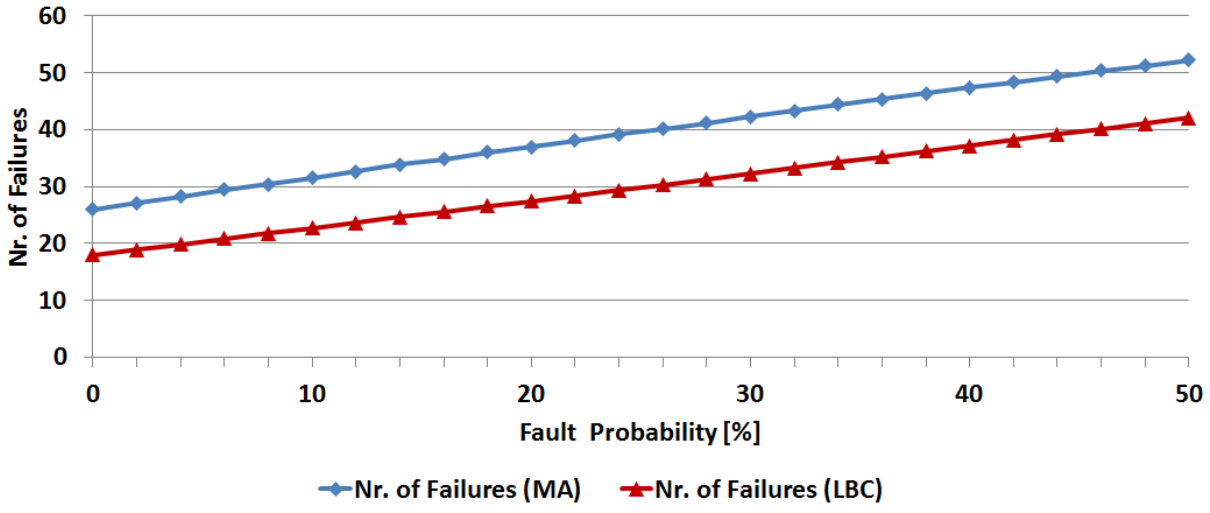



Figure 4.19: Number of failures, the sum of false positives and false negatives when random values are injected

### Discussion of experiment A - injecting zeroes

Injecting zeroes as administered in this case study will eventually lead to a decrease in the actual number of events present as an injected zero cannot constitute an additional event. Naturally, both EDAs exhibit a decrease in the number of detected events and both, the duration and the correct duration of the cumulative events. The interesting facet is, the extent to which the two EDAs differ in their behavior. A prominent example for an explicit difference is the cumulative duration of detected events, where the lines cross each other.

The final judgment is delivered by the concluding figure that displays the number of failures. The preceding metrics of false positives and false negatives viewed individually only show small differences between the EDAs, but the sum of these deviations culminates in a significant difference between them.

### Discussion of experiment B - injecting random values

In contrast to the previous experiment, injected random values of up to 1000 can look like additional events. To that effect, all metrics, except for the false negatives, display an increase.

Noteworthy in this experiment is that the LBC obviously is significantly more perceptive to immediately detecting an event when a high value was injected. This can be deduced from the fact that the number of failures as well as the cumulative duration of detected events are both considerably lower for the LBC, but still the LBC detects more events. In addition, the number of false positives is lower for the LBC. Combining this with the number of detected events indicates, that the MA receives more false positives per wrongly detected event than the LBC. That is, on average, the LBC rather detects a brief event for every injected value and the MA takes more samples to realize that the event has already ended.

### Concluding remarks

Looking at the combination of the number of detected events in both experiments and the detection outcomes in the golden run from Figure 4.7 suggests that the LBC reacts more quickly to changes in the input. This is nice in the fault-free case, but can lead to an unwanted high number of false positives, when a fault is recognized as an event.

We chose a wide variety of metrics for this case study in order to show that there cannot be an ultimate best candidate for all applications and that some properties can be discovered only by looking at more than one metric at once. In addition, the suitability of the EDA depends on the application's demands. These demands can be expressed in a subset of the aforementioned metrics, optionally ranked in importance. Consider for example that although the LBC shows rather better performance in this case study, the superiority ends when a low number of false alerts is desirable. Although the LBC displays the lower number of false positives in both experiments, the number of detected events is higher in both cases. This

means, that any actions triggered upon the detection of an event are carried out at a much higher rate than in the case of the MA.

Even if it is possible to conceive the general behavior of the EDAs when being confronted with erroneous data, only exhaustive experiments illustrate the actual magnitude of the impact of erroneous data. The AE benefits from this analysis in two ways. First, she can make an informed decision when choosing one of the two EDAs, possibly she can use the evaluation outcome for a risk analysis. Second, she can modify the LBC in order to make it less sensitive and run the analysis again.

## 4.6 Case study 2 - burst faults

This second case study also compares the LBC and the MA, but uses a scenario that is very different from the scenario examined in the first case study. Some of the metrics established previously are reconsidered and two new metrics targeting dependability are introduced. The parameters for the MA and LBC are specified in Table 4.2. The parameters differ from those in the first case study because they are optimized for a different form of event. While the first case study tested the capabilities of "all-round EDAs" that can detect long and brief events of diverse magnitudes, this case study focuses on the detection of a single, long event. In addition, burst faults instead of single faults are considered.

Table 4.2: Parameter sets for the EDAs in case study 2

| **MA** | |
|---|---|
| $k$ | 2 |
| $\Theta$ | 137.0 |

(a) Parameter set for the EDA based on a moving average

| **LBC** | |
|---|---|
| $\rho$ | 300 drops/t |
| $\Gamma$ | 395 drops |

(b) Parameter set for the EDA based on a leaky bucket counter

### 4.6.1 Input data

Figure 4.20 depicts the sensor's measurements in lux in blue and an additional red line indicates if an event is present for every instant in time. There is only a small latency between the start of the event and the increase of the sensor values' magnitudes. The input includes a total of 66 samples.

**Event definition** Again, an event takes place, when the slider for the event light exceeds 67. There is only one event with a duration of 33 samples and the slider is at position 70 during the entire event. Figure 4.21 shows the slider positions for both the event light and the ambient light during the experiment. Complementary to the first case study, there are no peaks in the ambient light, only a little noise is visible. The corresponding console output can be found in Appendix A.2.

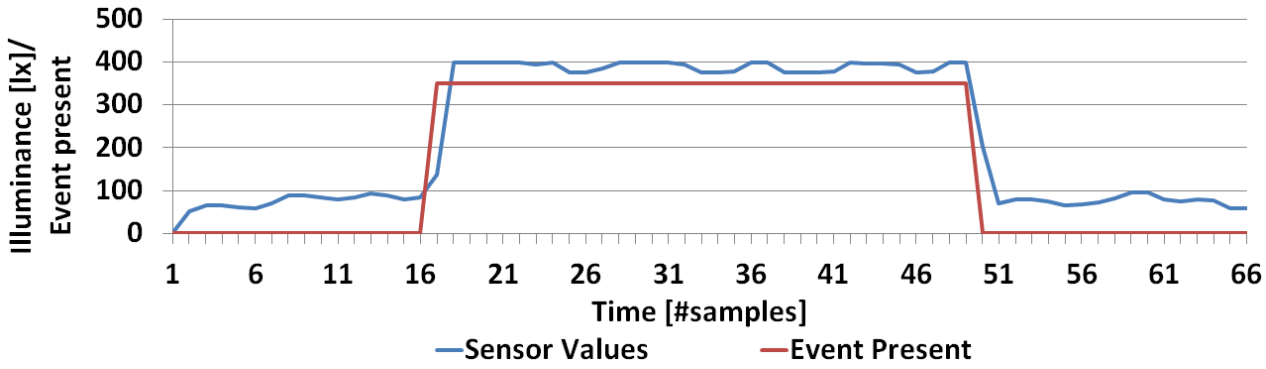



Figure 4.20: Input samples for case study 2 as captured by the RIPLECS infrastructure

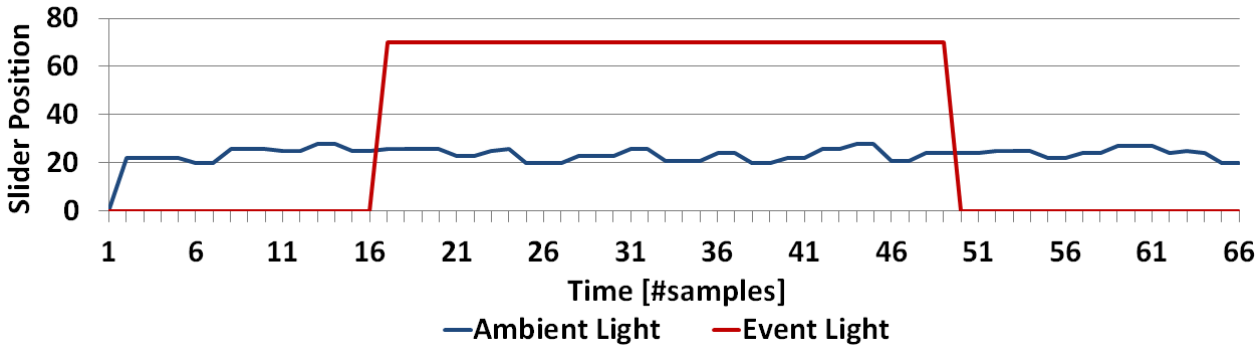



Figure 4.21: Positions for the sliders for the control of ambient light and event light in case study 2

### 4.6.2 Injected faults

In contrast to the first case study, we consider the injection of burst faults in the following experiment. As a matter of principle, there is a special case of a minimal burst duration of only one sample, which corresponds to the fault scenario in case study 1. However, a burst fault lasts in general for several samples, hence the name. We will conduct one experiment in this case study, where random values $\in$ [0;1000] are injected. We do not consider the injection of zeroes because both EDAs display nearly identical behavior under these circumstances.

Instead of varying the fault probability as was done in case study 1, we now vary the duration, i.e. the length, of the injected burst faults. That is, after the initial golden run, we inject exactly one burst fault that lasts for two samples. Subsequently, the burst length is increased by two with each run up to a maximum burst length of 66, which corresponds to the total number of samples in this scenario. During each run, only one burst fault is injected within these 66 samples.

### 4.6.3 Metrics

The first four of the metrics below have already been used and described in the previous case study. Two new metrics are introduced in the following, yielding a total of six considered metrics:

- number of detected events
- number of false positives
- number of false negatives
- number of failures
- number of active faults
- fault latency

The two new metrics aim at examining the EDAs' behaviors with regard to a dependability aspect. The number of active faults shows how many of the injected burst faults actually cause the EDAs' services to fail, i.e. to produce a false negative or a false positive. The fault latency illustrates how long it takes from injecting a fault until the EDAs start to provide incorrect service when a fault has become active.

### 4.6.4 Fault injection and evaluation process

As before, a golden run is performed first, i.e. a burst fault of length zero is injected. Subsequently, we inject the first real burst fault with a length of two. When injecting a burst fault, one of the 66 samples is randomly chosen as the start of the burst fault, that is, any sample is selected with a probability of $0.01\dot{1}\dot{5}\%$. The selected sample and its successors are then replaced by random values $\in$ [0;1000] until the

number of replaced values equals the length of the burst fault. The resulting vector is used as input for the MA and the LBC, so that both EDAs operate on exactly the same data. Finally, the metrics are calculated and the next fault vector is determined.

It may occur that due to a high index number of the sample that was chosen to be the first sample of the burst fault, there may not be enough samples left to accommodate the entire burst fault. In such a case, the burst fault is shortened to end with the $66^{th}$ sample. As the length of the burst fault increases, this situation arises more frequently. Taking this condition into account, we performed a total of 500.000 runs per length of burst fault and averaged the results.

The ground truth used for establishing the correctness of the detection results is determined by the slider positions depicted in 4.21. This applies to the number of false positives, number of false negatives and number of failures and the comparison of the number of detected events to the number of actual events. Contrary to that, the dependability related metrics of latency and number of active faults are compared to the golden run. Otherwise incorrect detection results that would also occur during a run performed with clean data would be wrongly attributed to being caused by injected faults. Analogous, a fault is an active fault only if an incorrect detection outcome that would not have been present in the golden run is caused by this injected fault.

### 4.6.5 Results of case study 2

Figure 4.22 shows the undisturbed detection outcome for the MA as well as for the LBC in combination with a line indicating the presence of the event. Both of the EDAs detect the start of the single event just one sample after the event started. The end of the event, however, is correctly detected by the LBC while the MA is one sample late.

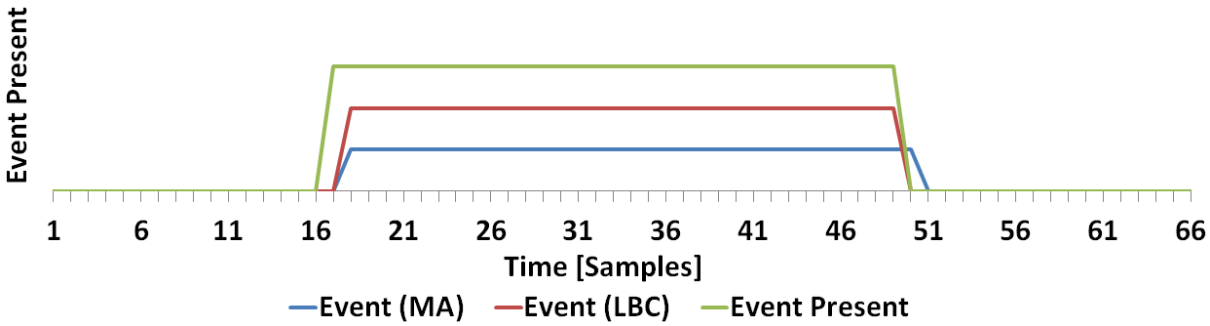



Figure 4.22: Detection results for both EDAs in the fault-free case, the golden run, in contrast to the actual event

**Number of detected events - Figure 4.23** The green line shows the ground truth - only one event is present - and the two EDAs exhibit very different behaviors. While both EDAs display a gain, the LBC detects close to eight events

for a maximum burst length, while the MA only doubles the number of detected events roughly from one to just a little over two. As the magnitude of any injected value ranges from zero to 1000, an injected burst fault can be detected as several events. This is the case if high and low values alternate. Obviously, the LBC is more sensitive to deviations of the input than the MA.

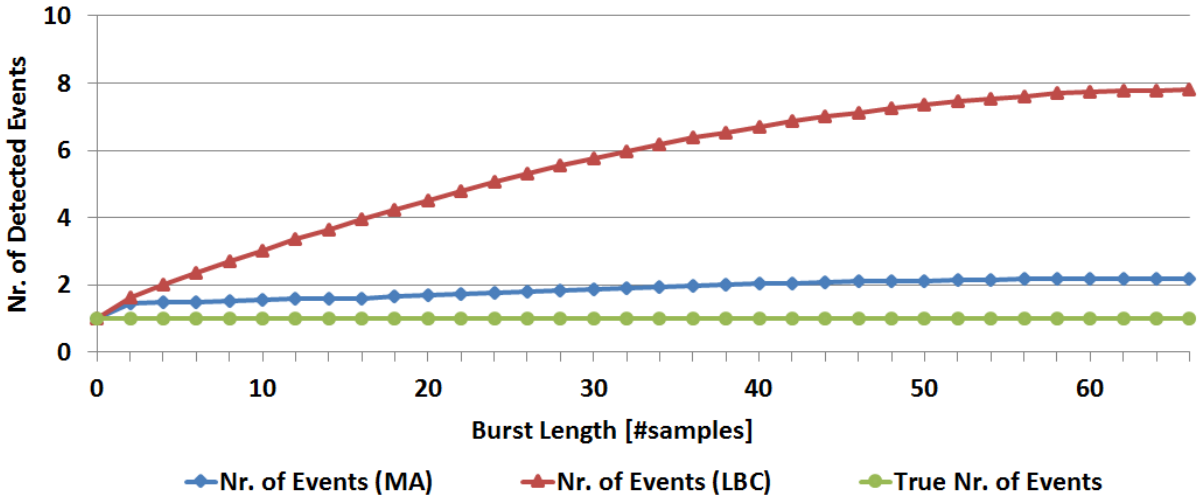



Figure 4.23: Number of detected events when bursts of random values are injected

**Number of false negatives - Figure 4.24** As the event in this scenario lasts for 33 samples, the maximum number of false negatives possible is also 33. The MA shows a nearly constant number of false negatives, rising from one to only 1.36. The LBC has a greater gain; also starting with a single false negative, this line reaches approximately 6.2 false negatives for a burst length of 44 and stays on this level.

**Number of false positives - Figure 4.25** The difference in behavior of the two EDAs is not especially exciting, both show comparable gain with an offset growing from one to more than five. Here, the MA constantly displays a higher number of false positives. This figure primarily was included to serve as complementary information for the following figure.

**Number of failures - Figure 4.26** This sum of false negatives and false positives is nearly identical for the two EDAs. The maximum deviation is only 1.24 for a burst length of two and the two lines intersect each other twice. The figure suggests that, concerning the failures, the two EDAs are of comparable quality. Combining this figure with the previous two indicates, once again, that each EDA has different strengths and weaknesses.

