Noname manuscript No.
(will be inserted by the editor)

Heuristic Strategies for Assessing Wireless Sensor Network Resiliency

An Event-based Formal Approach

Alessandro Testa · Marcello Cinque · Antonio Coronato · Giuseppe De Pietro · Juan Carlos Augusto

Received: date / Accepted: date

Abstract Wireless Sensor Networks (WSNs) are increasingly being adopted in critical applications. In these networks undesired events may undermine the reliability level; thus their effects need to be properly assessed from the early stages of the development process onwards to minimize the chances of unexpected problems during use.

In this paper we propose two heuristic strategies: *what-if analysis* and *robustness checking*. They allow to drive designers towards optimal WSN deployment solutions, from the point of view of the connection and data delivery resiliency, exploiting a formal approach based on the event calculus formal language.

The heuristics are backed up by a support tool aimed to simplify their adoption by system designers. The tool allows to specify the target WSN in a user-friendly way and it is able to elaborate the two heuristic strategies by means of the event calculus specifications automatically generated. The WSN reliability is assessed computing a set of specific metrics. The effectiveness of the strategies is shown in the context of three case studies.

A. Testa
Institute for High Performance Computing and Networking, CNR, via P. Castellino, 111 - Naples, 80131, Italy
Tel.: +39-081-6139515
Fax: +39-081-6139531
E-mail: alessandro.testa@na.icar.cnr.it

A. Coronato and G. De Pietro
Institute for High Performance Computing and Networking, CNR, via P. Castellino, 111 - Naples, 80131, Italy

M. Cinque
DIETI, University of Naples "Federico II", via Claudio 19 - Naples, 80127, Italy

J. C. Augusto
School of Science and Technology, Middlesex University, The Burroughs - London, UK

Keywords Wireless Sensor Networks; Heuristic; Resiliency; Formal language; Robustness Checking.

1 Introduction & Motivation

The extensive use of Wireless Sensor Networks (WSNs) in critical application scenarios stresses the need to verify their reliability properties at design time, in order to prevent wrong design choices that could affect the success of large-scale industrial applications. Examples of such applications are health monitoring (Hao and Foster, 2008), Ambient Intelligent (AmI) systems (Coronato and De Pietro, 2010) and environmental monitoring (Xu et al, 2004). The correct operation of a WSN is affected by several undesired events, such as the crash of a node due to the cheap hardware adopted (Cinque et al, 2012b), battery exhaustion (El Abdellaoui et al, 2012) and the unreliability of the wireless medium (Yu et al, 2012). These, in turn, may isolate whole portions of the network or cause packets to be lost during their traversal to the destination.

If not adequately considered at design time, these events may cause severe failures with dangerous consequences, such as, a health monitoring system not able to report critical alerts about a patient status to a medical center, or a structural monitoring system unable to report a developing crack in a structure.

Formal methods are widely adopted in the literature to verify the correctness of a WSN specification at design time (Boonma and Suzuki, 2010; Zoumboulakis and Roussos, 2011; Bromuri and Stathis, 2009; Blum and Magill, 2011). They appear to be particularly suitable for applying heuristic strategies adopted in the WSN field.

However, their practical use for the verification of reliability properties of WSNs has received little attention, due to the distance between system engineers and formal methods experts and the need to re-adapt the formal specification to different design choices. Even if some development teams would invest on the definition of a detailed specification of WSN correctness properties, a design change (e.g., different network topology, number of produced packets) could require to rethink the formal specification, incurring in extra undesirable costs. To address the issues listed above, we also introduce a tool for the automated application of the heuristic strategies in order to assess WSN resiliency facilitating the work of the designers.

The contribution of the work is manifold:

- we define a formal approach for failure detection in WSNs considering the formal specification of WSN correctness split in two logical sets: a *general correctness specification*, valid independently of the particular WSN under study, and a *structural specification* related to the properties of the target WSN (e.g., number of nodes, topology, channel quality, initial battery charge), designed to be generated automatically;

- we consider specific WSN reliability metrics, such as *connection resiliency*, *coverage*, *data delivery resiliency*, *network capacity* and *power consumption*, used as drivers to guide design choices;
- we propose two heuristic strategies (starting from the same specification): i) *what-if analysis*, to observe how the WSN behaves in response to a given sequence of events of interest for the designer, and ii) *robustness checking*, to verify the long term robustness of the WSN against random sequences of undesired events, useful to identify corner cases and reliability bottlenecks.
- we develop an automated verification tool, named *ADVISES* (**A**utomate**D** **V**er**I**fication of w**S**n with **E**vent calculu**S**), to simplify the adoption of the proposed approach. It is realized i) to automatically generate the structural specifications given the properties of a target WSN (e.g. topology), ii) to automatically perform the two heuristic strategies, iii) to perform the reasoning starting from the correctness and structural specifications and iv) to compute reliability metrics starting from the event trace produced by a reasoner;
- we show the usefulness of the proposal in the context of three case studies, to show how the proposed framework and tool can help system engineers to take decisions upon key design questions such as: "How many nodes are covered in the WSN if a given sequence of failures occurs?", "How many failures the WSN is able to tolerate so that a minimum coverage level is guaranteed?", "How the WSN behaves, in terms of delivered packets, if the channel quality changes (e.g., to consider environments with different levels of harshness)?".

This research is partially based on our previous results (Testa et al, 2012). In this paper we extend the formal specification considering new events: packet loss event and battery exhaustion event, we present the what-if analysis and robustness checking techniques as heuristic strategies applied using a formal approach; moreover, we provide additional experimental results that show how the two strategies practically work and allow to make interesting considerations; finally we report the results of detailed comparison of our approach with the related work in the area of WSN heuristics and resiliency assessment.

The rest of the paper is organized as follows. Section 2 presents related work, while in Section 3 we introduce the formal approach defining the general correctness and structural specifications of a WSN respectively. Heuristic strategies considered in this work are discussed in Section 4 and the ADVISES tool is shown in Section 5. Cases studies and results are presented in Section 6 and a comparison of our work with the cited related work is reported in Section 7. Finally Section 8 concludes the paper.

2 Related Work

Laprie (2008) defined resiliency as the persistence of reliability when facing changes. In facts, assessing the resiliency of a WSN means essentially to eval-

uate the effects of a change affecting the WSN, in terms of the variation on the metrics of interest or in terms of the overall behavior of the network.

Since the reliability was defined as the ability of a system to deliver a service that can justifiably be trusted (Avizienis et al, 2004), we can claim that the concept of resiliency is strictly related to the concept of reliability.

Typically a node failure in a WSN has the effect of modifying the system topology by the removal of a communication node and its corresponding links. A node may fail due to several reasons, such as, battery discharge (either natural or abnormal), hardware faults of the sensing and computation platforms (such as bit flips and stuck-at-zero) induced by the harsh environment, software defects of the sensor program, etc.

In WSNs several failures/recoveries of nodes influence the expected behavior and quality of the WSN (Cinque et al, 2013). For this reason, it becomes important to apply the concept of *connection resiliency*, here defined as *the persistence of network connectivity in spite of WSN changes*, related to the WSN topology. Since a real WSN configuration is not generally a fully connected graph, successive failures may result in a disconnection of the system, namely a disconnection failure, and therefore prevent a set of nodes from reaching the sink (i.e. isolated nodes).

However, the service delivered by the WSN does not encompass only the connection, but also the computation, i.e., even when sensor nodes are potentially connected (a path exists between nodes and sink node), packet losses can still occur, e.g., due to physical causes (wireless channel interference) or software causes (sender or receiver buffer overflow). For this reason, the concept of *data delivery resiliency* can be defined as *the persistence of the amount of data delivered to the sink node of the WSN, despite WSN changes.*

Finally, other metrics of interest to qualify the resilient operation of a WSN can be evaluated: the *power consumption* of nodes, because a node may fail due to battery discharge (either natural or abnormal), the *coverage* of the monitored area to analyze the time interval in which the WSN can operate, while preserving a given number of nodes connected to the sink node, and the *network capacity* to measure the average amount of packets received by the sink node

At present, most of the papers propose heuristic algorithms for optimum utilization of energy in WSNs. For instance, Arivubrakan and Dhulipala (2012) and Santos et al (2012) present techniques to minimize the power consumption of the wireless sensor nodes. In particular Arivubrakan and Dhulipala (2012) propose an algorithm for consume low energy without degrading the performance; Santos et al (2012) focus on the WSN organization by designing topologies based on clusters which minimize the power consumption of the wireless sensor nodes. However, they focus on the optimization of the power consumption of a wireless sensor node without evaluating the overall reliability of the WSN deployment in terms of coverage, connection resiliency and data delivery resiliency.

Yuan and Hollick (2012) propose and implement heuristics for the optimal configuration of number of channels and topology of the routing tree of a WSN.

Despite they consider a heuristic approach to assess resiliency of a WSN, it cannot be considered to evaluate the robustness of a WSN in terms of delivered packets.

Several approaches have been proposed in the literature for the quantitative evaluation of WSN properties.

Simulative approaches are reported by Titzer et al (2005), Levis et al (2003) and Zhang et al (2009). They are realized by means of behavioral simulators i.e., tools able to reproduce the expected behavior of a WSN-based system by means of a code-based description. These approaches are more oriented to the verification of behavioral properties or performance indicators, and not oriented to the observation of reliability properties. No heuristic method is proposed.

Lee et al (2004) and Di Martino et al (2012) adopt analytical approaches to assess the resiliency and performance of WSNs. Lee et al (2004) define a mathematical model of the energy consumption of nodes to study and forecast the network lifetime. Di Martino et al (2012) introduce an approach for the automated generation of WSN resiliency models, based on a variant of Petri nets. However, they do not apply heuristics and it is not always possible to observe non-functional properties of WSNs by means of analytical approaches, since models need to be redefined and adapted to the specific network.

