Exploring IoT Trickle-Based Dissemination Using Timed Model-Checking and Symbolic Execution

Boutheina Bannour, Arnault Lapitre, Pascale Le Gall

Exploring IoT Trickle-based Dissemination using Timed Model-checking and Symbolic Execution

Boutheina Bannour[1], Arnault Lapitre[1], and Pascale Le Gall[2]

[1] CEA LIST, Gif-sur-Yvette, France
email: {firstname.lastname}@cea.fr
[2] MICS Lab., Univ. Paris-Saclay, Gif-sur-Yvette, France
email: {firstname.lastname}@centralesupelec.fr

Abstract. We focus on studying an IoT algorithm called Trickle using a formal model-based approach. The algorithm has an essential role in traffic regulation across distributed networks of wireless sensors which are part of IoT. The algorithm allows efficient dissemination of information such as critical applicative data, firmware upgrades or security fixes. In this paper, we develop timed asynchronous computational models for Trickle. We show how reachability properties can be assessed on such models using an original combination of model-checking and symbolic execution implemented by the tools UPPAAL and DIVERSITY, respectively. Our experiments produce promising results on highlighting updated or outdated nodes situations during dissemination.

1 Introduction

Context. Sensors networks (WSN) play an essential role in the uptake of the Internet of Things (IoT) as they allow direct connection between the physical environment and the digital systems. They come with a reduced economical cost, and they can easily be deployed in inaccessible areas. WSN involve constrained-energy devices (sensors) which operate over long periods. The information dissemination across these networks is often subject to constraints to reduce the communication cost, with the objective not to exhaust the batteries of such nodes that are in general neither rechargeable nor replaceable after the deployment. Gossip paradigm has been recognized as being efficient in practice to control the communications of each node, roughly speaking: i) every node try quickly to transmit new data, and ii) in case of redundant data reception, the node reduces the transmissions frequency over time. The algorithm Trickle [23, **?**,**?**] is one of the most known: It comes as a standard library in TinyOS [22] and Contiki [15], two well-known firmware Operating Systems (OS) for WSN. The algorithm is involved in recently standardized WSN protocols namely the Multicast Protocol for Low Power and Lossy Networks (MPL) [16] and the IPv6 Routing Protocol for Low Power and Lossy Networks (RPL) [1]. There are many others like FireFly Gossip (FiGo) [7], Energy Efficient Gossiping (E-Gossip) [21], Multi Randomized Gossip-Consensus-based Sync (Multi RGCS) [29], and new ones continue to be proposed given the economic interest of WSN.

Case study: Trickle dissemination. The goal of Trickle is to reach a stable global state of the network where all the nodes have the same up-to-date information. Each node applies a set of rules to control its transmissions as follows [27]:

- each node maintains a current interval I, a counter c and a broadcasting time t in interval $[I/2, I[$,
- global parameters to all nodes are k the redundancy constant, I_{min} (resp. I_{max}) the smallest (resp. largest) interval,
- each node applies the following rules:
  1. at the start of a new interval, the timer and counter c are reset and t is randomly set to a value in $[I/2, I[$,
  2. if a received message is consistent with the information the node holds, the counter c is incremented,
  3. when the timer reaches t and $c < k$, a message carrying the node information is transmitted to neighbours in broadcast,
  4. when the timer expires at I, the interval length is increased by setting I to $\min(2 \cdot I, I_{max})$ and a new interval starts,
  5. when a received message is inconsistent with the node information, then I is set to I_{min}, and a new interval starts. Otherwise, nothing happens.

Trickle uses "polite gossip" to exchange information with its network neighbours. It breaks time into dynamically adjustable intervals, and at a random point in each interval, it considers broadcasting the information it holds. If Trickle has already heard several other nodes gossip the same information in this interval, it politely stays quiet: "repeating what someone else has said is rude" [23].

