Towards a Fault-Tolerant Wireless Sensor Network Using Fault Injection Mechanisms: A Parking Lot Monitoring Case

Golnaz Karbaschi, Francoise Sailhan, Stephane Rovedakis

Towards a Fault-Tolerant Wireless Sensor Network Using Fault Injection Mechanisms: a Parking Monitoring Case

Golnaz Karbaschi
SmartGrains SAS, Paris-FR
golnaz.karbaschi@smartgrains.com

Francoise Sailhan
Cedric Laboratory
CNAM, paris-FR
Francoise.sailhan@cnam.fr

Stephane Rovedakis
Cedric Laboratory
CNAM, paris-FR
stephane.rovedakis@cnam.fr

***Abstract*—A Wireless Sensor Network (WSN) requires a high level of robust and fault tolerant sensing and actuating capabilities, specially when the application aims to gather delicate and urgent data with reasonable latency. Hence, verifying the behavior properties under the presence of faults remains an important step in developing an application over a WSN. A comprehensive study on characterization and understanding of all the possible faults is required in order to generate and inject 'any' known error to the system. In order to ensure appearance of all the faults and possible bugs in the system, conception and developing a fault injector which generates and injects any requested fault to the system is promising. This becomes more important and critical when the fault happens very rarely, while due to Murphy's law it happens certainly along the network life. Considering that occurrence of faults depends heavily on the specifications of the use case, in this paper we concentrate on a sensor network which aims to detect the presence of vehicles on parking lots. We try to categorize and characterize the faults driven by this system as the first step of developing a fault injector[1].**

***Keywords*-Robustness; Wireless Sensor network; Fault injection; Monitoring a Parking;**

I. Introduction

Wireless sensor networks have been founded in order to sense and monitor physical phenomena in a distributed manner and disseminate the useful information via a wireless multi hop network. The explosion in the instrumentation of our environment is driving the need for developing sensor network that not only observes and measures the sensed data but also shows a high fault-tolerance behavior to harsh and hazardous environment around the network [2], [6].

Meeting the challenging task of developing dependable wireless sensor network (WSN) necessitates deploying a wide range of strategies including self-configuring, self-healing, defensive designing and development. Beside such tactical approach for developing a dependable WSN is putting in a concrete form through rigorous testing, evaluation and validation. Toward this goal, we present the foundation of a fault injection based evaluator, as a pragmatic approach of test, that adequately accelerates the occurrence of faults so as to easily judge the quality of error handling and hence, more generally, evaluate and analyze the dependability of the sensor network. As shown in Figure 1, this solution is to be grounded upon three main building blocks which are imposed on the system under test as follows [1]:

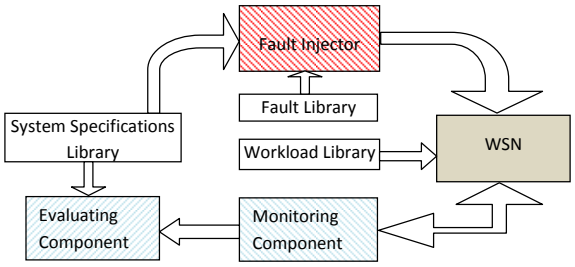

Figure 1. Fault injection and monitoring the system behavior

- A fault injection mechanism which inserts fault into one or several nodes.
- A monitoring component that looks for and reports misbehavior or dysfunction.
- An evaluation component which assesses whether correct behavior is being observed.

Therefore, a strict testing mechanism involves a two-steps-process. First, one should identify the failures that threaten a WSN. Second, grounded upon this understanding, one can deduct requirements and suggest an approach for evaluating dependability. As a first step upon this goal, we herein attempt to identify and characterise the faults that have been encountered through a WSN under the test. Identifying the possible faults starts by translating any deviation from system specifications into possible causing errors and finally providing an independent complete fault library. Alas, we should note that in general complete proving of sufficiency of the fault library is very difficult, if not impossible. In the following, we use the terms *fault* and *failure* defined as following. A *failure* in a system occurs when the service it delivers to the user deviates with the system specification for a period of time, while a *fault* is the cause of a *failure*.

Knowing the fact that the faults occurrence and its severity

[1]This research has been supported by Murphy ANR Project (http://cedric.cnam.fr/ sailhanf/murphy/).

on the network performance depends on the application case which is ran over the WSN, in this paper we have focused on a particular case study. The system under the test, named ParkSense[2], is a parking monitoring system which aims at detecting the presence of vehicles on parking lots by placing a sensor on each parking lot.

