International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014 pp.12-18
ISSN 2356-5608

Robustness testing of secure Wireless Sensor Networks

Maha NACEUR[#1], Lilia SFAXI[*2], Riadh ROBBANA[*3]

[#]*LIP 2 Laboratory*

[*] *University of Carthage*

[1] naceur.maha@gmail.com

[2] liliasfaxi@gmail.com

[3] riadh.robbana@gmail.com

Abstract— **A Wireless Sensor Network (WSN) is a type of ad-hoc networks. WSN are characterized by severely constrained computational and energy resources and an ad-hoc operational environment. Due to inherent resource and computing constraints, security in sensor networks poses different challenges comparing to traditional network computer security. To ensure a proper application of security protocols for WSN, it is necessary to validate them before their implementation. We are interested in testing the robustness of these protocols in a hostile environment. This paper presents a new approach to generate and execute test cases for robustness using reachability properties transformed for robustness test purposes. The testing process announces whether secure WSN is robust or not against unexpected events.**

Keywords— **Robustness Testing, security protocol, Network of Timed Automata**

I. Introduction

Wireless sensor networks (WSN) are related to ad-hoc networks [15]. A WSN is an adaptive embedded system composed of small embedded computers (called sensors) that communicate wirelessly to perform a particular task. Sensors can be deployed in large numbers in hostile conditions and their destruction can achieve the desired functionality of the WSN. Indeed, the sensors are able to organize themselves without predefined infrastructure (Self-organization). These collect and transfer some physical data of the ambient environment to one or more gateway nodes (called Sinks). This transfer is done via a multihop architecture. The sink turns the information via satellite or Internet to the computer center, so it can analyze data and take action. The disadvantages of WSN architecture are sensor limited memory, energy supply, processing capacity and wireless communication use [3]. Additional resources are also required to secure sensors. The code size should be very compact for any security solution to avoid memory waste. In other hand, the encryption and decryption are not necessarily performed, because of energy limitation (Data availability), and packets may be lost or damaged due to the wireless communication environment (Data confidentiality and integrity). Due to the harsh communication, old messages may be relayed. Thus, we need to ensure the freshness of each message (Data Freshness).

Sensors are deployed in a hostile environment which probably causes the partial or total degradation of the network. After deployment, the sensors establish cryptographic keys with their neighbors to provide some security services. This mechanism protects exchanged messages between sensors. Each message must be identified and quantified. In WSN, symmetric cryptography is usually used to establish trust. The latter reduces energy consumption of sensors. In addition, a pre-distributed method is used to load keys into sensors.

Most of the security mechanisms require the use of cryptographic keys shared between the communicating parties. Keys have to be installed in sensors to secure communications. However, traditional key-distribution schemes have the following shortcoming: either a single mission key or a set of separate n-1 keys, each being pair wise privately shared with another node, have to be installed in every sensor node. In pre-distribution key method, a big issue is how to load a set of keys (called key ring) into the limited memory of each sensor.

In this work, we are interested on a Localized Encryption and Authentication Protocol (LEAP). LEAP is a key management protocol considered in WSN [18]. It is a deterministic protocol using essentially a master key to derive keys between sensors. It reduces the security threat of a compromised node on its immediate neighbors. To ensure a proper application of security protocols for WSN, it is necessary to validate them before their implementation. We are interested in testing the robustness of these protocols in a hostile environment. Testing is an important validation activity [16]. It is a difficult, expensive, time-consuming, labor-intensive process and it should be repeated each time an implementation is modified [14]. A promising improvement of a testing process is to automatically generate tests from formal models of specification. Using tools may reduce the cost of the test process [17] [13]. Robustness testing helps answering the following question "How does the system react against unexpected events?" [12] [5]. Previous works on robustness testing can be classified into two categories: the first one uses models related to the input domain based on fault injection techniques, and dedicated to characterize the robustness, such as BALLISTA [9]. The second uses behavioral patterns based on the availability of a formal model describing the system behavior and some assumptions to assess its robustness, such as STRESS [6], Verimag Approach [4], Rollet-Fouchal Approach [12], [11].

A security protocol is a sequence of operations that ensure data protection. Used with a communication protocol, it provides secure delivery of data between two parties. It is concerned with properties such as integrity and secrecy. In general, the robustness of security protocols is their ability to respond correctly against network failures. In this paper, we present a new method developed to generate automatically robustness test cases.