Motivation and related work. As it may show from Trickle, popular gossip protocols are sophisticated and complex on the nature of applied control on node transmissions. For that reason, most of the validation effort of this class of protocols relies mainly on testbeds or simulations, and at a less extend on analytical methods, more exhaustive, yet they lack automation: among the latter, we can cite those developed for Trickle in [6, 18, 28, 11, 31]. On the other hand, formal methods and in particular model-checking [10] come with a high degree of automation. They consider computational models which, by its nature, delimit the perimeter of the analysis and the kind of properties to be verified. The survey [8] overviews many relevant works on applying formal methods on WSN protocols, including gossip-based protocols. The following papers [19, 32, 14, 34, 33]) have successfully applied model-checking on gossip protocols, among which [14, 34, 33] concern Trickle. The early work [14] proposed formal models for WSN based on classic process algebras. The work developed a simplified model of Trickle for illustration purposes, then translated into a network of Timed Automata (TA) [2] supported by the model-checker UPPAAL. In their model, Trickle intervals are not adjustable and are of fixed length, which restricts the coverage of performed analyses. More recently, authors in [34, **?**] have proposed model-checking techniques for WSN, and Trickle has been proposed for illustration as well. The work [34] provides formal semantics for a subset of NesC programs used to built TinyOS [22] applications; such semantics have been implemented in the model-checking framework PAT [24] which supports TA too. The work [33] focuses on

combining probabilistic model checking provided by the tool PRISM [20] with automated debugging algorithms in order to find pathological typologies which cause some failure: typically in case of Trickle, it is about finding the topology pattern that prevents recent information from being spread. In both works [34, **?**], Trickle models have not been given. In this paper, we are interested in the distributed and desynchronized nature of WSN nodes [26, **?**], that is they are dephased by some duration because often they do not share a common zero as classically in distributed systems. Gossip protocols, Trickle included, do not have any assumption on nodes that should be synchronized. The control they provide on transmission instants is usually implemented by introducing some time randomness to benefit from desynchronized nodes. Besides, inline with such assumption, we consider asynchronous communications which can take time to deliver data, all previous works have not taken into account such desynchronized hypothesis.

Contribution and paper outline. We provide formal models for Trickle as Extended Timed Automata (XTA) in which data can be used to define computations, random updates, constraints on clocks and communication actions. A network of XTA can then be designed to form the overall Trickle topology of nodes. The network is endowed with operational semantics in which XTA communicate their data via unbounded queues. In this paper, we are interested in highlighting some situations of updated or outdated nodes using reachability properties under the desynchronized assumption. To assess such properties on XTA, which are more expressive than classical TA because they introduce data, we propose to combine model-checking [10] with symbolic execution [17]. The latter virtually executes models (or code) for symbolic input parameters rather than concrete values. Each execution path is associated with logical constraints on those input parameters computed at each execution step, the so-called Path Conditions (PC). PCs are a compact representation of classes of actual values for input parameters for executing those paths; they can be solved using SMT solvers such as CVC4 [5] or Z3 [12]. Symbolic execution can be applied on timed models (e.g., [4]).

Yet enumerating all execution paths to check a property of interest is combinatorial, this is an identified problem of the technique that we have experimented as well on some early work on Trickle [30, **?**]. The idea is then to apply model-checking first on XTA in which data is numeric and random updates are restricted to some values, yet clock constraints are handled as usual by zone-based abstraction implemented in UPPAAL using the efficient Difference Bounded Matrices data structure (DBM) [13]. In case the property is verified, the corresponding sequence of transitions of a solution is used to guide symbolic execution. We experiment with the combined techniques in UPPAAL and in the symbolic execution tool DIVERSITY [25]. The rest of the paper is organized as follows. Section 2 introduces the formal model of the network of XTA and its operational semantics. Section 3 proposes a Trickle model designed as a network of XTA. Section 4 introduces our approach of combining model-checking

and symbolic execution to assess reachability properties and evaluates it on UPPAAL-DIVERSITY connection. Section 5 concludes the paper.

2 Network of extended timed automata

The model of Timed automata [3] is a well-established formalism for modelling the timing behaviour of systems. This section defines syntax and semantics of an extension of timed automata introducing data updates, communication actions with data transmission, and non-trivial data-dependent time constraints. Those appear in functional specification of systems, as in the specification of Trickle in the introduction.

Data domain. We use a a universal data domain D to abstract all values of time variables, called clocks as usual, and other data variables. Data variables can be of any type, whereas clocks are typed in a time domain $T \subseteq D$ which is isomorphic to $\mathbb{Q}_+$, the set of positive rational numbers.

Data valuations. For a set of data variables V, a data valuation is a type-preserving mapping $v : V \to D$. We canonically extend data valuation to usual arithmetical expressions defined over V, i.e., $v(e)$ is the value of e for the valuation v. We denote D^V the set of all such valuations.