The analysis of WSNs with formal approaches has recently started to be considered by the scientific community. Formal approaches are very useful to understand a particular behavior of a WSN on the basis of a reasoning performed considering an initial sequence of events.

Ölveczky and Thorvaldsen (2007) propose a formal language to specify a WSN and develop a tool to simulate it. However, the proposed solution is not adaptive and not heuristic-based: the formal specification that describes the behavior of a WSN has to be rewritten whenever the considered WSN changes. Man et al (2009) propose a methodology for modeling, analysis and development of WSNs using a formal language; they focus on power consumption as resiliency concern disregarding node crash/isolation and packet loss.

Other papers focus on formal modeling of WSN protocols. Chen et al (2013) provide an extensive review of automated formal verification techniques of ad hoc routing protocols for WSN; Katelman et al (2008) propose a formally-based system redesign methodology used to redesign a version of the LMST topology protocol that ensures network connectivity under realistic deployment conditions. Elleuch et al (2011) apply a probabilistic framework to formally reason about the expected values of coverage intensity in a WSN. In both cases they only check the connectivity of the network without considering data delivery resiliency or connection resiliency to understand the tolerance degree of a WSN. Meanwhile Fehnker et al (2009) propose a graphical specification style which by means of a visualization studies the effect of topologies in performance analysis presenting interference problem but without providing any reliability metrics (like the data delivery resiliency).

A model-driven performance engineering framework for WSNs, named Moppet, is introduced by Boonma and Suzuki (2010); it uses the event calculus

formalism to estimate the performance of WSN applications in terms of power consumption of each sensor node. Whilst our contribution shares similarities to the work presented by Boonma and Suzuki (2010), we focus however not only on power consumption but also on coverage and resiliency to undesired events.

By studying the current proposed formal approaches, we realize that no heuristic strategy has been introduced yet in order to assess the resiliency of a WSN. We assert that, applying a formal approach, heuristic techniques can be very helpful to easily lead designers towards good deployment choices.

We also propose an automatic process, supported by a user friendly tool, for adapting the specifications to the target WSN and for elaborating the heuristic strategies and computing reliability metrics automatically, starting from the analysis of the reasoning output.

Finally we note that an open issue with formal specifications of WSNs is that they need to be adapted when changing the target WSN configuration, e.g., in terms of the number of nodes and topology. To address this problem, in this work we provide separated specifications and thus conceive two logical sets of specifications (see Section 3).

3 Formal Approach

3.1 Event Calculus

Since the normal and failing behavior of a WSN can be characterized in terms of an event flow (for instance, a node is turned on, a packet is sent, a packet is lost, a node stops to work due to crash or battery exhaustion, etc.), we adopt an *event-based* formal language. In particular, among several event-based formal languages, we choose *event calculus* (Shanahan, 1999), since its simplicity, its wide adoption in the sensor networks arena (Boonma and Suzuki, 2010; Zoumboulakis and Roussos, 2011; Bromuri and Stathis, 2009; Blum and Magill, 2011) and the possibility to formally analyze the behavior of a system as event flows, offering simple ways to evaluate the reliability metrics of our interest.

Event calculus was proposed for the first time by Kowalski and Sergot (1986) and then it was extended by Miller and Shanahan (1996). It belongs to the family of logical languages and is commonly used for representing and reasoning about the events and their effects. *Fluent*, *event* and *predicate* are the basic concepts of event calculus (Shanahan, 1999). Fluents are formalized as functions and they represent a stable status of the system. For every timepoint, the value of fluents or the events that occur can be specified. This language is also named *narrative-based*: there is a single time line on which events occur and this event sequence represents the *narrative*. Narrative is useful to understand a particular behavior of a WSN. Reliability metrics can be valuated by analyzing the narrative generated by an event calculus reasoner based on the specification of the target WSN.

To check the proposed correctness properties defined in event calculus we use the *Discrete Event Calculus (DEC) Reasoner*. The DEC Reasoner (Mueller, 2005) uses satisfiability (SAT) solvers and by means of this we are able to perform reasoning like deduction, abduction, post-diction and model finding. It is documented in details in (Mueller, 2005) in which its syntax is explained (e.g. the meaning of the symbols used in the formulas).

3.2 General Correctness Specification

The formal approach is founded on the definition of a core formal specification in event calculus. The main idea is to formalize the correctness properties allowing engineers to verify if a given WSN design, specified in terms of number of nodes, position of nodes, channel quality and initial battery level, is able to satisfy given design constraints.

In particular, in this paper the evaluation of design constraints is based on the measurement of the following WSN reliability metrics proposed by Di Martino et al (2012) and by Chiasserini and Garetto (2004):

- *Connection Resiliency* represents the number of node failures and disconnection events that can be sustained while preserving a given number of nodes connected to the sink;
- *Coverage* is the time interval in which the WSN can operate, while preserving a given number of nodes connected to the sink;
- *Data Delivery Resiliency* is the number of node failures and disconnection events that can be sustained while preserving a given number of correct packets delivered to the sink;
- *Power Consumption* is the battery power consumed by each sensor, useful to estimate the expected lifetime of the WSN.
- *Network Capacity*, also defined as the overall throughput at the sink, is the total arrival rate of the packets coming from all the sensors and received by the sink.

Overall these metrics allow to evaluate the expected resiliency of a WSN in terms of its robustness to failure events, its capacity to cover a given area and its duration (lifetime). Clearly, other metrics can be defined for other purposes, following the same approach. The metrics are evaluated by analyzing the narrative generated by the reasoner based on the specification of the target WSN. To ease the modification of the specification upon changes in the WSN design, in terms of number and position of nodes, channel quality, etc., we define the specification as two logical sets:

- *General Correctness Specification* (described in this section) - a set of correctness properties specifications, valid independently of the particular WSN under study.
- *Structural Specification* (described in next section) - a set of specifications and parameters related to the properties of the target WSN, e.g., number of nodes, network topology, quality of the wireless channel (in terms of disconnection probability) and initial charge of batteries.

The general specification is defined once and used across different WSN designs. The structural specification, instead, has to be adapted when changing the target WSN. The proposed ADVISES tool, described later in the paper, aims to simplify this operation, by accepting the structural specification in a user friendly format and by generating automatically the files needed by the event calculus reasoner.

The general correctness specification is described in the following. It specifies that a WSN performs correctly if none of the following undesired events (or *failures*) happen: (i) *isolation event*, i.e., a node is no more able to reach the sink; (ii) *packet loss event*, i.e., a packet is lost during the traversal of the network; (iii) *battery exhaustion event*, i.e., a node stops to work since it has run out of battery. Events (i) and (ii) can be caused by more "basic events", such as the stop of one or more nodes (e.g., due to crash or battery exhaustion) or the temporary disconnection of a node to its neighbor(s) due to transmission errors. Event (iii) is generated by considering the initial battery charge and the energy request of nodes due to packet sending and receiving activities (in general assumed to be power demanding activities with respect to CPU activities). We concentrate on all of the three main events described previously considering the results of a Failure Modes and Effect Analysis conducted on WSNs by Cinque et al (2012a). Moreover the proposed approach is conceived to extend the specifications with other events, if needed.

3.2.1 Isolation event

The isolation event happens when a node is no more able to reach the *sink* of the WSN, i.e., the gateway node where data are stored or processed. The isolation might be caused by more simple, basic events, such as a stop of a node, due to an arbitrary crash, or battery exhaustion and the disconnection of a node from another node. For instance, let us consider the Figure 1 and let us suppose that node i is the only one allowing the transmission of data between the sink node and the subnet A. We want to check when the subnet A is isolated from the rest of network. We suppose that node i is connected with node j and k. If node i fails, the nodes j and k (and all the nodes of the subnet A) are alive but isolated and so the whole subnet A is isolated. More in general, if a subnet depends on a node and this node becomes isolated then all of the nodes of the subnet are isolated.

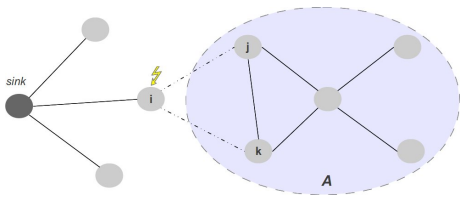


Fig. 1: Isolation of a WSN subnet

Table 1: Basic elements of the specification for the isolation event

Elements	**Name**	**Description**
Sorts	*sensor*	Reference sensor for events and fluents
	to_sensor	Sensor used in case of connection (i.e. a sensor connects **to** another sensor)
	from_sensor	Sensor used in case of disconnection (i.e. a sensor disconnects **from** another sensor)
Basic Events	*Start(sensor)*	Occurring event when a sensor turns on
	Stop(sensor)	Occurring event when a sensor turns off
	Connect(sensor, to_sensor)	Occurring event when a sensor connects to another sensor
	Disconnect(sensor, from_sensor)	Occurring event when a sensor disconnects from another sensor
Generated Events	*Isolate(sensor)*	Occurring event when a sensor is isolated from the network
	Join(sensor)	Occurring event when there is at least a connection between a sensor and one or more sensors
Fluents	*IsAlive(sensor)*	True when a *Start* event occurs for a sensor
	IsLinked(sensor, to_sensor)	True when a *Connect* event occurs
	IsReachable(sensor)	True when a sensor is reachable from the sink node

In Table 1 we report the basic elements (sorts, events and fluents) used for the specification. We distinguish basic events from generated events. These last events are generated by the reasoner on the basis of the specification and of the sequence of basic events actually occurred. Listing 1 shows the rules that represent the core of the specification for an isolation event. In lines 1-5 we define a rule to verify when a node becomes isolated. A sensor can be isolated if it is initially reachable, alive and, considering a link with another sensor, there is no sensor that is alive, reachable and connected with the sensor. Also