The paper is presented as follows: in the next section, we present the Robustness Testing architecture used to generate and execute robustness test cases. We present a nominal specification of a WSN modeled by a network of timed automata. Then, we present the increased specification obtained when adding suspension traces. Thereafter, we introduce the synchronized product between the increased specification and a robustness test purpose. Section 3 presents the generation and execution method of robustness test cases. Concluding remarks are presented in section 4.

II. TESTING APPROACH

Let's consider a hostile environment close to a real-life scenario. To the best of our knowledge, there is no robustness testing for security protocols in the literature. In this paper, we develop a new approach to generate automatically robustness test cases.

A. Robustness Testing Architecture

Formally, we describe the specification of sensors, using LEAP under nominal conditions, by timed automata [2]. By adding suspension traces, we increase this specification. These are useful actions that should be insured by the WSN under robustness test in stress conditions. To generate robustness test cases, we compute traces satisfying a specific need called robustness test purpose (RTP) from the increased specification. RTP describes some aspects of operations in the presence of unexpected events. The robustness test cases are generated from the synchronous product between an increased specification and a chosen RTP. Robustness testing architecture is presented in Figure1.

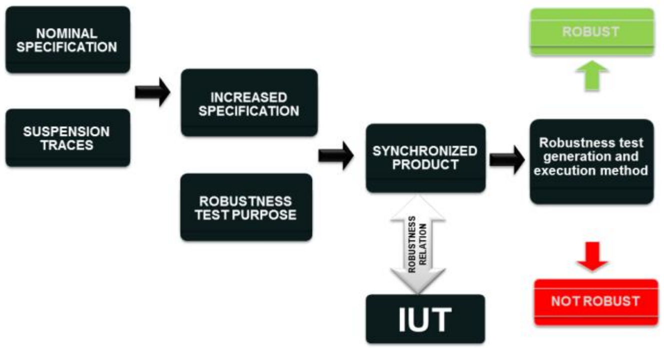

Figure 1 Robustness testing architecture

B. Formalisms

Sensors of WSN are considered as real-time systems and described with Timed Automata (TA), which is a popular formalism of real-time systems modeling [1]. It is an extended finite state machine led by a set of clocks. As time progresses, their values are automatically increased. In the following, we use X to denote the set of clocks and $G(X)$ the set of conjunctions over simple conditions of the form $x \lozenge c$ where $x \in X$, $c \in \mathbb{N}$ and $\lozenge \in \{<, \leq, =, >, \geq\}$. A clock valuation is a function defined as $u: X \rightarrow \mathbb{R}_+$. Let $\mathbb{R}^X$ be the set of all clock's valuations. $u_0(x) = 0$ for all $x \in X$. We consider guards and valuations as sets of clock's valuations and we note $u \in I(q)$ means that u satisfies $I(q)$.

Timed Automata : A TA is a tuple (Q, q_0, X, Act, E, I) where : Q is a finite set of locations, q_0 is the initial location, X is a finite set of clocks, $Act = Act_{In} \cup Act_{Out} \cup \{\tau\}$ is a finite set of input actions (denoted by $a?$), output actions (denoted by $a!$) and unobservable actions ξ, $E \subseteq Q \times Act \times G(X) \times 2^X \times Q$ is the set of edges between locations with actions, a guard and a set of clocks to be reset and $I : Q \rightarrow G(X)$ assigns invariants to locations.

Semantic of TA : TA is defined as a labeled transition system $(S, s_0, \hookrightarrow)$ where $S \subseteq Q \times \mathbb{R}^X$ is the set of states, $s_0 = (q_0, u_0)$ is the initial state and $\hookrightarrow \subseteq S \times (\mathbb{R}_+ \cup Act) \times S$ is the transition relation such that : $(q, u) \hookrightarrow^d (q, u + d)$ if $\forall d', 0 \leq d' \leq d \Rightarrow u + d' \in I(q)$, $(q, u) \hookrightarrow^a (q, u')$ if there exists $e \in E$, $u' = [r \rightarrow 0]u$ and $u' \in I(q')$ where $d \in \mathbb{R}_+$, $u+d$ maps each clock x in X to the value $u(x) + d$, and $[r \rightarrow 0]u$ denotes the clock valuation which maps each clock in r to 0 and agrees with u over $X \setminus r$.

