Integrated Fault Tolerance Framework for Wireless Sensor Networks

Dima Hamdan [a,b], Oum-El-Kheir Aktouf[b], Ioannis Parissis[b], Bachar El Hassan[a], Abbas Hijazi[a]

a. LASTRE Laboratory, Lebanese University
Tripoli, Lebanon

b. LCIS Laboratory, INP- Grenoble University
Valence, France

Abstract— Safety critical applications, such as fire detection or burglar alarm systems, require continuous and reliable operation of Wireless sensor networks (WSNs). However, validating that a WSN system will function correctly at run time is a hard problem. This is due to the numerous faults one may encounter in the resource-constrained nature of sensor platforms together with the unreliability of the wireless links networks. A holistic fault tolerance approach that addresses all fault issues does not exist. Existing fault tolerance work most likely misses some potential causes of system failures. In this paper, we propose an integrated fault tolerance framework (IFTF) that provides a complete picture of the system health with possibility to zoom in on the fault reasons of abnormal phenomena. IFTF diagnoses network failures, detects application level failures, identifies affected areas of the network and may determine the root causes of application malfunctioning. These goals are achieved efficiently through combining a network diagnosis service (component/element level monitoring) with an application testing service (system level monitoring). Thanks to these two complementary services, the maintenance operations will be more efficient leading to a more dependable WSN. From the design view, IFTF offers to the application many tunable parameters that make it suitable for various application needs. Simulation results show that the presented solution is efficient both in terms of memory use and power consumption. IFTF incurs a 4 %, on average, increase in power consumption (communication overhead) compared to using solely network diagnosis solutions.

Keywords: *wireless sensor networks; fault tolerance; diagnosis; test, semantic application level failures; node failures; link failures*

I. INTRODUCTION

Our work focuses on the issue of fault tolerance in Wireless Sensor Networks (WSNs). A WSN consists of spatially distributed autonomous *sensors* to monitor physical or environmental conditions, such as temperature or sound, and to cooperatively pass their data through the network to a main location. Sensor nodes are resource-constrained with limited processing power, limited memory and restricted energy. *Applications* over WSNs are growing quickly. However these emerging applications for WSNs do not limit themselves to a single function called *"sense and send"* with trivial local data processing tasks, but also include more complex collaborative decision-making applications [1], including aerospace, automation, medical monitoring, natural event monitoring, object and behavior tracking, monitoring product quality, combat field reconnaissance, military and mission-critical systems. While WSNs have significant potential in a range of varied applications, they also pose many challenges in terms of fault tolerance.

A. Problem Statement

Fault Tolerance in WSN is a crucial feature due to the diverse types of faults that may occur. The important question here is: which elements or which system inconsistencies must be considered by the diagnosis and debugging tools? The overall diagnosis for each individual system component is not possible due to the high energy consumption that will be required. Thus, diagnosing and fixing efficiently a few selected set of fault types can improve the system dependability. However, it couldn't address four major issues that will challenge the reliability of a system at run-time: (1) The positive assessments from diagnosis tools doesn't assure the enforcement of application semantics, (2) The negative assessments from diagnosis tools doesn't translate directly to the collapse of higher level applications due to the fault tolerance nature of the system. For example, an individual node failure may not necessarily poke a reliability hole in higher level applications, (3) The environmental changes, the