Listing 1: Correctness Specification for the Isolation event

```
1  [sensor,from_sensor, time]Neighbor(from_sensor,sensor) & HoldsAt(IsReachable(sensor),time)
2       & HoldsAt(IsAlive(sensor),time) & (!{from_sensor2} (HoldsAt(IsAlive(from_sensor2),
3       time) & HoldsAt(IsReachable(from_sensor2),time) &
4       HoldsAt(IsLinked(sensor,from_sensor2),time)) & Neighbor(from_sensor2,sensor)) ->
5  Happens(Isolate(sensor),time).
6
7  [sensor,from_sensor, time] ( !HoldsAt(IsReachable(sensor),time) & HoldsAt(IsAlive(sensor),
8        time) & HoldsAt(IsAlive(from_sensor),time) & HoldsAt(IsReachable(from_sensor),time))
9        & HoldsAt(IsLinked(sensor,from_sensor),time) & Neighbor(from_sensor,sensor) ->
10 Happens(Join(sensor),time).
11
12 [sensor,from_sensor, time] ((HoldsAt(IsAlive(from_sensor),time) &
13      HoldsAt(IsReachable(from_sensor),time) & HoldsAt(IsLinked(sensor,from_sensor),time))
14      | !HoldsAt(IsReachable(sensor),time) | !HoldsAt(IsAlive(sensor),time)) &
15      Neighbor(from_sensor,sensor) ->
16 !Happens(Isolate(sensor),time).
17
18 [sensor,from_sensor,time] ( HoldsAt(IsReachable(sensor),time) |
19      !HoldsAt(IsAlive(sensor),time) | !HoldsAt(IsLinked(sensor,from_sensor),time) |
20      !HoldsAt(IsAlive(from_sensor),time) | !HoldsAt(IsReachable(from_sensor),time)) &
21      Neighbor(from_sensor,sensor)->
22 !Happens(Join(sensor),time).
```

Table 2: Basic elements of the specification for the packet loss event

Elements	Name	Description
Sorts	*pkt*	Packet id
	source	Sensor that sends the packet
Basic Events	*Send(pkt,source)*	Occurring event when a sensor starts the delivery of a packet (*pkt*) towards the sink node
Generated Events	*Receive(pkt, source)*	Occurring event when sink node receives a packet
	Catch(pkt,source, sensor, from_sensor)	Occurring event when a sensor catches a packet from one of its neighbor sensors
	Forward(pkt,source, sensor, to_sensor)	Occurring event when a sensor, once caught a packet sent by a sensor, forwards it to its neighbor sensor
	PacketLoss(pkt, source)	Occurring event when a packet is lost
Fluents	*IsInDelivery(pkt, source)*	True when a packet delivery starts
	IsOnChannel(pkt,source, sensor, to_sensor)	True when a packet is in transmission on the channel between two nodes
	IsLost(pkt, source)	True when a packet is lost

we report (in lines 7-10) another rule which allows to check a *Join* event. In particular the rule declares that if a sensor is not reachable, because is isolated, and alive and its neighbor sensor is also alive and reachable and there is a connection between them then the sensor can join the network and becomes reachable again. In the last two rules (in lines 12-16 and 18-22) we define conditions in which *Isolate* and *Join* events cannot occur.

3.2.2 Packet Loss event

When there is a failure in a node or a link between a couple of nodes is disrupted then all of the packets that are in delivery towards this node are lost. In turn, these packets are not delivered to the sink. For instance, let us consider Figure 2. Node *A* sends a packet *pkt* to node *B*. If node B crashes it cannot receive the packet from node A and forward it to a node towards the sink; so the packet is lost.

In Table 2 we report the basic elements (sort, events and fluents) used for the specification related to the packet loss event. Again, events are divided in basic and generated ones.

In the Listing 2 there are the rules that represent the core of the specification for the packet loss event. By means of the rule defined in lines 1-2, we assert that when a *Send* event comes from the WSN, a *Forward* event is

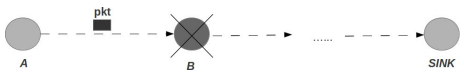


Fig. 2: Example of packet loss

generated and the packet delivery starts. In the lines 4-9 and 11-16, we define the *Catch* event that occurs when, being the packet on the channel between a couple of nodes, the receiving sensor is alive, the packet is not lost and there is a connection between the nodes. Note that we also included a concurrency control in order to manage packets that come from different nodes. Also, we can check the end of a packet delivery (lines 24-25) when it is caught by the sink node and so it is received.

Finally, in lines 27-32 we report the rule that checks when a packet is lost. Considering a packet, that is not lost, in delivery towards the sink node and on the channel between two nodes, when a disconnection event between the two nodes or the failure of receiving node occurs then there is a packet loss event.

3.2.3 Battery Exhaustion event

A battery exhaustion event happens when a node completely consumes its available energy. To this aim, we adopt a sort in the specification for each sensor node, called *level*, which represents the level of the battery of the node, and that is decremented each time the node sends, catches or forwards a packet.

Listing 2: Correctness Specifications for the packet loss event

```
1  [pkt,sensor,to_sensor,time]Happens(Send(pkt,sensor),time) & Neighbor(to_sensor, sensor) ->
2  Happens(Forward(pkt,sensor,sensor,to_sensor),time).
3
4  [sensor,from_sensor,source,pkt,time] HoldsAt(IsOnChannel(pkt,source,from_sensor,sensor)
5      ,time) & (!{sensor2,from_sensor2,source2,pkt2}
6      HoldsAt(IsOnChannel(pkt2,source2,from_sensor2, sensor2),time) & source!=source2 &
7      sensor=sensor2) & HoldsAt(IsAlive(sensor),time) &
8      HoldsAt(IsLinked(from_sensor,sensor),time) & !HoldsAt(IsLost(pkt,source),time) ->
9  Happens(Catch(pkt,source,sensor, from_sensor),time).
10
11 [sensor,from_sensor,pkt,from_sensor2, pkt2,source, source2, time]
12     HoldsAt(IsOnChannel(pkt,source,from_sensor, sensor),time) &
13     HoldsAt(IsOnChannel(pkt2,source2,from_sensor2, sensor),time) & source<source2
14     & HoldsAt(IsAlive(sensor),time) & HoldsAt(IsLinked(from_sensor,sensor),time) &
15     !HoldsAt(IsLost(pkt,source),time)->
16 Happens(Catch(pkt,source,sensor, from_sensor),time).
17
18 [sensor,to_sensor,from_sensor,source,pkt,time]
19     Happens(Catch(pkt,source,sensor, from_sensor),time) & Neighbor(sensor, from_sensor)
20     & Neighbor(to_sensor, sensor) & HoldsAt(IsAlive(sensor),time) &
21     !HoldsAt(IsLost(pkt,source),time) ->
22 Happens(Forward(pkt,source,sensor,to_sensor),time+1).
23
24 [from_sensor,source,pkt,time] Happens(Catch(pkt,source,1, from_sensor),time) ->
25 Happens(Receive(pkt,source),time).
26
27 [sensor,to_sensor,source,pkt,time] HoldsAt(IsInDelivery(pkt,source),time) &
28     !HoldsAt(IsLost(pkt,source),time) &
29     HoldsAt(IsOnChannel(pkt,source,sensor,to_sensor),time) & Neighbor(to_sensor, sensor) &
30     (!HoldsAt(IsLinked(sensor,to_sensor),time) | Happens(Stop(to_sensor),time) |
31     !HoldsAt(IsAlive(to_sensor),time) ) ->
32 Happens(PacketLoss(pkt,source),time).
```

Table 3: Basic elements of the specification for the battery exhaustion event

Elements	Name	Description
Sorts	*level*	The current battery level of a sensor
	levelnew	The new battery level of a sensor after RX/TX operations related to a packet
	capacity	Value which indicates the maximum battery capacity of a sensor, in terms of the maximum number of packets that it can send
Basic Events	-	-
Generated Events	*New_Consume(level,sensor)*	Occurring event when the battery level of a sensor decreases after the forwarding of a packet
	Old_Consume(level,sensor)	Event created for specification reasons
Fluents	*BatteryLevel(level,sensor)*	True when the battery charge of a sensor is equal to a certain *level*

In Table 3 we report the basic elements (sort, events and fluents) used for the specification related to the battery exhaustion event. For this specification, the basic events are the same shown for the previous specifications. Hence, we only report generated events.

In the Listing 3 there are the rules that represent the core of the specification for the battery exhaustion event. When a *Forward* event occurs (due to a sending of a packet) and the battery level of the sensor is positive then we consider the new consume level (lines 1-4). In the lines 6-7 we can see that when the battery level of a sensor, that is alive, is zero then a failure for the node occurs (*Stop* event) due to its battery exhaustion.

3.3 Structural Specification

General correctness specifications are complemented by a structural specification of the target WSN. This is mainly related to the topology of the WSN and completed by parameters regarding the initial level of batteries and the quality of channels. Differently from the specifications described in the previous Section, this specification varies on the basis of the characteristics of the target WSN.

To specify the topology, we use the predicate *Neighbor* (already used in the previous specifications) to indicate how nodes are linked in the topology. For instance, considering the topology in Figure 3, let us suppose node i is

Listing 3: Correctness Specification for the battery exhaustion event

```
1 [pkt,sensor,to_sensor,source,level,time] Happens(Forward(pkt, source, sensor,to_sensor),
2         time) & HoldsAt(BatteryLevel(level,sensor),time) & level>0 ->
3 Happens(Old_Consume(level,sensor),time) &
4 ({levelnew}levelnew=level-1 & Happens(New_Consume(levelnew,sensor),time)).
5
6 [sensor,time] HoldsAt(BatteryLevel(0,sensor),time) & HoldsAt(IsAlive(sensor),time) ->
7 Happens(Stop(sensor),time).
```

Listing 4: Example of Neighbor predicate Specification

```
1 [sensor1,sensor2] Neighbor(sensor1,sensor2) <-> (
2 (sensor1=i & sensor2=j) |
3 (sensor1=i & sensor2=k)
4 ).
```

connected with j and k and let us consider a tree graph where the sink node (root node) is the node i and the nodes j and k are child nodes.