The reminder of this paper is organized as follows: We first describe briefly the concerned use case of WSN and the developed software and platform run over it. An investigation on characterization of faults that are imposed from interior and exterior of sensors comes hereafter. Finally we conclude the paper in the last section.

II. Large Scale Wireless Parking Monitoring

We have studied a particular use case of a wireless monitoring system which detects the presence of vehicles in a parking. The network contains several magnetic sensor nodes stuck on each parking lot which are able to gather sensory information, communicate with other connected nodes and propagates the data information via other nodes towards a server data base. An important aspect to take into consideration is the fact that one goal is to design a system which is able to work on an large scale parking lot. This implies the spread of a high number of sensors to monitor the vehicles' presence. Therefore, as in many sensor networks, it is of crucial importance to develop a high dependable system, in particular with no or few maintenance to carry out on sensors because of the impossibility to repair each out of order sensor. We give now a brief description of this use case.

A. ParkSense Topology and Infrastructure

The employed magnetic sensor detects the presence of metallic objects due to their impacts on changing earth's magnetic fields. Once the vehicle presence is confirmed, this sensory information is routed through a multi-hop path towards a plugged sink node, from which a real-time map of parking occupancy is established. For this purpose, sensors are organised into a tree structure where the root corresponds to the sink.

In the case of extended network to a large one, in order to avoid the presence of isolated subnetworks, the whole system is partitioned into sub-networks each is governed by its own sink. All the subnetworks are connected via a tree-shape backbone with one principal root.

B. ParkSense Software /Hardware

ParkSense sensor nodes encompasses a magnetometer sensor, a micro-controller, a memory, a wireless communication interface and a ZigBee Radio antenna. Figure 2 shows a simplified description of internal structure of a

[2]ParkSense is one of the solutions provided by SmartGrains [3] and is capable of providing a real-time overview of parking space occupancy in a given zone, indoors or outdoors, in car parks and on the street.

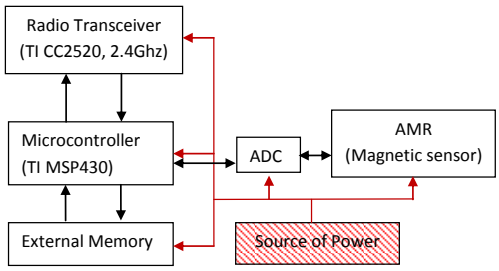

Figure 2. ParkSense Sensor Mote Structure

ParkSense mote. As shown in this figure, each sensor can have two external inputs: One, the sensed information via the magnetic sensor. Second, the received radio frequency (RF) message from the radio antenna. A battery source powers all the component of the mote. The network protocol stack is managed by a micro-controller (using the external memory) where a simple operating system (FreeRTOS) schedules the different tasks. The MAC and physical layer operates on ZigBee principals at the frequency of 2.4 Ghz [5].

ParkSense benefits from energy investment via synchronizing all the nodes in the system. Thanks to a precise unique timing among the nodes, each sensor that operates on battery can optimize its power consumption. Indeed it spends most of the time snoozing while it knows the exact moment of wake up and operating. As Synchronization must be maintained through the network, some frames called synchronization beacons must be sent periodically from a Master to its Slaves in order to ensure that the clock drift between them will not overtake a given value. Therefore, sensors operate in cyclic mode: sleep then receive mode. Duty cycle is programmable and defines access time to each sensor.

III. Faults Description

We classify the faults that are encountered or expected to happen, due to their respective origins, into two categories: external and internal origin, which are described below. To describe more precisely the faults which can be arisen in the case study, it is crucial to also characterize the impact of any fault on the system. To this end, we can describe a fault following three characteristics: its periodicity, its gravity and its extent. First, for the periodicity, related to the frequency at which a fault arises in the system, we can distinguish different degrees: transient (if the fault is unpredictable and occurs rarely), sporadic (if it occurs frequently following a period of time) and permanent (if a node is no more available in the network because of the fault). Second, the gravity of a fault informs on its consequences during runtime, which could be defined as: benign (if consequences are minor), severe (if consequences are problematic for the running of the system) and critical (if the system does no more provide

the expected service). Finally, the extent of a fault which is the ability of the fault to infect other nodes, following its severity: no contamination (if neighbouring nodes are not affected), local (if only neighbouring nodes can be affected) and overall (if the overall network can be affected by the fault). However, in the study of contagion of each fault, it should be considered that since the network has a tree-based structure and all the nodes rely on each other for connecting to the main backbone, a node's break down can disturb all the neighboring nodes and so the whole network. Loosing a node becomes more risky in the case of a sparse topology where there are not many alternative nodes instead of the ones already dead. Hence, a local fault can be easily contaminated to others.