Network of timed automata NTA: A state of WSN is defined by locations of all TA, the clock values, and discrete variables. Every TA may fire a transition separately or synchronize with another TAs, which leads to a new location. Consisting of n sensors modeled by TAs, $NTA = \{Q_i, q^0_i, X, Act, E_i, I_i\}$. A location vector is a vector $\bar{q} = (q_1, \ldots, q_n)$. We compose the invariant functions into a common function over location vectors $I(\bar{q}) = \wedge_i I_i(q_i)$ and we write $\bar{q}[q'_i / q_i]$ to denote the vector where the ith element q' of $\bar{q}$ is replaced by q'_i. In the following, we define the semantics of NTA.

Semantic of NTA: Let $SenA_i = \{Q_i, q_{0i}, X, Act, E_i\ I_i\}$ be a NTA of WSN. Let $\bar{q}_0 = (q^0_1, \ldots, q^0_n)$ be the initial location vector. The semantic is defined as a transition system $(S_Q, s_0, \hookrightarrow)$, where $S_Q = (Q_1 \times \ldots \times Q_n) \times \mathbb{R}^X$ is the set of states, $s_0 = (\bar{q}_0, u_0)$ is the initial state and, $\hookrightarrow \subseteq S \times S$ is the transition relation defined by :

- $(\bar{q}, u) \hookrightarrow^d (\bar{q}, u + d)$ if $\forall d',\ 0 \le d' \le d \Rightarrow u + d' \in I(\bar{q})$
- $(\bar{q}, u) \hookrightarrow^a (\bar{q}[q'_i/q_i], u')$ if there exists $q_i \hookrightarrow^{\tau g r} q'_i$, $u \in g$, $u' = [r \to 0]u$ and $u' \in I(\bar{q}[q'_i/q_i])$
- $(\bar{q}, u) \hookrightarrow^a (\bar{q}[q'_j/q_j, q'_i/q_i], u')$ if there exists $q_i \hookrightarrow^{c?g_i r_i} q'_i$, $q_j \hookrightarrow^{c!g_j r_j} q'_j$, $u \in (g_i \wedge g_j)$, $u' = [r_i \cup r_j \to 0]u$ *and* $u' \in I(\bar{q}[q'_j/q_j, q'_i/q_i])$.

C. Nominal specification of a wireless sensor network using LEAP

We start first by modeling a sensor using LEAP under nominal conditions by TA [2], using Uppaal [8] [7]. This tool allows real-time systems validation [10]. It is designed in order to check systems such as protocols or multimedia applications modeled as NTA which is useful for our work. Indeed, we consider that a WSN is a real-time system composed of a set of components (sensors). Sensors interact permanently with each other and with the environment or human operators, in order to provide services with time constraints. The external environment stimulates sensors with input actions, allowing them to react by producing output actions. During this interaction, a sensor must obey some time constraints. Nominal conditions are defined to allow a normal operation of the WSN. In our case, an attacker who sends a neighbor discovery pushes nodes to send an alert message to the Base Station (BS). In turn, the BS sends a message to all nodes to change the individual key. In [18], authors assume a secure and not compromised BS. Before deployment, the BS loads key into sensors. The authors in [18], assume that a node cannot be compromised before a period T_{min}. Since the deployment, the sensors send a message to discover their neighborhood and establish their pair wise keys. We also note that the energy is not taken into account which degrades performance of the sensor network. In our work, we need to automatically increment integer variables to define the time progression T_{min} and the energy depletion e. These are considered as clocks. A sensor using LEAP in normal conditions is modeled with UPPAAL by the TA in Figure 2. We can move from the initial location *idle* to *WaitAck* if we produce an input action (*Hello!*) or move from *idle* to *Ack* location if an output action (*Hello?*) is received with respect to clock's constraints such as $timeout < 20$ and $e > 30$. If the guard $ID == true$ is satisfied, receivers send an acknowledgement *Acc!*, otherwise, they send (*alerte!*) to the BS. Once all sensors end their neighborhood discovery (guard $PS == N$ is satisfied), they move to the *Connected* location. Figure 3 presents the sensor network we used in nominal conditions. This simulation is composed of 4 sensors. One of them is an attacker. An attacker may pretend to be a neighbor node, it sends a request for neighbor discovery, requires nodes to check the key. This can causes exhaustion of batteries by applying cryptographic functions and/or exhaustion of the time allotted for the neighbor's discovery