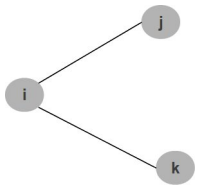


Fig. 3: Example of tree graph with 3 nodes

The resulting specification is reported in Listing 4, where *sensor1* is the parent node (i) and *sensor2* is the child node (for this example, j and k). Clearly, this specification can be adapted easily whenever the WSN topology changes.

The role of the *Neighbor* predicate is very important to understand when an axiom can be applied. Let us examine the axiom related at a possible isolation event (lines 1-5 of Listing 1) and let us apply it for the Figure 3. The described implication is true when, given a couple of nodes (*sensor*, *from_sensor*), the conditions about isolation are true and there is a link between the nodes (in this case, between node j and i or between node k and i). This, for instance, can never be true for the couple of nodes j and k, since there is not a physical link between them.

Regarding the parameters, their values can be used to check the correctness properties of the WSN under different conditions, i.e., under different assumptions on the initial charge of batteries (e.g., to verify a WSN in the middle of its life), or under different environmental conditions affecting the quality of the channels (impacting on the probability to have a disconnection event when checking the robustness of the WSN).

4 Heuristic Strategies

To provide useful support for resiliency assessment of a WSN, we perform two different heuristic strategies (starting from the same specification): i) *What-If Analysis*, to observe how the WSN behaves in response to a given sequence of events of interest for the designer, and ii) *Robustness Checking*, to discover

Listing 5: Example of initial event trace

```
Happens(Stop(j),1).
Happens(Stop(k),3).
Happens(Start(k),4).
Happens(Disconnect(k,i),5).

completion Happens
```

the long term robustness of the WSN against random sequences of undesired events, useful to identify corner cases and reliability bottlenecks.

4.1 What-if Analysis

The goal of a *What-If* scenario analysis is to observe the behavior of the target WSN under various circumstances. To this aim, we need to indicate an initial event sequence (*Event Trace*) in the structural specification along with the topology specification and all the values belonging to the target WSN. The Event Trace is a combination of predicates written in the event calculus language in the following form:

Happens(event, timepoint).

where *event* is an occurrence (i.e. a sensor stops or disconnects, ...) and *timepoint* is a number to fix the succession of the events. This type of heuristic strategy allows us to figure out *what* happens *if* certain events occur by means of the reasoning performed by the event calculus reasoner; we have observed that event calculus is particularly suitable to apply this strategy.

For example, as Listing 5 shows, by means of *Happens* predicates, we can declare that at timepoint 1 sensor j stops, at timepoint 3 sensor k stops, etc. In this way, by means of the event calculus reasoner, we can observe the consequences of any initial sequence of events of interest for the designer, e.g., to test the robustness of the designed topology against the temporary unavailability (failure/recovery) of a given set of nodes or to quantify to what extent a modification of the topology can be beneficial for the network.

4.2 Robustness Checking

For *Robustness Checking* we intend a heuristic strategy to statically analyze the behavior of a WSN in front of a number of failures that can occur during its operation. It is generated a failure tree (in which there are several failure combinations) and we perform a pruning of this tree on the basis of the desired reliability requirements.

We distinguish two types of robustness checking analysis. The first one aims to analyze the robustness of the network, in terms of coverage, against a variable number of failures (*Stop* and *Disconnection* events), from 1 to n, where n is selected by the user, considering all combinations without repetitions. It is particularly useful to evaluate the coverage and connection resiliency of the network and pinpoint weak points in the topology. The second one aims to analyze how the target WSN behaves during a periodic sending of packets if random failures happen, causing packets to be lost. It is useful to evaluate the data delivery resiliency of the network. These two types of robustness checking analysis are detailed in the following.

Coverage Robustness Checking. By means of coverage robustness checking, we can check how many node failures the network can tolerate, while guaranteeing a given minimum level of coverage. For example if we consider a network composed by m nodes and a threshold coverage equal to 50%, we may want to understand what are the sequences of failures causing more than $m/2$ nodes to be isolated (i.e., coverage under the specified threshold) and how the resiliency level varies when varying the sequences of failures. This allows to evaluate the maximum (and minimum) resiliency level reachable by a given topology and what are the critical failure sequences, i.e., the shortest ones causing a loss of coverage. These are particularly useful to pinpoint weak points in the network. We developed an algorithm to generate automatically the sequences of failures (*Stop* and *Disconnection* events) against which checking the robustness of the WSN. The algorithm is implemented by the ADVISES tool (introduced in the next Section), and it is aimed to reduce the number of failure sequences to be checked. The principles are to avoid repetitions and to end the sequence as soon as the coverage level becomes lower than the user defined threshold. For instance, we start considering all the cases when there is one failure.

By means of the reasoner we compute the coverage; if the coverage is above the threshold, the resiliency is surely greater than 1, because there is just one failure and it is tolerated in all cases. In the generic $k-th$ step, we consider sequences of k failures. If the generic sequence $\{f_1, f_2, ..., f_k\}$ leads to a coverage below the threshold, we do not consider sequences starting with a $\{f_1, f_2, ..., f_k\}$ prefix in the $(k+1)-th$ step. By considering the percentage of sequences with k failures where the coverage is above the threshold, let us say r_k%, we can say that the resiliency is k in r_k% of cases.

The described process is summarized by means of the algorithm for the computation of event sequences with n failures shown in listing 6.

Let *failures* be the maximum number of failures in a row that user wants to consider. At the first step, the algorithm considers all of the possible failure event sequences with only one failure performing reasoning on the basis of the event traces composed by one single failure.

Then, we have resiliency computation to check if for every obtained event trace, there is coverage and thus resiliency: if the experiment with trace i produces coverage lower than the set threshold, it is deleted otherwise this trace will be included among the *new_traces* ("good" traces). Percentage of resiliency: number of "good" traces (1 failure) out of the total number of the

Listing 6: Algorithm for the computation of event sequences with n failures

```
int f=1;
while (f<=failures)
{
       if f==1
       {
        Computation of all combinations with 1 failure
       gen_traces()

        Storing these traces in a combination file

       create_and_reason() //Create and make reasoning on these traces
       new_traces = compute_resiliency() // Resiliency computation

        Percentage of resiliency: number of traces with coverage upper the threshold (dim)
            / all of the events (total number of single failures)
       per_resiliency = (dim*100) / events_map.size();
       }

        2 or more failures
       else
       {
              for (int ev =0; ev<new_traces.size(); ev++)
              {
              gen_traces()

               Create and make reasoning on these traces
              create_and_reason(traces_pp,f_count,events_map, coverage,con_resiliency);

               Resiliency computation: if the experiment with the 'ev'-th trace produces
                    coverage false, it is deleted otherwise this trace will stay in '
                    traces_temp' (temporary)
              traces_temp=compute_resiliency(traces_pp, events_map,f_count,output_r);

               Insert the traces, contained in 'traces_temp' and related to this for cycle,
                     in a global file; in this last file there are all the traces to
                    consider for the experiments with upper number of failures.
              for (int traces_temp_ind =0; traces_temp_ind<traces_temp.size();
                    traces_temp_ind++)
              {
                     traces_all.addElement(traces_temp.elementAt(traces_temp_ind));
              }

               Calculate total number of combinations.

               Percentage of resiliency: number of traces with coverage upper the threshold
                     (dim) / all of the combinations (den)
              per_resiliency = dim * 100 / den;
              new_traces = traces_all;
       }

       f++;  // counter of the while cycle
}
```

possible one-failure sequences; this value is provided by the ADVISES to figure out the traces to bring in the next step of the algorithm.

In case of failure event sequences composed by two or more failures, the algorithm performs the same previous operations but in an extended version. We consider the combination of all the traces with the number of failures equal to the algorithm cycle. For instance, if we have to consider sequences of

2 failures, for every event trace computed at the step 1 (only one failure) we obtain n-1 event traces composed by two failures adding a single failure that has not already occurred in the sequence.

Thus, if after the first cycle we have the following event traces composed by single failures

$$\{f_1, f_2, f_3, \dots f_n\}$$

after the second cycle we have the following event traces composed by two failures:

$$\{f_1f_2, f_1f_3, f_1f_4, \dots f_1f_n, f_2f_1, f_2f_3, f_2f_4, \dots f_2f_n, \dots f_nf_1, \dots f_{n-1}f_n\}$$

Note that, to prune the search tree of the algorithm, we compute the new event traces only starting from previous event traces that lead to a coverage higher than the desired value.

We perform reasoning on the basis of the new obtained event traces which are composed by as many failures as it is the value of the current cycle.

The next step is the resiliency computation, to check if for every obtained event trace the coverage is still above the desired value: if the experiment with the "ev"-th trace produces a coverage below the desired threshold, it means that the network is not resilient to such an event trace, otherwise the trace will be kept in *traces_temp* (temporary); the new traces, contained in *traces_temp* are inserted in a global file; in this last file there are all the traces that have been considered as "good" (leading to a coverage higher that the set threshold).

To calculate the percentage of resiliency we have to compute the total number of the obtained combinations; this number depends on the number of failures that the algorithm is taking in account in the current cycle and the number of possible failures (Stop/Disconnect events).

For instance, if the number of *f_count*(cycle counter) is 2 and the total number of failures is 12, we have that all the possible combinations of sequences composed by 2 failures is 12!/(12-2)! = 12*11 = 132.

The algorithm ends when all the traces, composed by a number of failure events equal to value chosen by the user, have been analyzed.

Data Delivery Robustness Checking. In this case we want to check the robustness of the WSN in case of failures when there are packets being delivery on the network. To do this, we model each link of the WSN on the basis of the Gilbert-Elliott Channel Model (Elliott, 1963), and we associate to each link a disconnection probability (indicating the channel quality), set as a parameter by the user together with the structural specification. In this way, the user can simulate different scenarios, e.g., environments with different channel conditions.

