Chapter 26
Security Testing of Internet of Things for Smart City Applications: A Formal Approach

Moez Krichen, Mariam Lahami, Omar Cheikhrouhou, Roobaea Alroobaea, and Afef Jmal Maâlej

26.1 Introduction

Internet of Things (IoT) is a promising technology that permits to connect everyday things or objects to the Internet by giving them the capabilities to sense the environment and interact with other objects and/or human beings through the Internet. This evolving technology has promoted a new generation of innovative and valuable services. Today cities are getting smarter by deploying intelligent systems for traffic control, water management, energy management, public transport, street lighting, etc., thanks to these services. Nevertheless, these services can easily be compromised and attacked by malicious parties in the absence of proper mechanism for providing adequate security.

Recent studies have shown that the attackers are using smart home appliances to launch serious attacks such as infiltrating to the network or sending malicious email or launching malicious actions such as distributed denial of service (DDoS) attack. Therefore, security solutions need to be proposed, set up, and tested to mitigate these identified attacks.

M. Krichen (✉)
Faculty of CSIT, Al-Baha University, Al Baha, Saudi Arabia

ReDCAD Laboratory, University of Sfax, Sfax, Tunisia
e-mail: moez.krichen@redcad.org

M. Lahami · A. J. Maâlej
ReDCAD Laboratory, University of Sfax, Sfax, Tunisia
e-mail: mariam.lahami@redcad.org; afef.jmal@redcad.org

O. Cheikhrouhou · R. Alroobaea
Taif University, Taif, Saudi Arabia
e-mail: o.cheikhrouhou@tu.edu.sa; r.robai@tu.edu.sa

R. Mehmood et al. (eds.), *Smart Infrastructure and Applications*,
EAI/Springer Innovations in Communication and Computing,
https://doi.org/10.1007/978-3-030-13705-2_26

In this work, we aim to adopt a model-based security testing (MBST) approach to check the security of IoT applications in the context of smart cities. The MBST approach consists in specifying the desired IoT application in an abstract manner using an adequate formal specification language and then deriving test suites from this specification to find security vulnerabilities in the application under test in a systematic manner.

The work introduced here is an extension of a previous work [19] and it is a piece of a broader approach dealing with the security of IoT applications for smart cities and consisting of the following steps:

- Identify and assess the threats and the attacks in smart cities IoT applications.
- Design and develop security mechanisms for standard protocols at the application and the network layer.
- Evaluate the performance and the correctness of the proposed security protocols using simulation and implementation on real devices.

The rest of this paper is organized as follows. Section 26.2 introduces some preliminaries about IoT and smart cities. Section 26.3 defines the model of extended timed automata. Section 26.4 presents our conformance testing framework. Section 26.5 presents an overview about our proposed approach. Section 26.6 reports on related research efforts dealing with IoT security testing. Finally Sect. 26.7 concludes the paper.

26.2 Preliminaries

26.2.1 Internet of Objects

Recent advances in communication and sensing devices make our everyday objects smarter. This smartness is resulted from the capability of objects to sense the environment, to process the captured (sensed) data, and to communicate it to users either directly or through the Internet. Taking an example of the object "lamp," a classical lamp needs to be wired, linked to the electricity in order to produce light and it does not handle more than the on and off states. This lamp becomes smarter if it is equipped with sensors that can detect environment luminosity and adjust its brightness automatically based on the sensed value. Moreover, this lamp can be equipped with a communication system and therefore can be remotely controlled and supervised (e.g., energy consumption). This example can be generalized to any other thing (object) and therefore leading to the Internet of Things (IoT) concept. The IoT refers to the ability of everyday objects to connect to the Internet and to send and receive data. The integration of these smart objects to the Internet infrastructure is promoting a new generation of innovative and valuable services for people. These services include home automation, traffic control, public transportation, smart water

metering, waste and energy management, etc. When integrated in a city context, they make citizens live better and so form the modern smart city.

26.2.2 Smart Cities

In the recent years, several research works are shaping the smart cities concept [4, 34]. In October 2015, ITU-T's Focus Group on Smart Sustainable Cities (FG-SSC)[1] agreed on the following definition of a smart sustainable city: A smart sustainable city (SSC) is an innovative city that uses information and communication technologies (ICTs) and other means to improve quality of life, efficiency of urban operation and services, and competitiveness, while ensuring that it meets the needs of present and future generations with respect to economic, social, and environmental aspects. Based on this definition, the main goal for SSC is to enhance the quality of life of its citizens across multiple, interrelated dimensions, including (but not limited to) the provision and access to water resources, energy, transportation and mobility, education, environment, waste management, housing, and livelihoods (e.g., jobs), utilizing ICTs as the key medium. Therefore, the IoT as a promising ICT technology will play a major role in the development of these new smart cities. With IoT, objects like phones, cars, household appliances, or clothes become wirelessly connected and can sense and share data.

26.2.3 Threats

Indeed, connecting our everyday things to the public Internet opens these objects to several kinds of attacks. Take the example of a traffic control system. If the hackers could insert fake messages to these traffic control system devices, they can make traffic perturbations and bottlenecks. Another example related to home automation, if attackers gain access to smart devices such as lamps and doors, it could manipulate doors and steal the house properties. The main security threats in the IoT are summarized in [8] as follows:

- Cloning of smart things by untrusted manufacturers;
- Malicious substitution of smart things during installation;
- Firmware replacement attack;
- Extraction of security parameters since smart things may be physically unprotected;
- Eavesdropping attack if the communication channel is not adequately protected;
- Man-in-the-middle attack during key exchange;

[1] https://www.itu.int/en/ITU-T/focusgroups/ssc.

– Routing attacks;
– Denial-of-service attacks; and
– Privacy threats.

Therefore, a key challenge for IoT towards smart city applications is ensuring its reliability, security, and privacy, which represent the main scope of this work. Without guarantees that smart city IoT objects are: (1) sensing correctly the environment, (2) exchanging the information securely, (3) safeguarding private information, and (4) providing correct and not manipulated information, users are reluctant to adopt this new technology that will be a part of their everyday lives.

26.2.4 Challenges

Due to its specific characteristic, new issues are raised in the area of IoT. Trust management, which plays an important role in the IoT for reliable data fusion, qualified services, and enhanced user privacy and information security, is one of these main issues. Indeed, data collection trust is a crucial issue in the IoT. If the huge collected data is not trusted (e.g., due to the damage or malicious input of some sensors), the IoT service quality will be greatly influenced and hard to be accepted by users even though the network layer trust and the application layer trust can be fully provided [36]. On the other hand, in order to have intelligent context-aware services, users should disclose or have to share their personal data or privacy such as location, preferences, and contacts. Providing intelligent context-aware and personalized services and at the same time preserving user privacy are two conflicting objectives that induce a big challenges in the IoT. Another challenge faced when designing security solutions to the IoT is the limited resources of the IoT devices. Indeed, most of IoT devices are limited in terms of CPU, memory capacity, and battery supply. They often operate on lossy and low bandwidth communication channels. This renders the application of the conventional Internet security solutions not appropriate. As an example, the use of small packets (i.e., IEEE 802.15.4 supports only 127-bytes packets) may result in fragmentation of larger packets when using the standard protocols. This will quickly exhaust the lifetime of IoT devices and open new possibilities for DoS attacks [25]. Moreover, the limited resources of IoT devices render the use of complex cryptographic protocols inadequate. Finally, the inherent complexity of the IoT, where multiple heterogeneous entities located in different contexts can exchange information with each other, further complicates the design and the deployment of efficient, interoperable, and scalable security mechanisms [28]. The proposed security solutions should fulfill the following security requirements [26].

– Data confidentiality: make information inaccessible to unauthorized users. For example, a 6LoWPAN node should not leak some of its collected data to neighboring networks.

- Data authentication: since an adversary can easily inject messages, the receiver needs to ensure that data are originated from trusted sources.
- Data integrity: ensures that an adversary does not alter the received data in transit.
- Availability: ensures the survivability of network services to (only) authorized parties when needed, despite a DoS attack(s).
- Energy efficiency: a security scheme must be energy efficient so as to maximize the network lifetime.

26.3 Extended Timed Automata

We extend the framework presented in [18].