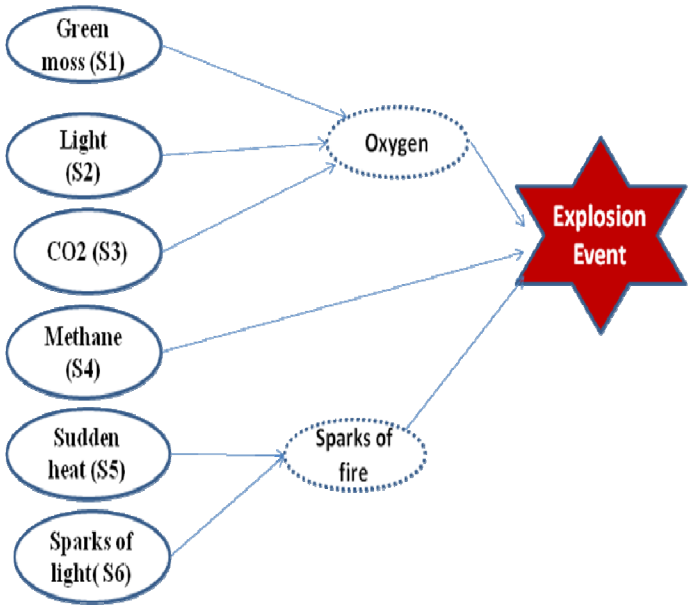

Figure 1: Coal mine explosion WSN application

978-1-4673-0747-5/12/$31.00 ©2012 IEEE

upgrades, the reprogramming and other conditions changing may lead to unexpected system failures at run-time, which will ultimately jeopardize the reliability of the system, (4) The sensor nodes can lose synchronization and their applications can easily reach arbitrary states due to the improper system configuration that lead to the collapse of certain design assumptions at run-time.

B. Fault types in WSN

Failures in wireless sensor networks can occur for various reasons. Sensor nodes are fragile, and they may fail due to depletion of batteries or destruction by an external event. In addition, nodes may capture and communicate incorrect readings because of environmental influence on their sensing components. Links are also failure-prone, causing network partitions and dynamic changes in network topology, leading to delays in both data and communications. Links may fail when permanently or temporarily blocked by an external [2] or environmental condition. Packets may be corrupted due to the erroneous nature of communication. When nodes are embedded or carried by mobile objects, nodes can be taken out of the range of communication. WSNs are also prone to malicious attacks, such as denial of service, injection of faulty packets, leading to unexpected behavior of the system and so on. Other than these predefined faults or failures (i.e., with known types and symptoms), many times the sensor networks exhibits silent failures [3] that are unknown beforehand and highly system-related. For example, the weak correlation between radio-on time and number of transmitted packets of a node suggests inconsistency on that node.

In our approach, we address two classes of faults. First, faults which lead to node and link failures (predefined faults). Second, faults which lead to system dysfunctionalties or application level failures. In this way, our approach allows to (1) diagnose network faults which are likely to happen in WSN deployments; (2) quickly assess the impact of faults on the whole system behavior; (3) to improve the fault detection rate by detecting some hidden causes of faults (silent or predefined faults); and (4) to validate the application after code upgrades or any changing to the operating conditions.

C. Related Works

Most fault tolerance solutions fall into two categories: *testing* of WSN applications and fault diagnosis of low-level system components. Testing is an important means to verify the correctness of an application. Debugging WSN applications is a complicated process and many different approaches exist. T-Check [4] finds bugs by exploring program states extensively via simulations. Marionette [5] is source-level debugger allowing access to source-level symbols. Sentomist [6] uses data mining techniques and relies on statistics and learned good behavior for outlier detection. However, all of these debugging mechanisms are either used prior to the deployment or in a post mortem manner, where data about the application is collected and then analyzed offline and where the size of the test suite is not as critical. Therefore, these techniques do not provide a way to monitor the application behavior at run time. Application level testing, which assesses the whole system end-to-end, receives very little attention. [7, 8] propose a testing methodology of WSN application behavior at run time. These solutions provide a positive affirmation of correct application-level operation. However, they don't identify the reasons of the system abnormal behavior. In contrast, Network diagnosis approaches are important to identify the network, node and data anomalies at run time. With Sympathy [9], a set of metrics including node connectivity and data flows are transmitted periodically to a central node to identify the fault reasons of packet loss in data collection applications. PAD [10] employs a packet marking algorithm that constructs and maintains a probabilistic inference model to infer the root causes of some abnormal phenomena. Some approaches [11-13] rely on node collaboration on the basis of periodic heartbeats in order to detect some deviations from expected behavior. On the basis on our literature survey of related research work, a framework that addresses all fault issues one may encounter in a WSN doesn't exist. Thus, our approach aims at providing a complete picture of the system health with possibility to get detailed reports about low level system components faults.

The remainder of this paper is organized as follows. Section II presents a motivating application scenario. Section III presents IFTF architecture. Section IV gives some fault scenarios. Section V evaluates the proposed design. Section VI concludes the paper.