**Number of active faults - Figure 4.27** In each run, only one fault is injected, it is only the length of this burst fault, that increases. Consequently, the number

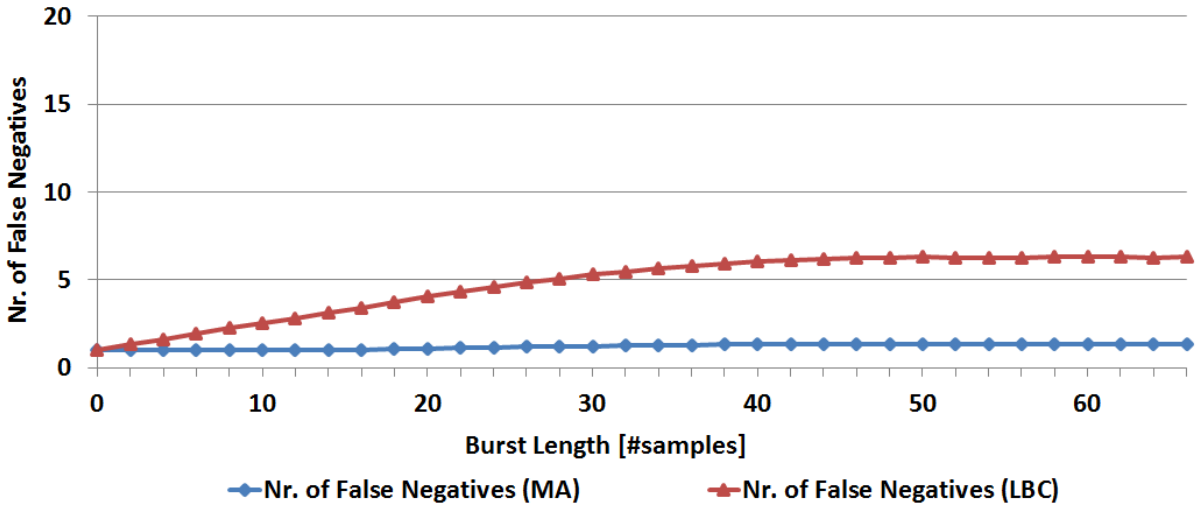



Figure 4.24: Number of false negatives when bursts of random values are injected

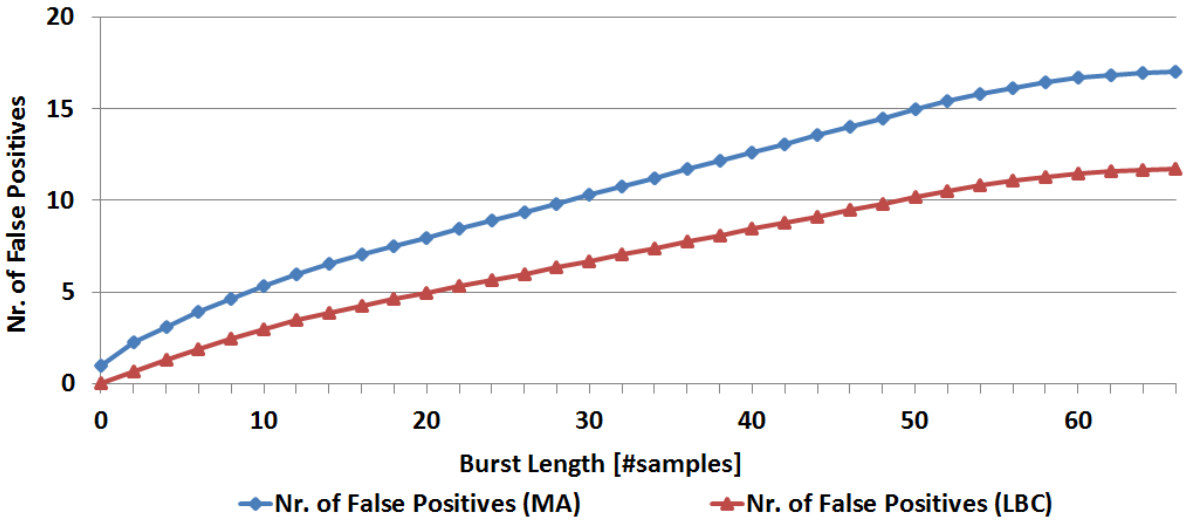



Figure 4.25: Number of false positives when bursts of random values are injected

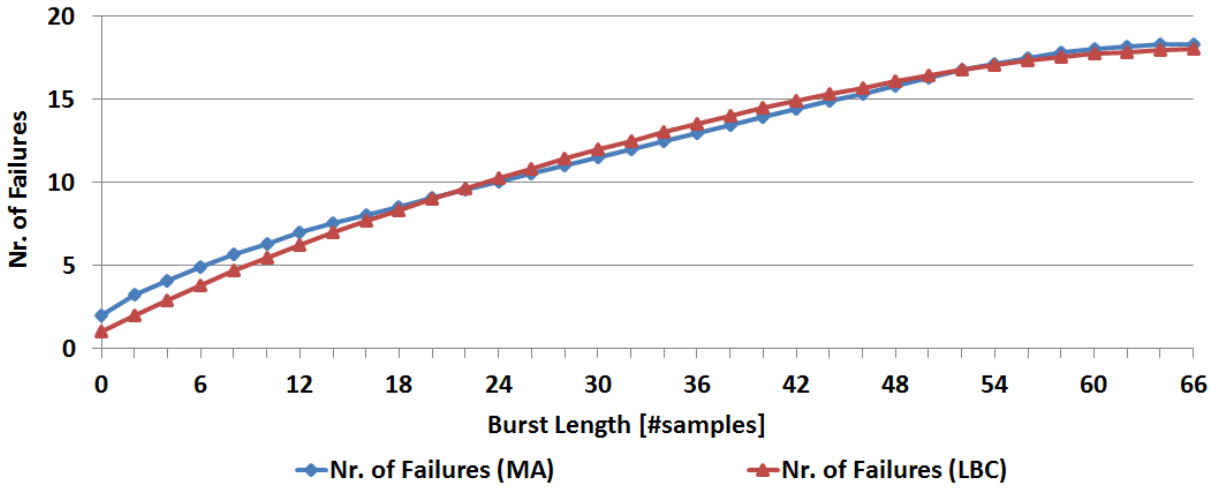



Figure 4.26: Number of failures, the sum of false positives and false negatives when bursts of random values are injected

of active faults cannot become more than one. The LBC already reaches this point for a burst length of 16, whereas the MA does not join in before burst length 34, suggesting the MA to be more resistant to injected faults with a burst length below 34 than the LBC. Each injected burst fault can become active only once, irrespective of the burst fault's length. That is, if a burst fault causes the detection outcome to become incorrect again after it had already been restored to a correct state, it is only accounted for as one active fault.

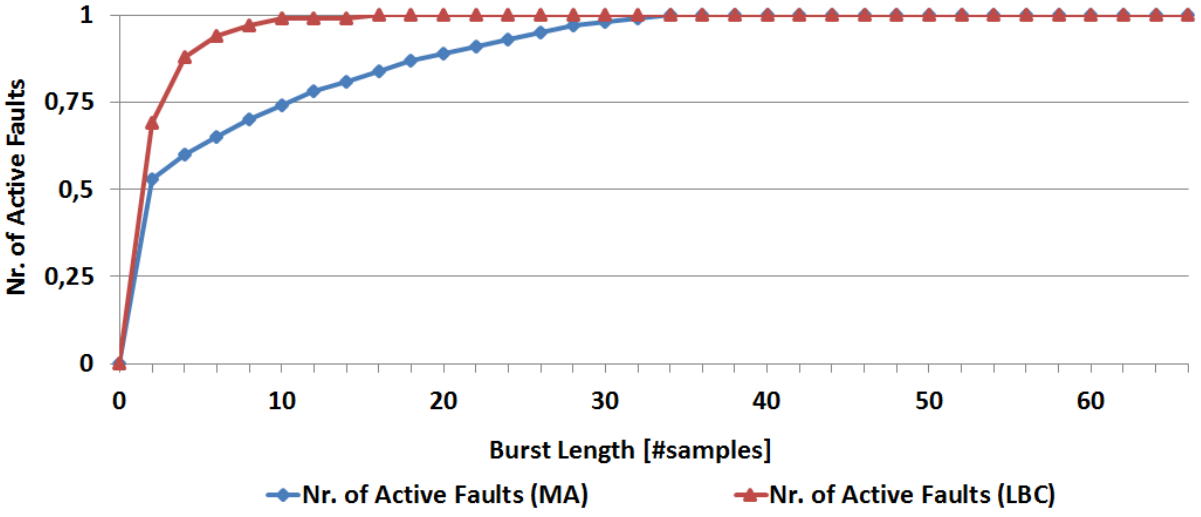



Figure 4.27: Number of active faults when bursts of random values are injected

**Fault latency - Figure 4.28** Both lines experience periods of steady gain followed by a sequence of values close to being constant. The trends change between burst lengths 34 and 36 for the MA and between burst lengths 16 and 18 for the LBC. The reason for the change in gain is explained by the corresponding number of active faults depicted in the previous figure: as soon as the number of active faults reaches one, i.e. all injected faults become active, the fault latency stays constant. The fault latency was averaged over the number of runs where a fault became active as opposed to the entire 500000 runs. During the time, where no event is present, both EDAs can wrongly detect an event if only a single fault is injected. Considering $\Theta = 137$ and $\Gamma = 395$, injecting a value greater than 274 and greater than 395, respectively, would already suffice. On average, injecting values of this magnitude is no rare occasion, as the average magnitude of the injected values is 500. Over

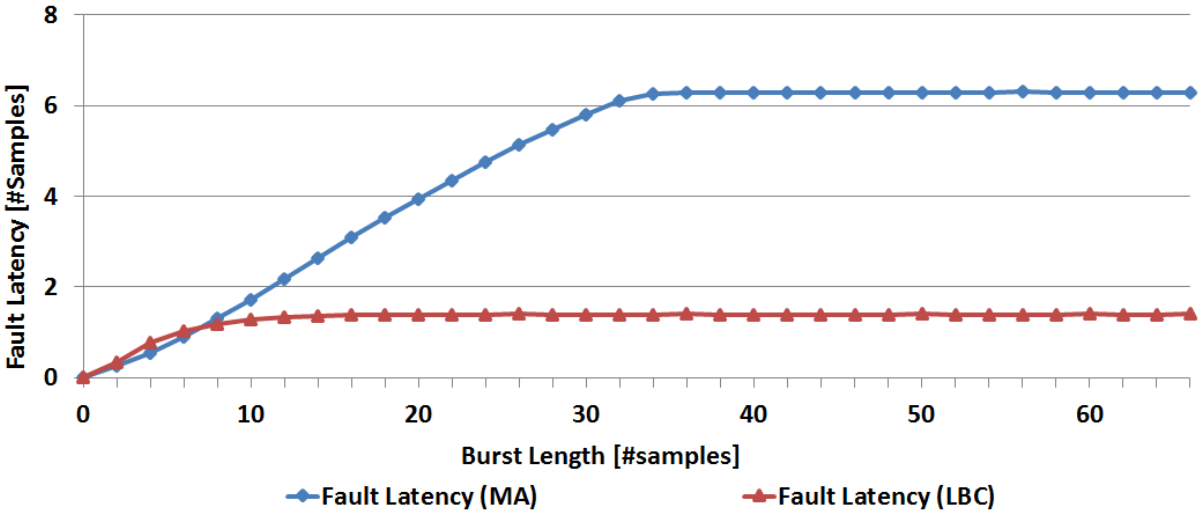



Figure 4.28: Fault latency when bursts of random values are injected

the period of time, where the event is present, injecting a single zero would not change the detection outcome of the MA. The reason for this is that the individual sensor values range up to nearly 400 lux, which yields an average of at least 200 for two consecutive samples, irrespective of the magnitude of the second value. The LBC, however, would already produce a false negative for a single injected value of roughly 300 or lower. This is why the LBC on average reacts considerably quicker to injected faults, within less than two samples for most of the burst lengths.

### Discussion of the experiment - injecting burst faults

Albeit the number of active faults is considerably different during the first third of the experiment, the number of failures is virtually equal for the two EDAs. The differences in the numbers of false positives and negatives underline that the actual consequences of the injected burst faults are quite different.

The dependability related metrics clearly show that the LBC is more responsive

to the injected burst faults. The LBC reaches the point where 100% of the injected burst faults become active significantly earlier than the MA and reacts on average considerably faster to the injected faults. The number of detected events is a good example that illustrates how differently the effects of the injected faults can manifest themselves. Here, the MA shows nearly a constant detection rate while the LBC displays a steeply rising detection rate.

## 4.7 Summary and discussion of the case studies

The two case studies examine the EDAs from different viewpoints. While in the first case study, many events of different characteristics were used for the analysis, the second case study features only one long event. In addition, the employed metrics differ partially in the two case studies.

The first case study shows for both experiments - the injection of zeroes and the injection of random values - a distinctly lower number of failures for the LBC. This is also visible when looking at the constituting elements of the number of failures, the numbers of false negatives and false positives. There is only one exception in this trend: when looking at the numbers of false negatives for injected random values, the LBC's number becomes higher for the second half of the experiment.

Table 4.3: Summary of results of case study 1, + denotes good performance and ++ refers to superior performance in comparison to the other EDA

| | **Experiment A** | | **Experiment B** | |
|---|---|---|---|---|
| | **MA** | **LBC** | **MA** | **LBC** |
| Detection quality in the golden run | + | | ++ | |
| Number of detected events | + | ++ | ++ | + |
| Cum. duration of detected events | + | + | + | + |
| Cum. duration of correctly detected events | + | ++ | + | + |
| Number of false negatives | + | ++ | + | + |
| Number of false positives | + | ++ | + | ++ |
| Number of failures | + | ++ | + | ++ |

In both case studies, the LBC's number of detected events rises more steeply than the MA's when random values are injected. The reason for this behavior is that the LBC is more sensitive to changes in the input. That is an advantage when the beginning and end of events are to be detected quickly, but turns into a disadvantage when faults just as quickly result in failures. The injection of zeroes causes a decrease in the LBC's number of detected events that is less steep than it is for the MA. At first sight, this looks like a positive fact, but is indeed due to fact that the LBC tends to identify one disrupted event as multiple events. Table 4.3 summarizes the results of case study 1.

The second case study confirms the pattern described in the first case study:

The LBC is very responsive, to events as well as to faults. The ultimate confirmation is presented by the number of active faults that clearly shows that, by trend, faults become active more easily when employing the LBC. In addition, the fault latency is smaller for the LBC. A smaller fault latency, however, provides an advantage: the consequences of faults can be observed more easily, as they take effect soon after their occurrence.

Table 4.4: Summary of results of case study 2, + denotes good performance and ++ refers to superior performance in comparison to the other EDA

| | **MA** | **LBC** |
|---|---|---|
| Detection quality in the golden run | + | ++ |
| Number of detected events | ++ (steady) | + (increasing) |
| Number of false negatives | ++ | + |
| Number of false positives | + | ++ |
| Number of failures | + | + |
| Number of active faults | ++ | + |
| Fault latency | higher | lower |

It remains to be answered which EDA is to be chosen. With good reason there were no absolute decision criteria mentioned at the beginning of this chapter, i.e. we did not specify how important a timely detection result was as opposed to robustness against faults. This was done in order to present a comprehensive overview of the EDAs' characteristics. That is, depending on the actual application, it may not be necessary to detect beginnings and ends of events as timely as the LBC does, but other attributes may be of more interest. In this light, it can be a good characteristic if the EDA shows a certain inertia, e.g. because the actions resulting from the detection of an event are complex or are heavy on resource requests.

A comparison of the golden runs indicates a comparable quality of detection results, with the LBC featuring slightly better values, especially with regard to case study 1. Applying the proposed methodology to the considered scenarios shows that the LBC is the better choice when the timely detection of events and a low number of false positives is desirable. On the other side, the MA's most prominent characterizing attributes include a slower responsiveness to events and to faults resulting in a high fault latency. In addition, for fault probabilities below 40%, the number of faults that become active is less for the MA, allowing it to provide correct detection results even in the presence of erroneous input data.

While the number of failures when bursts of random values are injected is nearly identical for both EDAs, there are considerable differences concerning false positives and false negatives. Employing the proposed methodology allows the AE to find out which of the EDAs is best suited for her purposes. In addition, the AE can derive feedback from the evaluation in order to revise the considered EDAs. The feedback of the conducted experiments is that the LBC reacts very quickly to changes in the magnitude of the input and that this is quite unfavorable if there is an erro-

neous input. From that, the AE can conclude that the dynamics of the LBC can be altered by introducing a maximum for the change of the magnitude, that is, a maximum total amount of change in the sensor's samples between two sampling intervals. Allowing only limited change in two successive sensor samples by using an intermediate value would cause a dampening of the LBC's reaction. Such an intermediate value could be defined as an absolute number, e.g. 500 lx, or alternatively, as a percentage of the difference of the two newest values with a predefined maximum.

Depending on the AE being satisfied with the presented characterization resulting from applying the methodology, the AE will either choose one of the EDAs or rerun the evaluation using the EDA that was modified as suggested.

# Chapter 5

# Conclusion

## 5.1 Summary

This dissertation presents a methodology for assessing EDAs for WSNs by means of fault injection. The outcome of this assessment provides an additional selection criterion when choosing an EDA or a parameter set for an EDA. Extending beyond traditional selection criteria such as quality of detection in the fault-free case, memory consumption or processing speed, the presented approach addresses a hitherto neglected dependability aspect.

The need for assessing the EDAs from a dependability inspired point of view originates in the WSNs' constant exposure to disturbing influences of the environment. These influences can cause the nodes to end up with inadequate reflections of reality. The effects of the external influences, referred to as disturbances, are modeled using a fault injection mechanism in order to show the effects' impacts on the EDAs' quality of detection. The proposed methodology allows the discovery of strengths and weaknesses of EDAs and to provide relevant feedback to the application engineer. Nevertheless, analyzing the EDAs' behaviors when being confronted with erroneous input is not only interesting for judging which EDA is better suited, but also to learn about the characteristics of the individual EDAs. In addition, the AE may derive inspiration from the interpretation of the evaluation results on how to optimize an EDAs' implementation, thus mitigating or even removing an EDA's weakness.

Two detailed case studies comparing two EDAs demonstrate the applicability of the proposed methodology. As an additional part of the contribution, in the course of the case studies the leaky bucket counter as a threshold mechanism for one of the considered EDAs is introduced. As the EDAs may display similar performances with regard to a subset of the considered metrics, not every metric that is chosen beforehand, may yield a conclusive result. Both case studies show that it is vital to thoroughly consider which metrics, and accordingly, which combinations of metrics are of interest and of significance. The evaluation of the conducted experiments repeatedly underline that in the considered scenarios, the EDA that is based on a leaky bucket counter performed better with respect to the considered metrics, while

the EDA based on a moving average was more robust against faults when it came to the number of detected events.

## 5.2 Future prospects

While the concept was successfully demonstrated by the two presented case studies, there remain a few possible improvements to the existing implementation.

**Graphical user interface** Providing an advanced graphical user interface (GUI) would facilitate the user interaction and would make it easier for new users to become acquainted with the system. The available prototype implementation of a GUI (cf. Section 3.5.5) offers a limited scope of operation only. Most notably, there is no visual output of the evaluation. Integrating the visual representation of the evaluation results into the main application would allow for a more convenient mode of operation. Examples of extended functionality include to provide the ability to freely select metrics of one's choice, to combine any selection of metrics in one diagram or to select a time interval via the GUI and calculate statistical parameters for this interval on the fly.

**Neighborhood vs. cluster vs. network** For large WSNs, it is advisable to divide the evaluation into different segments, according to types of relationships between the nodes, namely the neighbor, cluster and network level. If a cluster displays strange behavior that differs from that of the other clusters, looking at the evaluation of the individual nodes and their neighborhoods that are part of this cluster allows for a more fine grained analysis. Means for this kind of analysis have yet to be completed.

**Scoring approach** Another interesting idea is to implement a scoring approach as described in [ISO10a]. The evaluator assigns weights to the individual metrics and the associated evaluation process yields a single value for each evaluated EDA. This single value aggregates all considered metrics, thereby allowing further automation of the decision process.