26.3.1 Timed Labeled Transition Systems

Let R be the set of non-negative reals, Q the set of non-negative rationals, and N the set of non-negative integers. Given a finite set of *actions* Ac, the set $(\mathsf{Ac} \cup \mathsf{R})^*$ of all finite-length *real-time sequences* over Ac will be denoted $\mathsf{RT}(\mathsf{Ac})$. $\epsilon \in \mathsf{RT}(\mathsf{Ac})$ is the empty sequence. Given $\mathsf{Ac}' \subseteq \mathsf{Ac}$ and $\rho \in \mathsf{RT}(\mathsf{Ac})$, $\mathsf{P}_{\mathsf{Ac}'}(\rho)$ denotes the *projection* of ρ to $\mathsf{Ac}' \cup \mathsf{R}$, obtained by "erasing" from ρ all actions not in $\mathsf{Ac}' \cup \mathsf{R}$. Similarly, $\mathsf{DP}_{\mathsf{Ac}'}(\rho)$ denotes the (discrete) projection of ρ to Ac'. For example, if $\mathsf{Ac} = \{a, b\}$, $\mathsf{Ac}' = \{a\}$ and $\rho = a\,1\,b\,2\,a\,3$, then $\mathsf{P}_{\mathsf{Ac}'}(\rho) = a\,3\,a\,3$ and $\mathsf{DP}_{\mathsf{Ac}'}(\rho) = a\,a$. The time spent in a sequence ρ, denoted $\mathsf{duration}(\rho)$, is the sum of all delays in ρ, for example, $\mathsf{duration}(\epsilon) = 0$ and $\mathsf{duration}(a\,1\,b\,0.5) = 1.5$. In the rest of the document, we assume given a set of actions Ac, partitioned in two disjoint sets: a set of *input actions* $\mathsf{Ac}_{\mathsf{in}}$ and a set of *output actions* $\mathsf{Ac}_{\mathsf{out}}$. Actions in $\mathsf{Ac}_{\mathsf{in}} \cup \mathsf{Ac}_{\mathsf{out}}$ are called *observable* actions. We also assume there is an *unobservable* action $\tau \notin \mathsf{Ac}$. Let $\mathsf{Ac}_\tau = \mathsf{Ac} \cup \{\tau\}$. A *timed labeled transition system* (TLTS) over Ac is a tuple $(S, s_0, \mathsf{Ac}, T_d, T_t)$, where:

- S is a set of *states*;
- s_0 is the initial state; T_d is a set of *discrete transitions* of the form (s, a, s') where $s, s' \in S$ and $a \in \mathsf{Ac}$;
- T_t is a set of *timed transitions* of the form (s, t, s') where $s, s' \in S$ and $t \in \mathsf{R}$.

Timed transitions must be deterministic, that is, $(s, t, s') \in T_t$ and $(s, t, s'') \in T_t$ implies $s' = s''$. T_t must also satisfy the following conditions: $(s, t, s') \in T_t$ and $(s', t', s'') \in T_t$ implies $(s, t + t', s'') \in T_t$; $(s, t, s') \in T_t$ implies that for all $t' < t$, there is some $(s, t', s'') \in T_t$.

We use standard notation concerning TLTS. For $s, s', s_i \in S$, $\mu, \mu_i \in \mathsf{Ac}_\tau \cup \mathsf{R}$, $a, a_i \in \mathsf{Ac} \cup \mathsf{R}$, $\rho \in \mathsf{RT}(\mathsf{Ac}_\tau)$ and $\sigma \in \mathsf{RT}(\mathsf{Ac})$, we have

- General transitions:
  - $s \xrightarrow{\mu} s' \stackrel{Def}{=} (s, \mu, s') \in T_d \cup T_t$; $s \xrightarrow{\mu} \stackrel{Def}{=} \exists s' : s \xrightarrow{\mu} s'$;
  - $s \not\xrightarrow{\mu} \stackrel{Def}{=} \nexists s' : s \xrightarrow{\mu} s'$;
  - $s \xrightarrow{\mu_1 \cdots \mu_n} s' \stackrel{Def}{=} \exists s_1, \cdots, s_n : s = s_1 \xrightarrow{\mu_1} s_2 \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_n} s_n = s'$;
  - $s \xrightarrow{\rho} \stackrel{Def}{=} \exists s' : s \xrightarrow{\rho} s'$;
  - $s \not\xrightarrow{\rho} \stackrel{Def}{=} \nexists s' : s \xrightarrow{\rho} s'$.
- Observable transitions:
  - $s \stackrel{\epsilon}{\Rightarrow} s' \stackrel{Def}{=} s = s'$ or $s \xrightarrow{\tau \cdots \tau} s'$;
  - $s \stackrel{a}{\Rightarrow} s' \stackrel{Def}{=} \exists s_1, s_2 : s \stackrel{\epsilon}{\Rightarrow} s_1 \xrightarrow{a} s_2 \stackrel{\epsilon}{\Rightarrow} s'$;
  - $s \stackrel{a}{\not\Rightarrow} \stackrel{Def}{=} \nexists s' : s \stackrel{a}{\Rightarrow} s'$; $s \stackrel{a_1 \cdots a_n}{\Longrightarrow} s' \stackrel{Def}{=} \exists s_1, \cdots, s_n : s = s_1 \stackrel{a_1}{\Rightarrow} s_2 \stackrel{a_2}{\Rightarrow} \cdots \stackrel{a_n}{\Rightarrow} s_n = s'$;
  - $s \stackrel{\sigma}{\Rightarrow} \stackrel{Def}{=} \exists s' : s \stackrel{\sigma}{\Rightarrow} s'$;
  - $s \stackrel{\sigma}{\not\Rightarrow} \stackrel{Def}{=} \nexists s' : s \stackrel{\sigma}{\Rightarrow} s'$.

A sequence of the form $s_0 \xrightarrow{\mu_1} s \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_n} s'$ is called a *run* and a sequence of the form $s_0 \stackrel{a_1}{\Rightarrow} s \stackrel{a_2}{\Rightarrow} \cdots \stackrel{a_n}{\Rightarrow} s'$ an *observable run*.

26.3.2 Extended Timed Automata

We use timed automata [2] with deadlines to model urgency [18]. An *extended timed automaton over* Ac is a tuple $A = (Q, q_0, X, \mathsf{I}, \mathsf{Ac}, \mathsf{E})$, where:

- Q is a finite set of *locations*;
- $q_0 \in Q$ is the initial location;
- X is a finite set of *clocks*;
- I is a finite set of integer variables;
- E is a finite set of *edges*.

Each edge is a tuple $(q, q', \psi, r, inc, dec, d, a)$, where:

- $q, q' \in Q$ are the source and destination locations;
- ψ is the *guard*, a conjunction of constraints of the form $x\#c$, where $x \in X \cup \mathsf{I}$, c is an integer constant and $\# \in \{<, \leq, =, \geq, >\}$;
- $r \subseteq X \cup \mathsf{I}$ is a set of clocks and integer variables to *reset* to zero;
- $inc \subseteq \mathsf{I}$ is a set of integer variables (disjoint from r) to *increment* by one;
- $dec \subseteq \mathsf{I}$ is a set of integer variables (disjoint from r and inc) to *decrement* by one;
- $d \in \{\mathsf{lazy}, \mathsf{delayable}, \mathsf{eager}\}$ is the *deadline*;
- $a \in \mathsf{Ac}$ is the action.

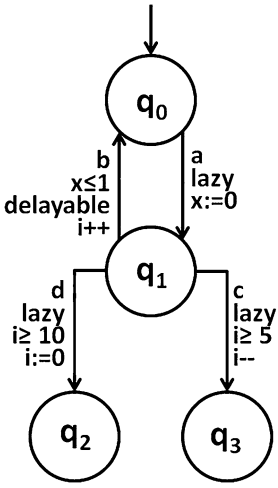

Fig. 26.1 An example of an extended timed automaton

An example of an extended timed automaton $A = (Q, q_0, X, \mathsf{I}, \mathsf{Ac}, \mathsf{E})$ over the set of actions $\mathsf{Ac} = \{a, b, c, d\}$ is given in Fig. 26.1 where:

- $Q = \{q_0, q_1, q_2, q_3\}$ is the set of locations;
- q_0 is the initial location;
- $X = \{x\}$ is the finite set of clocks;
- $\mathsf{I} = \{i\}$ is the finite set of integer variables;
- E is the set of edges drawn in the figure.

The figure uses the following notation:

- "$x := 0$" means resetting the clock x to 0;
- "$i := 0$" means resetting the integer variable i to 0;
- "$i + +$" means incrementing i by 1;
- "$i - -$" means decrementing i by 1.