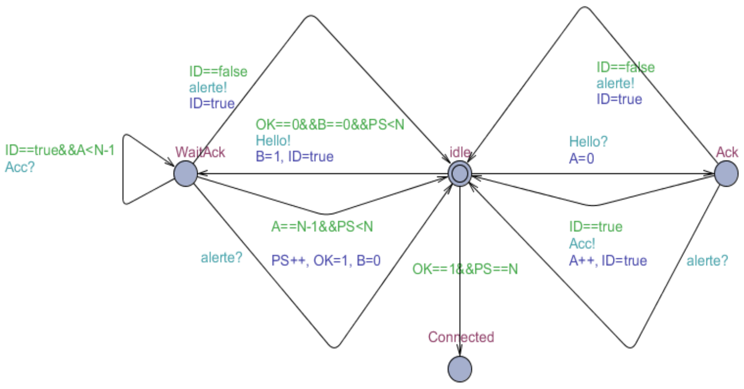

Figure 2 TA describing a sensor in nominal conditions

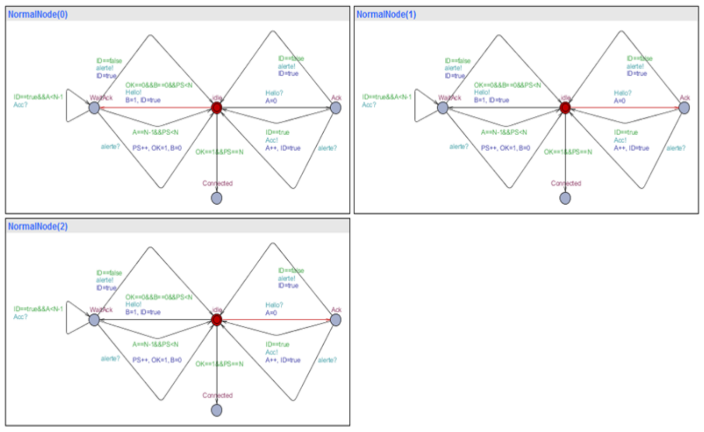

Figure 3 NTA of a WSN in nominal conditions

D. Increased Specification

Sensors are deployed in a hostile environment which certainly causes the partial or total degradation of the network. This is due to the accumulation of either time or energy depletion. In this context, we consider that the degradation could be caused before the T_{min} imposed by the authors in [18]. We increase the nominal specification of a sensor by adding suspension traces. These are useful actions that should be insured by the implementation under test. Suspensions traces are defined as follows:

- **Deadlock**: the system cannot evolve because ($e > 0$) is not satisfied, we add a location called ***DEAD*** (guard $e == 0$) to which the system converges if the sensor's energy is spent.
- **Livelock:** the system diverges to an infinite sequence of actions because the guard ($timeout < T_{min}$) is not satisfied, we add the location ***Blocked*** to which the system evolves if T_{min} runs out.

An ***increased TA*** is a tuple $A=(Q_A, q_0, X, Act_A, E_A, I_A)$ where : $Q_A = Q \cup \{DEAD\} \cup \{Blocked\}$, $Act_A = Act \cup K$ is a finite set of actions with K is a combine action, $E_A \subseteq Q_A \times Act_A \times G(X) \times 2^X \times Q_A$ is the set of edges between locations with actions, a guard and a set of clocks to be reset and $I_A : QA \rightarrow G(X)$ assigns invariants to locations. The increased specification of a sensor using LEAP is presented in Figure 4.

An ***increased NTA*** is defined as follows: $SenA_i = \{Q_{Ai}, q^0_{i}, X, Act_{Ai}, E_{Ai}, I_{Ai}\}$, $\forall i \in [0, n[$ with n the number of all increased TA interacting. Figure 5 presents the increased NTA in a hostile environment.