II. AN APPLICATION SCENARIO

A. Needs for a WSN application for explosion prediction

Let us consider an underground coal mine that reports from time to time that some explosions happen. Through inspections, the major cause of these explosions is due to mixing enough amount of oxygen with methane in presence of a tiny spark of fire (harsh environment). Normally, the mine can generate plenty of gases, such as methane and CO2, and a negligible amount of oxygen. But due to some green moss growing in the mine, a photosynthesis process can produce oxygen when the sunshine gets through the holes on top of the mine. The danger of this situation can be reduced by setting up a wireless sensor network to give an alert enough before an explosion occurs. The system must not use oxygen sensors directly because if it can detect O2, it might be already too late. Instead, the system must detect the photosynthesis process to have enough time to take the necessary precaution steps. In such an application, six types of detection sensors are required (Figure 1): the presence of green moss (S1), the sunshine (S2), the CO2 gas (S3), the high concentration of methane gas (S4), the sudden generated heat (S5) and the sparks of light (S6). The photosynthesis process is detected through the sensors S1, S2 and S3 and the spark of light through the sensors S5 and S6. The explosion can then be detected when the three events take place: the photosynthesis process, the high concentration of methane and the spark of light.

B. Problems after system deployment

The deployment of WSN saves many times the life of miners and avoids many potential damages in the mine and the nearby villages. However, the complex fault scenarios of wireless sensor networks (WSN) lead to miss reporting some real explosions and other times to generate false alarms explosion which make the rescue squad uselessly dispatched to the mine after every report of explosion events.

C. Need for fault tolerance solution with IFTF features

Hence, an efficient diagnosis tool is required to maintain the network performance. Infield investigations conduct to locate the most frequent failure reasons which are node failures due to energy depletion and link problems. The diagnosis tool contributes to alleviate the problem, but does not guarantee the correct functionalities of the system at runtime. Consider the case where the network reports of an oxygen event without the detection of green moss. From the application semantics, an oxygen event can only happen if there has been a green moss event as one of its necessary triggers. Or the case, where the network reports the photosynthesis process during night. From the application semantics, the photosynthesis process (which triggers the O2 event detection) must only happen during daytime. In addition the system may generate false alarms even though the diagnosis tool shows positive assessments for the node and link states. Thus, it is required to identify efficiently other causes of failures that highly affect the correct application functioning. Hence, an application testing tool is introduced to complement the diagnosis tool functionalities. This requirement is common to many collaborative decision making applications dealing with the complex fault scenarios of wireless sensor networks. Thanks to these two complementary and orthogonal tools, the maintenance operations will be more efficient leading to a more dependable WSN system. This is the basic idea of our solution that is called IFTF (Integrated Fault Tolerance Framework).

III. IFTF Architecture

IFTF system is built around the following three components: the IFTF manager, the testing subservice and the diagnosis subservice (Figure 2).

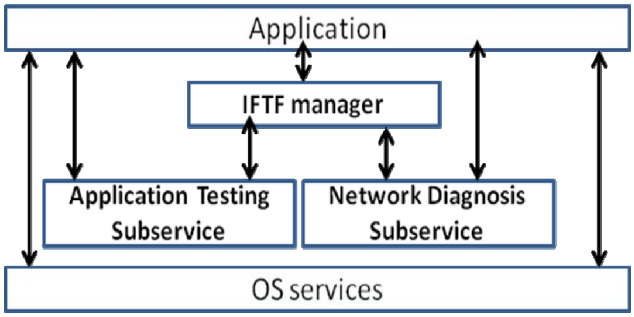

Figure 2: IFTF Software Architecture

A. IFTF manager

IFTF manager takes control of triggering the two subservices according to a hard-coded schedule, on demand of the administrator or based on the result of the underlying subservice. In addition, this component ensures the synchronization between nodes to perform the tests at the same time.

Interfaces**:** IFTF manager provides interfaces to interact with the application and the two subservices. First, a configuration command is provided so that the application can determine and adjust the subservices schedules anytime. Second, an interface is provided to get the current node status, e.g. "Normal operation" and "test mode".