26.3.3 Semantics of Extended Timed Automata

An extended timed automaton $A = (Q, q_0, X, \mathsf{I}, \mathsf{Ac}, \mathsf{E})$ defines an infinite TLTS which is denoted $L_A = (S_A, s_0^A, \mathsf{Ac}, T_d^A, T_t^A)$.

- Its states S_A are tuples $s = (q, v_X, v_\mathsf{I})$, where:
  - $q \in Q$;
  - $v_X : X \to \mathsf{R}$ is a *clock valuation*;
  - and $v_\mathsf{I} : \mathsf{I} \to \mathsf{N}$ is a *integer variable valuation*.

- $s_0^A = (q_0, \mathbf{0}_X, \mathbf{0}_I)$ is the initial state, where:
  - $\mathbf{0}_X$ is the valuation assigning 0 to every clock of A;
  - $\mathbf{0}_I$ is the valuation assigning 0 to every integer variable of A.
- *Discrete transitions* are of the form $(q, v_X, v_I) \xrightarrow{a} (q', v'_X, v'_I)$ where $a \in \mathsf{Ac}$ and there is an edge $(q, q', \psi, r, inc, dec, d, a)$ such that (v_X, v_I) satisfies ψ and (v'_X, v'_I) is obtained by:
  - resetting to zero all clocks and integer variables in r;
  - incrementing integer variables in *inc* by one;
  - decrementing variables in *dec* by one;
  - leaving all other variables unchanged.
- *Timed transitions* are of the form $(q, v_X, v_I) \xrightarrow{t} (q, v_X + t, v_I)$ where $t \in \mathsf{R}, t > 0$ and there is no edge $(q, q'', \psi, r, inc, dec, d, a)$ such that:
  - either $d = \mathsf{delayable}$ and there exist $0 \leq t_1 < t_2 \leq t$ such that $(v_X + t_1, v_I) \models \psi$ and $(v_X + t_2, v_I) \not\models \psi$;
  - or $d = \mathsf{eager}$ and $(v_X, v_I) \models \psi$.

Lazy edges do not impact the semantics. They denote that an edge is neither delayable nor eager. More precisely, lazy edges cannot block time progress, whereas delayable and eager edges can. We do not allow delayable edges with guards of the form $x < c$ since there is no *latest* time when the guard is still true. Similarly, we do not allow eager edges with guards of the form $x > c$ since there is no *earliest* time when the guard becomes true.

A state $s \in S_A$ is *reachable* if there exists $\rho \in \mathsf{RT}(\mathsf{Ac})$ such that $s_0^A \xrightarrow{\rho} s$. The set of reachable states of A is denoted $\mathsf{Reach}(A)$.

For instance, for the TA presented in Fig. 26.1, the initial state is $(q_0, 0, 0)$. A possible timed transition of the system is $(q_0, 0, 0) \xrightarrow{5} (q_0, 5, 0)$ which corresponds to the fact that the system spends 5 time units at node q_0. A possible discrete transition is $(q_0, 5, 0) \xrightarrow{a} (q_1, 0, 0)$ which corresponds to the execution of action a and results in the reset of clock x to 0. Another possible discrete transition is $(q_1, 0, 0) \xrightarrow{b} (q_0, 0, 1)$ by which the integer variable i is incremented for the first time. The time constraint $x \leq 1$ means that the execution of b must happen at most 1 time unit after the execution of a. The deadline delayable associated with this constraint means that time is blocked after one time unit and that it is compulsory to execute action b before that limit. It is not difficult to see that action b needs to be executed at least 5 times (resp., 10 times) in order to execute action c (resp., d).

26.3.4 Extended Timed Automata with Inputs and Outputs

An *extended timed automaton with inputs and outputs* (ETAIO) is an extended timed automaton over the partitioned set of actions $\mathsf{Ac}_\tau = \mathsf{Ac}_{\mathsf{in}} \cup \mathsf{Ac}_{\mathsf{out}} \cup \{\tau\}$. For clarity, we will explicitly include inputs and outputs in the definition of an ETAIO A and write $(Q, q_0, X, \mathsf{I}, \mathsf{Ac}_{\mathsf{in}}, \mathsf{Ac}_{\mathsf{out}}, \mathsf{E})$ instead of $(Q, q_0, X, \mathsf{I}, \mathsf{Ac}_\tau, \mathsf{E})$.

- An ETAIO is called *observable* if none of its edges is labeled by τ.
- Given a set of inputs $\mathsf{Ac}' \subseteq \mathsf{Ac}_{\mathsf{in}}$, an ETAIO A is called *input-enabled* with respect to Ac' if it can accept any input in Ac' at any state:

$$\forall s \in \mathsf{Reach}(A) \,.\, \forall a \in \mathsf{Ac}' \,:\, s \xrightarrow{a} .$$

  It is simply said to be input-enabled when $\mathsf{Ac}' = \mathsf{Ac}_{\mathsf{in}}$.
- A is called lazy-*input* with respect to Ac' if the deadlines on all the transitions labeled with input actions in Ac' are lazy. It is called lazy-input if it is lazy-input with respect to $\mathsf{Ac}_{\mathsf{in}}$. Note that input-enabled does not imply lazy-input in general.
- A is called *deterministic* if:

$$\forall s, s', s'' \in \mathsf{Reach}(A) \,.\, \forall a \in \mathsf{Ac}_\tau \,:\, s \xrightarrow{a} s' \wedge s \xrightarrow{a} s'' \Rightarrow s' = s''.$$

- A is called *non-blocking* if:

$$\forall s \in \mathsf{Reach}(A) \,.\, \forall t \in \mathsf{R} \,.\, \exists \rho \in \mathsf{RT}(\mathsf{Ac}_{\mathsf{out}} \cup \{\tau\}) \,:\, \mathsf{duration}(\rho) = t \wedge s \xrightarrow{\rho} .$$

  This condition guarantees that A will not block time in any environment.

  The set of *timed traces* of an ETAIO A is defined to be

$$\mathsf{TTr}(A) = \{\rho \mid \rho \in \mathsf{RT}(\mathsf{Ac}_\tau) \wedge s_0^A \xrightarrow{\rho}\}.$$

  The set of *observable timed traces* of A is defined to be

$$\mathsf{OTTr}(A) = \{\mathsf{P}_{\mathsf{Ac}}(\rho) \mid \rho \in \mathsf{RT}(\mathsf{Ac}_\tau) \wedge s_0^A \xrightarrow{\rho}\}.$$

The TLTS defined by an ETAIO is called a *timed input–output LTS* (TIOLTS). From now on, unless otherwise stated, all the considered ETAIO are defined with respect to the same sets $\mathsf{Ac}_{\mathsf{in}}$ and $\mathsf{Ac}_{\mathsf{out}}$ and unobservable action τ. As for ETAIO, a given TIOLTS L is denoted $(S, s_0, \mathsf{Ac}_{\mathsf{in}}, \mathsf{Ac}_{\mathsf{out}}, T_d, T_t)$ instead of $(S, s_0, \mathsf{Ac}_\tau, T_d, T_t)$. The two operators $\mathsf{TTr}(\cdot)$ and $\mathsf{OTTr}(\cdot)$ are extended in a natural way to the case of TIOLTS.

26.3.5 Parallel Composition of ETAIO with Shared Integer Variables

The parallel composition we propose here is similar to the parallel composition for classical timed automata. The new thing here is that we consider shared variables between the different elements to compose. The shared variables can be incremented and decremented by any participant in the composition. These variables are used to formulate the constraints of the different automata. In this way the behaviors of the different components of the system are related to each other and depend on each other. For instance, the shared variables may represent the shared resources of the system.

Let n be a non-negative integer such that $n \geq 2$. We consider n ETAIO $(A_i)_{1 \leq i \leq n}$ where $A_i = (Q^i, q_0^i, X^i, \mathsf{I}, \mathsf{Ac}_{\mathsf{in}}^i, \mathsf{Ac}_{\mathsf{out}}^i, \mathsf{E}^i)$. That is the set of integer variables I is shared between all the considered ETAIO $(A_i)_{1 \leq i \leq n}$ while no other element from Q^i, X^i, $\mathsf{Ac}_{\mathsf{in}}^i$, and $\mathsf{Ac}_{\mathsf{out}}^i$ is shared with the other ETAIO $(A_j)_{j \neq i}$.