We simulate that, assuming a periodic WSN, every sensor sends periodically a packet to the sink. At the same time we randomly generate failures (failures node and/or disconnection events), taking into account disconnection probabilities.

The data delivery resiliency is computed as the maximum number of failures that can be sustained while a given fraction of packets (above a given threshold) is still received at the sink.

5 The ADVISES Tool

A Java-based tool, called *ADVISES* (**A**utomate**D** **V**er**I**fication of w**S**n with **E**vent calculu**S**), has been designed and implemented to facilitate the application of the proposed heuristic strategies.

By means of a graphical user interface (GUI), shown in Figure 4, the user can simply specify i) the topology of the target WSN, using a connectivity matrix (topology section of the GUI), ii) the formal correctness specifications (for checking isolation events, packet losses and battery exhaustion), iii) the temporal window size to consider, in terms of the number of timepoints (EC section), iv) the number of packets that each sensor can send, v) the number of failures (in case of coverage robustness checking as chosen heuristic strategy), vi) the battery capacity of a sensor and the needed energy for RX/TX operations (both expressed in Joule), vii) the metrics to calculate (for coverage and data delivery resiliency it is necessary also the threshold value), viii) the channel model (in case of data delivery robustness checking), ix) the initial battery level, x) the initial event trace (to perform what-if scenario analysis as heuristic strategy).

Figure 5a shows an example of a dialog associated with the *Channel Model* button, where the user can specify the disconnection probability of every link of the WSN. In Figure 5b we show how a user can choose the initial battery level of each sensor. These values are expressed in percentages. Figure 6 shows an example of dialog associated with the *New Trace Event* button, where the user can specify the initial event trace to conduct what-if analysis. Note that the user can completely specify the WSN and values associated with his case

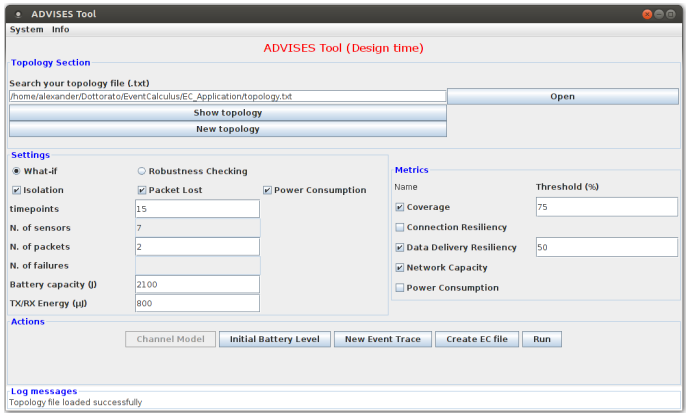
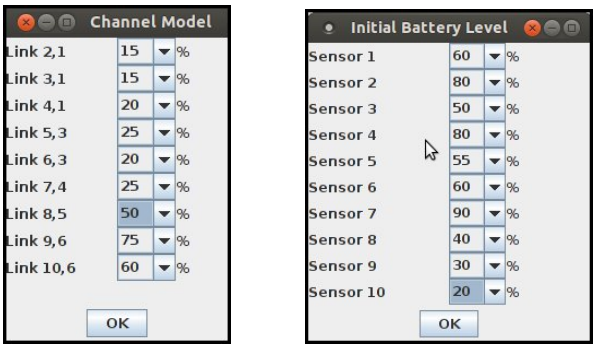


Fig. 4: ADVISES GUI



(a) Channel Model (b) Initial Battery Level

Fig. 5: Channel Model and Initial Battery Level

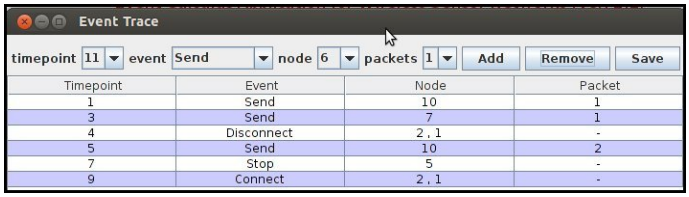


Fig. 6: Example of an initial Event Trace specified by the user

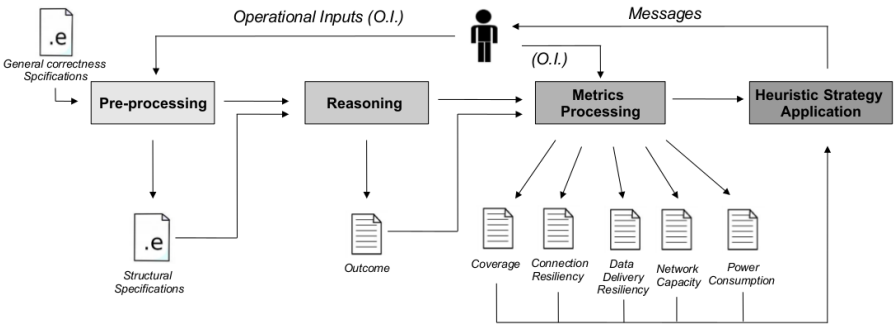


Fig. 7: Workflow

study, without the need to know the details about the underlying formalism, which is hidden by the ADVISES tool.

Figure 7 shows the workflow of the ADVISES tool. Once user inputs are inserted and settings have been chosen (i.e. *Channel Model*, *Initial Battery Level* and initial *Event Trace*), the ADVISES tool is able to automatically generate in a file (that can be visualized pressing the *Create EC file* button) the structural specification with initial conditions (timepoints, number of sensors and packets, battery levels, etc.) in terms of event calculus formalism. This

file, together with the general correctness specification, is then provided by the ADVISES tool as input to the DECReasoner (the most used event calculus reasoner), which produces the outcome as the result of the reasoning (this happens when the user presses the *Run* button). The obtained trace (more than one, in case of robustness checking) is analyzed by the ADVISES tool to evaluate the reliability metrics, namely coverage, connection resiliency, data delivery resiliency, network capacity and power consumption (when the user selects the respective checkboxes in the *Metrics* section of the GUI). Finally ADVISES performs a heuristic strategy chosen by the designer and then sends messages in order to inform or alert him about resiliency related to a particular WSN deployment.

5.1 Metric computation

By means of a parser that analyzes the traces produced by the DECReasoner, the ADVISES tool calculates the reliability metrics. The computation of *Coverage* and *Connection Resiliency* depends on a threshold parameter, to be indicated as a percentage by the user in the GUI (see the "Threshold" text field in the Metrics section in figure 4). The threshold expresses the fraction of failed and isolated nodes that the user can tolerate, given its design constraints. For instance, over a WSN of 20 nodes, a threshold set to 100% means that all the 20 nodes have to be connected, whereas 50% means that the user can tolerate at most 10 isolated nodes.

Considering the threshold value, we calculate the *Coverage* analyzing the *IsReachable(sensor)* and *IsAlive(sensor)* fluents found to be true in the event trace produced by the reasoner: if a *-IsReachable(x)* or a *-IsAlive(sensor)* fluent is true in the event trace, this means that node x became isolated or it stopped. For example in the case of coverage at 50%, for a WSN with 7 nodes, there is coverage when at least 4 nodes are not isolated (i.e., they are reachable). Hence, as soon as 4 different nodes are no reachable nor alive (looking at the fluents), the network is not covered anymore. The coverage can be then evaluated as the interval $[0, t]$, being t the timepoint of the last failure or disconnection event before the isolation (e.g., the timepoint of the event that caused the isolation of a number of nodes exceeding the threshold).

The *Connection Resiliency* can then be evaluated as the number of failure and disconnection events (namely, *Stop(sensor)* and *Disconnect(sensor, from_sensor)* events) that happen within the coverage interval, excluding the last failure/disconnection event, that is, the one that actually leads the number of isolated nodes to overcome the threshold. For example, if we have coverage in the interval [0, 6], and during this period 3 failure/disconnection events can be counted, than the connection resiliency is 2, that is, the WSN was able to tolerate 2 failures or disconnections while preserving more than 50% of the nodes connected.

Even the computation of the *Data Delivery Resiliency* depends on a threshold parameter indicated by the user as the percentage value. By means of this

percentage value, the user can express the fraction of the number of packets that can be lost. For instance, over a WSN of 30 nodes, a sensors sends 10 packets; if this threshold is equal to 60%, it means that at most 4 packets can be lost. Considering the threshold value, the ADVISES tool calculates the data delivery resiliency analyzing the *IsInDelivery(pkt,source)* fluent in the event trace produced by the reasoner: if a *+IsInDelivery(pkt,source)* fluent is true in the event trace, this means that a packet has been sent by a sensor and it is in delivery; if a *-IsInDelivery(pkt,source)* fluent is true, this means that a packet has been received by the sink node and it is not in delivery anymore. If the number of instances where the fluent *-IsInDelivery(pkt,source)* is equal to the number of instances where the fluent textit+IsInDelivery(pkt,source), then every packet has been correctly received by the sink node, otherwise this means that some packet is lost. As soon as the fraction of lost packets becomes lower than the threshold value then the ADVISES tool verifies the number of failures (*Stop* and *Disconnect* events) occurred; the number of the occurred failures represents the value of the data delivery resiliency.

Network Capacity is calculated in a similar way to the *Data Delivery Resiliency*. Also in this case, the event trace produced by the reasoner is analyzed by searching the *IsInDelivery(pkt,source)* fluent to check if all the packets have been received by the sink node. Respect to the previous metric, the network capacity is represented as a normalized value ranging between 0 (if sink node does not receive any packet) and 1 (if sink node receives all the packets sent by the nodes). This value represents the average of the delivery rate of all the packets sent by all the nodes of a network.