It should be added that characteristics like periodicity, gravity and extension of faults can be used as important parameters to give priority to implying the faults and conceiving an intelligent fault injector. Indeed at the conception of a fault injector, a permanent critical fault that can affect the whole network should not have the same place as a rare benign fault with no contamination[3].

We present now the faults related to the studied use case due to their origins.

A. External Origin

We call any fault with the origin from exterior of sensors as an external fault. In general these faults are mostly imposed by nature phenomena or they are human related faults.

Nature related faults: One of the phenomena that can potentially impact the system performance is ambient temperature variations. Indeed the oscillation frequency of the quartz crystal that is embedded on each sensor leads to a quartz-crystal-based clock drift over the time. This quartz frequency, like all the other electrical components, has a typical working temperature interval in which it works linearly and has a predictable functionality. The clock drifts is highly temperature dependent and should be tuned regularly by environment variations. This problem is more sever in outdoor parkings where some nodes are potentially exposed to direct sun light while the rest are shaded. This leads to different on-board clocks for the sensors. In precisely synchronous networks such as ParkSense case, the drifts in quartz clock becomes more vital and draws more attention. This is because these networks (for the energy economy reasons) work based on having synchronized sensors that tune and lock their internal clock regularly by the network master clock. Any slide in clocks may lead to consume a lot of energy for the sensors in order to looking for the actual network global clock. Once they are relocked they follow again the sleep/receive mode cycle. Therefore desynchronization may strongly reduces the network life time. Drift in quartz is a fault arisen with a transient periodicity and it can have severe consequences on the system runtime affecting potentially other nodes of the network.

Moreover, high temperature also affects the antenna gain, as it is seen in one of the parking equipped by ParkSense sensors and confirmed by antenna's data sheet [4]. Adapting the antenna setting will reduce the variation in output power over temperature. This fault has the same characteristics as drift in quartz cock fault, that is a transient periodicity, severe consequences on the system runtime, but no contamination risks for other nodes of the network.

Another disturbing external fault is the nearby electrical and electromagnetic fields around the system under the test. Electromagnetic sensors are mostly subject to the advert effect of this parameter. Indeed electrical and electromagnetic radiations set up electromagnetic noises and so interferences that are characterised by a natural (such as atmospheric or cosmic radiation) or artificial origin (such as human technologies implanted for communication or power transmission). As observed in a parking, most of the time artificial interferences superpose to natural noises, possibly leading to excessive interference/noise together sensed by the magnetic sensors and potentially interpreted (depending on the pulse shape in the time and spatial domain) as a change in the occupancy of the parking space. Depending on the degree of severity of the interference/noise, the accuracy of the gathered information is hence potentially compromised. More specifically, transient electromagnetic interferences which lead to a flip-flop effect (i.e. free-occupied message) are critical because any change of occupancy lead to an emission of a notification and hence to an increase in traffic/energy-consuming and reduce in bandwidth availability and sensor life time.

Noises and interferences that occur across the radio frequency spectrum affect the speed, accuracy and range of communication. The difficulty comes at a wireless sensor that is unable to distinguish a desired signal due to the concomitant reception of a (strong) spurious signal which comes from a device operating within the shared band/channel or closely spaced frequency band/channel. Electrical or electromagnetic interferences noises and interferences may have several reasons including presence of reflecting obstacles, a (possibly malicious) device that operates within or around the frequency/channel of the system under the test. In practice, they lead to message corruption, lost, and deletion. Electromagnetic noise and interferences can occur in a transient or sporadic manner. They could have a severe impact on the system (as described above, flip-flop effect, message lost, high traffic/energy consumption, etc.) and this could affect locally or overall the network. Some other disturbing factors like the mutipath effect, variable quality of wireless links and fading issues can also imposed to the network by nature.