B. Application testing Subservice

The testing subservice verifies the ability of the system to perform correctly its functionalities. The IFTF approach is based on functional testing. Motivation for this type of test comes from considering the sensor as a black box component. Services are tested by feeding the nodes with test inputs and examining the outputs to compare them to the expected ones. In this case, the verification of the correct behavior of the application doesn't consider the internal application structure. Tests can include different kinds of computations, complex collaborative decision making, message passing, interaction between nodes and other distributed operations required to produce output values and actions, and which can be compared to the expected outputs. They can be end to end tests, which validate whether the flow of application from the starting point (nodes) to the end point (the gateway) is happening as required. They can also be used to verify a specific part of the distributed application logic (partial tests). Therefore, the application contains the test suite and emulates the nodes responses or notification to neighboring nodes or gateway application, e.g., a success or error condition. When the test session starts, the testing polls its virtual readings and marks the application messages. When normal operation resumes after the session, it ensures that any parameters relating to the node are restored.

Interfaces: the testing subservice uses a clearly defined interface to interact with the application and the framework manager.

- Configuration command is provided to the application layer to determine and to adjust the service parameters (test cases, test mode, urgent reading, lowest valid reading, and highest valid reading). The first three parameters are application specific and the second two parameters are sensor specific but are influenced by the deployment. The test mode can be blocking mode, switching mode or abandonment of the test in case of urgent reading.
- Start command is provided to the IFTF manager to trigger a particular test. It sets the node status to the test mode. By default, it triggers the "end to end" test.
- Stop command is provided to the IFTF manager to cancel a specific test. It changes the node status to the normal mode operation.
- Read command which is a call to get a new reading to the application. Depending on the current status

node, the application receives back the real or the virtual reading. In case of normal mode operation, the application receives a real reading with a validity field. The validity field indicates whether the reading is valid or not and it is up to the application to filter messages based on its own policy. For example, the application may wait for several erroneous readings before alerting the server or it may alert the server on the first erroneous readings and send any more until the error is fixed. This field allows a significant decrease in network overhead.

- Send command is provided to the application to send its output values in a network packet. Depending on the current node status, it can be an application generated or a test packet.
- Receive command lets the application receiving the test and effective packets transparently.
- Report is a provided interface to notify the application about the test result.

Testing subservice components

- Input Handler component: the input handler component manages the inputs directed (provided) to the application. Depending on the current state mode and the test mode, the input handler polls either the physical sensors (sensors drivers) or redirects the request to the test data or virtual sensors. These virtual sensors have a buffer to store test readings and can also be sampled. This component can also deal with several different virtual sensors, e.g. temperature and green moss presence. Depending on the test mode selected by the application, when a node starts the test checks, this component can stop the sampling from real sensors, or continue sampling the real sensors even in test mode or abandon test in case of sensing of an urgent reading, e.g. the high methane concentration.
- Communications handler component: the communication handler tags the application messages with current state node. This allows distinguishing the real packets from the test packets. Also, it allows verifying the state of the neighboring nodes when checking the tag on receiving the application messages.
- Test checker component: this component verifies the test output against the expected output.

C. Network Diagnosis subservice

Our diagnosis subservice targets two failures that are likely to happen in WSN deployments which are the node failure due to energy depletion, and the link failure due to poor connectivity with neighbors.

Failure detection phases: Our subservice leverages the insights gained from existing distributed approaches [11-13] and uses a similar two-phase detection algorithm: the local detection phase and the consensus phase.

- In the local detection phase, each node monitors its neighbors (or a specific number of neighbors) and waits for a certain period of time for a "heartbeat" message. To avoid the impact of transient failures [13], the period must be large enough to allow transmitting n1 messages from each selected neighbor. Thus, only when n1 messages are missed from a particular node, will that neighbor node be considered suspicious.
- In the consensus phase, the monitoring node corroborates its findings with other neighbors. This phase is particularly important to decrease the false alarm rates. Again, the period of the second phase must be large enough to allow exchanging n2 messages before deciding the final verdicts and sending them to the application layer.