Power Consumption is computed analyzing the *BatteryLevel (level, sensor)* fluent. For each timepoint, if a *+BatteryLevel(level,sensor)* fluent is true then the ADVISES tool updates the battery consumption of the related sensor with the new value equal to *level*. After the last timepoint, for each sensor the ADVISES tool computes the percentage of power consumption. If the behavior of nodes is known (e.g., each node sends packets periodically, with a known period), this information can be used to evaluate the lifetime of the network as the time when the number of alive and not isolated nodes falls below the coverage threshold.

6 Case studies & Results

In this section we report the results from three representative case studies considered to show the application of our two heuristic strategies for assessing WSN resiliency exploiting our formal approach.

We have focused on the criticality of WSN-based healthcare systems involved in patient monitoring scenarios as reported by Ko et al (2010), Hande et al (2006) and Chipara et al (2010). The considered systems are composed by WSNs containing 7/15 nodes.

The case studies have been realized taking in account indoor environments and TMOTE SKY XM1000 wireless sensors (running TinyOS, as operating

system for WSNs, that implements a contention based MAC protocol) which follow the IEEE 802.15.4 standard (ZigBee). We have chosen these wireless sensors since they offer high data rate requiring ultra low power.

In the table 4 we summarize the main features considered in order to perform the case studies. In particular, the type of MAC layer is the Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA). We can assert the MAC mechanism provided by our IEEE 802.15.4-based wireless sensors successfully reacts on connectivity of the network since it supports retransmission and automatic synchronization to networks mechanisms.

The first case study is based on what-if scenario analysis and its aim is to show how this heuristic strategy, utilizing the ADVISES tool, can be adopted to test a WSN configuration empirically observing its behavior given a certain event sequence. The last two case studies are based on robustness checking. In these cases, the aim is to show how a heuristic strategy, as the robustness checking, can be helpful to analyze a WSN design and drive engineers' choices before the real deployment.

In particular, the third case study has been realized to check the robustness related to data delivery of packets sent by sensors of a WSN exploiting the Gilbert-Elliott Channel Model (Elliott, 1963).

Table 4: Features of the sensor nodes involved in the case studies

TMOTE SKY XM10000	
Feature	**Description**
Standard	IEEE 802.15.4 (ZigBee)
Frequency band	2.4 GHz (ISM)
Transmission rate	250 Kbps
Radio Chipset	CC2420 RF which provides reliable wireless communications
Power Consumption	18.8 mA (RX) 17.4 mA (TX)
MAC Layer features	Encryption with AES-128 CSMA-CA Retransmission Synchronization to networks Security
Microcontroller	TI MSP430F1611
Operating System	TinyOS
Storage	10KB RAM 48 KB Flash

6.1 Case study 1 (what-if analysis): *A WSN realized in our laboratory*

As first case study, we consider the topology of a WSN constructed by mounting ten sensor nodes placed according to a tree schema (Figure 8); node 1 (the root of the tree) is the sink of the network. The distance between the couples of nodes (in parent-child relationship) is set to about 2 meters.

In this case study we are interested to evaluate the behavior of the network, in terms of coverage, connection resiliency and data delivery resiliency, when the following sequence of basic events occurs: *Send(1,5)* at timepoint 1, *Send(1,9)* at timepoint 2, *Stop(4)* at timepoint 4 and *Disconnect(3,1)* at timepoint 7. For a coverage threshold set at 70%, we should observe that coverage interval is equals to $[0, 7]$ (i.e., when node 3 disconnects from node 1 at time point 7, 4 nodes are not reachable, namely 3, 4, 8 and 9) and the connection resiliency is equals to 1 (i.e., only the *Stop(4)* event is tolerated).

For a data delivery resiliency threshold set at 60%, we should observe that the packets sent by node 5 and node 9 are received by the sink without being lost. The initial event trace produced by the ADVISES tool is given in Listing 7.

The ADVISES tool generates the values for the range of sensors and timepoints, again analyzing user inputs. In this case, we know that the network is composed by 10 nodes that can send at most 1 packet and we want to observe what it could happen in 10 timepoints.

Also in this case, the outcome produced by the DECReasoner (see Listing 8) reports the event trace by means of which we can validate our assumptions.

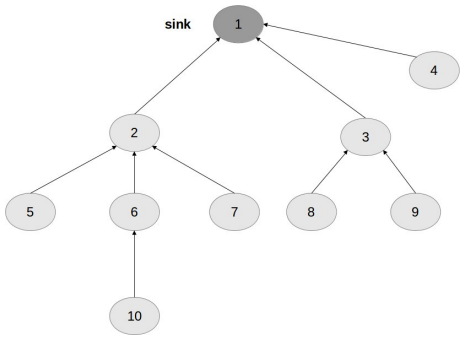


Fig. 8: Topology of the WSN realized in our laboratory (case study 1)

Listing 7: Initial event trace for the WSN of case study 1

```
1 Happens(Send(1,5),1).
2 Happens(Send(1,9),2).
3 Happens(Stop(4),4).
4 Happens(Disconnect(3 , 1),7).
5
6 completion Happens
```

Listing 8: Outcome of the DECReasoner for the case study 1

```
0
1
Happens(Forward(1, 5, 5, 2), 1).
Happens(Send(1, 5), 1).
2
+IsInDelivery(1, 5).
+IsOnChannel(1, 5, 5, 2).
Happens(Catch(1, 5, 2, 5), 2).
Happens(Forward(1, 9, 9, 3), 2).
Happens(Send(1, 9), 2).
3
-IsOnChannel(1, 5, 5, 2).
+IsInDelivery(1, 9).
+IsOnChannel(1, 9, 9, 3).
Happens(Catch(1, 9, 3, 9), 3).
Happens(Forward(1, 5, 2, 1), 3).
4
-IsOnChannel(1, 9, 9, 3).
+IsOnChannel(1, 5, 2, 1).
Happens(Catch(1, 5, 1, 2), 4).
Happens(Forward(1, 9, 3, 1), 4).
Happens(Receive(1, 5), 4).
Happens(Stop(4), 4).
5
-IsAlive(4).
-IsInDelivery(1, 5).
-IsOnChannel(1, 5, 2, 1).
+IsOnChannel(1, 9, 3, 1).
Happens(Catch(1, 9, 1, 3), 5).
Happens(Receive(1, 9), 5).
6
-IsInDelivery(1, 9).
-IsOnChannel(1, 9, 3, 1).
7
Happens(Disconnect(3, 1), 7).
8
-IsLinked(3, 1).
Happens(Isolate(3), 8).
9
-IsReachable(3).
Happens(Isolate(8), 9).
Happens(Isolate(9), 9).
10
-IsReachable(8).
-IsReachable(9).
```

In fact, we can observe that when node 4 stops, there are still 9 reachable nodes. When there is a disconnection between nodes 3 and 1, there are 6 reachable nodes. For this reason the coverage is 7 because in the interval $[0, 7]$ the covered nodes are 9 on 10 (90%); the connection resiliency is equal to 1.

Finally, the data delivery resiliency is 100%; the packets sent by node 5 and node 9 are correctly received by the sink.

6.2 Case study 2 (robustness checking): *Is it worth to add a node?*

As a first case study for robustness checking let us consider a simple WSN with 6 nodes (Figure 9a). This topology is commonly adopted to monitor a

Table 5: Results of simple linear topology

Outcome	**Simple linear topology**							
	6 sensors				**6 sensors + 1 extra**			
No. of Failures	1	2	3	4	1	2	3	4
Connection Resiliency	1	0	0	0	1	2	0	0
Percentage of Cases	20%	-	-	-	66%	24%	-	-

linear structure, such as a tunnel or an oil pipeline (Jawhar et al, 2008). We have considered every couple of adjacent sensors is placed at a distance of 4 meters following a linear schema.

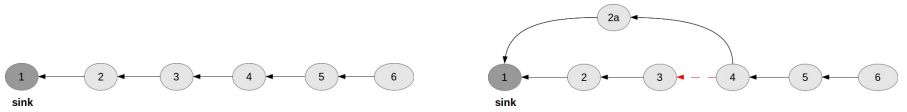
From the figure, it is intuitive to conclude that node 2 represents the most critical reliability bottleneck for this topology, since it has to route the packets from all other nodes to the sink.

The simplicity of the topology allows to reason on reliability bottlenecks and on potential improvements. In particular, the objective of the case study is to quantify the benefits, in terms of connection resiliency, of adding one extra node (see Figure 9b) between the sink node (node 1) and node 4 at a distance equal to 6 meters from these two nodes.

In order to obtain the results, shown in Table 5, we have considered in both topologies a reasoning performed on 10 timepoints, a coverage threshold value equal to 70% (5 reachable nodes) and a number of failures from 1 to 4.

For both topologies we report the number of failures, the connection resiliency and the percentage of the cases in which the connection resiliency is not 0. For example in the simple linear topology (without extra node), exploring all of the cases in which 1 failure occurs, we have connection resiliency equal to 1 in 20% of the cases. In the other cases, the coverage is below 70%. If 2 failures occur, we have no cases in which coverage is above 70% and so, the maximum connection resiliency level achievable is 1. This confirms numerically that the topology is extremely fragile and susceptible to failures in the majority of cases when undesired events occur.

If we add an extra node then we gain benefits because we triple the chances (66%) to have connection resiliency with coverage >70% in case of 1 failure, and we have coverage upper than threshold value also when 2 failures occur (in the 24% of the cases). Hence, in this case the maximum connection resiliency level is 2. In this way we can assert that adding a node (accounting for 17%



(a) WSN with line topology (b) WSN with extra node in line topology

Fig. 9: WSN with line topology (case study 2)

of extra cost) in the proximity of the sink allows to significantly boost the robustness of the WSN, by triplicating the chances of survival in case of 1 failure, and by doubling the maximum connection resiliency achievable.