The TIOLTS $L^P = (S^P, s_0^P, \mathsf{Ac}_{\mathsf{in}}^P, \mathsf{Ac}_{\mathsf{out}}^P, T_d^P, T_t^P)$ generated by the parallel product of the ETAIO $(A_i)_{1 \leq i \leq n}$ is defined as follows:

- $s_0^P = ((q_0^1, \cdots, q_0^n), (\mathbf{0}_{X^0}, \cdots, \mathbf{0}_{X^n}), \mathbf{0}_{\mathsf{I}})$;
- $\mathsf{Ac}_{\mathsf{in}}^P = \bigcup_{1 \leq i \leq n} \mathsf{Ac}_{\mathsf{in}}^i$, $\mathsf{Ac}_{\mathsf{out}}^P = \bigcup_{1 \leq i \leq n} \mathsf{Ac}_{\mathsf{out}}^i$;
- and S^P, T_d^P, and T_t^P are the smallest sets such that
  - $s_0^P \in S^P$;
  - For $s^P = ((q^1, \cdots, q^n), (v_{X^0}, \cdots, v_{X^n}), v_{\mathsf{I}}) \in S^P$ and $\delta \in \mathsf{R}$:

$$\forall 1 \leq i \leq n : (q_i, v_{X^i}, v_I) \xrightarrow{\delta} (q_i, v_{X^i} + \delta, v_I) \in T_t^i$$

$$\Rightarrow s'^P = ((q^1, \cdots, q^n), (v_{X^0} + \delta, \cdots, v_{X^n} + \delta), v_{\mathsf{I}}) \in S^P \; and \; s^P \xrightarrow{\delta} s'^P \in T_t.$$

  - For $s^P = ((q^1, \cdots, q^n), (v_{X^0}, \cdots, v_{X^n}), v_{\mathsf{I}}) \in S^P$, $1 \leq i \leq n$ and $a_i \in \mathsf{Ac}_\tau{}^i = \mathsf{Ac}_{\mathsf{in}}^i \cup \mathsf{Ac}_{\mathsf{out}}^i \cup \{\tau\}$):

$$(q_i, v_{X^i}, v_I) \xrightarrow{a_i} (q_i', v_{X^i}', v_I') \in T_d^i$$

$$\Rightarrow s'^P = (q'^p, v_X'^p, v_{\mathsf{I}}') \in S^P \quad \wedge \quad s^P \xrightarrow{a_i} s'^P \in T_d$$

where

$$q'^p = (q^1, \cdots, q^{i-1}, q'^i, q^{i+1}, \cdots, q^n)$$

and

$$v_X'^p = (v_{X^0}, \cdots, v_{X^{i-1}}, v_{X^i}', v_{X^{i+1}} \cdots, v_{X^n}).$$

It is worth noticing here that it is possible to define the parallel composition of n copies $(A_i)_{1 \leq i \leq n}$ of the same ETAIO A. In this case we assume it is possible to distinguish the sets of inputs and outputs of the different instances by a particular identifier corresponding to each instance. Obviously, the n instances share the set of integer variables of the ETAIO A. The obtained TIOLTS is denoted L_n^P.

26.4 Conformance Testing Framework

In this section, we are going to define a new extended timed input–output conformance relation, etioco. Then, we propose a new approach for deriving analog-clock tests from the SUT specification. Finally, we discuss both test execution and correctness requirements.

26.4.1 Conformance Relation

In order to formally define the conformance relation, we define a number of operators. Given a TIOLTS $L = (S^L, s_0^L, \mathsf{Ac}_{\mathsf{in}}^L, \mathsf{Ac}_{\mathsf{out}}^L, T_d^L, T_t^L)$ and a timed trace $\sigma \in \mathsf{RT}(\mathsf{Ac}^L)$ L after σ is the set of all states of L that can be reached by some timed sequence ρ whose projection to observable actions is σ. Formally: L after $\sigma = \{s \in S^L \mid \exists \rho \in \mathsf{RT}(\mathsf{Ac}_\tau{}^L) : s_0^L \xrightarrow{\rho} s \wedge \mathsf{P}_{\mathsf{Ac}}(\rho) = \sigma\}$. Given state $s \in S^L$, $\mathsf{elapse}(s)$ is the set of all delays which can elapse from s without L making any observable action. Formally: $\mathsf{elapse}(s) = \{t > 0 \mid \exists \rho \in \mathsf{RT}(\{\tau\}) : \mathsf{duration}(\rho) = t \wedge s \xrightarrow{\rho}\}$. Given state $s \in S^L$, $\mathsf{out}(s)$ is the set of all observable "events" (outputs or the passage of time) that can occur when the system is at state s. The definition naturally extends to a set of states S. Formally: $\mathsf{out}(s) = \{a \in \mathsf{Ac}_{\mathsf{out}}^L \mid s \xrightarrow{a}\} \cup \mathsf{elapse}(s)$ and

$$\mathsf{out}(S) = \bigcup_{s \in S} \mathsf{out}(s).$$

The specification of the system to be tested is given as a non-blocking ETAIO A_S while the implementation can be modeled as a non-blocking, input-enabled ETAIO A_I. For $n \geq 1$, let $L_{S,n}^P$ (resp., $L_{I,n}^P$) be the parallel composition of n copies of A_S (resp., A_I). Input-enabledness is required so that the implementation can accept inputs from the tester at any state. The *extended timed input–output conformance relation*, denoted etioco, is an extension of our previous conformance relation tioco [17, 18]. The new relation etioco is defined as A_I etioco A_S *iff* $\forall n \geq 1 \wedge \sigma \in \mathsf{OTTr}(L_{S,n}^P) : \mathsf{out}(L_{I,n}^P \ \mathsf{after}\ \sigma) \subseteq \mathsf{out}(L_{S,n}^P \ \mathsf{after}\ \sigma)$. The relation states that an implementation A_I conforms to a specification A_S iff for any number of copies n of A_S and any observable behavior σ of $L_{S,n}^P$, the set of observable outputs

of $L^P_{I,n}$ after any behavior "matching" σ must be a subset of the set of possible observable outputs of $L^P_{S,n}$. Notice that observable outputs are not only observable output actions but also time delays. Also notice that in case we consider only $n = 1$, the definitions of etioco and tioco become the same.

26.4.2 Analog-Clock Tests

A test (or *test case*) is an experiment performed on the implementation by an agent (the *tester*). There are different types of tests, depending on the capabilities of the tester to observe and react to events. In general, one may consider either *analog-clock* or *digital-clock* tests [11]. In this work, we consider only analog-clock tests. The latter can measure precisely the delay between two observed actions and can emit an input at any point in time.

It should be noted that we consider *adaptive* tests (following the terminology of [21]), where the action the tester takes depends on the observation history. For $n \geq 1$, let Ac^n (resp., $\mathsf{Ac}^n_{\mathsf{in}}$) denote the union of all observable actions (resp., all input actions) of n copies of the specification A_S. An analog-clock test for n parallel executions of A_S is a total function $T_n : \mathsf{RT}(\mathsf{Ac}^n) \rightarrow \mathsf{Ac}^n_{\mathsf{in}} \cup \{\mathsf{Wait}, \mathsf{Pass}, \mathsf{Fail}\}$. $T_n(\rho)$ specifies the action the tester must take once it observes ρ:

- If $T_n(\rho) = a \in \mathsf{Ac}^n_{\mathsf{in}}$, then the tester emits input a.
- If $T_n(\rho) = \mathsf{Wait}$, then the tester waits (lets time elapse).
- If $T_n(\rho) \in \{\mathsf{Pass}, \mathsf{Fail}\}$, then the tester produces a verdict (and stops).

26.4.3 Test Execution and Correctness Requirements

The execution of the test T_n on the implementation A_I can be defined as the *parallel composition* of the TIOLTS defined by T_n and $L^P_{I,n}$ the TIOLTS corresponding to n copies of A_I, with the usual *synchronization* rules for transitions carrying the same label. We will denote the product TIOLTS by $L^P_{I,n} \| T_n$. The execution of the test reaches a pass/fail verdict after bounded time. Formally, we say that A_I *passes* the test, denoted A_I passes T_n, if state Fail is not reachable in the product $L^P_{I,n} \| T_n$. We say that an implementation passes (resp. fails) a set of tests (or *test suite*) $\mathcal{T}$ if it passes all tests (resp. fails at least one test) in $\mathcal{T}$. We say that an analog-clock test suite $\mathcal{T}$ is *sound with respect to* A_S if

$$\forall A_I \ : \ A_I \text{ etioco } A_S \Rightarrow A_I \text{ passes } \mathcal{T}.$$

We say that $\mathcal{T}$ is *complete with respect to* A_S if

$$\forall A_I \ : \ A_I \text{ passes } \mathcal{T} \Rightarrow A_I \text{ etioco } A_S.$$

26.5 Proposed Approach

In this section, we define a workflow that covers the different steps of a classical model-based testing process, namely: model Specification, test generation, test selection, test execution, and evaluation activities as depicted in Fig. 26.2.