Human related faults: Some external faults are imposed

[3]Note that determining the priority of faults to be included in fault injector is done by the owner of the system under test.

to the system by active DoS attacks which attempt to make the system to a down level without available resources to operate normally. Knowing that for providing a fully robustness and resilient operation for the system, one should also take these kinds of faults into account. Therefore, external attack-like faults have to be included in the fault library in order to inject them to the system and immunize it, as much as possible. Attack-like faults have a transient or sporadic periodicity whose impact on the system can be severe or critical depending of the amount of system resource harvesting. Moreover, it may lead also to a local or overall system contamination.

B. Internal Origin

The faults coming from the sensor's interior can be due to hardware or software deficiency.

Hardware faults: The use of off-the-shelf components aggravates the vulnerabilities of sensors. The faulty manufacturing and assembling of electronic components leads to instabilities and so to erroneous data stored in memory (memory corruptions) or erroneous computations (e.g., faulty sensing values). Furthermore, some other hardware faults such as not-correctly assembling a component, faulty electronic design, shorts or opens in electrical circuits, Bus error, etc are included in the library of hardware faults. These kinds of faults are permanent internal faults which involve severe or critical gravity. However, the presence of these faults could not infect other nodes of the system by its nature.

Battery failure resulting from natural depletion causes a hardware supply failure on the sensor. Indeed battery life time is not linear (energy level starts with high and constant level and falls drastically at the near end of life of the battery). Thus, based on the energy level, one cannot foresee the battery failure. Once detected, the brutal fall of energy level cannot be notified because of insufficient remaining energy to send a message. Battery failure at a node could drive it to a breakdown state. This permanent fault may remain non-contagious but it could have severe or critical consequences on the service provided by the system, since some parts of the network could not respond any more.

Software faults: Memory faults are the common software faults that are encountered in embedded systems. Faults such as memory leaks, illegal memory (de)allocation, buffer overflow, Null pointer references, reading a non-initialized memory and memory (code) corruption are among the most important software bugs. Besides, any bug in the code of the protocol stack software or operating system (OS), wrong interrupt from other units or Watch Dog Timer may cause a deviation from system specifications. Memory faults while may be hidden from the neighboring nodes, may appear in a transient manner and can have a severe or critical impact on the system.

Another possible software faults occurrence lies on configuration and network setting parameters. Among these parameters we can point to refreshing rate of tables which keep the status of neighboring area. All the nodes in a multi-hop sensor network have to store some information on their immediate neighbors in order to keep connected to the whole network and to monitor their local area. This information is stored in a table, generally called a *neighboring table*. A wireless network may have potential environmental and/or topology variations. In order to follow the changes, having an efficient routing decisions, preventing the loops, not forwarding a message to an isolated node, choosing the best clock master, etc. the neighboring abstract should be reactive and updated enough [4]. Having always the fresh tables and updating the stale values requires a minimum refreshing rate to be able to follow the vital variations around. Hence, the periodic time to update the neighboring table is an important parameter of the network that should be chosen carefully. Generally system configuration faults can have a permanent non-contagious impact and can greatly reduce the network performance.

IV. Conclusion

Sensor networks are used in a growing number of applications. Usually these networks are deployed in uncertain environments and are subject to several faults, in part due to environmental conditions or due to resource constraints of embedded systems. These faults generate the errors in the system which leads to mismatching to system specifications and failures. Although with the sensor networks the key constraint is resource saving, but additional requirements relating to robustness and fault resilient have been arisen by new smart applications. In this paper, a particular use case of a large-scale parking lot wireless monitoring is considered. A brief description from topology and infrastructure as well as some particularities in hardware and software allows to any similar embedded system to benefit from the achieved fault library to include them to its own fault injector.

Besides of periodicity, gravity and duration of faults which are the important distinctions allowing to develop an intelligent fault injector, the faults can be divided regarding to their origins in relation with the system: faults with external origins or internal origins. This characterization is not exhaustive but it describes the crucial observed faults-failures during the system development.

External faults have a vast range and are generally imposed by the natural phenomena or active attacks. Faults such as high environmental temperature and its impact on embedded quartz and RF antenna, interferences on magnetic fields and radio transmissions are included in this category. Failures caused by these faults contains erroneous sensed

[4]Depending on the design, routing information may be stored in a separate table called *routing table* that similarly needs to be refreshed frequently.

values, loosing clock synchronization, message collisions, link failures, etc. These symptoms may lead to deviation from system specifications, high resource consumption and short network life. Internal fault may happen on software or hardware of the sensors. Any faults in electrical circuits and components and battery supply, etc can refer to hardware faults. Software faults commonly lie on access or manipulation to the memory, algorithm faults, wrong network configuration settings, etc.