6.3 Case study 3 (robustness checking): *Robustness checking in harsh environments*

In the last case study we consider a WSN with 8 nodes reported in Figure 10. In particular we want to observe how the WSN would react and, consequently, how the percentage of data delivery would change with different disconnection probabilities. This is useful to check the robustness of the WSN in different deployment scenarios, e.g., from environments with a good channel quality (for instance, an outdoor scenario in good whether conditions, with no interferences) to harsh environments (such as, indoor scenario with fading due to obstacles and walls and electromagnetic interferences due to the presence of other wireless devices).

To do this, we model each link of the WSN on the basis of the Gilbert-Elliott Channel Model Elliott (1963), and we associate to each link a disconnection probability (indicating the channel quality), set as a parameter by the user together with the structural specification. In this way, the user can simulate different scenarios, e.g., environments with different channel conditions.

We simulate that, assuming a periodic WSN, every sensor sends periodically a packet to the sink. At the same time we randomly generate failures (failures node and/or disconnection events), taking into account disconnection probabilities. The data delivery resiliency is computed as the maximum number of failures that can be sustained while a given fraction of packets (above a given threshold) is still received at the sink.

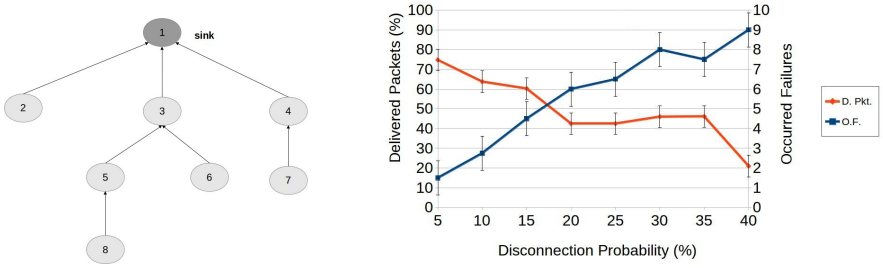
Specifically, we analyze the percentage of delivered packets when every link has a probability of disconnection ranging from 5% to 40% with step 5%. We expect that, the more the disconnection probability grows, the more failures occur, and the less is likely that a packet is delivered to the sink. In particular, we want to check under which operational conditions the network is still able to deliver more than 50% of packets to the sink.

The graph in Figure 10b shows the results of the study, where *O.F.* (*Occurred Failures*) represents the number of the occurred failures and *D. Pkt.* (*Delivered Packets*) the percentage of delivered packets. Both the values are reported as a function of the disconnection probability from 5% to 40%. Each point of the graph is obtained by repeating the experiment 3 times, letting the ADVISES tool generate different random sequences starting from the disconnection probability set by the user. For this reason, each point represents a mean value and also standard deviation bars are reported. From the graph, we can observe the expected inverse relationship between the trend of the occurred failures and the trend of delivered packets: when the probability of disconnection increases, also the number of the failures increases, whereas the number of the delivered packets decreases.

Table 6: Network capacity of the sink on the basis of disconnection probability (Case study 3)

Disconnection Probability	Network Capacity
5%	0.747
10%	0.638
15%	0.603
20%	0.465
25%	0.440
30%	0.425
35%	0.425
40%	0.210

It is interesting to observe a certain resilience of the network for disconnection probability values ranging from 20% to 35%. In this range it seems that, even if the number of failures keeps increasing as expected, the network is redundant enough to tolerate them. Hence the percentage of delivered packets remains the same, and we need to stress the network up to a 40% of disconnection probability to observe a more significant loss in the percentage of delivered packets. Considering instead our requirement on checking up to which conditions the WSN is able to deliver more than 50% of packets to the sink, we can observe that this requirement is satisfied up to a disconnection probability of 15% and a number of failures below 5. This means that the data delivery resiliency for this network is 5 (with the threshold set to 50%), and that the WSN is able to conform to expectations only if deployed in an environment where quality of channels is such that the disconnection probability of links does not overcome the critical level of 15%.



(a) WSN topology with 8 nodes (b) Results of the case study

Fig. 10: Topology and results of the case study 3

For this case study, we have also computed the network capacity metric since we have considered the packet delivery. From the table 6, evaluating the result of the figure 10b, it is worth to claim that the higher is the disconnection probability of the links in the WSN, the lesser is the network capacity of the sink, or in other words the total arrival rate of packet received by the sink; this is due to a higher loss of the packets during their delivery. Even in the case of the lowest disconnection probability (5 %) the WSN does not exploit all the available throughput (the network capacity is 0.747); this means that considered data flows do not saturate the sink.

7 Related Work Comparison

In order to appreciate the innovative contribution of our heuristic strategies applied using a formal approach, we compare (in Table 7) the characteristics of the analyzed work, discussed in Section 2, with ADVISES tool. The aim of this comparison is to show what are the advantages that other work cannot benefit.

For each work we analyzed if some formal method has been used, if some heuristic strategy has been adopted, which reliability metrics have been considered, if specifications have been separated (in case of work based on formal approach), if the work is supported by some tool, and finally if some case study has been presented.

From our related work we can see that few apply formal approaches to study the behavior of WSN from the same perspective we do in this work. Therefore a better understanding of how formal approaches have been applied in the WSN area can be interesting. We have noticed that among the most important reliability metrics, the power consumption is the only one that has been considered extensively. Instead data delivery resiliency and connection resiliency are the least analyzed; our aim has been to consider all the four reliability metrics and until now there is no work that considers all these that we have considered in this paper in order to assess WSN resiliency exploiting the power of the formal methods and heuristic strategies.

The majority of papers propose a tool and present results by means of some case study. The *Separated specifications* column, one of the main advantages of the ADVISES tool, emerges: for the first time specifications are separated considering the general specifications on one side and the structural specifications dependent on the WSN topology on another side.

Therefore we can confirm that the ADVISES tool meets several important characteristics that allow us to easily apply heuristic strategies for assessing WSN resiliency using an event-based formal approach demonstrating its novelty in the field of reliability research for WSNs.

Table 7: Related work comparison

Work	**Formal Method**	**Heuristic Strategy**	**Reliability metrics**	**Separated specifications**	**Tool**	**Case study**
ADVISES	YES	YES	Coverage Conn. Resil. Data Del. Resil. Pow. Cons.	YES	YES	YES
Avrora (Titzer et al, 2005)	NO	NO	None	NO	YES	YES
GRASP-based metaheuristic (Yuan and Hollick, 2012)	NO	YES	Pow. Cons.	NO	NO	YES
HOL theorem prover (Elleuch et al, 2011)	YES	NO	Coverage	NO	NO	NO
LMST (Katelman et al, 2008)	YES	NO	Coverage	NO	NO	NO
Math. Analysis work (Lee et al, 2004)	NO	NO	Coverage Conn. Resil. Pow. Cons.	NO	NO	YES
Moppet (Boonma and Suzuki, 2010)	YES	NO	Pow. Cons.	NO	YES	YES
Ns2 (Zhang et al, 2009)	NO	NO	None	NO	YES	YES
Petri nets Framework (Di Martino et al, 2012)	NO	NO	Coverage Conn. Resil. Data Del. Resil. Pow. Cons.	NO	YES	YES
PRISM-based framework (Fehnker et al, 2009)	YES	NO	None	NO	YES	NO
Real-time Maude tool (Ölveczky and Thorvaldsen, 2007)	YES	NO	Coverage Conn. Resil. Pow. Cons.	NO	YES	NO
Tepawsn (Man et al, 2009)	YES	NO	Pow. Cons.	NO	YES	NO
Tossim (Levis et al, 2003)	NO	NO	None	NO	YES	YES
Tree convergecast scheduling (Santos et al, 2012)	NO	YES	None	NO	NO	YES
WSN heuristic algorithm (Arivubrakan and Dhulipala, 2012)	NO	YES	Pow. Cons.	NO	NO	YES

8 Conclusion

This paper has described two heuristic strategies for assessing WSN resiliency: What-If analysis and Robustness Checking. These strategies have been applied exploiting an event-based approach, and related ADVISES tool, for the automated reliability assessment of WSNs using event calculus as formal language. Using this formalism we have presented a set of general correctness specifications, valid for any WSN that are integrated with a structural specification, automatically generated for the particular WSN under study. In this way, the same core specification can be re-used to elaborate heuristics for a WSN under design, without requiring system engineers to have knowledge of the adopted formalism.

The effectiveness of the proposed heuristic strategies has been shown in the context of three case studies. They have shown how the results are useful to understand the behaviour of a WSN when a particular event sequence occurs, to drive design choices (e.g., whether is worth to add a node) and to check limit operational conditions (the minimum channel quality required to let the WSN work to expectations).

The definition of the general correctness specifications (for isolation, packet loss and battery exhaustion event) has been an hard and long task since we had to write their informal descriptions in event calculus formalism, to realize some initial simple scenario and verify/test them by means of the DECReasoner.

Among them we found particular criticality of the packet loss event specification and thus battery exhaustion event specification (because the last one is linked to the packet loss event). While the reasoning performed on the specification for the isolation event takes a reasonable time to be performed, the reasoning performed on the specification for packet loss (and for battery exhaustion) requires orders of magnitude longer times. This is due to the more complex logic that characterizes this specification.

This issue is also reflected on case studies. We have considered topologies of WSNs adopted in typical critical scenarios, such as healthcare and indoor surveillance (with 7/15 nodes). During the experimental phase, we have also tested the correct functioning of the specification with topologies with more nodes (about 100) but the reasoning time becomes impractical on our commodity hardware, due to the state space explosion problem. While this issue does not affect the conceptual validity of the approach and its use on typical critical WSNs, it undermines its practical adoption for large WSNs. To face this scalability problem, we are currently conceiving a method to divide a large topology in several sub-topologies (equivalent to WSN clusters), perform reasoning on each sub-topology in parallel and finally join the results taking in account the dependences between the sub-topologies.

The outcomes obtained by the event calculus reasoner report events happening in a given timepoint and their cause-effect relationships, without any information about the actual time (e.g. it is not known the exact temporal instant of a certain event in seconds).