26.5.1 Test Generation and Selection

Test Generation We adapt the untimed test generation algorithm of [30]. Roughly speaking, the algorithm builds a test in the form of a tree. A node in the tree is a set of states S of the specification and represents the "knowledge" of the tester at the current test state. The algorithm extends the test by adding successors to a leaf node, as illustrated in Fig. 26.3. For all *illegal* outputs a_i (outputs which cannot occur from any state in S) the test leads to Fail. For each legal output b_i, the test proceeds to node S_i, which is the set of states the specification can be in after emitting b_i (and possibly performing unobservable actions). If there exists an input c which can be accepted by the specification at some state in S, then the test may decide to emit this input (dashed arrow from S to S'). At any node, the algorithm may decide to stop the test and label this node as Pass.

Analog-clock tests cannot be directly represented as a finite tree, because there is an a-priori infinite set of possible observable delays at a given node. To remedy this, we use the idea of [31]. We represent an analog-clock test as an *algorithm*. The latter essentially performs subset construction on the specification automaton, during the

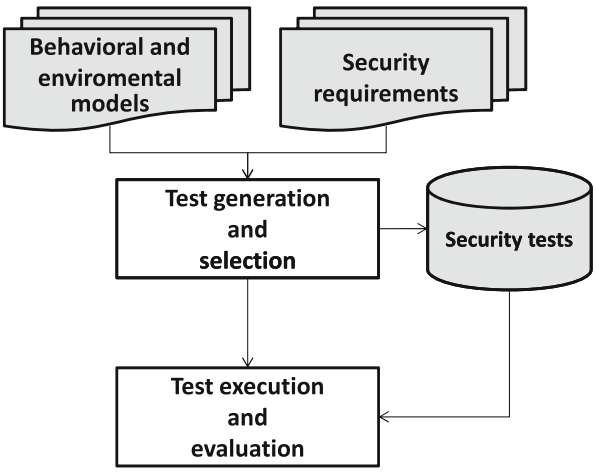

Fig. 26.2 Model-based security testing process

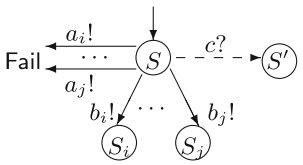

Fig. 26.3 Test generation principle [18]

Algorithm 1 On-the-fly analog-clock test generation

```
1  S ← tsucc({s^P_{n,0}}, 0);
2  while(not Fail)
3    x ← 0; /* x is a clock measuring elapsing time */
4    await(output b is received at x < T or x = T)
5    if (b received at x)
6      S ← dsucc(tsucc(S, x), b);
7    else
8      S ← tsucc(S, T);
9    endif;
10   if (S = ∅)
11     announce Fail;
12     exit ;
13   endif;
14   if (validinputs(S) ≠ ∅)
15     i ← pick({0, 1}); /* 0 to send an input and 1 to continue observation */
16   endif;
17   if (i = 0)
18     a ← pick(validinputs(S));
19     S ← dsucc(S, a);
20   endif;
21 endwhile;
```

execution of the test. Thus, our analog-clock testing method can be classified as on-the-fly or *on-line*, meaning that the test is generated at the same time it is executed. More precisely, the tester will maintain a set of states S of the TIOLTS $L^P_{S,n}$.

S will be updated every time an action is observed or some time delay elapses. Since the time delay is not known a-priori, it must be an input to the update function. We define the following operators:

$$\mathsf{dsucc}(S, a) = \{s' \mid \exists s \in S \ : \ s \xrightarrow{a} s'\}$$

and

$$\mathsf{tsucc}(S, t) = \{s' \mid \exists s \in S \,.\, \exists \rho \in \mathsf{RT}(\{\tau\}) \ : \ \mathsf{duration}(\rho) = t \wedge s \xrightarrow{\rho} s'\}$$

where $a \in \mathsf{Ac}^n$ and $t \in \mathsf{R}$. $\mathsf{dsucc}(S, a)$ contains all states which can be reached by some state in S performing action a. $\mathsf{tsucc}(S, t)$ contains all states which can be reached by some state in S via a sequence ρ which contains no observable actions and takes exactly t time units. The test operates as follows. It starts at state $S_0 = \mathsf{tsucc}(\{s^P_{n,0}\}, 0)$ where $s^P_{n,0}$ is the initial state of $L^P_{S,n}$. Given current state S:

- if output a is received t time units after entering S, then S is updated to $\mathsf{dsucc}(\mathsf{tsucc}(S, t), a)$.
- If ever the set S becomes empty, the test announces Fail.
- At any point, for an input b, if $\mathsf{dsucc}(S, b) \neq \emptyset$, the test may decide to emit b and update its state accordingly.

On-line analog-clock test generation is performed by Algorithm 1. The algorithm keeps running as long as no non-conformance is detected. At any time the tester can stop testing and declare Pass. The algorithm uses the following notation. Given a nonempty set X, $\mathsf{pick}(X)$ chooses randomly an element in X.

Given a set of states S, $\mathsf{validinputs}(S)$ is defined as the set of valid inputs at S, that is: $\mathsf{validinputs}(S) = \{a \in \mathsf{Ac}^n_{\mathsf{in}} | \mathsf{dsucc}(\mathsf{tsucc}(S, 0), a) \neq \emptyset\}$. Following the same methodology as in [18] we can prove that the proposed test generation algorithm is both sound and complete. Indeed both frameworks and both approaches are based on IOLTS and at this level the algorithms are the same and the conformance relations are equivalent. The difference between the two frameworks is only at syntactic and structural levels. In [18] the authors consider only one instance of the system whereas in our case we consider many instances which interact with each other and the behaviors of which are influenced by the total number of active components and the state of the shared resources.

The used test generation technique is based on model checking. The main idea is to formulate the test generation problem as a reachability problem that can be solved with the model checker tool UPPAAL [3]. However, instead of using model annotations and reachability properties to express coverage criteria, the observer language is used.

In this direction, we reuse the finding of Hessel et al. [14] by exploiting its extension of UPPAAL, namely UPPAAL CO$\sqrt{}$ER.[2] This tool takes as inputs a model, an observer, and a configuration file. The model is specified as a network of timed automata (.xml) that comprises a SUT part and an environment part. The observer (.obs) expresses the coverage criterion that guides the model exploration during test case generation. The configuration file (.cfg) describes mainly the interactions between the system part and the environment part in terms of input/output signals. It may also specify the variables that should be passed as parameters in these signals. As output, it produces a test suite containing a set of timed traces (.xml).

[2] http://user.it.uu.se/~hessel/CoVer/index.php.

Our test generation module is built upon these well-elaborated tools. The key idea here is to use UPPAAL CO$\surd$ER and its generic and formal specification language for coverage criteria to generate tests for security purposes.

Test Selection Different coverage criteria have been proposed for software, such as statement coverage and branch coverage [24]. In the TA case existing methods attempt to cover either finite abstractions of the state space (e.g., the region graph [29]) or structural elements of the specification such as edges or locations [14]. Here, we propose a technique for covering states, locations, edges, actions, or shared variables of the specification:

- State coverage: As already mentioned each node of a given test case corresponds to a set of states S of A_S^{Tick}. We say that the node *covers* S. We say that such a test covers the union of all sets of states covered by its nodes. We say that a set of tests (or *test suite*) achieves *state coverage* if every reachable state of $L_{S,n}^P$ is covered by some test in the suite.
- Location coverage: A test suite achieves *location coverage* if every reachable location of A_S is covered by some test in the suite.
- Edge coverage: Every edge of a test case can be associated with an edge of $L_{S,n}^P$. In particular, an edge $S \xrightarrow{a} S'$ will be associated with all edges which are visited during the reachability algorithm which computes S' from S. We say that a test suite achieves *edge coverage* if every reachable edge of $L_{S,n}^P$ is covered by some test in the suite.
- Action coverage: We also define *action coverage* as follows. If a given edge $S \xrightarrow{a} S'$ is reachable, then the corresponding observable action a is said to be reachable as well. Action coverage is achieved if all the reachable observable actions are covered by the considered test suite.
- Shared integer variable coverage: Finally we define *shared integer variable coverage* which consists in generating tests which cover the different possible values of the system variables.