Apart from improving and enhancing the existing implementation there are also several interesting aspects concerning the further development of the general domain. With the advent of ever more sophisticated techniques for event detection, the way of describing events may also shift from being as exact as possible towards a more abstract representation. Furnishing the EDA with example events, formed by a combination of several features, and letting the EDA discover subsequent events on its own, would foster autonomous learning as well as autonomous decision making.

In detection schemes that are based on cooperation between the nodes, the nodes acquire a certain knowledge, or at least a vague idea about what is going on in their vicinity. As envisioned in [HW09], a self-diagnosis mechanism can make use of this information, deciding whether the individual node is malfunctioning or not. Alternatively, a higher instance, like a clusterhead or even the base station, can

conclude from the nodes' messages if any nodes are likely to be faulty. In order to attain results of better quality, the higher instance instructs the potentially faulty nodes to switch to another EDA that is known to perform better in the presence of faults.

A new and, when viewed from a dependability related perspective, rather intriguing concept is *inexact design*. This principle, described as being *beyond fault tolerance* [Ant13], aims to design electronic circuits in a way so that they deliver inaccurate results, with the amount of inaccuracy being a design parameter. The benefit of this accuracy is that these circuits consume considerably less energy than a circuit yielding an accurate result would do. The potential deviations from the correct result are taken into account consciously and no attempt for correction is made. Inexact design is especially interesting when tradeoffs between properties like processing speed, power consumption, memory usage or accuracy, i.e. the quality of detection are concerned. Applying inexact design to EDAs would make it possible to have algorithms implemented that, compared to the current range of algorithms, are much more complex and intensive in terms of resource demands.

# Appendices

# Appendix A

# Log data

## A.1 Console output for case study 1

The corresponding experiment was conducted on February, 24th 2014.

```
11:20:47.879 statusExercise started.
11:20:47.883 laboratoryNode 1 enabled.
11:20:47.887 laboratoryNode 2 enabled.
11:20:47.891 laboratoryNode 3 enabled.
11:20:47.895 laboratoryNode 4 enabled.
11:20:48.555 statusNode 1: value=0
11:20:53.560 statusNode 1: value=0
11:20:56.564 statusNode 2: value=0
11:20:57.886 laboratoryNode 2 disabled.
11:20:58.459 laboratoryNode 3 disabled.
11:20:58.565 statusNode 1: value=0
11:20:59.147 laboratoryNode 4 disabled.
11:21:02.405 laboratoryAmbient light set to 24.
11:21:03.571 statusNode 1: value=2
11:21:08.756 statusNode 1: value=73
11:21:10.472 laboratoryAmbient light set to 21.
11:21:13.799 statusNode 1: value=69
11:21:16.414 laboratoryAmbient light set to 27.
11:21:18.843 statusNode 1: value=69
11:21:22.908 laboratoryAmbient light set to 23.
11:21:23.892 statusNode 1: value=94
11:21:27.594 statusNode 1: value=73
11:21:32.600 statusNode 1: value=70
11:21:36.604 laboratoryAmbient light set to 99.
11:21:37.605 statusNode 1: value=135
11:21:39.125 laboratoryAmbient light set to 0.
11:21:42.610 statusNode 1: value=149
11:21:43.912 laboratoryAmbient light set to 21.
```

```
11:21:47.115 statusNode 1: value=37
11:21:51.123 statusExercise stopped.
11:21:52.119 statusNode 1: value=63
11:21:57.124 statusNode 1: value=63
11:22:02.129 statusNode 1: value=63
11:22:06.633 statusNode 1: value=63
11:22:11.639 statusNode 1: value=63
11:22:16.643 statusNode 1: value=63
11:22:21.650 statusNode 1: value=63
11:22:25.731 laboratoryAmbient light set to 0.
11:22:26.155 statusNode 1: value=50
11:22:31.159 statusNode 1: value=0
11:22:36.164 statusNode 1: value=0
11:22:37.107 statusExercise started.
11:22:37.111 laboratoryNode 1 enabled.
11:22:37.115 laboratoryNode 2 enabled.
11:22:37.119 laboratoryNode 3 enabled.
11:22:37.122 laboratoryNode 4 enabled.
11:22:40.989 laboratoryNode 2 disabled.
11:22:41.168 statusNode 1: value=0
11:22:41.479 laboratoryNode 4 disabled.
11:22:42.881 laboratoryNode 3 disabled.
11:22:45.672 statusNode 1: value=0
11:22:48.946 laboratoryAmbient light set to 22.
11:22:50.678 statusNode 1: value=13
11:22:55.682 statusNode 1: value=68
11:22:59.567 laboratoryAmbient light set to 20.
11:23:00.688 statusNode 1: value=67
11:23:05.192 statusNode 1: value=58
11:23:05.986 laboratoryAmbient light set to 23.
11:23:10.197 statusNode 1: value=68
11:23:15.201 statusNode 1: value=70
11:23:15.544 laboratoryAmbient light set to 99.
11:23:20.206 statusNode 1: value=314
11:23:22.965 laboratoryAmbient light set to 21.
11:23:24.711 statusNode 1: value=287
11:23:29.716 statusNode 1: value=63
11:23:29.965 laboratoryAmbient light set to 23.
11:23:34.721 statusNode 1: value=69
11:23:37.963 laboratoryAmbient light set to 22.
11:23:39.726 statusNode 1: value=69
11:23:44.431 statusNode 1: value=68
11:23:47.357 laboratoryAmbient light set to 99.
11:23:49.435 statusNode 1: value=158
```

```
11:23:54.440 statusNode 1: value=373
11:23:55.635 laboratoryAmbient light set to 23.
11:23:58.944 statusNode 1: value=193
11:24:03.948 statusNode 1: value=70
11:24:08.954 statusNode 1: value=70
11:24:09.276 laboratoryAmbient light set to 25.
11:24:13.959 statusNode 1: value=76
11:24:15.356 laboratoryAmbient light set to 22.
11:24:18.464 statusNode 1: value=72
11:24:23.468 statusNode 1: value=68
11:24:24.605 laboratoryAmbient light set to 99.
11:24:28.473 statusNode 1: value=257
11:24:33.478 statusNode 1: value=373
11:24:37.981 statusNode 1: value=373
11:24:39.491 laboratoryAmbient light set to 21.
11:24:42.986 statusNode 1: value=191
11:24:47.992 statusNode 1: value=63
11:24:49.413 laboratoryAmbient light set to 25.
11:24:52.996 statusNode 1: value=72
11:24:57.501 statusNode 1: value=79
11:25:00.353 laboratoryAmbient light set to 23.
11:25:02.506 statusNode 1: value=74
11:25:07.511 statusNode 1: value=70
11:25:12.515 statusNode 1: value=70
11:25:15.181 laboratoryEvent light set to 99.
11:25:17.199 statusNode 1: value=263
11:25:22.254 eventNode 1 triggered the event with value 553.
11:25:27.299 statusNode 1: value=553
11:25:28.198 laboratoryAmbient light set to 21.
11:25:32.350 statusNode 1: value=538
11:25:36.539 statusNode 1: value=528
11:25:37.203 laboratoryAmbient light set to 22.
11:25:41.545 statusNode 1: value=528
11:25:45.347 laboratoryAmbient light set to 20.
11:25:46.550 statusNode 1: value=528
11:25:51.555 statusNode 1: value=528
11:25:52.248 laboratoryAmbient light set to 26.
11:25:56.561 statusNode 1: value=548
11:26:01.653 statusNode 1: value=553
11:26:02.800 laboratoryAmbient light set to 23.
11:26:06.705 statusNode 1: value=553
11:26:07.594 laboratoryEvent light set to 0.
11:26:11.757 statusNode 1: value=263
11:26:15.705 laboratoryEvent light set to 100.
```

```
11:26:16.810 statusNode 1: value=70
11:26:20.585 eventNode 1 triggered the event with value 523.
11:26:22.949 laboratoryEvent light set to 0.
11:26:25.591 statusNode 1: value=267
11:26:28.569 laboratoryAmbient light set to 21.
11:26:30.596 statusNode 1: value=68
11:26:35.601 statusNode 1: value=63
11:26:36.769 laboratoryAmbient light set to 22.
11:26:40.106 statusNode 1: value=67
11:26:45.111 statusNode 1: value=65
11:26:48.573 laboratoryEvent light set to 85.
11:26:50.116 statusNode 1: value=147
11:26:55.121 eventNode 1 triggered the event with value 467.
11:26:56.654 laboratoryAmbient light set to 24.
11:26:59.626 statusNode 1: value=477
11:27:04.631 statusNode 1: value=492
11:27:05.235 laboratoryAmbient light set to 22.
11:27:09.635 statusNode 1: value=472
11:27:14.640 statusNode 1: value=467
11:27:15.603 laboratoryEvent light set to 0.
11:27:19.145 statusNode 1: value=232
11:27:21.724 laboratoryEvent light set to 84.
11:27:24.150 statusNode 1: value=224
11:27:27.302 laboratoryEvent light set to 0.
11:27:29.155 statusNode 1: value=392
11:27:33.665 laboratoryAmbient light set to 23.
11:27:34.160 statusNode 1: value=65
11:27:38.678 statusNode 1: value=70
11:27:43.683 statusNode 1: value=70
11:27:44.241 laboratoryAmbient light set to 20.
11:27:48.689 statusNode 1: value=60
11:27:53.694 statusNode 1: value=58
11:27:53.887 laboratoryEvent light set to 70.
11:27:58.198 eventNode 1 triggered the event with value 375.
11:28:03.203 statusNode 1: value=375
11:28:08.209 statusNode 1: value=375
11:28:13.213 statusNode 1: value=375
11:28:14.996 laboratoryEvent light set to 0.
11:28:17.718 statusNode 1: value=177
11:28:21.501 laboratoryEvent light set to 66.
11:28:22.725 statusNode 1: value=116
11:28:23.200 laboratoryEvent light set to 68.
11:28:24.107 laboratoryEvent light set to 0.
11:28:27.729 statusNode 1: value=183
```

```
11:28:32.735 statusNode 1: value=58
11:28:34.244 laboratoryAmbient light set to 22.
11:28:37.240 statusNode 1: value=62
11:28:42.245 statusNode 1: value=65
11:28:46.215 laboratoryAmbient light set to 26.
11:28:47.250 statusNode 1: value=66
11:28:50.115 laboratoryAmbient light set to 20.
11:28:52.254 statusNode 1: value=79
11:28:56.758 statusNode 1: value=58
11:28:57.499 laboratoryEvent light set to 100.
11:29:01.764 eventNode 1 triggered the event with value 433.
11:29:03.186 laboratoryEvent light set to 0.
11:29:06.770 statusNode 1: value=251
11:29:07.643 laboratoryAmbient light set to 23.
11:29:11.775 statusNode 1: value=67
11:29:12.736 laboratoryAmbient light set to 21.
11:29:16.280 statusNode 1: value=66
11:29:20.302 laboratoryAmbient light set to 24.
11:29:21.285 statusNode 1: value=64
11:29:26.290 statusNode 1: value=76
11:29:27.144 laboratoryEvent light set to 100.
11:29:31.296 eventNode 1 triggered the event with value 423.
11:29:36.301 statusNode 1: value=553
11:29:37.613 laboratoryEvent light set to 0.
11:29:40.806 statusNode 1: value=266
11:29:45.154 laboratoryAmbient light set to 23.
11:29:45.812 statusNode 1: value=76
11:29:50.817 statusNode 1: value=71
11:29:55.821 statusNode 1: value=70
11:30:00.326 statusNode 1: value=71
11:30:03.596 laboratoryEvent light set to 100.
11:30:05.331 statusNode 1: value=157
11:30:10.337 eventNode 1 triggered the event with value 553.
11:30:12.281 laboratoryEvent light set to 0.
11:30:15.342 statusNode 1: value=353
11:30:17.393 laboratoryAmbient light set to 21.
11:30:19.845 statusNode 1: value=67
11:30:24.851 statusNode 1: value=63
11:30:29.856 statusNode 1: value=63
11:30:32.794 laboratoryEvent light set to 100.
11:30:34.860 statusNode 1: value=164
11:30:37.725 laboratoryAmbient light set to 23.
11:30:39.365 eventNode 1 triggered the event with value 538.
11:30:44.370 statusNode 1: value=553
```

```
11:30:47.203 laboratoryEvent light set to 0.
11:30:49.375 statusNode 1: value=389
11:30:54.380 statusNode 1: value=70
11:30:55.245 laboratoryAmbient light set to 24.
11:30:58.885 statusNode 1: value=73
11:31:01.196 laboratoryAmbient light set to 21.
11:31:03.890 statusNode 1: value=70
11:31:06.722 laboratoryAmbient light set to 20.
11:31:08.896 statusNode 1: value=61
11:31:13.901 statusNode 1: value=58
11:31:17.804 laboratoryEvent light set to 70.
11:31:18.406 statusNode 1: value=53
11:31:23.411 eventNode 1 triggered the event with value 375.
11:31:23.804 laboratoryEvent light set to 0.
11:31:28.415 statusNode 1: value=125
11:31:33.221 laboratoryEvent light set to 70.
11:31:33.419 statusNode 1: value=58
11:31:37.924 eventNode 1 triggered the event with value 336.
11:31:39.646 laboratoryEvent light set to 0.
11:31:42.929 statusNode 1: value=185
11:31:47.934 statusNode 1: value=58
11:31:52.938 statusNode 1: value=58
11:31:54.232 laboratoryEvent light set to 69.
11:31:56.722 laboratoryEvent light set to 69.
11:31:57.443 statusNode 1: value=248
11:32:00.375 laboratoryEvent light set to 69.
11:32:02.447 statusNode 1: value=376
11:32:05.237 laboratoryEvent light set to 70.
11:32:07.453 statusNode 1: value=375
11:32:09.393 laboratoryEvent light set to 0.
11:32:12.458 statusNode 1: value=231
11:32:16.962 statusNode 1: value=58
11:32:21.967 statusNode 1: value=58
11:32:24.586 laboratoryAmbient light set to 23.
11:32:26.972 statusNode 1: value=63
11:32:31.977 statusNode 1: value=70
11:32:32.262 laboratoryAmbient light set to 22.
11:32:36.482 statusNode 1: value=66
11:32:37.271 statusExercise stopped.
11:32:46.492 statusNode 1: value=65
```

## A.2 Console output for case study 2

The corresponding experiment was conducted on March, 19th 2014.

```
09:56:51.413 statusExercise started.
09:56:51.417 laboratoryNode 1 enabled.
09:56:51.421 laboratoryNode 2 enabled.
09:56:51.425 laboratoryNode 3 enabled.
09:56:51.428 laboratoryNode 4 enabled.
09:56:53.156 statusNode 1: value=0
09:56:56.803 laboratoryNode 2 disabled.
09:56:57.417 laboratoryNode 3 disabled.
09:56:58.162 statusNode 1: value=0
09:56:58.447 laboratoryNode 4 disabled.
09:57:02.666 statusNode 1: value=0
09:57:02.910 laboratoryAmbient light set to 22.
09:57:07.672 statusNode 1: value=52
09:57:12.676 statusNode 1: value=65
09:57:17.681 statusNode 1: value=65
09:57:19.234 laboratoryAmbient light set to 20.
09:57:22.186 statusNode 1: value=61
09:57:27.190 statusNode 1: value=58
09:57:29.323 laboratoryAmbient light set to 26.
09:57:32.195 statusNode 1: value=70
09:57:37.201 statusNode 1: value=89
09:57:41.706 statusNode 1: value=89
09:57:43.116 laboratoryAmbient light set to 25.
09:57:46.711 statusNode 1: value=83
09:57:51.715 statusNode 1: value=79
09:57:53.813 laboratoryAmbient light set to 28.
09:57:56.720 statusNode 1: value=84
09:58:01.224 statusNode 1: value=94
09:58:03.808 laboratoryAmbient light set to 25.
09:58:06.229 statusNode 1: value=88
09:58:11.233 statusNode 1: value=79
09:58:13.341 laboratoryAmbient light set to 26.
09:58:16.238 statusNode 1: value=85
09:58:19.810 laboratoryEvent light set to 70.
09:58:20.742 statusNode 1: value=138
09:58:25.748 eventNode 1 triggered the event with value 398.
09:58:30.753 statusNode 1: value=398
09:58:34.811 laboratoryAmbient light set to 23.
09:58:35.757 statusNode 1: value=398
09:58:40.262 statusNode 1: value=399
09:58:41.459 laboratoryAmbient light set to 25.
```

```
09:58:45.267 statusNode 1: value=398
09:58:47.421 laboratoryAmbient light set to 19.
09:58:48.819 laboratoryAmbient light set to 26.
09:58:50.272 statusNode 1: value=393
09:58:54.847 laboratoryAmbient light set to 20.
09:58:55.276 statusNode 1: value=398
09:59:00.281 statusNode 1: value=375
09:59:04.786 statusNode 1: value=375
09:59:06.855 laboratoryAmbient light set to 23.
09:59:09.790 statusNode 1: value=384
09:59:14.795 statusNode 1: value=399
09:59:19.800 statusNode 1: value=399
09:59:19.824 laboratoryAmbient light set to 26.
09:59:24.305 statusNode 1: value=398
09:59:29.310 statusNode 1: value=398
09:59:32.843 laboratoryAmbient light set to 21.
09:59:34.315 statusNode 1: value=393
09:59:39.319 statusNode 1: value=375
09:59:43.824 statusNode 1: value=375
09:59:46.852 laboratoryAmbient light set to 24.
09:59:48.829 statusNode 1: value=379
09:59:53.834 statusNode 1: value=398
09:59:57.917 laboratoryAmbient light set to 20.
09:59:58.840 statusNode 1: value=398
10:00:03.344 statusNode 1: value=375
10:00:07.525 laboratoryAmbient light set to 22.
10:00:08.347 statusNode 1: value=375
10:00:13.352 statusNode 1: value=375
10:00:16.416 laboratoryAmbient light set to 26.
10:00:18.357 statusNode 1: value=379
10:00:22.862 statusNode 1: value=398
10:00:26.874 laboratoryAmbient light set to 28.
10:00:27.867 statusNode 1: value=397
10:00:32.872 statusNode 1: value=397
10:00:35.883 laboratoryAmbient light set to 21.
10:00:37.877 statusNode 1: value=393
10:00:42.381 statusNode 1: value=375
10:00:46.245 laboratoryAmbient light set to 24.
10:00:47.386 statusNode 1: value=379
10:00:52.391 statusNode 1: value=398
10:00:57.397 statusNode 1: value=398
10:00:58.926 laboratoryEvent light set to 0.
10:01:01.901 statusNode 1: value=203
10:01:02.897 laboratoryAmbient light set to 22.
```

```
10:01:05.348 laboratoryAmbient light set to 25.
10:01:06.906 statusNode 1: value=70
10:01:11.911 statusNode 1: value=79
10:01:16.916 statusNode 1: value=79
10:01:20.134 laboratoryAmbient light set to 22.
10:01:21.420 statusNode 1: value=76
10:01:26.426 statusNode 1: value=65
10:01:28.985 laboratoryAmbient light set to 24.
10:01:31.432 statusNode 1: value=68
10:01:36.437 statusNode 1: value=73
10:01:38.457 laboratoryAmbient light set to 27.
10:01:40.941 statusNode 1: value=81
10:01:45.946 statusNode 1: value=95
10:01:50.951 statusNode 1: value=95
10:01:51.931 laboratoryAmbient light set to 24.
10:01:55.956 statusNode 1: value=80
10:01:59.283 laboratoryAmbient light set to 25.
10:02:00.460 statusNode 1: value=74
10:02:05.466 statusNode 1: value=79
10:02:09.558 laboratoryAmbient light set to 20.
10:02:10.471 statusNode 1: value=78
10:02:15.476 statusNode 1: value=58
10:02:20.483 statusNode 1: value=58
10:02:21.492 statusExercise stopped.
```

# Appendix B

# Publications

**Publication 1:** Karima B. Hein, Leander B. Hörmann, Reinhold Weiss, *Using a Leaky Bucket Counter as an Advanced Threshold Mechanism for Event Detection in Wireless Sensor Networks*, 10th International Workshop on Intelligent Solutions in Embedded Systems (WISES), Klagenfurt, Austria, July, 5th – 6th 2012.