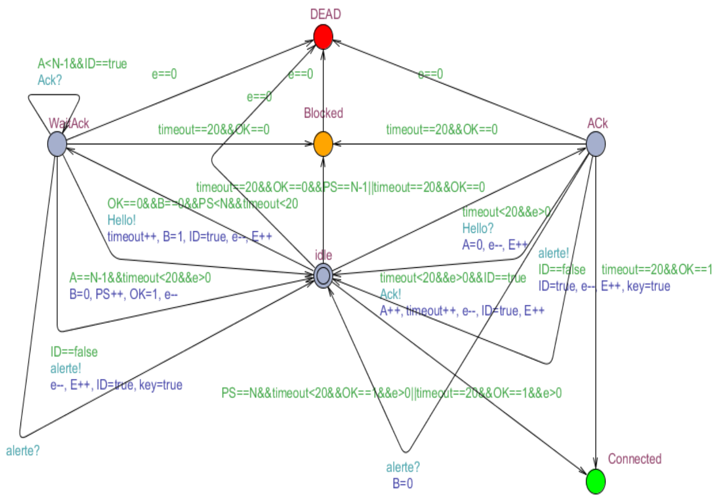

Figure 4 Increased TA of a sensor in hostile environment

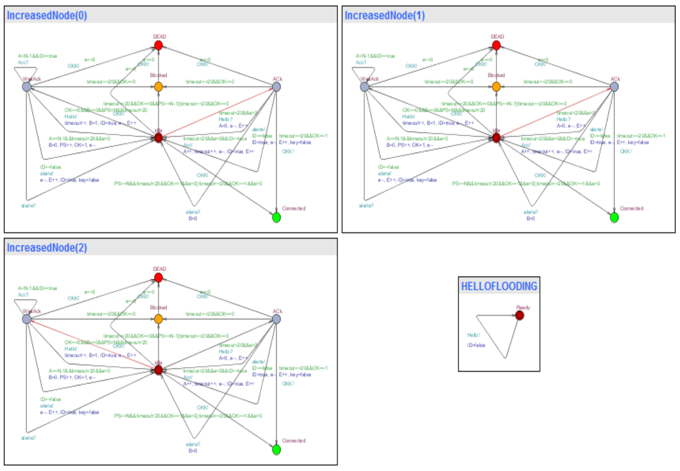

Figure 5 Increased NTA of a WSN in a hostile environment

E. Synchronized Product

1. Robustness test purpose

Informally, the test purpose describes the behavior of the implementation that the tester intends to test. This phase allows deriving and selecting execution traces that assimilate invalid entries, inappropriate entries and acceptable exits [12]. The generation of robustness test cases is done to compute traces satisfying a specific need called robustness test purpose RTP from the increased NTA. An RTP corresponds to a property or a need that the tester wishes to observe on the implementation under test. Under the robustness test, an RTP describes some aspects of operations in the hostile environment. In our approach, RTP is defined as a reachability property which asks whether a given state formula Ψ possibly can be satisfied by any reachable location. In other words: "*Is a path starting at the initial state, such as ψ, is eventually satisfied along that path?*". In UPPAAL [10], we express properties using $E <> \psi$ and $A[]\ \psi$.

RTP is a TA with a final location "*Accept*" describing the behavior accepted by the System Under Robustness Testing and optionally another final state "*Reject*" describing the behavior rejected by the System Under Robustness Testing. *RTC* corresponds to a path starts from the initial location from *Syn* and stops in the state "*Accept*" or "*Reject*". In our work, if all sensors reach the *Connected* location, *RTP* reaches *Accept* location. Otherwise, it reaches *Reject* location. RTP is shown in Figure 6.

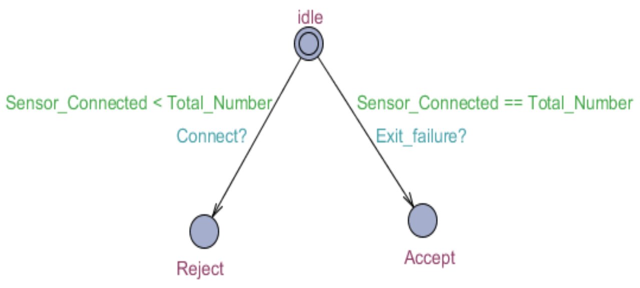
idle
Sensor_Connected < Total_Number
Connect?
Sensor_Connected == Total_Number
Exit_failure?
Reject
Accept