26.5.2 Test Execution and Verdict Analysis

For the execution of the obtained security tests, we aim to use a standard-based test execution platform, called TTCN-3 test system for runtime testing (TT4RT), developed in a previous work [20]. To do so, security tests should be mapped to the TTCN-3 notation since our platform supports only this test language. Then, test components are dynamically created and assigned to execution nodes in a distributed manner.

Each test component is responsible for (1) stimulating the SUT with input values, (2) comparing the obtained output data with the expected results (also called oracle), and (3) generating the final verdict. The latter can be pass, fail, or inconclusive. A pass verdict is obtained when the observed results are valid with respect to

the expected ones. A fail verdict is obtained when at least one of the observed results is invalid with respect to the expected one. Finally, an inconclusive verdict is obtained when neither a pass nor a fail verdict can be given. After computing for each executed test case its single verdict, the proposed platform deduces the global verdict.

26.5.3 Cloud Testing

The emergent paradigm, cloud computing, is formally defined by U.S.NIST (National Institute of Standards and Technology) [23] as follows. Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

This cloud model is characterized with three service models: software as-a-service (SaaS), platform as-a-service (PaaS), and infrastructure as-a-service (IaaS). The SaaS refers to the capability provided to the consumer to use the provider's applications running on a cloud infrastructure. With PaaS, the consumer is able to deploy his own applications without installing any platform or tools since he uses provided platform layer resources, including operating system support and software development frameworks. Regarding IaaS, it provides a collection of resources such as servers, storage, networks, and other computing resources in the form of virtualized systems, which are accessed through the Internet.

It is worthy to note that public cloud providers like Amazon Web Services[3] and Google Cloud Platform[4] offer a cloud infrastructure made up essentially of availability zones and regions. As shown in Fig. 26.4, a region is a specific geographical location in which public cloud service providers' data centers reside. Each region is further subdivided into availability zones. Several resources can live in a zone, such as instances or persistent disks. In the context of Google Cloud Platform, the us-central1 region, for example, denotes a region in the Central United States that has four zones, namely us-central1-a, us-central1-b, us-central1-c, and us-central1-f.

Cloud computing has been used in the context of software testing to encounter the lack of resources and the expensiveness of building a distributed test environment during the testing process. As a result, the concept of *cloud testing* is newly emerging in order to provide cost-effective testing services. According to [7], it refers to testing activities, essentially test generation, test execution, and test evaluation on a cloud-based environment. The latter supports on-demand resource allocation to large-scale testers whenever and wherever they need by following the

[3] https://aws.amazon.com/fr/.

[4] https://cloud.google.com/.

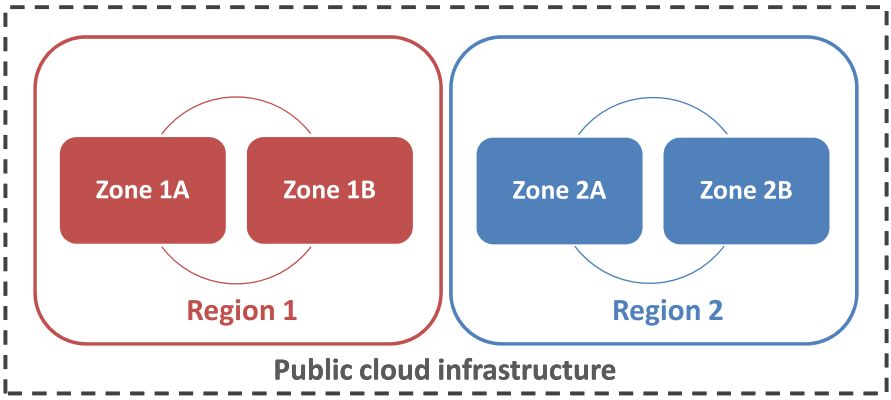

Fig. 26.4 Illustration of cloud partitioning in regions and zones

pay-per-use business model. Such virtualized and shared resources may reduce effectively the cost of building a distributed test environment for the runtime validation of dynamically adaptive systems.

Testing as-a-service (TaaS) is an innovative concept that provides end users with testing services such as test case generation, test execution, and test result evaluation. It has been proposed to improve the efficiency of software quality assurance. Notably, it is used for software systems that are remotely deployed in a virtualized runtime environment using shared hardware/software resources, and hosted in a third-party infrastructure (i.e., a cloud). One of the primary objectives is to reduce the cost of software testing tasks by providing on-demand testing services and also on-demand test environment services (i.e., establishing the required virtual (or physical) cloud-based computing resources and infrastructures for testing purposes).

26.5.4 Test Execution Platform as-a-Service

The proposed approach is built based on TaaS concepts. Figure 26.5 outlines an overview of its different constituents.

– Test management GUI: This component offers a graphical user interface (GUI) charged with managing the overall testing process: the automatic creation/deletion of VM instances, the dynamic allocation of test components to the appropriate VMs, the start-up of test component execution, and the computation of the final verdict. Moreover, it is responsible for querying the runtime monitoring component for information about the usage of resources in running VMs.

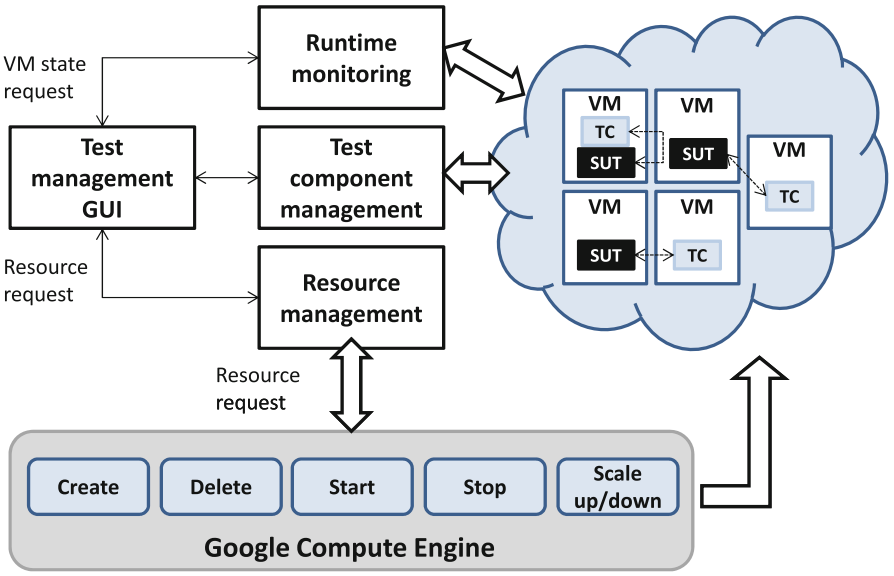

Fig. 26.5 Test execution platform overview

- Resource management: This component enables flexibility and elasticity during the testing process. If there is no adequate VM to handle the execution of a test component, a new VM can be created and started automatically. Moreover, it is possible to scale up or scale down an existing VM. The unused one can be released as well.
- Test component management: This component offers services for creating/deleting test components and starting/stopping their execution. A test component is an entity that interacts with the SUT to execute the available test cases (i.e., a set of input values and expected results) and to observe its response related to this excitation. Its main role consists of stimulating the SUT with the input values, comparing the obtained output values to the expected results and generating the final verdict that can be pass, fail, or inconclusive.
- Runtime monitoring: This component monitors VM instances during or even before the test execution and gives the status of each VM in terms of computing resources (such as CPU, memory, and storage).

As already discussed, several VM instances are created and started in the proposed cloud infrastructure in which several components under test are running too and can be evolved at runtime. To perform runtime tests in a cost-effective manner, several test components should be deployed in the final execution environment. The major question to be tackled here is how to assign efficiently test components to the existing VM instances?

Before answering to this question, we should mention that the components under test are distributed in several VM instances that can be located in the same region

and in the same zone as well as in different zones and even in different regions of the cloud infrastructure. Such information is provided via a SUT deployment descriptor. In this file, the SUT manager defines, for each component under test, the VM instance hosting its execution and also its main characteristics (i.e., its IP address, its corresponding zone, and region). Hereafter, we denote the VM hosting a component under test by VM under test (VMUT) and the VM hosting the test component by test VM (T_VM).

26.6 Related Work

In this section we give an overview of contributions from the literature and from our previous work related to the security of IoT applications in smart cities.