**Publication 2:** Karima B. Hein, Leander B. Hörmann, Reinhold Weiss, *Analysis of Threshold-Based Event Detection Algorithms for Wireless Sensor Networks by Fault Injection*, 10th International Conference on Ubiquitous Intelligence and Computing and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC), Vietri sul Mare, Italy, December, 18th – 21st 2013.

**Publication 3:** Karima B. Hein, Reinhold Weiss, *Minesweeper for Sensor Networks – Making Event Detection in Sensor Networks Dependable*, International Conference on Computational Science and Engineering (CSE), Vancouver, Canada, August 29th – 31st 2009.

10th International Workshop on Intelligent Solutions in Embedded Systems, 2012

# Using a Leaky Bucket Counter as an Advanced Threshold Mechanism for Event Detection in Wireless Sensor Networks

Karima B. Hein, Leander B. Hörmann, Reinhold Weiss
*Institute for Technical Informatics*
*Graz University of Technology*
*Graz, Austria*
*Email: {Hein,Leander.Hoermann,RWeiss}@TUGraz.at*

***Abstract*—In this work, we show how to use a leaky bucket counter (LBC) as a sophisticated threshold mechanism for detecting events in wireless sensor networks. After introducing the LBC and elaborating on various special cases for different possibilities of event detection, we present a case study. Using varying parameters, we compare the performance of the LBC approach to that of a moving average approach and a simple threshold-only mechanism. These mechanisms are of comparable computational complexity and have similar resource demands. The comparison underlines the differences in how old measurements influence the actual detection outcome in different ways. We also explain under which conditions an LBC is suited for event detection and when it is not.**

***Keywords*-Sensor networks; event detection; leaky bucket counter; threshold modeling**

## I. Introduction

Event detection in wireless sensor networks (WSNs) is a widely used type of application. In the process of event detection, the system (the single node or a base station or cluster head) has to have some knowledge about what is normal and what is not. Building upon this knowledge the system can detect anomalies in its surroundings. Popular examples for event detection in WSNs include emergency warning systems for wild fires [1] and volcano monitoring [7] as outdoor applications and office surveillance [6] as a representative for indoor surveillance.

Typically, a large number of nodes form a WSN. Each node is equipped with one or more sensors and periodically measures the quantity of one or more attributes. The (optionally preprocessed) measurements are forwarded to a base station for further processing or, alternatively, the node autonomously performs appropriate actions. No matter where the nodes' data are processed, some instance has to decide whether an event has taken place or not. This is done by some kind of threshold mechanism. It is crucial that this mechanism not only correctly identifies events, but also properly recognizes faults and noise as not being events. In this work, we will demonstrate how a leaky bucket counter (LBC) can be employed as a sophisticated threshold mechanism with two parameters.

In order to correctly detect an event and also the absence of an event, it is important to clearly define what an event is. The LBC is a proper choice if the events sought after are defined not only by a momentarily peak but also by the recent behavior of input samples. Consequently, the LBC is not adequate for all kinds of events. Simple events, where checking if the measured or calculated value is larger than a given threshold, can surely be detected by the LBC approach. But such simple events can also easily be detected with a standard threshold mechanism that entails less computational overhead than the LBC approach.

The demands on event detection have evolved in recent years: not every feature that is measurable may be of interest and not every change in a feature's progression may be an event. As different kinds of events have different characteristics, not every method is suited for every event. Having available a priori knowledge about the event can simplify the process: if it is known, that an event has a minimal duration of five samples, it is easy to filter out peaks that manifest themselves in only one sample. If there is no a priori knowledge about the shape (in the sense of peaks) available, it is mostly not possible to distinguish between a short event and a faulty sensor reading.

## II. Related Work

The intuitive solution for detecting an event is to compare a measured value to a threshold. Much work deals with improving this simple approach. In [5] the authors present an event detection algorithm, where a second, lower threshold is introduced. A node that passes only the lower threshold is checked once if it has a neighbor which passes the high threshold. If so, the node becomes a member of the number of nodes that have detected the event.

The authors of [8] present an event-driven and decentralized approach with two thresholds. The first threshold is used to decide whether the event detecting node should interrogate its neighbors. A low threshold guarantees that the system detects weak signals, but the probability of a false alarm also rises. The second threshold is used to decide about the neighbors' observations. Two different strategies are reviewed for the second mechanism.

Contrary to our approach, both of the two approaches above use communication with the node's neighbors. Com-

munication in WSNs is a costly feature in terms of energy. Additionally, message exchange opens the door to transmission errors and the need for retransmissions.

Escalation [9] is an early data mining approach. Complex events are defined as "sets of data points that correspond to interesting or unusual patterns in the underlying phenomenon that the network monitors". The authors' work is inspired by time-series data mining techniques. Data mining needs a lot of resources and therefore, cannot be performed on a node. Therefore, all the data has to be forwarded to a base station, which also entails high overhead costs.

Liang and Wang discuss in [4] how to use fuzzy logic for acoustic event detection in WSNs. They also introduce a double sliding window, so that the detection does not depend on the sensed signal energy (the level of noise power is the problem), but on the ratio of two consecutive windows.

The authors of [1] present a machine learning approach using decision trees for distributed event detection in WSNs for disaster management. The proposed techniques are of low complexity and considerable performance, but due to the employed voting, communication with neighbors again is necessary.

## III. A Leaky Bucket Counter as an Advanced Threshold Mechanism for Event Detection

An LBC traditionally is used in network applications to ensure that bandwidth limits are not surmounted or for the detection of anomalous behavior [2], i.e. if bursts in the traffic flow occur with a frequency that is too high.

Linking this concept to event detection, the LBC is an instrument to distinguish between real events and noise. A so called real event refers to an event of sufficient significance in the context of previous samples, i.e. a small peak may be an event if more activity has been reported recently. But this small peak may be treated like noise, if no adequate recent activity has been perceived.

### A. Events and event characteristics

Roughly speaking, an event in a WSN takes place if the trend of the magnitude of some measured feature(s) trigger(s) a threshold mechanism often enough. The decision process of determining if a threshold has been exceeded can range from a very simple comparison (is $x > 5$?) to a complex procedure that entails calculating multiple intermediate results and even the consultation of other nodes.

The features are measured with the nodes' sensors. The thresholds are either predetermined by an application engineer or acquired during a learning phase. Additionally, there can be a post processing phase, where the results are checked for feasibility (through experience or by comparison with neighboring nodes' results [3]).

*Multi feature events:* If an event is constituted by the combination of multiple features we call it a multi feature event (MFE). Different features may be of different severity. We consider events as MFEs if a strong overall magnitude of different features suffices, i.e. it is not necessary for all features to be present if the subset of features that are present show strong magnitudes. The detection of MFEs is established via multi modal sensor fusion. The different sensors usually are located on the same node, but could be distributed on several other nodes, although the latter case incurs some overhead due to the additional communication for retrieving the other nodes' measurements.

The next section will explain how an LBC works and show how events that are defined by only a single feature and MFEs can efficiently be dealt with by an LBC.

### B. Basic mode of operation of a leaky bucket counter

The basic idea behind the LBC is a bucket where drops (also referred to as $tokens$) aperiodically fall into and water leaks out constantly through a hole. Only the size of the hole determines the rate of emptying as the pressure of the present filling is neglected. The contents of the bucket leaks out with a constant rate until it is empty but the inflow is variable. If the bucket is empty, the outflow is zero. If the outflow cannot compensate a strong inflow, the bucket eventually overflows. This occurs if the inflow is a little more than the outflow during a large time window or if a burst fills the bucket during a short period of time.

This mechanism is suited for distinguishing between permanent and transient events. A transient event is an event with a relative short duration and a magnitude that is not too high. If the duration of the event is not long enough to bring the bucket to overflow, than the event is apparently transient and, consequently, not reported. An example would be a network, where the nodes measure the indoor room temperature. Opening a door lets in cool air from outside and the temperature at a node located closely to the door drops for a small number of samples. After closing the door, the room temperature soon stabilizes again.

In case of a permanent event, the threshold is long enough exceeded or the feature(s) are so intense that this behavior is categorized as an event. In the door example, this corresponds to the door being open long enough, so that the temperature at the node in question stays low for a sufficient long period of time.

### C. Handling multi feature events

In case of an MFE, the contribution of different features may be of different severity. A basic feature with a basic severity is defined and for each feature, the severity of the feature's contribution is defined as a factor *Severity*. If the *Severity* of a feature is double the *Severity* of the basic feature, than the number of drops that fall into the bucket

upon measuring the feature, is double the number of a basic feature.

Additionally, different magnitudes of one feature can also be described by different severities. Four example, the measured temperature can be *cold* (*Severity*=1), *adequate* (*Severity*=2) or *hot* (*Severity*=3).

### *D. Detecting moderate peak events with a multistage LBC*

With a modification, a one-time inflow of high amount (but less than the bucket's capacity), a so called moderate peak event, can be recognized as an event. This can be detected by a multistage LBC, where a small bucket that floats inside the large bucket is added. The small bucket has to float, because otherwise, the larger bucket's drain would be clogged when the small bucket overflows.

The inflow goes directly into the small bucket. Accordingly, if an amount more than the small bucket's capacity goes into the small bucket, it overflows and, consequently, a peak event is detected. The overflown water, however, is inflow for the large bucket. The small bucket's entire contents has to be drained into the large bucket after a peak event has been detected. Otherwise, a single drop arriving directly after the peak event would be recognized as another peak event.

## IV. Implementation

### *A. The LBC's functionality*

The functionality of the LBC is composed of six steps:

1) Input goes in
2) Filling level update
3) Optional: overflow occurs $\rightarrow$ an event is detected
4) Outflow occurs
5) Filling level update
6) Ready for next input

There are two special cases to mention. In the first case, the filling level just before the subtraction of the outflow is smaller than the outflow: the new filling level is zero, as the bucket cannot have a negative filling level. The second special case is if an overflow occurs: the overflown tokens disappear and will not be included in the bucket, i.e. the resulting filling level after step five is the bucket's capacity minus the flow rate.

The filling level of the bucket is a counter that is increased for every drop that falls into the bucket. A threshold indicates when the bucket is filled. The bucket's leaking is modeled as a periodic decrease of the counter, but the counter is not decreased beyond zero as the bucket cannot contain less than zero drops. Consequently, the parameters that have to be specified are the bucket's capacity (number of drops that fit into the bucket), the *Severity* of each feature (different drop sizes, i.e. the number of tokens), and the size of the bucket's hole (frequency and amount of counter decrease). The overflow check has of course to occur before the counter decrease.

Finding the ideal parameters and, consequently, dynamics is up to the application engineer as this person defines the characteristics of the events of interest.

### *B. Modifying the leaky bucker counter for 'regular' event detection*

An LBC can also be used in a dual mode, where it is possible to switch between two modes of operation: standard mode (as described before) and basic thresholding (see also section V-C). In the second mode, only one sample is considered at a time and the bucket capacity and leakage rate is set to the threshold $\Theta$. That way, after each input the bucket is emptied and only an input that has a magnitude greater than $\Theta$ results in the detection of an event.

## V. Case Study

In this section, we will demonstrate how an LBC can be employed for event detection. We compare the performance of detection by using an LBC to that of employing a moving average window and to that of a simple threshold mechanism. We chose to compare these approaches, as all three of them are computationally simple and have comparable resource requirements.

The considered scenario is a series of temperature measurements close to a warmth emitting component. The same 51 samples were used for the analysis of every mechanism. We refrain from comparing the total number of detected events for each approach, as with a small offset to the thresholds, this number can be changed. Instead we graphically present the characteristics of the different mechanisms. In the corresponding graphs, it is easy to visualize how the actual number of detected events would change with a modification of the threshold.

### *A. Detection using an LBC*

The capacity of the bucket is eight tokens and the outflow is two tokens per time unit. Figure 1 shows the filling level before and after the outflow. As the outflow per definition takes place after a potential overflow, the filling level after the outflow is never more than six (cf. the violet line with the crosses). An event is detected every time the green line (triangles) crosses the horizontal red line representing the bucket capacity.

The shape of the filling level before the outflow (FLBO) roughly resembles the shape of the input. If a few consecutive input samples have a low magnitude, the FLBO also has a comparable magnitude. But for samples of higher magnitude, the peaks of the FLBO are of even higher magnitude. These peaks are compensated by the overflow automatic, i.e. the peak is clipped.

In spite of the lost tokens at each overflow, the bucket has some kind of memory. Consider the peak at time t=20, the magnitude of the following three samples is only in the order of the leakage rate, which causes the bucket to stay

nearly filled. The small peak with a magnitude of five at t=24 therefore causes another overflow and an event is detected. This example beautifully underlines how an event, i.e. a high peak, reverberates and influences future decisions.

The contrary case can be seen at time t=48, where another input of magnitude five is sampled. Although the last detected event only was at t=43, the ensuing inputs did not keep the bucket's filling level high enough to detect another event. To that effect, his peak is considered as a transient event or a fault.

### *B. Detection using a moving average window*

Using a moving average window also needs two parameters for defining event detection, the threshold $\Theta$ and the window size $k$. This window size $k$ defines the maximum number and, consequently, the maximum age of values used for calculating the mean. For window size $k = 2$ that means, only the current value and the one measured before that are used for determining the mean.

For a large $k$, many values are used for calculating the mean and the graph's shape will become flatter. For a small $k$, only few values are employed and the general shape will approach a slightly dampened and offset version of the input values' trend. Due to this dampening, short peaks are filtered out. To underline this behavior and in order to compare the moving average and the LBC, we have chosen two different parameter sets.

*1)* ***Window size*** $k = 5$*:* Figure 2 shows the input samples (the blue line with diamonds), the moving average for window size $k = 5$ and the threshold $\Theta = 5$. Every time the threshold $\Theta$ is crossed, an event is detected. The graph of the mean has a different course compared to the graph of the input samples. The reduction and broadening of the peaks stems from the fact that the peaks of the inputs are not as broad as the window size. That way, an event is only recognized if a substantial high input has occurred. Due to the slow reaction of the moving average, three single peaks (cf. t=14, t=19 and t=24) have been fused together.

*2)* ***Window size*** $k = 3$*:* Refining the behavior of the moving average in the aforementioned example, we will now take a look at the behavior for window size $k = 3$ depicted in Figure 3. As the considered window size is rather small, the mean follows the trend of the input values closer than for $k = 5$. Peaks in the input are easier distinguishable in the resulting mean. Short peaks surrounded by low values (cf. t=24 and t=42) are still dampened. This behavior is desirable if a very short peak (here only one sample) is in principle not to be considered as an event.

### *C. Detection using a basic threshold mechanism*

The term basic threshold mechanism refers to comparing the current input value to a given threshold. As only one of the samples is used for comparison, this method does not take into account any trend in the preceding samples and is memoryless. This approach can be seen as a special case of the moving average with $k = 1$.

An event is always detected when the input $Severity$ (again the blue line with diamonds) is more than the threshold. By moving the horizontal line $Threshold$ or $Bucket\ Capacity$ in any of the figures 1,2, or 3 up or down, the mechanism can be made more or less sensitive and, consequently, detects more or less events.

### *D. Analysis and discussion*

Comparing the moving average approach to the LBC shows that the LBC allows perceiving the influence of a high peak immediately. This cannot simply be seen as an advantage or a disadvantage, but as a different characteristic. Depending on what the application engineer wants to capture, an appropriate detection mechanism has to be chosen.

As the presented example shows, the definition of an event is crucial, i.e. when an event is considered permanent and when it is considered transient. If the event of interest can simply be described by a statement of the kind *if temp* $> 10\,^{\circ}C$, employing an LBC would introduce pointless overhead, and making use of the simple threshold mechanism would suffice. Additionally, an event may be defined by its mean over the last $k$ samples and the height of occurring peaks may only be secondary. We will now compare the behavior of the different approaches in a number of essential aspects.

*1) Influence of single inputs on the detection outcome:* The behavior of the LBC is defined by its capacity and flow rate. The time until a specific input is gone (leaked out) depends on the flow rate and the magnitude of this very input. Consequently, we know beforehand only an upper limit of how long an input has influence on the detection process, that is, the bucket capacity divided by the leakage rate. If the input is larger than the bucket's capacity, the bucket overflows anyway and remains filled, so there is no violation of this upper limit.

Considering the moving average, reverberation of a peak depends only on the window size $k$. That means, after $k$ ticks of the clock, the influence of this input is gone for sure. We know beforehand exactly how long an input influences the mean and this period of time does not depend on the sample's magnitude. As with each time step, an old value is dropped, the magnitude of the decrease of the sum of old inputs (the leaking in LBC terms) is not constant like it is with the LBC.

The simple threshold approach only considers one current sample and any previously sampled values have no influence on the detection outcome.