Figure 6 Robustness Test Purpose for WSN using LEAP

2. Synchronized product between increased NTA and an RTP

In the previous sections, we defined formalisms modeling an increased NTA and RTP of an implementation under test. In this section, we present the synchronized product between these. Intuitively, a synchronized product of two transition systems describing increased NTA and robustness test purpose is defined as ***Syn*** such that all timed words are recognized by both increased specification and robustness test purpose transition systems. Let ***SenA*** be a transition system describing the *increased NTA* and *RTP* is a transition system describing the robustness test purpose. The synchronized product is a NTA such as $Syn = SenA \otimes RTP$ where $Q_{Syn} = \{(q_1, q_2) \mid q_1 \in Q_{SenA}, q_2 \in Q_{RTP}\}$, $q^0_{Syn} = (q^0_{SenA}, q^0_{RTP})$, $X_{Syn} = X_{SenA} \cup X_{RTP}$, $Act_{Syn} = Act_{SenA} \cup Act_{RTP}$, $E_{Syn} = E_{SenA} \cup E_{RTP}$ and $I_{Syn} = I_{SenA} \cup I_{RTP}$. The synchronized product is defined as a transition system $(Syn_Q, s_0, \hookrightarrow)$, where $\overline{Syn_Q} = (Q_1 \times Q_n \times Q_{RTP}) \times \mathbb{R}^X_{Syn}$ is the set of states, $s_0 = ((\overline{q_0}, q), u_0)$ is the initial state where $(\overline{q_0}, q) \in Q_{Syn}$ and $\hookrightarrow \subseteq$ Syn × Syn. The synchronized product is shown in Figure 7.

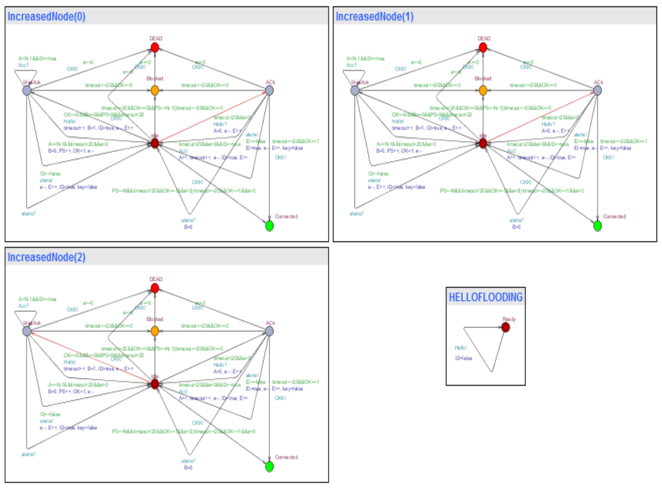

Figure 7 Synchronized product

III. Generation of robustness test cases

A. *Robustness Test Case*

A robustness test case ***RTC*** is represented by a timed word of the NTA corresponding to the synchronized product between increased Specification and a chosen RTP.

Computation sequence: A computation sequence is a finite sequence of pairs (s_i, τ_i) where $s_i = ((\overline{q_i}, q)\, u_i)$ with $u_i \in I(\overline{q_i})$ and τ_i is the observation clock value. Let C be a set of configurations over *Syn*. A computation sequence is defined as follows $\sigma = (s_0, \tau_0) \rightsquigarrow (s_1, \tau_1) \rightsquigarrow \ldots \rightsquigarrow (s_n, \tau_n)$. *CS(Syn)* is the set of computation sequences.

Timed Word: A timed word is a finite sequence of timed actions. A timed action is $\boldsymbol{a\tau}$ pair a where $a \in Act_{Syn}$ and $\tau \in \mathbb{R}_+$, meaning that action a takes place when the observation clock is equal to τ. A timed word is a sequence $\omega = a_1\tau_1 a_2\tau_2 \ldots a_n\tau_n$ where a_i is an action and τ_i is a value of the observation clock. We notice that $\tau_i \leq \tau_{i+1}$. Let *L(Syn)* be the set of timed words of *Syn*,

$$\omega = a_1\tau_1 a_2\tau_2 \ldots a_n\tau_n \in \mathrm{L(Syn)}$$
$$\Leftrightarrow$$
$$\sigma = (s_0, \tau_0) \rightsquigarrow (s_1, \tau_1) \rightsquigarrow \ldots \rightsquigarrow (s_n, \tau_n)$$

We write $(s_0, \tau_0) \overset{\omega}{\rightsquigarrow} (s_n, \tau_n)$. Let ω be a timed word and $a \in$ Act, $\tau \in \mathbb{R}_+$ such that $\tau_n \leq \tau$, then we denote by $\omega.a\tau$ the timed word obtained by adding $a\tau$ to ω and we have $\omega.a\tau = a_1\tau_1 a_2\tau_2 \ldots a_n\tau_n\, a\tau$.