V. Future Work

Studying on the influential faults that cause the system failures, is the first step towards designing an injector of fault which can accelerate their occurrence. This allows to improve the dependability of the system and to immunize it by using mechanisms which allow to be more resistant to these faults. Despite careful design and concerns about the possible faults before deployment, our experiment shows that there still may be some unexpected faults that show up after a long run time and can be added to the fault library at that time. It is also confirmed in [7]. Therefore, there may be a closed loop between the fault library and the evaluating component in which evaluating the system always enriches the fault library.

We believe that the next remaining issue is to know which fault to be injected and where, when and how to inject them. As mentioned before, not all the faults have the same importance to be considered as the first crucial faults. Besides, some related work on designing a fault injector ([9], [10] and [8]) show that for having a full coverage of considering most of the faults, injection should be applied on both hardware and software components[6].

Testing the system aims at making a robust network which works well at almost all the local conditions such as heavy rain or intense cold and in sever tests such as a heavy traffic load. Therefore, a fault injector have to do a stress testing on the system by injecting some intense external faults to show the limits of the system and to remove the possible failures as much as possible.

Most of the hardware related faults are intended to be checked before a real deployment. The first step for testing the system functionality is deploying a *home network* using simple single-hop routing protocol with a quite large number of nodes. This deployment allows to avoid the added complexity of multi hopping and to test the hardware and the major software functions such as creating neighbor tables, sending the sensory information, etc. The corresponding faults to check the sensors' basic functionality can be injected at this stage. In addition, the stress testing (environmental or human related), verifying the memory and injecting any internal software faults can be executed. To this aim a hard-coded fault injector module can be envisaged on the chips. Triggering this module can be done by different approaches (ex. a wireless command, a time-out event, etc.). The next step is deployment in a real site. This allows to test the system in actual situations which most of the time lead to face unexpected problems. Magnetic sensors particularly sense different fields in indoor/ourdoor parkings. Once confident in hardware and software, the networking and multi-hopping issues can be verified.

Acknowledgments

This work is supported by the French National Agency (ANR) under contract ANRBLAN-SIMI10-LS-100618-6-01.

References

[1] F. Sailhan, T. Delot, A. Pathak, A. Puech and M. Roy, *On Providing Fault Injection for Assessing the Dependability of Wireless Sensor-Actuators Network*, Submited to INFORSID GEDSIP Workshop, 2010.

[2] M. C. Hsueh, T. K. Tsai, and R. K. Iyer, *Fault Injection Techniques and Tools*, IEEE Computer, vol. 30, no. 4, April 1997, pp. 75-82.

[3] SmartGrains Company, *www.smartgrains.com*.

[4] Texas Instrument CC2520 Datasheet, *2.4 GhZ IEEE 802.15.4/ ZigBee RF Tranceiver*.

[5] ZigBee Alliance, *http://www.caba.org/standard/zigbee.html*.

[6] A. Benso, P. Prinetto, *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*, Frontiers in Electronic Testing, No. 23, 245p., Kluwer Academic Publishers, London, UK, 2003.

[7] G. Barrenetxea, F. Ingelrest, G. Schaefer and M. Vetterli, *The hitchhiker's guide to successful wireless sensor network deployments*, in Proceedings of the 6th ACM conference on Embedded network sensor systems, pp. 43-56, 2008.

[8] J. Carreira, H. Madeira and J.G. Silva, Xception: Software Fault injection and Minitoring in Processor Functional Units, in Proc. IEEE Int'l Working Conference Dependable Cmputing for Critical Applications, PP.135-149. 1995.

[9] J. Aidemark, J. Vinter, P. Folkesson, J. Karlsson, *GOOFI: Generic Object-Oriented Fault Injection Tool*, IEEE Int. Conf. on Dependable Systems and Networks, Gteborg, Sweden 2001, pp. 71-76

[10] R.M. Lefever,M. Cukier and W.H Sanders, *Loki: a state-driven fault injector for distributed systems*, In Proc. International Conference on Dependable Systems and Networks, 2000, PP.37-242.