*2) Sampling a magnitude of zero:* When employing an LBC, an input with magnitude zero does not increase the bucket's filling level. The filling level can only become less (due to outflow) or remain constant. Consequently, a

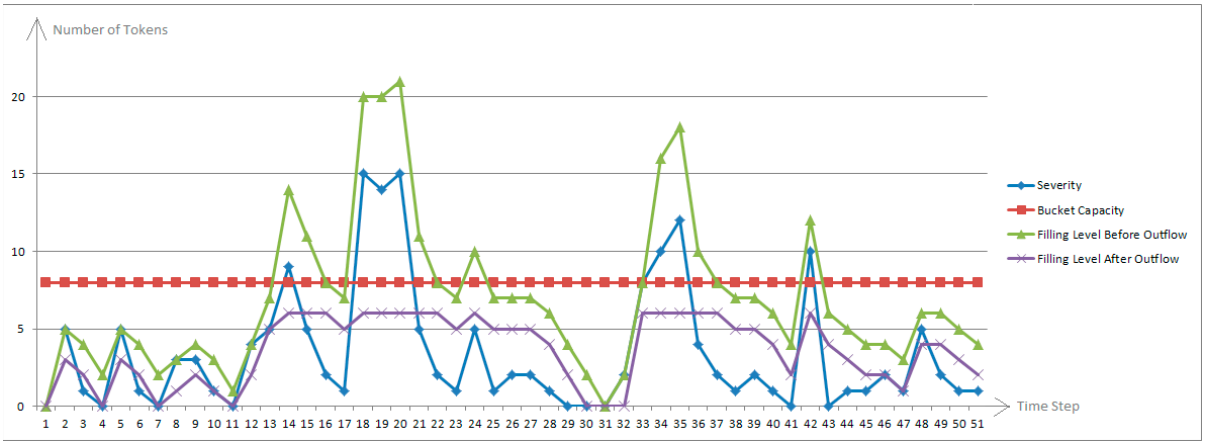



Figure 1. Filling level of the bucket before and after the outflow and overflow combined with the input samples and the bucket capacity. If the green line (dotted with triangles) crosses the horizontal 'capacity' line, the bucket overflows and an event is detected.

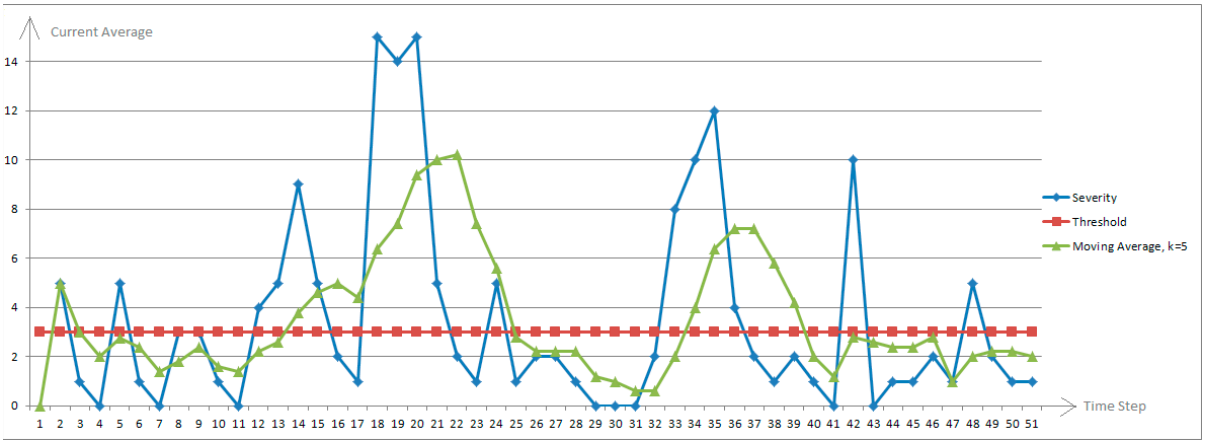



Figure 2. Moving average for window size $k = 5$ (green line with triangles), the horizontal line represents the threshold . If the green line crosses the threshold, an event is detected. The blue line (diamonds) represents the input samples.

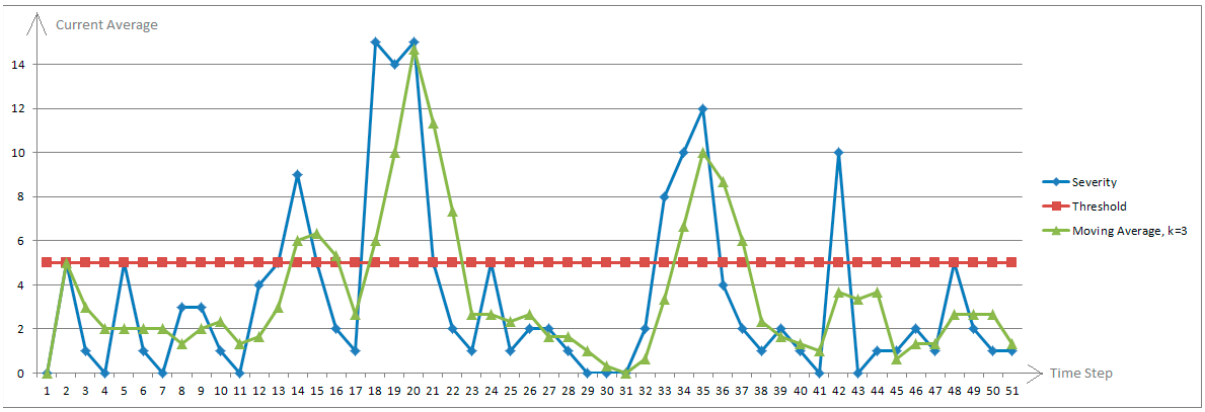



Figure 3. Moving average for window size $k = 3$ (green line with triangles), the horizontal line represents the threshold . If the green line crosses the threshold, an event is detected. The blue line (diamonds) represents the input samples.

magnitude of zero can never result in the detection of an event.

Things are different when employing a moving average: Newly arriving values that are smaller than the old values with index number greater than $k$ drag down the average, but it is possible that the average is sufficiently high, so that even a new value of zero would still result in an exceeding of the threshold. This behavior could be avoided by limiting the maximum input value by some kind of sanity check. However, it may be a defining characteristic of the considered environment that extremely high peaks take place from time to time and that a simple limiting of the maximum may contort the result.

As the simple threshold approach only takes into account this zero-sample, no event can be detected.

*3) Sampling a magnitude higher than the capacity or threshold:* For the LBC, this is always identified as an event because the bucket overflows no matter what the previous filling level was.

As the moving average calculates a mean over multiple samples, the necessary magnitude of a peak is considerably higher and depends on the magnitude of the previous $k-1$ samples. In the worst case, there were only samples of magnitude zero, than the peak would have to be of magnitude greater than $k \cdot \Theta$.

The simple threshold approach per definition reports an event.

*4) The minimal magnitude of an input in order to detect an event if an event just has been detected:* For an LBC the input has to exceed the leakage rate.

For the moving average, it has to be equal to the magnitude of value $k+1$. In this case, detection very much depends on the timely distribution of previous peaks, as the oldest is replaced with the newest.

The simple threshold approach needs an input with a magnitude higher than the threshold, no matter if an event has just been detected or not.

*5) Data management:* LBC: It is only necessary to store the current filling level (the flow rate and bucket size are parameters).

Moving average: It is necessary to store $k$ elements ($k$ and the threshold $\Theta$ are parameters, the resulting mean does not have to be stored).

For the simple threshold approach, no data has to be stored.

By shortening the sampling interval, it may be easier to detect events correctly. With a shorter delay between two consecutive measurements, the shape of the signal's peaks can be identified with more accuracy. Fine tuning the length of the sampling interval will lead to a balance between energy consumption and detection accuracy. It is notable that the emptying rate of the LBC has to be adjusted with respect to the sampling interval.

## VI. Conclusion and Future Work

This work showed how an LBC can be used for event detection in WSNs. We compared this approach to that of a moving average window and a basic threshold mechanism and highlighted how the mechanisms behave in different ways. We showed in particular, how past measurements can affect the detection outcome.

Our plans include implementing an application based upon event detection by LBC running on Mica2 nodes in the near future.

## References

[1] Majid Bahrepour, Nirvana Meratnia, Mannes Poel, Zahra Taghikhaki, and Paul J. M. Havinga. Distributed event detection in wireless sensor networks for disaster management. In *Intelligent Networking and Collaborative Systems (INCOS), 2010 2nd International Conference on*, pages 507–512, November 2010.

[2] Anup K. Ghosh, Aaron Schwartzbard, and Michael Schatz. Learning program behavior profiles for intrusion detection. In *Proceedings of 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, pages 51–62, 1999.

[3] Karima B. Hein and Reinhold Weiß. Minesweeper for sensor networks – making event detection in sensor networks dependable. In *International Conference on Computational Science and Engineering 2009*, volume 1, pages 388–393. IEEE Computer Society, August 2009.

[4] Q. Liang and L. Wang. Event detection in wireless sensor networks using fuzzy logic system. In *Computational Intelligence for Homeland Security and Personal Safety, 2005. CIHSPS 2005. Proceedings of the 2005 IEEE International Conference on*, pages 52–55, April 2005.

[5] Stefania Sardellitti, Sergio Barbarossa, and Luca Pezzolo. Distributed double threshold spatial detection algorithms in wireless sensor networks. In *Signal Processing Advances in Wireless Communications, 2009. SPAWC '09. IEEE 10th Workshop on*, pages 51–55, June 2009.

[6] Byunghun Song, Haksoo Choi, and Hyung Su Lee. Surveillance tracking system using passive infrared motion sensors in wireless sensor network. In *Information Networking, 2008. ICOIN 2008. International Conference on*, pages 1 –5, jan. 2008.

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

[8] Sung-Jib Yim and Yoon-Hwa Choi. Fault-tolerant event detection using two thresholds in wireless sensor networks. In *Dependable Computing, 2009. PRDC '09. 15th IEEE Pacific Rim International Symposium on*, pages 331 – 335, November 2009.

[9] Michael Zoumboulakis and George Roussos. Escalation: complex event detection in wireless sensor networks. In *Proceedings of the 2nd European conference on Smart sensing and context*, pages 270–285, 2007.

2013 IEEE 10th International Conference on Ubiquitous Intelligence & Computing and 2013 IEEE 10th International Conference on Autonomic & Trusted Computing

# Analysis of Threshold-Based Event Detection Algorithms for Wireless Sensor Networks by Fault Injection

Karima B. Hein, Leander B. Hörmann, Reinhold Weiss
*Institute for Technical Informatics*
*Graz University of Technology*
*Graz, Austria*
*Email: {Hein,Leander.Hoermann,RWeiss}@TUGraz.at*

***Abstract*—Wireless sensor networks are often deployed in harsh environments where they are exposed to extreme conditions. The influences and faults resulting from these conditions are often overlooked. As there are many possibilities for the occurrence of faults, dependability considerations are of high importance. In this work we present an approach for choosing the most adequate parameter set out of a number of qualified parameter sets for an event detection algorithm (EDA). Instead of only considering the performance of the EDA for data that does not contain errors, we analyze the EDA's reaction to erroneous data by injecting different kinds of faults. In a case study, we show how a seemingly optimal parameter set becomes apparent to be having a weak point. We also show how an application engineer can be provided with feedback about weaknesses of the employed EDA and its parameter set, respectively.**

## I. Introduction

In our modern world, wireless sensor networks (WSNs) are a well established technology representing a popular approach for distributed and cooperative problem solving. An essential branch of applications of WSNs is event detection. Event detection algorithms (EDAs) are crucial to many installments of WSNs, indoors as well as outdoors. Established applications range from glacier monitoring [2] over emergency warning systems for wild fires [1] to also include office surveillance [9].

This list of applications already indicates that WSNs are often deployed in harsh environments. The character of the WSN's environment can affect the condition of the WSN's hardware. The resulting effects can include cracks in the casing that entail shorts due to high humidity levels or accelerated component deterioration. Erratic behavior of a truly unexpected degree can be the consequence as was shown by the authors of [10], who led the famous WSN installation on Great Duck Island. They reported that due to humidity problems, failing temperature sensors not only reported persistent readings of 0 °C, but also readings out of the sensors' operating ranges.

Additionally, when installing a WSN in the field, nodes may become damaged without this being noticed or may be installed in a way that leads to the nodes behaving in a notion that was unanticipated by the application engineer. During their experiments in the field, the authors of [8] were faced with large plant leafs covering the antennas that dramatically reduced the radio range, fragile antennas that came loose easily and spontaneous resets of the nodes.

Finally, a WSN may become victim of an attacker. An attack can be carried out by a person that actually wants to destroy something, but can also be motivated by ignorant motives. An example for an ignorant motivation would be a member of the cleaning staff getting a ladder to climb up to ceiling mounted sensor nodes in order to clean them thoroughly.

Summing up, there is a wide variety of possibilities for the introduction of faults emphasizing that the occurrence of faults is simply natural. This fact often is overlooked and application designers often act on the assumption that the values that are delivered by the sensor or via the radio are correct numbers. Consequently, when looking for an EDA with an optimal parameter set, only the performance in the fault-free case is considered. In this paper, we investigate the analysis of the behavior of an EDA when being exposed to erroneous data. Analyzing this behavior can yield valuable feedback to the application engineer. Our approach shall support the application engineer's selection process of a parameter set from a predefined set of parameter sets. This is achieved by using fault injection to show what kinds of errors are to be expected for different kinds of faults. We present a case study where we illustrate this process.

## II. Related Work

We already mentioned in the introduction a number of papers that deal with the importance of the awareness of faults that occur due to environmental conditions. The authors of [2] even tried to create conditions for the nodes' components similar to those they would have to face on the glacier by putting them into a refrigerator. Nevertheless, software fault injection as we employ it is not mentioned in any of the papers.

Fault injection for event detection in WSNs was first applied in [7] for testing a distributed Bayesian algorithm. This approach envisions the sensor node as one unit and does not consider where faults come from and how they can



978-1-4799-2481-3/13 $31.00 © 2013 IEEE
DOI 10.1109/UIC-ATC.2013.28

develop. In addition, little attention is paid to the EDA that is actually employed. Previous work of ours has been inspired by this approach [5]. A shortcoming in both of these works is that only snapshots of one instant in time are considered.

Fault injection in WSNs has been investigated in [4], where a tool, AVR-INJECT for automated fault injection is presented. AVR-INJECT emulates hardware fault injection on assembly level and the effect of bit flips on system operations in several operating systems is analyzed in detail. In our considerations, we separate the EDA from any other software layer and focus on the EDA's reaction to erroneous data.

In [3], the authors illustrate the influence of temperature on low-power communications, reporting a major effect on signal strength and link quality. Allthough the motivation for this work comes from the same background as ours, the focus is on networking and EDAs do not play any part in this.

What is novel about our approach is that we use fault injection to understand how a change in the nodes' environment can have an impact on the EDA's service quality. We compare the changes in the EDA's service quality for different parameter sets in order to give a recommendation about the individual parameter sets' suitability.

## III. Analysis of EDAs by means of fault injection

The concern of our research starts with the point where the application engineer has already decided on either a specific EDA or a selection of EDAs along with suitable parameter sets. There are four different cases, as shown in Table I.

| | Parameter set(s) already fixed | Different parameter set(s) considered |
|---|---|---|
| One algorithm | **A** | **B** (focus of this paper) |
| Several algorithms | **C** | **D** |

Table I
The four categories in the process of choosing an EDA

***Case A:*** If the application engineer has already chosen one specific algorithm and also a parameter set, all decisions have already been made. In this case, carrying out fault injection experiments helps to understand how this specific combination of EDA and parameters would perform when being exposed to erroneous data, i.e. how long correct service would be sustained in case of emergency. These results can be used for risk assessment.

***Case B:*** The application engineer has already decided on a certain EDA and has a number of suitable parameter sets at hand, but is not sure which parameter set to choose. Especially if multiple metrics are employed, there may not be one optimal set. This case is the focus of this paper, Section IV presents a detailed case study on this topic.

***Case C:*** The application engineer has chosen several EDAs that are suited for the WSN's task and for each algorithm, a parameter set that is considered optimal has been provided. As different EDAs show different dynamics and weak points, the performances in this case are more difficult to compare. Moreover, if only one parameter set is considered, any bad performance can be either due to the EDA's functionality but also due to properties of the parameter set.

***Case D:*** If several EDAs are available and multiple parameter sets are available for each EDA, the application engineer quickly is faced with a large number of possibilities. For a large number of combinations of EDAs and parameter sets the data resulting from this approach might become too bulky.

### A. Selection process

Figure 1 illustrates the procedure of selecting an appropriate parameter set (Case **B**). The application engineer defines a topology, a sequence of events to take place and a benchmark, which is a set of disturbances (see below for additional details about benchmarks). The EDA is then simulated on the defined topology, with the defined events taking place. The disturbances interfere with normal operations, so that the detection outcome of the EDA(s) is disturbed. These disturbances are modeled by a fault injection mechanism. The simulation is performed once for each parameter set. The following evaluation is of a statistical nature (e.g. number of false positives) and can also be displayed graphically. Finally, by considering the evaluation, the most adequate parameter set is selected.

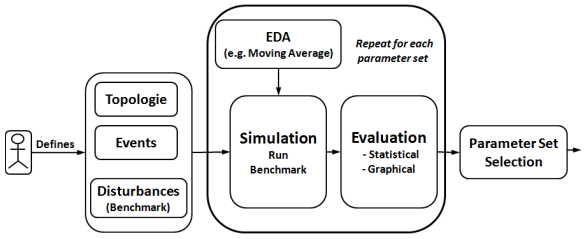



Figure 1. The process of finding the most adequate parameter set

### B. Disturbances and benchmarks

A disturbance represents an unwanted but unavoidable influence on the sensor node. We focus our considerations of consequences of the disturbances on value faults in the sensor(s) of the nodes. Each disturbance is characterized by

1) Start point in time with respect to the begin of the simulation
2) Geographic spread (time dependent)
3) Area covered
4) Movement direction (0 if stationary)

5) Speed (0 if stationary)
6) Duration
7) Effect of the disturbance, the contortion function
8) Fault probability

The contortion function defines the unwanted effect on the sensor node's behavior. For example, this can be set to a sampled value to zero. The fault probability defines the probability of a sample being contorted.

A benchmark is a set of disturbances. That is, the EDA is faced with a number of disturbances of potentially different effects and probabilities. When the application engineer defines a benchmark for an experiment, he also defines which events are to take place during the experiment. Events are characterized by the first six attributes that describe disturbances, as metioned before. In addition, events have a feature, which is sampled by the sensor, i.e. the luminous flux.

## IV. Case Study

The data used for this case study was captured from a part of the infrastructure of the RIPLECS project [6] that is dedicated to WSNs. The RIPLECS project provides remote-labs for e-learning targeting the subject of information and communication systems.

The installation features six sensor nodes, two light sources, equipment for energy harvesting and a base station. The intensity of the light sources can be regulated via a web interface. The periodic measurements of the luminous flux by the nodes are displayed in the web interface as soon as they have reached the base station. If desired, the measurements are also logged in a file for offline access. The purpose of the installation is to detect light events.