Although the security protocols are well elaborated in the Internet domain, it is still not fully clear how these existing IP security protocols and architectures can be adapted and deployed in the context of distributed and heterogeneous environment like the Internet of Things (IoT). From its appearance as a promiscuous technology, several works are addressing the security problems of the IoT [4, 8, 12, 13, 26, 28, 32, 36], but until now there is no sufficient solutions that meet the users' requirements and performance needs. In the following, we will provide an overview of the most important related works and clarify the contributions of our proposal in comparison to the state of the art.

The authors in [4] presented the security contributions of the SMARTIE work, which aims to provide secure IoT data management for smart cities. The authors first classified attacks to internal and external. Internal attacks are caused by devices and/or users of the smart city environment. Internal attackers are more dangerous than external ones as they have detailed knowledge about the infrastructure, they have access to part of the systems and they hold some keys. Internal attackers can be users or administrators of the smart city systems or any hackers that succeed to compromise a component of the system. Note that, due to the diversity of the IoT devices and their spread in different locations, device compromise attack is easier in IoT than in classical networks. On the other hand, external attackers may try to access private data from users, components, or subsystems of the IoT environment. Moreover, as in IoT some components are controlled remotely, attackers can exploit these features to gain control and manipulate victims devices. Two main security mechanisms are defined in [4] to address the above issues. First, DCapBAC [13] is an authorization scheme that takes access control decisions before the actual service is accessed. It does this by giving a signed authorization token to a user who is asking for any particular service or functionality offered by a thing. The authorization token is sent along with a request to the thing that verifies the validity of the request and the authorization token, delivering the requested data, if successful. Second, PrivLoc [33] offers secure location-based services, in particular a secure geo-fencing service that alerts users if objects enter or leave a defined area. Location-based services are increasingly gaining importance. Not only end

users but also companies can make use of location data to track assets (e.g., public transport services, users looking for transportation, or logistics companies). PrivLoc scrambles location information in a way that allows computation on intersections of scrambled geometric objects, which is the main operation behind a geo-fencing service.

In paper [32], the authors proposed OSCAR (object security architecture for the Internet of Things), an architecture for end-to-end security in the Internet of Things. It is based on the concept of object security that relates security with the application payload. The architecture includes authorization servers that provide clients with access secrets that enable them to request resources from constrained nodes. The nodes reply with the requested resources that are signed and encrypted. Although this architecture solves some of disadvantages of Datagram Transport Layer Security (DTLS) since it supports multicast, asynchronous traffic, and caching, it has certain limitations. Indeed, OSCAR is vulnerable to the replay attack and its performance is affected with the eventual usage of larger elliptic curve cryptography (ECC) curves. In [12], an architectural reference model-compliant framework was proposed. This framework emphasizes on security and privacy aspects to be used on smart buildings scenarios. Additionally, authors proposed an extension of the security functional components of the reference architecture in order to enable more flexible sharing models, in which the physical context information is considered as a first-class component in order to realize the so-called context-aware security on IoT scenarios. As an instantiation of this framework, a platform for services management on smart buildings has been deployed and extended to offer both user-centric services, like comfort and energy saving, and discovery and security functionality for such services. The feasibility of the proposed mechanisms has been demonstrated through the instantiation of the platform and its evaluation in a smart building used as reference [4].

From a standardization perspective, the recently standardized constrained application protocol (CoAP) was proposed as a lightweight alternative to the HTTP protocol for web-based IoT applications, but security does not keep up. For this purpose, the IETF has thus taken a position to reuse the datagram transport layer security (DTLS), the all-round point-to-point security protocol, to secure the communication channel between a constrained device running CoAP and a client [32]. However, apart from its current incompatibility with caching and multicast traffic, the DTLS approach has an important impact on scalability: Memory limitations of constrained nodes restrict the number of DTLS sessions. In IoT scenarios such as smart cities in which a large number of clients may communicate with constrained CoAP nodes, the limitations lead to a considerable load that translates to an increased energy consumption and a shortened lifetime [32]. Several works have thus been proposed to overcome these limitations of DTLS.

The DTLS in constrained environments (DICE), an IETF working group, was formed to add multicast security to DTLS [15]. In [15], the authors present a method for securing IPv6 multicast communication based on the DTLS which is already supported for unicast communication for CoAP devices. They deal with the adaptation of the DTLS record layer to protect multicast group communication,

assuming that all group members already have the group security association parameters in their possession. The adapted DTLS record layer provides message confidentiality, integrity, and replay protection to group messages using the group keying material before sending the message via IPv6 multicast to the group [15]. However, the authors did not present how group members can agree on the group security association.

In [33], the authors presented Lithe—an integration of DTLS and CoAP for the IoT. Lithe proposes a novel DTLS header compression scheme that aims to reduce the energy consumption by leveraging the 6LoWPAN standard based on reducing the number of transmitted bytes while maintaining DTLS standard compliance.

Granjal et al. [9], described mechanisms to enable security at the network layer, based on the IPSec protocol, and at the application layer, based on the DTLS protocol, and performed an extensive experimental evaluation study with the goal of identifying the most appropriate secure communication mechanisms and the limitations of current sensing platforms for supporting end-to-end secure communications in the context of Internet-integrated sensing applications [9]. These results showed a similar performance of the two approaches, except in the case when DTLS is additionally used to exchange keys with the elliptic curve Diffie-Hellman exchange.

Heer et al. [10] discussed the applicability and limitations of existing Internet protocols and security architectures in the context of IoT. They presented challenges and requirements for IP-based security solutions and highlighted specific technical limitations of standard IP security protocols. It was indicated that for supporting secure IoT, its security architecture should fit the life cycle of a thing and its capabilities, and scale from small-scale ad-hoc security domains of things to large-scale deployments, potentially spanning several security domains. Security protocols should further take into account the resource-constrained nature of things and heterogeneous communication models. Lightweight security mechanisms and group security that are feasible to be run on small things and in IoT context should be developed, with particular focus on possible DoS/DDoS attacks. In addition, cross layer concepts should be considered for an IoT-driven redesign of Internet security protocols.

The authors in [35] addressed the routing protocol for low-power and lossy networks (RPL) attacks and they provided a comprehensive analysis of IoT technologies and their new security capabilities that can be exploited by attackers or IDSs. One of the major contributions in [35] is the implementation and demonstration of well-known routing attacks against 6LoWPAN networks running RPL as a routing protocol. The implemented attacks are selective-forwarding attacks (where malicious nodes selectively forward packets and therefore can achieve a DoS attack), sinkhole attacks (where a malicious node advertises an artificial beneficial routing path and attracts many nearby nodes to route traffic through it), HELLO flood attacks (where the attackers by broadcasting a HELLO message with strong signal power and a favorable routing metric can introduce himself as a neighbor to many nodes, possibly the entire network), wormhole attacks, clone ID, and

Sybil attacks. In order to mitigate these attacks, the authors proposed an intrusion detection system (IDS), called SVELTE [27].

SVELTE [27], an intrusion detection system for the IoT was designed, implemented, and evaluated against routing attacks such as spoofed or altered information, sinkhole, and selective-forwarding. SVELTE's overhead is small enough to deploy it on constrained IoT nodes with limited energy and memory capacity. However, SVELTE assumes that it has access to the border router of the network to place heavyweight IDS parts there. This assumption is not always possible. For example in a smart city application, the messages can be routed over a cellular station that belongs to the network of another owner and therefore we did not have access to the border router.

The previously cited proposed solutions can be classified to three categories which are application layer security solutions [15, 33], network layer security solutions [27, 35], and context-aware security solutions [4, 12, 32]. Context-aware security solutions are very dependent to the specific characteristics of the applications use case and so suffer from the lack of inter-operability. Moreover, the existing solutions have a high computation and communication cost that make them inadequate to resource-constrained things. Network security solutions are limited to attacks related to network layer and so cannot mitigate attacks that target the application layer and cannot provide some security services such as authentication and access control.

Contrary to [27, 35] that proposed an IDS that is limited to routing attacks. In this work we aim to extend the functionality of the IDS and to address also the application layer attacks that target the CoAP protocol. This kind of IDS will present a first line of defense and will mitigate several attacks such as the DoS attack. In addition, this work will focus on providing security based on the DTLS protocol as there are several attempts to make this protocol as the standard for security in the IoT. Therefore, we will propose enhancements to the DTLS protocol to fit the IoT objects. Moreover, we will focus on not resolved aspects such as group key management and multicast communication.

The authors of [6] propose a good survey on more than one hundred publications on model-based security testing extracted from the most relevant digital libraries and classified according to specific criteria. Even though this survey reports on a large number of articles about MBST it does not contain any reference to IoT applications or smart cities. Contrary to that the authors of [1] propose a model-based approach to test IoT platforms (with tests provided as services) but they do not deal with security aspects at all.