Diagnose failure root cause: the root causes reported by the diagnosis subservice are: battery energy depletion, link failure, and unknown failure. Distinction among them is based on remaining energy information, and link quality indicators. The remaining energy is transmitted by each packet sent by a node. The link quality between two neighbor nodes is computed by a node upon receiving a packet from each of its neighbors. If the remaining energy is below a low battery threshold level, the first cause is the reported root cause. Similarly, if the link quality is below to a predefined link quality threshold, the link failure is the reported root cause.

Interfaces: the diagnosis subservice provides several well defined interfaces to interact with the application and the framework manager.

- Configuration command is provided to the application to determine and to adjust the service parameters (n1, n2, neighbors Number, low battery threshold, link quality threshold).
- Start command is provided to the IFTF manager to start the diagnosis communication.
- Stop command is provided to the IFTF manager to stop the communication.
- Report is provided to the application layer to get the report information.

IV. IFTF AND FAULT MINE SCENARIOS

To illustrate our approach, let us return to the mine explosion WSN application. The application provides several types of services (functionalities) such as the explosion prediction, the oxygen detection, the sparks of fire detection, the high concentration of methane detection, and so on. The main goal of the testing service is to allow us to verify that the application can provide its services as expected. In other words, it allows answering to questions such as: Does the explosion prediction service function correctly in the mine? And if not, what causes its malfunctioning? Does the oxygen detection service function correctly at the area X? What about the system performance and global system metrics, e.g. notification latency?

A. Scenario 1: Oxygen detection at area X at daytime

Service Description **:** A cluster head node notifies the gateway about the event O2 detection in case of receiving positive readings from the green moss sensor (S1), CO2 sensor (S2) and the sunshine (S3) sensor and while the time is daytime (as the photosynthesis process happens during daytime only) (Figure 3) . Now, let us see which types of faults that each subservice can detect in each of the following case
s:

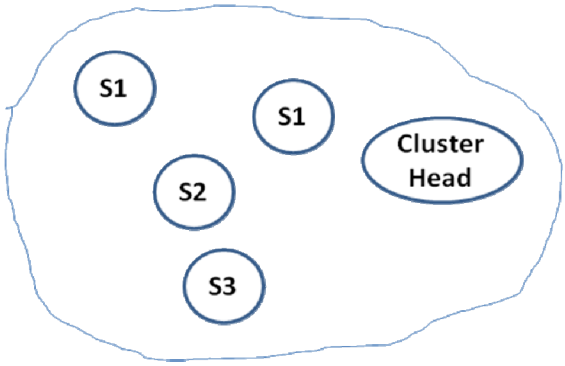

Figure 3: Area X topology

- Case 1 : Node or sensor crashes
  If S1 crashes the diagnosis subservice detects this failure and sends an alert to the gateway when the reporting time fires. The testing subservice doesn't consider it as a problem, even if it is able to detect it, as it does not affect the O2 detection event. However, the crash of S2 would be considered as a problem for the two subservices because in this case the node or sensor failure will affect the O2 detection event.
- Case 2: Location error
  Suppose that the location of S3 is changed to another area. Such a location change leads to a failure as it is against the application's requirements. In this case, the testing subservice signals that there is *potential problem* (an application level failure) to the gateway. However, for the diagnosis subservice *nothing goes wrong*!!
- Case 3: Clock drift
  Case3 is another semantic application failure, suppose that the cluster head receives positive readings at day from the three types of sensors (S1, S2 and S3) that affirm that three events that lead to the photo have occurred. In such case, the network must report the O2 event. However, due to a clock drift on the cluster head the network doesn't report anything because the time at cluster head indicates a wrong time (a night time) to the photosynthesis process. Although, the assessments of the diagnosis subservice here were positive (node responsiveness) the testing service is able to detect incorrect behavior of the O2 detection service.

B. Scenario2: Explosion Detection service

Service Description: A cluster head node notifies the gateway about the eventual mine explosion upon receiving positive affirmations about the three events: O2 detection service, high methane concentration service and sparks of fire detection service (Figure 1).