The contextual setting of this case study is a meeting room, where the sensor nodes detect the luminous flux, perform an EDA and forward their findings to a base station. With the help of other sensors (motion, sound ...) the base station decides if actions concerning the shutters and the lightning are to be taken. This is a complex system, and we will focus our considerations solely on the event detection of the light sensors. The actual decision process and acquisition of additional sensor data is not part of this work.

We conduct two experiments. In the first experiment, only zeroes are injected in order to model how the light sensor is obscured in irregular intervals. In the second experiment, random values are injected. That way, we model erratic behavior due to shorts that occur because of water that entered the node through a crack in the node's casing.

### A. Considered algorithm, parameters and data provision

This case studies considers a basic thresholding algorithm that employs a sliding window, a so called moving average (MA). The MA needs two parameters, the threshold $\Theta$ and the window size $k$. If the mean of the last k values exceeds $\Theta$, an event is detected.

For both parameters we have two values to choose from. According to the above mentioned case **B**, the application engineer has already decided to use the MA and has found parameter sets that perform well. The parameter set that is best suited for the needs of the application has to be chosen. To support the decision process, the performance of the different parameter sets is analyzed via fault injection.

The magnitude of all nodes' measurements for different light levels was captured from the RIPLECS project. However, as it is difficult to operate the sliders of the light regulators to exact points, the captured values were arranged to create an arbitrary behavior of the magnitude of the luminous flux.

In order to keep the presentation of the evaluation concise, only one node of the installation is considered. This corresponds to a base case of the proposed methodology: The stationary disturbance is so small that it only covers one node. The disturbances persist during the entire experiments.

### B. Experiments

The parameters available for the MA are $k = \{3, 4\}$ and $\Theta = \{140, 150\}$. All four combinations of these parameters deliver satisfactory detection results. The demands on the EDA are that events should be detected with rather low delay, the end of an event, however, can be detected with a delay. The first attribute can be quantified by looking at the false negatives produced by the algorithm. Allowing a delay in the detection of the end of an event means that the number of false positives can be slightly elevated and accordingly that a small number of false positives is only a secondary decision criterion. Additionally, a certain inertness of the EDA is desirable, so that two events that are separated by a small number of samples be identified as one.

Injecting a fault refers to changing a measured value (the input samples for the EDAs) to a faulty value. In each experiment, the first run with a fault probability of 0% is referred to as the $clean\ run$ as no faults are injected. The results of the clean run demonstrate how well the detection algorithm performs in an undisturbed environment.

In the first experiment, only zeroes are injected. In the second experiment, we inject numbers that are uniformly distributed in a range from zero to 1000. The reason for choosing the lower limit is that luminous flux per definition is never a negative number. The reason for the upper limit is that in the WSN installment, the maximum possible magnitude of the luminous flux measurable by the node in question is roughly 1000 lm. Sanity checks can ensure the adherence to these bounds. Values that are out of bounds can easily be replaced by the appropriate bound by the EDA.

Both experiments consist of 25 runs with increasing fault probabilities. Starting with a fault probability of two percent, in each run the fault probability is raised by two percentage points. Consequently, in the last run, every value has a 50% chance to be substituted. The upper level of 50% was chosen

for demonstrativeness, so that the trend of each metric can be observed clearly.

### C. Evaluation process

The goal of the evaluation process is to show how the application engineer can be supported in finding the optimal parameter set. In order to compare the suitability of the different parameter sets, we use the following metrics:

- Number of detected events
- False negative samples (no event was detected although there was one present)
- False positives samples (an event was detected although there was none present)

The results of the first run correspond to the fault-free case, they are displayed in the graph as the first data point (probability = 0). The behavior of the EDA for the different parameter sets in the clean run corresponds to that displayed in Figure 3.

For each fault probability, a data vector that contains the disturbed data is created. Subsequently, this data vector is used to perform event detection with all four parameter sets. Note that for each parameter set, exactly the same data vector is used. For each fault probability we averaged the results from 1000 runs.

In experiment 1, the contortion function is to set the affected values to zero. That is, for each sample, the probability of being zero corresponds to the fault probability. In experiment 2, the affected values are assigned a random value from the range of 0 to 1000.

Determining false positives and false negatives demands that there is a ground truth defining what an event is or when an event takes place, respectively. An event takes place when the dimmer switch has been switched on long or often enough with a sufficient magnitude. If a node samples the light sensor just after the light was switched off, a value greater than zero may still be sampled although no event is present.

We model this ground truth by annotating every measurement with a boolean that indicates whether an event takes place at that particular instant of time, i.e. if the light switch is on, or not. This additional boolean provides the possibility to decide whether a detected event has been correctly detected or if it is a false positive. Figure 2 displays the input for the two experiments, there are six events with a cumulative duration of 43 samples. In order to check the affinity of an EDA to overlook a short interval between two events, there are short intervals of 1, 2 and 4 samples included in the input vector.

Figure 3 outlines the detection outcomes for the MA with each of the four parameter sets. In accordance with Figure 2, the lowest line (turqoise annotated with stars) shows when an event is present, the ground truth. This figure only shows the detection outcome in the undisturbed case and shows how well the parameter set fits the undisturbed input. The first event is the only event that is detected by all four instances at the same time, at the very beginning of the event. The purple line (annotated with an 'x', k=3, $\theta$=140) is always the first to detect an event. The green line (annotated with triangles, k=3, $\theta$=150) nearly as good as the purple line in detecting the start of an event, but is definitely quicker in detecting the end of an event. The green line is the only one to identify events five and six as two distinct events.

The red (annotated with squares; k=4, $\theta$=140) and blue (annotated with diamonds; k=4, $\theta$=150) are both slower in detecting an event. This inertness is due to the larger window size.

Based on these facts, the best choice would be the purple line, as it matches the description best. The next subsections will discuss the findings from the fault injection campaign and consequently give a recommendation which parameter set to choose.

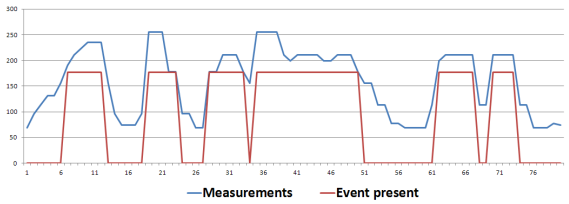



Figure 2. The 80 samples of the undisturbed input include six events with different distances [ # of samples] between each other. There are a total of 43 samples where an event takes place.

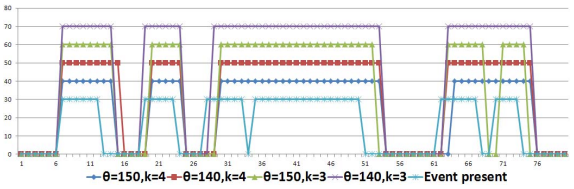



Figure 3. Detection results for the MA with different parameter sets, the turquoise line indicates when an event actually is present. For all lines applies, that an event is detected, when the value is greater than zero.

### D. Results

It applies for both experiments that out of the six events that are actually present, no instance of the MA detects all six events. This stems from the fact that the third and fourth event are only one sample apart. As we are looking for an inert algorithm, it is advantageous for all parameter sets that are considered, that the MA identifies these two events as one. The last two events are identified as one event by all parameter sets except for the set $\{\theta = 150, k = 3\}$ (the green line).

The detailed results of both experiments are plotted below. For experiment 1, where only zeroes are injected, Figure

4 shows the number of detected events, Figure 5 shows the number of false positives and Figure 6 the number of false negatives. Analogous, Figures 7, 8 and 9 display the corresponding results for the second experiment, in which random values ranging from 0 to 1000 are injected.

*1) Discussion of experiment 1 - injection of zeroes:* As already mentioned, all but one parameter set detect only four of the six events in the clean run. Up to a fault probability of 26%, the number of detected events slightly increases for all parameter sets. Subsequently, the number decreases to roughly the initial values. An exception is presented by the purple line, $\{\theta = 140, k = 3\}$, that sets itself apart from the other lines. Starting with four detected events, this line has a steeper rise, detecting more than eight events for a fault probability of 26%. This means that this parameter set is more sensitive than the other sets when it comes to disruptions of the sensed events via an injection of a zero. The same, but to a less extreme extent, applies to the green line.

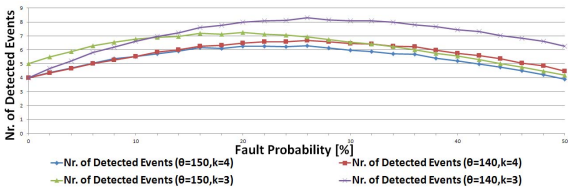



Figure 4. Number of detected events for the injection of zeroes. All parameter sets detect less than the six events that are actually present. The purple line (marked with 'x's) sets itself apart from the others because with these parameters, an injected zero is frequently interpreted as the beginning of a new event.

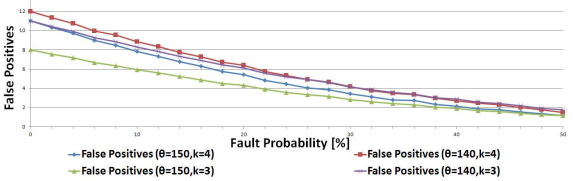



Figure 5. Number of false positives for the injection of zeroes. The green parameter set shows the best performance.

The green line is the all time winner when it comes to false positives. Starting at 8 false positives, the other parameter sets have 11 or 12 false positives. All four lines trend towards 1 for a fault probability of 50%, but the green line always is the lowest.

Things look different when considering the number of false negatives: All parameter sets have a comparable degree of gain. As the lowest line, the purple line always performs best and the green line always performs worst.

We conclude in this first experiment that the purple parameter set is not recommendable, because it tends to

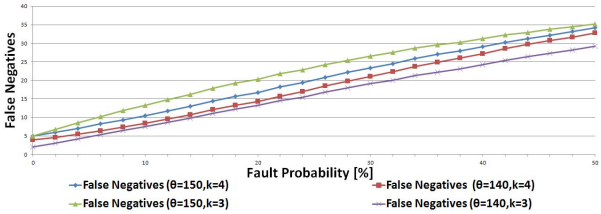



Figure 6. Number of false negatives for the injection of zeroes. Here, the purple line performs best.

identify the disruption of one event as the start of a new event. The green parameter set also has this nature, but not as extreme. The recommendation is to choose the red parameter set as the detection outcome approaches the outcome of the green parameter set and the rate of false negatives is second best to the purple set. Although the blue line performs better at false positives, it is not recommended as false negatives are the stronger decision criterion.

*2) Discussion of experiment 2 - injection of random values:* There is no outlier as in experiment one. The green parameter set tends to detect the highest number of events. This number only tops the true number of six elements only slightly. The purple line starts like the blue and red line at four events, but rises a little higher than the other two to a little more than five. The decrease of the purple line starting at a fault probability of 24% is less steep than at the other lines. The two parameter sets with the larger window size tend to detect fewest events.

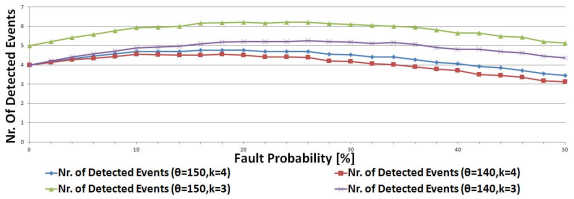



Figure 7. Number of detected events for the injection of random values ranging from 0 to 1000. The green parameter set is closest to the true number of events.

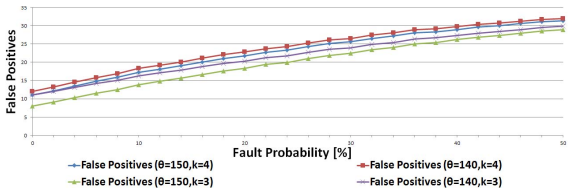



Figure 8. Number of false positives for the injection of random values ranging from 0 to 1000 - very different from experiment 1. Due to the injection of random values ranging from 0 and 1000, the average number of false positives raises with increasing fault probability.

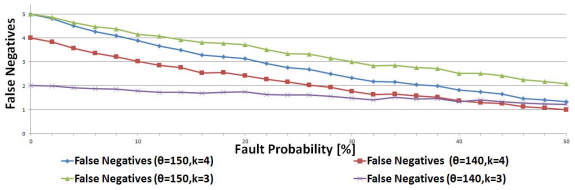



Figure 9. Number of detected events for the injection of random values ranging from 0 to 1000. The recommendation - the purple line - shows a close to steady performance.

As in experiment 1, the green parameter set performs best at false positives. The other three lines are relatively close to each other in the beginning, with the purple line having a smaller gain than the other two lines. With raising fault probability, the purple line slowly approaches the green line. It is noteworthy that in experiment 1, the number of false positives decreases, whereas in experiment 2 it increases. The reason for this behavior is that with the injection of zeroes, an injected value automatically means that an event is disrupted, if the zero is injected during an event. In experiment 2, a non-event sample can be converted to an event sample, if the injected value is sufficiently high.

The purple parameter set clearly performs best with respect to false negatives. This line starts out lowest and stays lowest nearly to the end of the experiment.

Based on these findings, the recommendation is the purple line due to the good performance at false negatives.

*3) Conclusion of discussion of results:* The initial analysis of the four parameters sets yielded that the purple parameter set is best suited for the application in question. Experiment 1 recommended the red parameter set and explicitly discouraged the use of the purple parameter set due to its sensitivity to injected zeroes. The second experiment produced a recommendation for the purple parameter set.

As here are two votes out of three in favor of the purple parameter set, this would be the final choice. Nevertheless, the experiments show that no parameter set is optimal under all circumstances. While the purple line shows very good results in the undisturbed case and in experiment 2, it is the only parameter set that shows a definite weakness in one of the experiments. The resulting feedback to the application engineer is that this parameter set is suited for an undisturbed environment and also for the disturbances described in experiment two. But his parameter set is not appropriate when instead of a correct measurement, zero is received. The best reaction to this insight would be to enhance the algorithm, so that upon the reception of an extremely low or extremely high value, an additional sanity check is performed.

## V. Conclusion

In the world of WSNs, there are numerous sources of faults. It is therefore important to pay attention to the behavior and performance of the EDA when being exposed to erroneous data when selecting an EDA's parameter set for event detection in a WSN.

In this work, we elaborated on how to pick the best suited parameter set out of a number of qualified parameter sets and underlined this process by an application-oriented case study. The case study showed how a parameter set that is well suited for a fault-free situation can lose its excellent performance when being exposed to erroneous data.

In addition, we showed, how the proposed approach can be used to provide the application engineer with feedback about an EDA's weak point.

## References

[1] M. Bahrepour, N. Meratnia, M. Poel, Z. Taghikhaki, and P. J. M. Havinga. Distributed event detection in wireless sensor networks for disaster management. In *Intelligent Networking and Collaborative Systems (INCOS), 2010 2nd International Conference on*, pages 507–512, 2010.

[2] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The hitchhiker's guide to successful wireless sensor network deployments. In *in Proceedings of ACM SenSys*, 2008.

[3] C.A. Boano, N. Tsiftes, T. Voigt, J. Brown, and U. Roedig. The impact of temperature on outdoor industrial sensornet applications. *Industrial Informatics, IEEE Transactions on*, 6 (3):451–459, 2010.

[4] M. Cinque, D. Cotroneo, C. Di Martino, and A. Testa. An effective approach for injecting faults in wireless sensor network operating systems. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 567–569, 2010.

[5] Karima B. Hein and Reinhold Weiß. Minesweeper for sensor networks – making event detection in sensor networks dependable. In *International Conference on Computational Science and Engineering 2009*, volume 1, pages 388–393. IEEE Computer Society, August 2009.

[6] L.B. Hörmann, M. Steinberger, M. Kalcher, and C. Kreiner. Using a remote lab for teaching energy harvesting enhanced wireless sensor networks. In *Global Engineering Education Conference (EDUCON), 2013 IEEE*, pages 1109–1117, 2013.

[7] B. Krishnamachari and S. Iyengar. Distributed bayesian algorithms for fault-tolerant event region detection in wireless sensor networks. In *IEEE Transactions on Computers*, volume 53, 2004.

[8] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *In Int. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, 2006.

[9] Byunghun Song, Haksoo Choi, and Hyung Su Lee. Surveillance tracking system using passive infrared motion sensors in wireless sensor network. In *Information Networking, 2008. ICOIN 2008. International Conference on*, pages 1 –5, 2008.

[10] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a sensor network expedition. In *European Conference on Wireless Sensor Networks 2004*, pages 307–322, 2004.

2009 International Conference on Computational Science and Engineering

# Minesweeper for Sensor Networks - Making Event Detection in Sensor Networks Dependable

Karima B. Hein, Reinhold Weiss
*Institute for Technical Informatics*
*Graz University of Technology*
*Graz, Austria*
*Email: {Hein,RWeiss}@TUGraz.at*

***Abstract*—Event detection using Wireless Sensor Networks (WSNs) has become a new field of research in the past years, increasing the need for dependability and fault tolerance. Our work exploits the massive redundancy of large WSNs in combination with neighbours' relations to identify faulty nodes. We present a new approach to categorize nodes in being faulty or fault free based on the event detection results of the nodes' neighbours and the nodes adjacent to the neighbours. For error probabilities $<$ 0.2 our algorithm performs closely to other work in the field, and performs considerably better for error probabilities up to 0.5.**

***Keywords*-Dependability, sensor networks, fault tolerance**

## I. Introduction

Sensor Networks are a widespread technology that has gained considerable importance in recent years. Particularly, *Wireless* Sensor Networks (WSN) are used for ensuring the safety of human beings i.e. patient monitoring or structural health monitoring [1]. A WSN is a network consisting of spatially distributed devices, so-called nodes. These nodes are equipped with a number of sensors to cooperatively monitor environmental conditions. The nodes forward their measurements as raw or preprocessed data to a base station. As the user's interface to the WSN, a base station is a device much better equipped with energy and computational resources than a regular node.

Environmental monitoring with WSNs encompasses a vast area of applications, with the most seminal project being habitat monitoring on Great Duck Island [4]. Related applications concerning event detection include forest fire detection [2] and monitoring of volcanic sites [6].

Faulty nodes can propagate incorrect values, misleading neighbouring nodes. In a WSN concerned with event detection, the application running on the base station is often only interested in the presence of events of a certain size. A small number of malfunctioning nodes could convince their neighbours of an incorrect spread of the event, consequently preventing a positive report to the base station.

Especially with WSNs deployed outdoors there are countless possibilities for the introduction of faults starting from incorrect deployment to transient errors caused by dew or shadow. Hence, it is clearly desirable to have more dependable, and thus more robust WSNs.