B. *Robustness Relation*

Sensor Network Under Robustness Test (SNURT) is the implementation under test of a WSN. SNURT is modeled by a NTA such that:

- $ACT^{SYN}_{IN} \subseteq ACT^{SNURT}_{IN}$
- $ACT^{SYN}_{OUT} \subseteq ACT^{SNURT}_{OUT}$

Let SNURT can be an implementation of the synchronous product. The robustness relation is defined as follows:

$$\text{SNURT } \mathbf{RoBuST} \text{ Syn}$$
$$\equiv$$
$$\forall \sigma \in \mathrm{CS(Syn)} \Leftrightarrow \mathrm{OUT(SNURT}, \sigma) \subseteq \mathrm{OUT(Syn}, \sigma)$$

Test cases can be generated with respect to the robustness relation.

C. *Generation and Execution of RTCs*

The generation and execution of the test is carried out at the same time. During the generation of test cases:

- if a path reaches *Accept*, then we conclude that the system is *ROBUST*
- if a path reaches *Reject*, then we conclude that the system is *NOTROBUST.*

We check reachability properties described as follows:

- *A[]RTP.Accept* meaning "for all paths on *SNURT*, *RTP* reaches the state *Accept*". If this property is satisfied, then we conclude that *SNURT* is *ROBUST*.
- *E <> RTP.Reject* meaning "Does there exist a path on *SNURT* in which *RTP* reaches the state *Reject*". If this property is satisfied, then we conclude that *SNURT* is *NOTROBUST*.

Let ω be an observation. It is a sequence of input and output actions that are either executed or produced by *SNURT*, and followed with its occurrence time. The tester operates as follows: Given ω an observation recorded on *SNURT*, we compute the set of input and output timed actions that can be taken by SNURT after observing ω. The *SNURT* is *NOTROBUST* if an output action is in Reject. It is *ROBUST* if an output action is in Accept. The robustness test continues and the observation ω is updated by concatenating the timed action taken by *SNURT*. Figure 8 illustrates a simulation trace in which an *RTC* is generated where *SNURT* is *NOTROBUST* and Figure 9 shows an RTC where *SNURT* is *ROBUST.*