We can apply the same fault scenarios as Scenario1. However, this time the service's requirements increase and more complex operations need to occur (collaborative decision making, messages passing, interaction between nodes, and so on). However, the more the complexity increases and the application functionalities are distributed among the nodes the more the benefits of our approach will be. This is mainly due to the *faults isolation* process that allows triggering efficient tests to pinpoints the potential reasons of a potential failure. For example, in case the explosion detection service doesn't work as expected, we can try to test the main service(s) of its necessary triggering. For example, if IFTF begins the tests with the O2 service and if it gets a positive assessment, it can test another relevant service, and so on. These sequential tests reduce the energy consumption in locating the cause(s) of the malfunctioning behavior. In this case, a set of dependency rules between the tests must be set to efficiently trigger the tests.

V. EVALUTION

To evaluate the performance of our approach, we have decided to use the TinyOS 2.x [14]. TinyOS is a free and open source component-based operating system and platform targeting wireless sensor networks (WSNs). What motivates to use TinyOS is its portability to many hardware platforms and its programming structure based on components and interfaces which accommodate the IFTF layer structure. Then, the performance of our solution can be easily studied using the TOSSIM tool [15]. TOSSIM is a TinyOS simulation tool which simulates WSN physical and link layer features accurately. This allows validating the solution under realistic WSN deployment conditions.

A. Memory overhead

One main concern of our approach is memory cost. IFTF resides on every node of the network. The required memory of the diagnosis subservice for TinyOS implementation on TelosB platform is approximately 13 Kbytes in ROM and 1 Kbytes in RAM. Implementation of testing subservice involves mainly a mapping table to store virtual readings, a timer to schedule tests, handlers to manage the inputs and outputs tests. The size of mapping table depends on the number of services to test and the test cases for each service. The test cases must be selected carefully to not overload the limited resources of WSN. Therefore, we choose to emulate the positive and negative readings for each type of service. This gives the opportunity to test the application behavior in a field with the real WSN capacity and dynamicity without overloading the limited resources of WSN. Concerning the handlers, they just call existing interfaces to generate simulated services or events. The required memory for the testing subservice using TinyOS implementation on TelosB is

approximately 20 Kbytes in ROM and 3 Kbytes in RAM. Some advanced events may need some event parameters for event processing. In this case, such parameters should be predefined and stored in the mapping table. This may add more memory cost, but provides more accurate validation.

B. Energy Overhead

Another important performance parameter is the energy consumption. Since network overhead consumes three times more energy than computations [16], only, the radio communication cost was considered in our study. So, we have studied the number of messages for energy overhead estimation. Diagnosis uses heartbeats from neighbor nodes to determine if a node is functional. Nodes in testing subservice are required to exchange messages to report virtual events. To study the impact of redundancy, we vary the number of nodes per area or cluster. The testing service trigger tests based on a hard coded timer and on a rotation basis (a different cluster/area is picked each time a test should be run).The experiment lasts 10 minutes in which the testing subservice runs a test for the sparks of fire service and the diagnosis subservice runs a test every minute. We have measured the message overhead of the IFTF approach and the results are shown in Figure 4. We see that the IFTF approach introduces on average a message overhead of 4% more than using solely the diagnosis subservice. The message overhead of the testing service increases slightly when the number per area or cluster increases due to the multi-hop communications overhead, in contrast of the one hop communications in the diagnosis subservice.