The service of the WSNs we consider is to detect event regions. Accordingly, a service failure of the system corresponds to the reporting of false positives or false negatives. The algorithm we propose is concerned with diagnosis, paving the way for fault tolerance as well as fault removal. The benefit of fault tolerance is that in spite of the presence of faulty nodes, the network can continue to work correctly. Fault removal can be realised by either flagging or rebooting faulty nodes.

There are two main ways to provide fault tolerance in WSNs: Either exclusively considering the single node or considering the network in connection with the base station. Fault tolerance conducted by the base station implies lots of bidirectional communication, i.e. the base station has to gather many data packets from a large number of



978-0-7695-3823-5/09 $26.00 © 2009 IEEE
DOI 10.1109/CSE.2009.146

nodes (usually all nodes) to be able to arrive at an informed decision. Contrarily, considering solely single nodes leads to an autonomous decision involving less overhead.

The core idea of this work is to provide an algorithm for autonomously answering the question 'Is the node faulty or non-faulty?'. Our node-centred system considers the node's own findings, the findings of the node's neighbours and also the findings of the nodes adjacent to the neighbours. The title of this paper was inspired by the Minesweeper game. Our simulator shows a digit at every node (cf. Figure 1 on page 6) indicating the number of neighbours that detect an event, in this manner bearing a resemblance to Minesweeper.

## II. Related Work

One of the first publications tackling fault tolerance in event detection is [3], where a distributed Bayesian algorithm is introduced. The authors argue that "measurements due to faulty equipment are likely to be uncorrelated, while environmental conditions are spatially correlated". The authors fail to mention that measurements due to faulty equipment can be correlated with environmental conditions. Consider a cluster of nodes deployed in the field that is shaded by a hedge or a mould: morning dew will stay longer on the shadowed nodes than on their sunbathed neighbours. This is an example where environmental conditions can cause equipment to become (transiently) faulty. In addition, the authors concentrate their considerations exclusively on nodes that lie in the inner region of the event, whereas our work is concerned with nodes situated at the boundary of the region as well.

In [5], Ould-Ahmed-Vall et al. further develop the ideas of [3] by introducing different failure probability levels for each node. Two different error models are considered: one takes into consideration that "nodes that are closer to each other have a higher spatial correlation than nodes that are farther apart"' and the other model considers how the neighbours are geographically distributed around the node. The authors don't comment on any particular circumstances concerning nodes on the boundary. In contrast to [5], our scheme does not only take the node's neighbours' decisions into account, but also the decisions of the nodes adjacent to the neighbours.

This idea of "neighbours' neighbours" has also been considered in [7], where Xiang et al. introduce a distributed weighting scheme for the detection of event regions. The proposed algorithm weights the node's neighbour's findings and their neighbours' findings with two different weights depending on the detection outcome and the distance to the node in question. Complementary to simple weighting, our approach considers weighted proportions of distinctive characteristics of the node and its eight neighbours. Nevertheless, as the idea described in [7] is closest to our approach, we compare our results to those of [7] in Section IV.

## III. Minesweeping for Fault Diagnosis

Only the correct identification of a fault enables mitigation of the fault's impact. Consequently, the crucial first step in dealing with faulty nodes is diagnosis.

### A. Problem Formulation

We consider a large WSN with the nodes installed in a grid topology. The nodes perform binary event detection, meaning a node only determines if an event is present or not. The measurements taken by a node can be incorrect, resulting in a false positive if an event is detected although there is no event present. Analogous, an event that goes undetected is called a false negative. The aim of this work is to explore if the event detection of each individual node is correct. This is achieved by comparing its decision with the decisions of the nodes in the closer vicinity, i.e. the two-hop-neighbourhood.

### B. Assumptions and Context

We only consider binary event detection, that is if the sensor reading is above a threshold, the sensing application's output is '1', i.e. the node detects the event. Otherwise, if a node does not detect the event, the sensor reading is not above the threshold, and the output is '0'. A faulty node outputs the opposite of the truth, that is a false

negative or a false positive respectively. In addition, only events of a reasonable size are considered, spanning at least a couple of nodes.

We also assume that each node knows its one-hop-neighbours. This knowledge is established in an initialization phase. WSNs are characterized by their severe restraints on energy, and unfortunately, radio communication is rather power-hungry. We therefore consider WSNs characterized by a high quantity of node-to-node communication. In such systems, the data required by our algorithm can be piggy-backed on data packets of the regular data flow. In such scenarios, we assume that reasonable energy supplies are provided in the form of large energy reservoirs and/or energy harvesting devices. Standard batteries can also be a sufficient energy supply for WSNs with a shorter mission time.

### *C. Fault Diagnosis - by Means of Playing Advanced Minesweeping*

Each node has a total of 8 neighbours. The node and each of its neighbours either detect the event or not. Nodes detecting an event are displayed dark in figures. The digits at each node refer to the number of neighbouring nodes that detect an event. The consideration of these numbers was inspired by the Minesweeper game, where digits show the number of mines in the adjacent fields. A mine in Minesweeper corresponds to a neighbouring node that detects an event in our algorithm.

Table I summarizes the nomenclature used. The nomenclature consists of terms adopted from [7] complemented with terms newly introduced for our algorithm.

**Definition** A $positive$ node or neighbour refers to a node that detects an event.

A node is faulty if the binary variable denoting the presence of an event at node $a$, $T_A$, is complementary to the binary variable $S_a$ that indicates if an event has been detected by node $a$: $T_a \neq S_a$

Every node is assigned a value $v(a)$, characterizing the node's inclination to be faulty. If $v(a)$ is larger than a threshold $\Theta$, the node is believed not to have detected the event, resulting in $R_a$ = 1, otherwise $R_a$ = 0. A fault was found by the diagnosis algorithm if $T_a \neq S_a$ and $R_a = T_a$ and a newly introduced fault corresponds to $T_a = S_a$ and $R_a \neq T_a$.

Table I
SUMMARY OF THE USED NOMENCLATURE

| Symbol | Definition |
|---|---|
| $a$ | The currently considered node |
| $b_i$ | A node adjacent to node $a$; {B} is the set of all $b_i$ |
| $c_i$ | A node adjacent to node $a$ that is detecting an event; {C} is the set of all $c_i$ |
| $T_a$ | Binary variable indicating if an event is present at node $a$ |
| $S_a$ | Binary variable indicating the detection result of node $a$ |
| $R_a$ | Binary variable indicating if the node considers to have detected an event after the diagnosis algorithm has been run |
| $N$ | The number of neighbours of node $a$ that detect an event |
| $P(all)$ | The number of all neighbours' adjacent nodes that detect an event |
| $P(N)$ | Same as P(all), but considering only positive neighbours of node $a$ |
| *Dom(a)* | The maximum number of positive neighbours any positive node adjacent to node $a$ has |
| $\omega(a)$ | The weighting factor $\omega(a) = \frac{8-Dom(a)}{N}$ |
| $\Theta$ | Threshold |
| $v(a)$ | The decision value compared to $\Theta$ |

*1) Computation of the Decision Value $v(a)$:* The set {B} contains all nodes $b_i$ that are adjacent to the node $a$. The set {C} contains all nodes $c_i$, that are adjacent to the node $a$, with $S_{c_i}$ = 1 $\rightarrow$ {C} $\subset$ {B}. The number of neighbours of a node x that detect an event is generally computed by

$$positives(a) = |C| = N$$

Computing

$$P(all) = \sum_{i=1}^{|B|} positives(b_i)$$

$$P(N) = \sum_{i=1}^{|C|} positives(c_i)$$

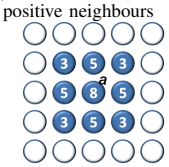
$$Dom(a) = max_{c_i \in C}\big(positives(c_i)\big)$$

$$w = \frac{8 - Dom(a)}{N}$$

leads to the decision value $v(a)$

$$v(a) = \begin{cases} \omega(a) \cdot \frac{P(N)}{P(all)} & P(N) \neq 0 \\ \infty & P(N) = 0 \end{cases}$$

Figure 1. $Dom(a)$ = 8 for all neighbours of the central node $a$, because $a$ has 8 positive neighbours




finally resulting in the decision $R_a$

$$R_a = \begin{cases} 1 & v(a) < \Theta \\ 0 & otherwise \end{cases}$$

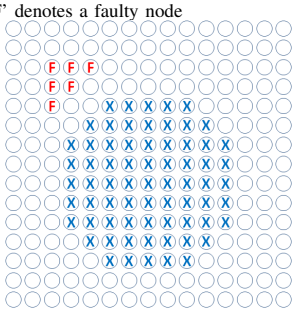
$P(N)$ yields zero if $N$ = 0 that is, no neighbour detects an event. This means that the node is the only one detecting the event within a one-hop-radius. Assigning an infinitely large $v(a)$ value to a node entails a guaranteed exceeding of the threshold. Therefore, the node is believed not to have detected an event, agreeing with our precondition about minimal event size.

The other extreme case is $Dom(a)$ = 8, as this implies $\omega(a)$ = 0, and consequently $v(a)$ = 0. $Dom(a)$ being maximal means that there is at least one node among the elements of {C} that features 8 positive neighbours. Note that this can also refer to the node $a$ itself. Assuming an error free scenario, a node having 8 positive neighbours is situated within the event region, at least one hop away from the boundary. The weighting scheme considers nodes in such a position as part of the event region. In the scenario depicted in Figure 1, dark nodes sense the event, while white nodes do not. All neighbours of node $a$ are assigned $\omega(a)$ = 0, because for all neighbours $x_i$ applies $Dom(x_i)$ = 8. Note that a node can only be assigned $Dom(a)$ = 8 if the node itself detects an event. The depicted scenario has the minimum number of nodes, where assigning $\omega(a)$ = 0 is possible.

*2) Threshold Selection:* Our diagnosis algorithm is optimised for the detection of convex event regions. Stating a minimum number of positive nodes necessary in the vicinity is not meaningful. This is because $v(a)$ generally is influenced by the number of detections in a two-hop-radius. In addition, we do not target small events that are detected by less than seven nodes. We found that a threshold $\Theta$ = 0.1775 performs well in experiments with convex event regions.

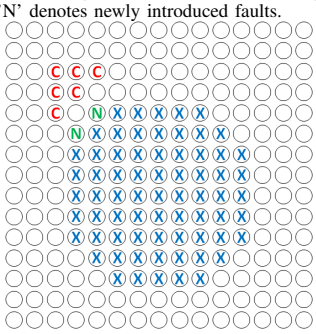
*3) Sample Scenario:* Figure 2 shows a scenario with a circular event region at the centre and six faulty nodes in the top left. An event is present at nodes represented by an 'X' and nodes outside the event region are denoted by an 'O'. Faulty nodes are depicted by an 'F'.

Figure 2. Sample scenario: 'X' denotes a node that detects an event and 'F' denotes a faulty node



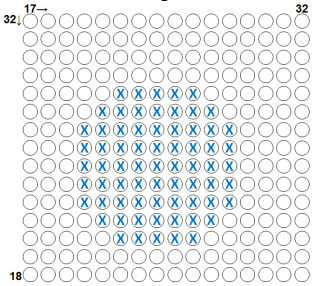
The corresponding output of our diagnosis algorithm is depicted in Figure 3.

Figure 3. Diagnosis for the sample scenario in Figure 2 obtained from our algorithm. 'C' denotes correctly identified faults and 'N' denotes newly introduced faults.



The algorithm correctly identifies all six faulty nodes (indicated as 'C'), but introduces two new faults (indicated as 'N'). This happens because the two newly introduced faults $n_1$ and $n_2$ are situated between two clusters of positive nodes. The larger cluster consists of nodes that really detect an event

Figure 4. The evaluation scenario similar to the one in [7] with only the event region in the upper right quadrant shown. The numbers indicate the column and line with the origin located in the lower left corner, the entire evaluated WSN grid consists of 32x32 nodes. Nodes detecting an event are denoted by 'X'




and the smaller cluster consists of nodes that incorrectly report to have detected an event. The diagnosis algorithm does not distinguish between 'detecting' and 'believing to detect'. Nodes $n_1$ and $n_2$ are considered to be faulty because $v(n_1)$ = 0.146 and $v(n_1)$ = 0.150 which is relatively close to the threshold $\Theta$ = 0.1775

## IV. Experimental Results

To evaluate the performance of our algorithm, we created a scenario equivalent to the one used in [7]. For better comparability, we adopted the metrics of evaluation from [7] and [3] respectively.

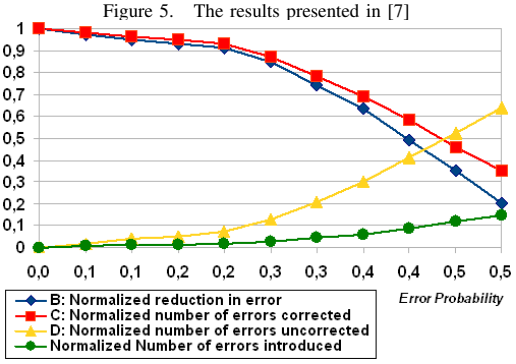
### A. Setup and Experiment

We have developed a Java application for the simulation and evaluation of our algorithm. We simulate a WSN of dimension 32x32 nodes with the nodes arranged in a grid topology, as depicted in Figure 4. As before, nodes, where the event is present are represented by an 'X', nodes outside the event region are denoted by an 'O'. The circular event region with a diameter of 9 nodes is situated in the upper right quadrant. Starting with an initial error rate of $p_{err}$ = 0.0 we increase $p_{err}$ in steps of 0.05 up to a maximum of $p_{err}$ = 0.5. which corresponds to every second node being faulty. We performed 100 simulation runs for every error rate in the interval [0,0.5] and averaged the individual results. The threshold was set to $\Theta$ = 0.1775.

Table II
Employed metrics, adapted from [3] and [7]

| Symbol | Definition | Evaluation Function |
|---|---|---|
| $\alpha$ | Average number of errors after decoding | B = $\frac{1-\alpha}{np}$ |
| $\beta$ | Average number of errors corrected | C = $\frac{\beta}{np}$ |
| $\gamma$ | Average number of errors uncorrected | D= $\frac{\gamma}{np}$ |
| $\delta$ | Average number of new errors introduced | E = $\frac{\delta}{np}$ |

Figure 5. The results presented in [7]




### B. Evaluation

Table II lists the employed metrics. We compare our results only to the findings of [7], displayed in Figure 5 as the results for all evaluation functions in [7] are better than those presented in [3]. The graphs plotted in Figure 6 illustrate the performance of our approach. Figure 7 compares the average number of errors after decoding (Evaluation Function 'B') of our algorithm to the results presented in [7]. This metric considers undetected errors as well as newly introduced errors. For lower error probabilities, our algorithm performs slightly worse, but for $p_{err} > 0.25$ our algorithm performs considerably better. The strength of our algorithm is that the number of corrected errors is nearly constant and the number of newly introduced errors and uncorrected errors is low. In order to mistake a fault-free node for a faulty one, there has to be a larger accumulation of nodes that detect an event,

Figure 6. The results achieved by our algorithm.

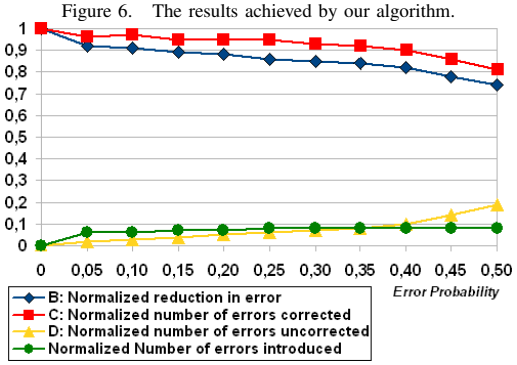
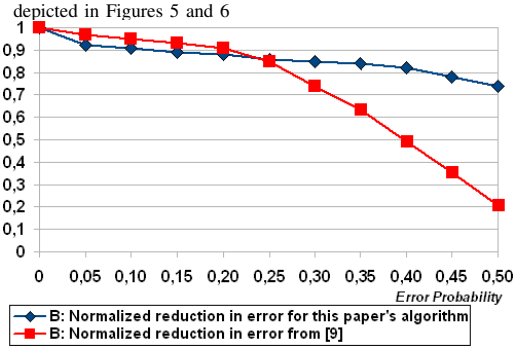



Figure 7. Exclusive comparison of Evaluation Function B depicted in Figures 5 and 6




no matter if they are right or not. This effect is illustrated by Figure 2.

We have learned from previous experiments that algorithms that perform very well at detecting errors tend to introduce a lot of new errors, yielding a large total of errors after decoding. We believe that our approach shows a good balance between the amount of detected errors and newly introduced errors.

For an even improved performance we plan to explore a hybrid scheme that combines these two approaches into one algorithm. As the nodes are not aware of the actual error rate or the number of detected errors, this scheme would require a supervising instance, e.g. a clusterhead. This clusterhead would be aware of the number of detected errors within its cluster and would select an algorithm depending on the number of detected errors.

## V. Conclusion and Future Work

We presented a dependability concept targeting WSNs that perform event detection. Our considerations concentrated on furnishing nodes with means to perform fault diagnosis autonomously, this being one step towards attaining dependability. The key feature was the comparison of the findings of the considered node, the node's neighbours and also the findings of the nodes adjacent to the neighbours. We showed that our algorithm performs nearly as good as the one presented in [7] for an error probability of $p_{err} < 0.25$, but performs considerably better for higher error probabilities of up to p=0.5.

Further work will include exploring ways to describe event characteristics such as non-convexity by processable statements from which optimal thresholds can be deducted.

## References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4), 2002.

[2] D. M. Doolina and N. Sitara. Wireless sensors for wildire monitoring. In *SPIE Symposium on Smart Structures & Materials NDE 2005*, March 2005.

[3] B. Krishnamachari and S. Iyengar. Distributed bayesian algorithms for fault-tolerant event region detection in wireless sensor networks. In *IEEE Transactions on Computers*, volume 53, 2004.

[4] J. Kumagi. Life of birds. *IEEE Spectrum*, 41, 2004.

[5] E. Ould-Ahmed-Vall, G. F. Riley, and B. S. Heck. A distributed fault-tolerant algorithm for event detection using heterogeneous wireless sensor networks. In *45th IEEE Conference on Decision and Control*, 2006.

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

[7] Y. Xiang, H. Li, Z. Xie, and P. Li. Distributed weighting fault-tolerant algorithm for even region detection in wireless sensor networks. In *International Conference on Communications, Circuits and Systems*, 2008.

# Bibliography


[ACGV11] Giuseppe Amato, Stefano Chessa, Claudio Gennaro, and Claudio Vairo. Efficient Detection of Composite Events in Wireless Sensor Networks: Design and Evaluation. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pages 821–823, 2011.