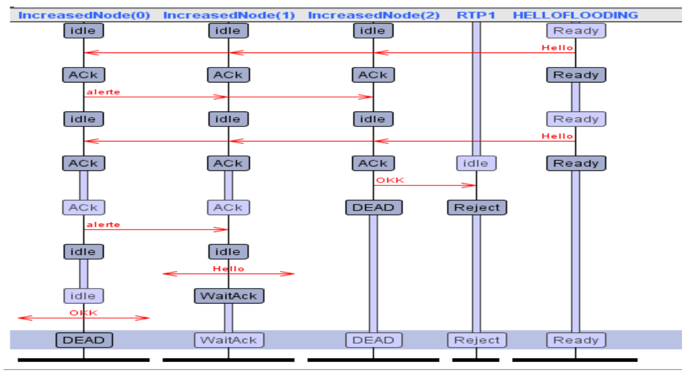

Figure 8 NOT ROBUST RTC

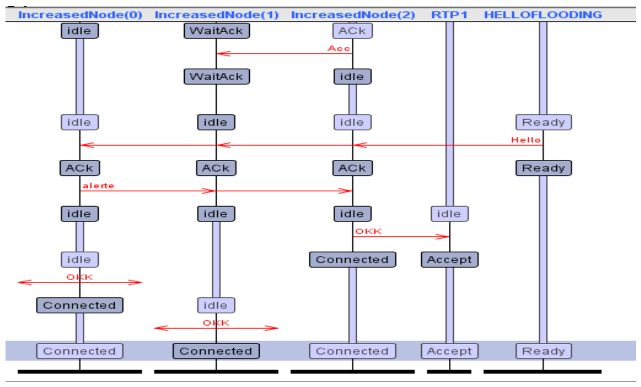

Figure 9 ROBUST RTC

IV. Conclusions

We have introduced a new method to generate and execute robustness test cases. We focus on security protocols for wireless sensor networks. Since sensors are deployed in a hostile environment, we applied our method to demonstrate if WSN is still ROBUST, even with unexpected events. We used the network of timed automata as formalism to model a nominal specification of WSN. Then, we presented an increased specification describing the behavior of a nominal one in a hostile environment. The tester uses a robustness test purpose which is created from a reachability property. Robustness test cases are generated from the increased specification when RTP reaches states REJECT or ACCEPT. After applying the robustness relation, we can infer if an implementation under test of a wireless sensors network using a security protocol is ROBUST or NOT. In our research, we test the robustness of security protocols in a hostile environment. We are primarily interested in systems using security requirements such as confidentiality and integrity of exchanged data. We are planning to test the non-interference property in component based systems. In order to ensure that this type of system is robust to unexpected events, the data exchanged must respond to non-interference property. Therefore, confidentiality and integrity must be ensured throughout the communication process.

References

[1] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking for real-time systems. In LICS, pages 414_425, 1990.
[2] R. Alur and D. L. Dill. A theory of timed automata. Theoretical Computer Science, 126(2):183 _ 235, 1994.
[3] D. Culler, D. Estrin, and M. Srivastava. Guest editors' introduction: Overview of sensor networks. Computer, 37(8):41_49, Aug.
[4] J.-C. Fernandez, L. Mounier, and C. Pachon. A model-based approach for robustness testing. In Proceedings of the 17th IFIP TC6/WG 6.1 international conference on Testing of Communicating Systems, TestCom'05, pages 333_348, Berlin, Heidelberg, 2005. Springer-Verlag.
[5] Y. Fu and O. Koné. Security and robustness by protocol testing. Systems Journal, IEEE, PP(99):1, 2012.
[6] A. Helmy and D. Estrin. Simulation-based 'stress' testing case study: A multicast routing protocol. In Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '98, pages 36_43, 1998.
[7] A. Hessel, K. G. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, and A. Skou. Online testing of real-time systems using uppaal. In International workshop on formal approches to testing of software co-located with IEEE conference on automates software engineering, 2004.
[8] A. Hessel, K. G. Larsen, B. Nielsen, P. Pettersson, and A. Skou. Time-optimal real-time test case generation using uppaal. In In FATES.03, pages 114_130. Springer.Verlag, 2003.
[9] P. Koopman, K. DeVale, and J. DeVale. Interface robustness testing: Experiences and lessons learned from the ballista project, 2008.
[10] K. G. Larsen, M. Mikucionis, and B. Nielsen. Online testing of real-time systems using uppaal. In Proceedings of the 4th international conference on Formal Approaches to Software Testing, FATES'04, pages 79_94, Berlin, Heidelberg, 2005. Springer-Verlag.
[11] A. Rollet. Testing robustness of real time embedded systems.
[12] A. Rollet and H. Fouchal. Testing protocol robustness. T.; Heyer, G. & Unger, H. *(Eds.)*, Innovative Internet Community Systems, volume 2877 of Lecture Notes in Computer Science, pages 201_215. Springer Berlin Heidelberg, 2003.
[13] A. Shahrokni and R. Feldt. Robustest: A framework for automated testing of software robustness. In Software Engineering Conference (APSEC), 2011 18th Asia Paci_c, pages 171 _178, dec. 2011.
[14] J. Springintveld, F. Vaandrager, and P. R. D'Argenio. Testing timed automata. In IN B. JONSSON AND J. PARROW (EDS.), PROC. FTRTFT'96, LNCS 1135, pages 130_147. Springer, 1996.
[15] J. Stankovic. Wireless sensor networks. Computer, 41(10):92_95, Oct.
[16] M. Timmer, E. Brinksma, and M. Stoelinga. Model-based testing.
[17] J. Tretmans and A. Belinfante. Automatic testing with formal methods, 2000.
[18] S. Zhu. Leap: Efficient security mechanisms for large-scale distributed sensor networks. *pages 62_72. ACM Press, 2003.*