26.7 Conclusion

In this work we aimed to combine these two directions, namely: model-based testing and security testing for IoT applications in smart cities. For that purpose we took advantage of our previous findings [5, 16, 20, 22] related to these fields. Moreover,

we extended the notions proposed in the survey [5] to the case of IoT applications. We also exploited our previous results about test techniques of dynamic distributed systems [16, 20].

Our work is at its beginning and a lot of efforts are needed at all levels on both theoretical and experimental aspects. First we need to deal with modeling issues. In this respect we need to extend our modeling formalism and to identify the particular elements of IoT applications to model (using extended timed automata). Models must not be big in order to avoid test number explosion. For that purpose we need to keep an acceptable level of abstraction. As a second step we have to adapt our test generation and selection algorithms to take into account security requirements of the applications under test. The new algorithms must be validated theoretically and proved to be correct. In the same manner we need to upgrade our tools to implement new obtained algorithms. We also need to validate our approach with concrete examples with realistic size. Finally we propose to adopt the same methodology as in [22] to combine security and load tests for IoT applications.

References

1. Ahmad, A., Bouquet, F., Fourneret, E., Le Gall, F., Legeard, B.: Model-based testing as a service for IOT platforms. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications, pp. 727–742. Springer, Cham (2016)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**, 183–235 (1994)
3. Behrmann, G., David, A. and Larsen, K.G.: A tutorial on uppaal. In: Bernardo, M., Corradini, F. (eds.)International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004. Revised Lectures, vol. 3185, LNCS, pp. 200–237. Springer, Berlin (2004)
4. Bohli, J.-M., Skarmeta, A., Moreno, M.V., García, D., Langendörfer, P.: Smartie project: secure IoT data management for smart cities. In: 2015 International Conference on Recent Advances in Internet of Things (RIoT), vol. 00, pp. 1–6 (2015)
5. Cheikhrouhou, O.: Secure group communication in wireless sensor networks: a survey. J. Netw. Comput. Appl. **61**, 115–132 (2016)
6. Felderer, M., Zech, P., Breu, R., Büchler, M., Pretschner, A.: Model-based security testing: a taxonomy and systematic classification. Softw. Test. Verif. Reliab. **26**(2), 119–148 (2016)
7. Gao, J., Bai, X., Tsai, W.-T.: Cloud testing- issues, challenges, needs and practice. Softw. Eng. Int. J. **1**(1), 9–23 (2011)
8. Garcia-Morchon, O., Kumar, S., Keoh, S.L., Hummen, R., Struik, R.: Security Considerations in the IP-Based Internet of Things, Internet-Draft draft-garcia-core-security-06, Internet Engineering Task Force, Fremont (2013). Work in Progress
9. Granjal, J., Monteiro, E., Sá Silva, J.: On the effectiveness of end-to-end security for internet-integrated sensing applications. In: 2012 IEEE International Conference on Green Computing and Communications, pp. 87–93 (2012)
10. Heer, T., Garcia-Morchon, O., Hummen, R., Keoh, S.L., Kumar, S.S., Wehrle, K.: Security challenges in the ip-based internet of things. Wirel. Pers. Commun. **61**(3), 527–542 (2011)
11. Henzinger, T.A., Manna, Z., Pnueli, A.: What good are digital clocks? In: Kuich, W. (ed.) Automata, Languages and Programming, pp. 545–558. Springer, Berlin (1992)
12. Hernández-Ramos, J.L., Moreno, M.V., Bernabé, J.B., Carrillo, D.G., Skarmeta, A.F.: SAFIR: secure access framework for IoT-enabled services on smart buildings. J. Comput. Syst. Sci. **81**(8), 1452–1463 (2015)

13. Hernández-Ramos, J.L., Jara, A.J., Marin, L., Gómez, A.F.S.: Dcapbac: embedding authorization logic into smart things through ECC optimizations. Int. J. Comput. Math. **93**(2), 345–366 (2016)
14. Hessel, A., Larsen, K.G., Nielsen, B., Pettersson, P., Skou, A.: Time-optimal real-time test case generation using uppaal. In: Petrenko A., Ulrich, A. (eds.) Formal Approaches to Software Testing, pp. 114–130. Springer, Berlin (2004)
15. Keoh, S., Kumar, S., Garcia-Morchon, O., Dijk, E., Rahman, A.: DTLS-Based Multicast Security for Low-Power and Lossy Networks (LLNs). Internet-Draft Draft-keoh-dice-multicast-security-08, Internet Engineering Task Force, Fremont (2014). Work in Progress.
16. Krichen, M.: A formal framework for black-box conformance testing of distributed real-time systems. IJCCBS **3**(1/2), 26–43 (2012)
17. Krichen, M., Tripakis, S.: Black-box conformance testing for real-time systems. In: Graf, S., Mounier, L. (eds.) Model Checking Software, pp. 109–126. Springer, Berlin (2004)
18. Krichen, M., Tripakis, S.: Conformance testing for real-time systems. Form. Methods Syst. Des. **34**(3), 238–304 (2009)
19. Krichen, M., Cheikhrouhou, O., Lahami, M., Alroobaea, R., Jmal Maâlej, A.: Towards a model-based testing framework for the security of internet of things for smart city applications. In: Mehmood, R., Bhaduri, B., Katib, I., Chlamtac, I. (eds.) Smart Societies, Infrastructure, Technologies and Applications, pp. 360–365. Springer, Cham (2018)
20. Lahami, M, Krichen, M., Jmaïel, M.: Safe and efficient runtime testing framework applied in dynamic and distributed systems. Sci. Comput. Program. **122**(C), 1–28 (2016)
21. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE **84**(8), 1090–1123 (1996)
22. Maâlej, A.J., Krichen, M.: A model based approach to combine load and functional tests for service oriented architectures. In: Proceedings of the 10th Workshop on Verification and Evaluation of Computer and Communication System, VECoS 2016, Tunis, October 6–7, 2016, pp. 123–140 (2016)
23. Mell, P., Grance, T.: The Nist Definition of Cloud Computing (2011)
24. Myers, G.J., Sandler, C.: The Art of Software Testing, Wiley, Hoboken (2004)
25. Nguyen, K.T., Laurent, M., Oualha, N.: Survey on secure communication protocols for the internet of things. Ad Hoc Netw. **32**, 17–31 (2015)
26. Park, S.D., Kim, K.-H., Haddad, W., Chakrabarti, S., Laganier, J.: IPv6 over Low Power WPAN Security Analysis. Internet-Draft draft-daniel-6lowpan-security-analysis-05, Internet Engineering Task Force, Fremont (2011). Work in Progress
27. Raza, S., Wallgren, L., Voigt, T.: Svelte: real-time intrusion detection in the internet of things. Ad Hoc Netw. **11**(8), 2661–2674 (2013)
28. Roman, R., Zhou, J., Lopez, J.: On the features and challenges of security and privacy in distributed internet of things. Comput. Netw. **57**(10), 2266–2279 (2013)
29. Springintveld, J., Vaandrager, F., D'Argenio, P.R.: Testing timed automata. Theor. Comput. Sci. **254**(1), 225–257 (2001)
30. Tretmans, J.: Testing concurrent systems: a formal approach. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR'99 Concurrency Theory, pp. 46–65. Springer, Berlin (1999)
31. Tripakis, S.: Fault diagnosis for timed automata. In: Damm, W., Olderog, E.R. (eds.) Formal Techniques in Real-Time and Fault-Tolerant Systems, pp. 205–221. Springer, Berlin (2002)
32. Vucinic, M., Tourancheau, B., Rousseau, F., Duda, A., Damon, L., Guizzetti, R.: OSCAR: object security architecture for the internet of things. CoRR, abs/1404.7799 (2014)
33. Vučinić, M., Tourancheau, B., Rousseau, F., Duda, A., Damon, L., Guizzetti, R.: Oscar. Ad Hoc Netw. **32**(C), 3–16 (2015)
34. Walewski, J.: Internet-of-Things Architecture IOTA Project Deliverable d1.2 - Initial Architectural Reference Model for IOT (2018)
35. Wallgren, L., Raza, S., Voigt, R.: Routing attacks and countermeasures in the RPL-based internet of things. Int. J. Distrib. Sens. Netw. **9**(8), 794326 (2013)
36. Yan, Z., Zhang, P., Vasilakos, A.V.: A survey on trust management for internet of things. J. Netw. Comput. Appl. **42**, 120–134 (2014)