[AGPB+12] Alvaro Araujo, Jaime García-Palacios, Javier Blesa, Francisco Tirado, Elena Romero, Avelino Samartín, and Octavio Nieto-Taladriz. Wireless Measurement System for Structural Health Monitoring With High Time-Synchronization Accuracy. *IEEE Transactions on Instrumentation and Measurement*, 61(3):801–810, March 2012.

[ALRL04] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 01(1):11–33, 2004.

[Ant13] Gary Anthes. Inexact Design: Beyond Fault-Tolerance. *Communications of the ACM*, 56(4):18–20, 2013.

[ASSC02] Ian F. Akyildiz, Wei Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 38(4):393–422, March 2002.

[Avi13] Algirdas Avižienis. The Architecture of a Resilience Infrastructure for Computing and Communication Systems. In *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*, pages 1–2, 2013.

[BBH+10] Carlo A. Boano, James Brown, Zhitao He, Utz Roedig, and Thiemo Voigt. Low-Power Radio Communication in Industrial Outdoor Deployments: The Impact of Weather Conditions and ATEX-Compliance. In *Sensor Applications, Experimentation, and Logistics*, volume 29 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 159–176. Springer Berlin Heidelberg, 2010.

[BGG08] Kenneth Bannister, Gianni Giorgetti, and Sandeep K. S. Gupta. Wireless Sensor Networking for "Hot" Applications: Effects of Temperature on Signal Strength, Data Collection and Localization. In *Proceedings of the fifth Workshop on Embedded Networked Sensors*, June 2008.

[BISV08] Guillermo Barrenetxea, François Ingelrest, Gunnar Schaefer, and Martin Vetterli. The Hitchhiker's Guide to Successful Wireless Sensor Network Deployments. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys08)*, pages 43–56, 2008.

[BKM+12] Nouha Baccour, Anis Koubâa, Luca Mottola, Marco Antonio Zú niga, Habib Youssef, Carlo Alberto Boano, and Mário Alves. Radio Link Quality Estimation in Wireless Sensor Networks: A Survey. *ACM Transactions on Sensor Networks*, 8(4), September 2012.

[BMH09] Majid Bahrepour, Nirvana Meratnia, and Paul J. M. Havinga. Sensor Fusion-based Event Detection in Wireless Sensor Networks. In *Mobile and Ubiquitous Systems: Networking Services, MobiQuitous, 2009. MobiQuitous '09. 6th Annual International*, pages 1–8, 2009.

[BMH11] Majid Bahrepour, Nirvana Meratnia, and Paul J. M. Havinga. Online Unsupervised Event Detection in Wireless Sensor Networks. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2011 Seventh International Conference on*, pages 306–311, 2011.

[BMP+10] Majid Bahrepour, Nirvana Meratnia, Mannes Poel, Zahra Taghikhaki, and Paul J. M. Havinga. Distributed Event Detection in Wireless Sensor Networks for Disaster Management. In *Intelligent Networking and Collaborative Systems (INCOS), 2010 2nd International Conference on*, pages 507–512, 2010.

[BTV+10] Carlo A. Boano, Nicolas Tsiftes, Thiemo Voigt, James Brown, and Utz Roedig. The Impact of Temperature on Outdoor Industrial Sensornet Applications. *Industrial Informatics, IEEE Transactions on*, 6(3):451–459, 2010.

[BVN+11] Carlo A. Boano, Thiemo Voigt, Claro Noda, Kay Römer, and Marco Zuniga. JamLab: Augmenting Sensornet Testbeds with Realistic and Controlled Interference Generation. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pages 175–186, 2011.

[CB10] Marcus Chang and Philippe Bonnet. Monitoring in a High-Arctic Environment: Some Lessons from MANA. *IEEE Pervasive Computing*, 9(4):16–23, October 2010.

[CCM+09] Marcello Cinque, Domenico Cotroneo, Catello Di Martino, Stefano Russo, and Alessandro Testa. AVR–INJECT: A Tool for Injecting

Faults in Wireless Sensor Nodes. In *23rd IEEE International Symposium on Parallel and Distributed Processing*, pages 1–6, 2009.

[CLF05] Qingchun Chen, Kam-Yiu Lam, and Pingzhi Fan. Comments on "Distributed Bayesian Algorithms for Fault-Tolerant Event Region Detection in Wireless Sensor Networks". *IEEE Transactions on Computers*, 54(9):1182–1183, 2005.

[CLL+08] Joyce Coleman, Tony Lau, Bhushan Lokhande, Peter Shum, Robert Wisniewski, and Mary Peterson Yost. The Autonomic Computing Benchmark. In *Dependability Benchmarking for Computer Systems*, pages 1–21. Wiley & Sons, Inc, 2008.

[CMT12] Marcello Cinque, Catello Di Martino, and Alessandro Testa. Analyzing and Modeling the Failure Behavior of Wireless Sensor Networks Software under Errors. In *8th IEEE International Wireless Communications and Mobile Computing Conference*, pages 567–569. IEEE, 2012.

[Con05] Cristian Constantinescu. Dependability Benchmarking Using Environmental Test Tools. In *Annual Reliability and Maintainability Symposium*, pages 567–571, 2005.

[Con13] RIPLECS Project Consortium. RIPLECS Project. `http://riplecs.dipseil.net/`, 2011-2013. Last accessed: 2014-01-17.

[CRS99] Joao Carlos Cunha, Mário Zenha Rela, and Joao Gabriel Silva. Can Software Implemented Fault-Injection be Used on Real-Time Systems? In *Dependable Computing*, volume 1667 of *Lecture Notes in Computer Science*, pages 209–226. Springer Berlin Heidelberg, 1999.

[CWJ+10] Peter Corke, Tim Wark, Raja Jurdak, Wen Hu, Philip Valencia, and Darren Moore. Environmental Wireless Sensor Networks. *Proceedings of the IEEE*, 98(11):1903–1917, 2010.

[dAFRG09] David de Andrés, Jesus Friginal, Juan Carlos Ruiz, and Pedro J. Gil. An Attack Injection Approach to Evaluate the Robustness of Ad Hoc Networks. In *15th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 228–233. IEEE Computer Society, 2009.

[DEM+12] Vladimir Dyo, Stephen A. Ellwood, David W. Macdonald, Andrew Markham, Niki Trigoni, Ricklef Wohlers, Cecilia Mascolo, Bence Pásztor, Salvatore Scellato, and Kharsim Yousef. WILDSENSING: Design and Deployment of a Sustainable Sensor Network for Wildlife Monitoring. *ACM Transactions on Sensor Networks*, 8(4):29:1–29:33, September 2012.

[dQMS10] Diego de Queiroz Macedo and Jaime S. Sichman. Analysis of Von Neumann Neighborhoods in Parallel Multi-agent Simulations. In *Second Brazilian Workshop on Social Simulation*, pages 27–32, Oct 2010.

[EPF13] EPFL. Sensorscope: Sensor Networks for Environmental Monitoring. `http://infoscience.epfl.ch/record/180186?ln=en`, 2013. Last accessed: 2013-12-12.

[FdARG11] Jesus Friginal, David de Andrés, Juan-Carlos Ruiz, and Pedro Gil. Using Performance, Energy Consumption, and Resilience Experimental Measures to Evaluate Routing Protocols for Ad Hoc Networks. In *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*, pages 139–146, 2011.

[FdARM11] Jesus Friginal, David de Andrés, Juan-Carlos Ruiz, and Regina Moraes. Using Dependability Benchmarks to Support ISO/IEC SQuaRE. In *Proceedings of the 2011 IEEE 17th Pacific Rim International Symposium on Dependable Computing*, pages 28–37. IEEE Computer Society, 2011.

[FEDV08] Niclas Finne, Joakim Eriksson, Adam Dunkels, and Thiemo Voigt. Experiences from Two Sensor Network Deployments Self-Monitoring and Self-Configuration Keys to Success. In *Wired/Wireless Internet Communications*, volume 5031 of *Lecture Notes in Computer Science*, pages 189–200. Springer Berlin Heidelberg, 2008.

[GST+08] Jayant Gupchup, Abhishek Sharma, Andreas Terzis, Al Burns, and Alex Szalay. The perils of detecting measurement faults in environmental monitoring networks. In *IEEE International Conference on Distributed Computing in Sensor Systems*, 2008.

[HC02] Jason L. Hill and David E. Culler. Mica: A Wireless Platform for Deeply Embedded Networks. *Micro, IEEE*, 22(6):12–24, 2002.

[HDCJ12] Wen Hu, Tuan Le Dinh, Peter Corke, and Sanjay Jha. Outdoor Sensornet Design and Deployment: Experiences from a Sugar Farm. *Pervasive Computing, IEEE*, 11(2):82–91, 2012.

[HHW12] Karima B. Hein, Leander B. Hörmann, and Reinhold Weiß. Using a Leaky Bucket Counter as an Advanced Threshold Mechanism for Event Detection in Wireless Sensor Networks. In *Proceedings of the Tenth Workshop on Intelligent Solutions in Embedded Systems*, pages 51–56, 2012.

[HHW13] Karima B. Hein, Leander B. Hörmann, and Reinhold Weiß. Analysis of Threshold-Based Event Detection Algorithms for Wireless Sensor Networks by Fault Injection. In *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International*

*Conference on Autonomic and Trusted Computing (UIC/ATC)*, pages 662–667. IEEE Computer Society, December 2013.

[HSKK12] Leander B. Hörmann, Michael Steinberger, Michael Kalcher, and Christian Kreiner. Educational Remote Lab Concept for Energy Harvesting Enhanced Wireless Sensor Networks. In *5th European DSP Education and Research Conference (EDERC)*, pages 95–99, 2012.

[HSKK13] Leander B. Hörmann, Michael Steinberger, Michael Kalcher, and Christian Kreiner. Using a Remote Lab for Teaching Energy Harvesting Enhanced Wireless Sensor Networks. In *Global Engineering Education Conference (EDUCON), 2013 IEEE*, pages 1109–1117, 2013.

[HSX+12] Renjie Huang, Wen-Zhan Song, Mingsen Xu, Nina Peterson, Behrooz Shirazi, and Richard LaHusen. Real-World Sensor Network for Long-Term Volcano Monitoring: Design and Findings. *IEEE Transactions on Parallel and Distributed Systems*, 23(2):321–329, Feb 2012.

[HTI97] Mei-Chen Hsueh, Timothy K. Tsai, and Ravishanar K. Iyer. Fault injection techniques and tools. *Computer*, 30(4):75–82, 1997.

[HV11] Zhitao He and Thiemo Voigt. Precise Packet Loss Pattern Generation by Intentional Interference. In *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*, pages 1–6, 2011.

[HW09] Karima B. Hein and Reinhold Weiß. Minesweeper for sensor networks – making event detection in sensor networks dependable. In *International Conference on Computational Science and Engineering 2009*, volume 1, pages 388–339. IEEE Computer Society, August 2009.

[IBS+10] François Ingelrest, Guillermo Barrenetxea, Gunnar Schaefer, Martin Vetterli, Olivier Couach, and Marc Parlange. SensorScope: Application-specific Sensor Network for Environmental Monitoring. *ACM Transactions on Sensor Networks*, 6(2):17:1–17:32, March 2010.

[IEC13] IEC. *International Electrotechnical Vocabulary. Chapter 191: Dependability and quality of service*, 1990–2013.

[Inc07] Crossbow Technology Inc. MTS/MDA Sensor Board Users Manual, Revision A. `http://www.memsic.com/userfiles/files/Datasheets/WSN/mts_mda_datasheet.pdf`, 2007. Last accessed: 2014-05-07.

[Inc14a] MEMSIC Inc. MICAz Wireless Measurement System. `www.memsic.com/userfiles/files/Datasheets/WSN/micaz_datasheet-t.pdf`, 2014. Last accessed: 2014-05-07.

[Inc14b] MEMSIC Inc. MTS/MDA Sensor, Data Acquisition Boards. `www.memsic.com/userfiles/files/Datasheets/WSN/micaz_datasheet-t.pdf`, 2014. Last accessed: 2014-01-22.

[ISO10a] ISO/IEC. *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Evaluation module for recoverability*, 2010.

[ISO10b] ISO/IEC. *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*, 2010.

[KI04] Bhaskar Krishnamachari and Sitharama Iyengar. Distributed Bayesian Algorithms for Fault-Tolerant Event Region Detection in Wireless Sensor Networks. *IEEE Transactions on Computers*, 53(3):241–250, 2004.

[KPSV02] Farinaz Koushanfar, Miodrag Potkonjak, and Alberto Sangiovanni-Vincentell. Fault Tolerance Techniques for Wireless Ad Hoc Sensor Networks. In *Sensors 2002, Proceedings of IEEE*, volume 2, pages 1491–1496, June 2002.

[KPSV03] Farinaz Koushanfar, Miodrag Potkonjak, and Alberto Sangiovanni-Vincentelli. On-line Fault Detection of Sensor Measurements. In *IEEE Sensors*, volume 2, pages 974 – 979, 2003.

[KS08] Karama Kanoun and Lisa Spainhower. *Dependability Benchmarking for Computer Systems*. Wiley & Sons, Inc, 2008.

[Lap08] Jean-Claude Laprie. From Dependability to Resilience. In *38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks*, 2008.

[LAV+10] Yingshu Li, Chunyu Ai, Chinh T. Vu, Yi Pan, and Raheem Beyah. Delay-Bounded and Energy-Efficient Composite Event Monitoring in Heterogeneous Wireless Sensor Networks. *Parallel and Distributed Systems, IEEE Transactions on*, 21(9):1373–1385, 2010.

[LBV06] Koen Langendoen, Aline Baggio, and Otto Visser. Murphy Loves Potatoes: Experiences from a Pilot Sensor Network Deployment in Precision Agriculture. In *International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, 2006.

[LCZ09] Steven Lai, Jiannong Cao, and Yuan Zheng. PSWare: A Publish / Subscribe Middleware Supporting Composite Event in Wireless Sensor Network. In *IEEE International Conference on Pervasive Computing and Communications*, pages 1–6, 2009.

[LYC09] Myeong-Hyeon Lee, Sung-Jib Yim, and Yoon-Hwa Choi. Grid-based Fault-Tolerant Event Detection in Wireless Wensor Networks. In *TENCON 2009 - 2009 IEEE Region 10 Conference*, pages 1–5, 2009.

[Mai10] Philipp Maierl. SeNetSim - Simulator für Dependability-Betrachtungen für Event-Detection in Sensornetzwerken. Bachelor's Thesis, Institute for Technical Informatics, Graz University of Technology, 2010.

[MCC12] Catello Di Martino, Marcello Cinque, and Domenico Cotroneo. Automated Generation of Performance and Dependability Models for the Assessment of Wireless Sensor Networks. *Computers, IEEE Transactions on*, 61(6):870–884, 2012.

[MCP+02] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John 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. ACM Press, 2002.

[MMA05] H. Morikawa M. Minami, T. Morito and T. Aoyama. Solar Biscuit: A Battery-less Wireless Sensor Network System for Environmental Monitoring Applications. In *Proceedings of the 2nd International Workshop on Networked Sensing Systems*, pages 51–56, 2005.

[MR13] Dorothy N. Monekosso and Paolo Remagnino. Data reconciliation in a smart home sensor network. *Expert Systems with Applications*, 40(8):3248 – 3255, 2013.

[MS06] Fernando Martincic and Loren Schwiebert. Distributed Event Detection in Sensor Networks. In *International Conference on Systems and Networks Communications*, 2006.

[PWC+11] W.-B. Pottner, L. Wolf, J. Cecilio, P. Furtado, R. Silva, J.S. Silva, A. Santos, P. Gil, A. Cardoso, Z. Zinonos, J. do O, B. McCarthy, J. Brown, U. Roedig, T. O'Donovan, C. J. Sreenan, Z. He, T. Voigt, and A. Jugel. WSN Evaluation in Industrial Environments First Results and lessons learned. In *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*, pages 1–8, 2011.

[RCK+05] Nithya Ramanathan, Kevin Chang, Rahul Kapur, Lewis Girod, Eddie Kohler, and Deborah Estrin. Sympathy for the Sensor Network Debugger. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, SenSys '05, pages 255–267. ACM, 2005.

[SCL08] Byunghun Song, Haksoo Choi, and Hyung Su Lee. Surveillance Tracking System Using Passive Infrared Motion Sensors in Wireless Sensor Network. In *International Conference on Information Networking*, pages 1–5, January 2008.

[SM14] Samrat Sarkar and Koushik Majumder. A Survey on Power Aware Routing Protocols for Mobile Ad-Hoc Network. In *Proceedings of the International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2013*, volume 247 of *Advances in Intelligent Systems and Computing*, pages 313–320. Springer International Publishing, 2014.

[Som12] Ian Sommerville. *Software Engineering*. Pearson Studium, Ninth edition, 2012.

[SPMC04] Robert Szewczyk, Joseph Polastre, Alan Mainwaring, and David Culler. Lessons From A Sensor Network Expedition. In *European Conference on Wireless Sensor Networks (EWSN)*, pages 307–322, 2004.

[SS92] Daniel P. Siewiorek and Robert S. Swarz. *Reliable computer systems - design and evaluation (2. ed.)*. Digital Press, 1992.

[Tan96] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 3rd edition, 1996.

[Top14] Top500.org. About The Linpack Benchmark. `http://www.top500.org/project/linpack/`, 2014. Last accessed: 2014-01-15.

[Tur86] Jonathan S. Turner. New directions in communications (or which way to the information age?). *IEEE Communications Magazine*, 24(10):8–15, October 1986.

[WBL12] Matthias Woehrle, Martin Bor, and Koen Langendoen. 868 MHz: A noiseless environment, but no free lunch for protocol design. In *2012 Ninth International Conference on Networked Sensing Systems (INSS)*, pages 1–8, 2012.

[YC09] Sung-Jib Yim and Yoon-Hwa Choi. Fault-Tolerant Event Detection Using Two Thresholds in Wireless Sensor Networks. In *15th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 331 – 335, November 2009.

[ZMH10] Yang Zhang, Nirvana Meratnia, and Paul Havinga. Outlier Detection Techniques for Wireless Sensor Networks: A Survey. *Communications Surveys Tutorials, IEEE*, 12(2):159–170, April 2010.

[ZR07] Michael Zoumboulakis and George Roussos. Escalation: Complex Event Detection in Wireless Sensor Networks. In *Smart Sensing and Context*, volume 4793 of *Lecture Notes in Computer Science*, pages 270–285. Springer Berlin Heidelberg, 2007.

[ZR09] Michael Zoumboulakis and George Roussos. Efficient Pattern Detection in Extremely Resource-Constrained Devices. In *Sensor, Mesh and*

*Ad Hoc Communications and Networks, SECON '09. 6th Annual IEEE Communications Society Conference on*, pages 1–9, 2009.