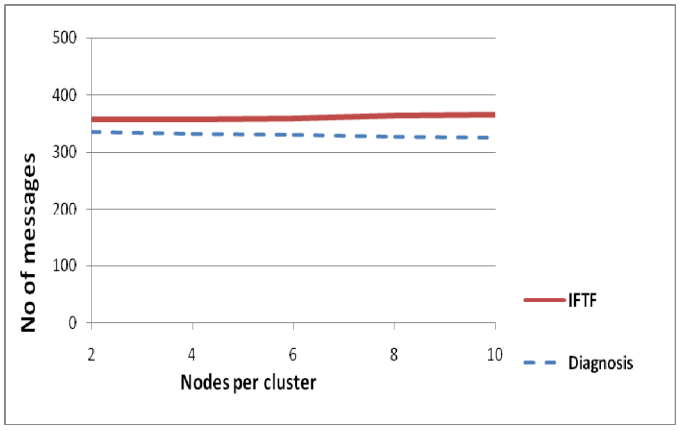

Figure 4: On average, IFTF uses 4% more messages than using solely diagnosis

VI. CONCLUSION

Traditional fault tolerance approaches focus on certain states of system inconsistencies. For example, the system can observe a data loss and diagnoses it by identifying whether it comes from software crashes, node crashes, link problems, etc. The detection system can also detect a software crash, and diagnose it by identifying whether it comes from a stack overflow, a race condition, etc. So we may have different levels to consider. Each problem at whatever level can be detected and diagnosed. However, the more the diagnosis process goes in depth the more the energy consumption will be higher. The reason is simple: more details mean more elements to monitor and more information to be collected and sometimes to be exchanged. However, our approach, in combining the application testing service to the network diagnosis service, allows system monitoring at different levels without significant extra overhead. In addition, it gives more valuable information to the network administrator: Which faults affect the application service? Which faults need to be recovered quickly (as they affect the system performance)? And which faults can be ignored even temporarily?

REFERENCES

[1] Arampatzis, Th., Lygeros, J., Manesis, S.," A Survey of Applications of Wireless Sensors and Wireless Sensor Networks". Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation

[2] L. Paradis and Q. Han, "A Survey of Fault Management in Wireless Sensor Networks," Journal of Network and Systems Management, Volume 15 Issue 2, June 2007.

[3] Xin Miao, Kebin Liu, Yuan He, Yunhao Liu, Dimitris Papadias, "Agnostic Diagnosis: Discovering Silent Failures in Wireless Sensor Networks", IEEE INFOCOM, Shanghai, China, April 10-15, 2011.

[4] P. Li and J. Regehr, "T-Check: Bug finding for sensor networks,"in Proc. of the ACM/IEEE IPSN, Stockholm, Sweden,Apr. 2010, pp. 174–185.

[5] Yang, J., Soffa, M.L., Selavo, L., Whitehouse, K.:" Clairvoyant: a comprehensive source-level debugger for wireless sensor networks". In: Proceedings of the 5th international conference on Embedded networked sensor systems (SenSys 2007).

[6] Y. Zhou, X. Chen, M. Lyu, and J. Liu, "Sentomist: Unveiling transient sensor network bugs via symptom mining," in *Proc.of the IEEE ICDCS*, Genoa, Italy, Jun. 2010, pp. 784–794.

[7] Y. Wu, K. Kapitanova, J. Li, J. Stankovic, S. Son and K. Whitehouse, "Run Time Assurance of Application-Level Requirements in Wireless Sensor Networks", In Proceedings of the 9th ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN 10), SPOTS track, April 2010.

[8] Pennington, S., Bauge, T.,Murray, B. "Integrity-Checking Framework: An In-situ Testing and Validation Framework for Wireless Sensor and Actuator Networks ". Sensor Technologies and Applications, 2009. Third International Conference SENSORCOMM '09.

[9] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, "Sympathy for the sensor network debugger",. In SenSys, 2005.

[10] Y.Liu, K. Liu, M. Li, "Passive Diagnosis for Wireless Sensor Networks", IEEE/ACM Transactions on Networking (TON), Vol. 18, No. 4, August 2010, Pages 1132-1144.

[11] S. Rost and H. Balakrishnan," Memento: A Health Monitoring System for Wireless Sensor Networks", In 3rd Annual IEEE communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON'06), Reston, VA, U.S.A., 2006.

[12] C. Hsin, M. Liu, "Self-monitoring of wireless sensor networks", Computer Communications 29 (2006) 462- 476.

[13] I. Urteaga, K. Barnhart, and Q. Han." Redflag a run-time distributed, flexible, lightweight, and generic fault detection service for data-driven wireless sensor applications". In IEEE PerCom, 2009.

[14] TinyOS (2007), Url : http://www.tinyos.net/.

[15] P. Levis, N. Lee, M. Welsh, D. Culler, "Tossim: Accurate and scalable simulation of entire tinyos applications," in Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003), 2003.

[16] H. Karl, A. Willis," Protocols and Architectures for Wireless Sensor Networks", John Wiley and Sons, 2005.