For this reason, in our paper we have focused on the metrics which are not time-dependent (i.e. connection resiliency, data delivery resiliency, network capacity and power consumption). In the case of coverage metric we have computed it considering the number of the timepoints.

We base our approach on sensors that are fixed (such as beacons) and with a established data routing reducing a topology like a spanning tree that is valid for a WSN. As future work we plan to extend the use of the specification also for mobile scenarios specifying further events that notify wireless sensor movements within clusters and from cluster to cluster.

References

Arivubrakan P, Dhulipala V (2012) Energy consumption heuristics in wireless sensor networks. In: Computing, Communication and Applications (ICCCA), 2012 International Conference on, pp 1–3, DOI 10.1109/ICCCA.2012.6179194

Avizienis A, Laprie JC, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. Dependable and Secure Computing, IEEE Transactions on 1(1):11–33

Blum J, Magill E (2011) Telecare service challenge: Conflict detection. In: Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2011 5th International Conference on, pp 502–507

Boonma P, Suzuki J (2010) Moppet: A model-driven performance engineering framework for wireless sensor networks. Comput J 53(10):1674–1690

Bromuri S, Stathis K (2009) Distributed agent environments in the ambient event calculus. In: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, ACM, New York, NY, USA, DEBS '09, pp 12:1–12:12, DOI 10.1145/1619258.1619275, URL http://doi.acm.org/10.1145/1619258.1619275

Chen Z, Zhang D, Zhu R, Ma Y, Yin P, Xie F (2013) A review of automated formal verification of ad hoc routing protocols for wireless sensor networks. Sensor Letters 11(5):752–764, DOI doi:10.1166/sl.2013.2653, URL http://www.ingentaconnect.com/content/asp/senlet/2013/00000011/00000005/art00002

Chiasserini CF, Garetto M (2004) Modeling the performance of wireless sensor networks. In: INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, vol 1, pp –231, DOI 10.1109/INFCOM.2004.1354496

Chipara O, Lu C, Bailey TC, Roman GC (2010) Reliable clinical monitoring using wireless sensor networks: experiences in a step-down hospital unit. In: Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, ACM, New York, NY, USA, SenSys '10, pp 155–168, DOI 10.1145/1869983.1869999, URL http://doi.acm.org/10.1145/1869983.1869999

Cinque M, Coronato A, Testa A (2012a) Dependable services for mobile health monitoring systems. IJACI 4(1):1–15

Cinque M, Martino CD, Testa A (2012b) Analyzing and modeling the failure behavior of wireless sensor networks software under errors. In: IWCMC, pp 1136–1141

Cinque M, Coronato A, Testa A, Martino CD (2013) A survey on resiliency assessment techniques for wireless sensor networks. In: Proceedings of the 11th ACM international symposium on Mobility management and wireless access (MobiWac '13), pp 73–80

Coronato A, De Pietro G (2010) Formal design of ambient intelligence applications. Computer 43(12):60 –68, DOI 10.1109/MC.2010.335

Di Martino C, Cinque M, Cotroneo D (2012) Automated generation of performance and dependability models for the assessment of wireless sensor networks. IEEE Trans Comput 61(6):870–884, DOI 10.1109/TC.2011.96, URL http://dx.doi.org/10.1109/TC.2011.96

El Abdellaoui S, Debbah M, Fakhri Y, Aboutajdine D, et al (2012) Increasing network lifetime in an energy-constrained wireless sensor network. International Journal of Sensor Networks (IJSNET) 12(4)

Elleuch M, Hasan O, Tahar S, Abid M (2011) Formal analysis of a scheduling algorithm for wireless sensor networks. In: Qin S, Qiu Z (eds) ICFEM, Springer, Lecture Notes in Computer Science, vol 6991, pp 388–403, URL http://dblp.uni-trier.de/db/conf/icfem/icfem2011.html#ElleuchHTA11

Elliott EO (1963) Estimates of Error Rates for Codes on Burst-Noise Channels. Bell System Technical Journal 42:1977–1997

Fehnker A, Fruth M, Mciver AK (2009) Methods, models and tools for fault tolerance. Springer-Verlag, Berlin, Heidelberg, chap Graphical Modelling for Simulation and Formal Analysis of Wireless Network Protocols, pp 1–24, DOI 10.1007/978-3-642-00867-2_1, URL http://dx.doi.org/10.1007/978-3-642-00867-2_1

Hande A, Polk T, Walker W, Bhatia D (2006) Self-powered wireless sensor networks for remote patient monitoring in hospitals. Sensors 6(9):1102–1117

Hao Y, Foster R (2008) Wireless body sensor networks for health-monitoring applications. Physiological Measurement 29(11):R27–R56, DOI 10.1088/0967-3334/29/11/R01, URL http://dx.doi.org/10.1088/0967-3334/29/11/R01

Jawhar I, Mohamed N, Shuaib K, Kesserwan N (2008) Monitoring linear infrastructures using wireless sensor networks*. In: Wireless Sensor and Actor Networks II, Springer, pp 185–196

Katelman M, Meseguer J, Hou J (2008) Redesign of the lmst wireless sensor protocol through formal modeling and statistical model checking. In: Proceedings of the 10th IFIP WG 6.1 international conference on Formal Methods for Open Object-Based Distributed Systems, Springer-Verlag, Berlin, Heidelberg, FMOODS '08, pp 150–169, DOI 10.1007/978-3-540-68863-1_10, URL http://dx.doi.org/10.1007/978-3-540-68863-1_10

Ko J, Lim JH, Chen Y, Musvaloiu-E R, Terzis A, Masson GM, Gao T, Destler W, Selavo L, Dutton RP (2010) Medisn: Medical emergency detection in sensor networks. ACM Trans Embed Comput Syst 10(1):11:1–11:29, DOI 10.1145/1814539.1814550, URL http://doi.acm.org/10.1145/1814539.1814550

Kowalski R, Sergot M (1986) A logic-based calculus of events. New Gen Comput 4(1):67–95, DOI 10.1007/BF03037383, URL http://dx.doi.org/10.1007/BF03037383

Laprie JC (2008) From dependability to resilience. 38th IEEE/IFIP Int Conf On Dependable Systems and Networks, Anchorage, Alaska, June 2008, Sup Vol, pp G8-G9

Lee JJ, Krishnamachari B, Kuo CCJ (2004) Impact of energy depletion and reliability on wireless sensor network connectivity. In: In Proceedings of the SPIE Defense and Security

Levis P, Lee N, Welsh M, Culler D (2003) Tossim: accurate and scalable simulation of entire tinyos applications. In: Proceedings of the 1st international conference on Embedded networked sensor systems, ACM, New York, NY, USA, SenSys '03, pp 126–137, DOI 10.1145/958491.958506, URL http://doi.acm.org/10.1145/958491.958506

Man K, Vallee T, Leung H, Mercaldi M, van der Wulp J, Donno M, Pastrnak M (2009) Tepawsn - a tool environment for wireless sensor networks pp 730 –733, DOI 10.1109/ICIEA.2009.5138301

Miller R, Shanahan M (1996) Reasoning about discontinuities in the event calculus. In: in Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96, Morgan Kaufmann, pp 63–74

Mueller ET (2005) Decreasoner. http://decreasoner.sourceforge.net

Ölveczky PC, Thorvaldsen S (2007) Formal modeling and analysis of the ogdc wireless sensor network algorithm in real-time maude. In: Proceedings of the 9th IFIP WG 6.1 international conference on Formal methods for open object-based distributed systems, Springer-Verlag, Berlin, Heidelberg, FMOODS'07, pp 122–140, URL http://dl.acm.org/citation.cfm?id=1772150.1772161

Santos A, Duhamel C, Belisário L, Guedes L (2012) Strategies for designing energy-efficient clusters-based wsn topologies. Journal of Heuristics 18(4):657–675, DOI 10.1007/s10732-012-9202-x, URL http://dx.doi.org/10.1007/s10732-012-9202-x

Shanahan M (1999) The Event Calculus Explained. Lecture Notes in Computer Science 1600:409–430, URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.43.3267

Testa A, Coronato A, Cinque M, Augusto JC (2012) Static verification of wireless sensor networks with formal methods. In: Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on, IEEE, pp 587–594

Titzer BL, Lee DK, Palsberg J (2005) Avrora: scalable sensor network simulation with precise timing. In: Proceedings of the 4th international symposium on Information processing in sensor networks, IEEE Press, Piscataway, NJ, USA, IPSN '05, URL http://dl.acm.org/citation.cfm?id=1147685.1147768

Xu N, Rangwala S, Chintalapudi KK, Ganesan D, Broad A, Govindan R, Estrin D (2004) A wireless sensor network for structural monitoring. In: Proceedings of the 2nd international conference on Embedded networked sensor systems, ACM, New York, NY, USA, SenSys '04, pp 13–24, DOI 10.1145/1031495.1031498, URL http://doi.acm.org/10.1145/1031495.1031498

Yu C, Fiske R, Park S, Kim WT (2012) Many-to-one communication protocol for wireless sensor networks. Int J Sen Netw 12(3):160–170, DOI 10.1504/IJSNET.2012.050454, URL http://dx.doi.org/10.1504/IJSNET.2012.050454

Yuan D, Hollick M (2012) Tree-based multi-channel convergecast in wireless sensor networks. In: World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a, pp 1–9, DOI 10.1109/WoWMoM.2012.6263713

Zhang J, Li W, Cui D, Zhao X, Yin Z (2009) The ns2-based simulation and research on wireless sensor network route protocol. In: Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on, pp 1 –4, DOI 10.1109/WICOM.2009.5302699

Zoumboulakis M, Roussos G (2011) Complex event detection in extremely resource-constrained wireless sensor networks. Kluwer Academic Publishers, Hingham, MA, USA, vol 16, pp 194–213, DOI 10.1007/s11036-010-0268-0, URL http://dx.doi.org/10.1007/s11036-010-0268-0