Data formulae. The set $\mathcal{F}(V)$ of data formula f over V is either: an atomic formula of the form $true$, $false$, $e_1 = e_2$, $e_1 \prec e_2$, and $e_1 \succ e_2$ with $\prec \in \{<, \leq\}$ and $\succ \in \{>, \geq\}$: or built over those using usual connectives: conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$).

Sequential data updates. We consider sequential updates defined as follows:

$$u ::= \text{skip} \mid x := e_1 \mid e_1 \prec : x :\prec e_2 \mid u_1 ; u_2 \mid \text{if}(f)\text{then}\{u_1\}\text{else}\{u_2\} \mid \text{repeat}(n)\{u_1\}$$

skip is the null update; $x := e_1$ assigns the variable x with a new value denoted by e_1; $e_1 \prec : x :\prec e_2$ assigns x with a new random value bounded from below by the value denoted by e_1, the value of x is also bounded from above by the value denoted by e_2. Moreover, updates can be built using usual control primitives: sequence (;), condition (if ... then ... else ...) or counted-loop (repeat(n)) allowing the repetition of the enclosed update n times.

The set of update functions $\mathcal{U}(V)$ is defined by functions $[\![u]\!]$ from D^V to $2^{(D^V)}$. The set of valuations $[\![u]\!](v) \in 2^{(D^V)}$ is defined on the form of u as follows[3]:

$$[\![\text{skip}]\!](v)=\{v\} \tag{1}$$

$$[\![x:=e_1]\!](v)=\{v[x\rightarrow v(e_1)]\} \tag{2}$$

$$[\![e_1 \prec : x : \prec e_2]\!](v)=\{v[x\rightarrow y] | v(e_1) \prec y \prec v(e_2)\} \tag{3}$$

$$[\![u_1;u_2]\!](v)=\{v' \mid \exists v'' \in u_1(v),\, v' \in u_2(v'')\} \tag{4}$$

$$[\![\text{if}(f)\text{then}\{u_1\}\text{else}\{u_2\}]\!](v)=u_1(v) \quad v(f) \tag{5}$$

$$[\![\text{if}(f)\text{then}\{u_1\}\text{else}\{u_2\}]\!](v)=u_2(v) \quad \neg v(f) \tag{6}$$

$$[\![\text{repeat}(n)\{u_1\}]\!](v)=[\![u_1;\text{repeat}(n-1)\{u_1\}]\!](v) \quad n>0 \tag{7}$$

$$[\![\text{repeat}(0)\{u_1\}]\!](v)=\{v\} \tag{8}$$

Clock formulae. Given a set of clocks Cl disjointed from V ($Cl \cap V = \emptyset$), a clock valuation is a mapping $w : Cl \rightarrow T$. With previous notation, the set $\mathcal{G}(Cl, V)$ of clock formulas g over Cl and V is either an atomic formulas of the form $true$, $false$, $clk \prec e$ or $clk \succ e$ where e is an expression over V typed in time domain T; or a conjunction of those. The set of clock invariants $\mathcal{I}(Cl)$ is defined by conjunctions of formulas of the form $clk \prec e$. We define a universal valuation $v \oplus w : V \cup Cl \rightarrow D$ as the resulting valuation which coincide with v and w on V and Cl respectively. This valuation can be canonically extended to formulas as usual.

Communication actions with data. Given a set of interaction points, often called ports, P, the set of communication actions $\mathcal{C}(P, V)$ contains two kind elements: output actions of the form $p!e$ which denotes an emission on some port p of a piece of data corresponding to the current valuation of e; or input actions of the form $p?x$ which denotes a reception of a piece of data that is stored in the variable x. Moreover, we consider the special action ϵ which denotes the absence of communication action. The valuations of actions of the form $p!e$, $p?x$, and ϵ, are defined by $v(p!e) = p!v(e)$, $v(p?x) = p?v(x)$, and $v(\epsilon) = \epsilon$ respectively.

Extended timed automaton. An extended timed automaton (XTA in short) is a tuple $(L, l_0, V, Cl, P, Tr, Inv)$ where L is a finite set of locations, $l_0 \in L$ is the initial location, V is a set of variables, Cl is a set of clocks, P is a set of ports, $Tr \subseteq L \times \mathcal{F}(V) \times \mathcal{G}(Cl, V) \times (\mathcal{C}(P, V) \cup \{\epsilon\}) \times \mathcal{U}(V) \times 2^{Cl} \times L$ is a set of transitions, and $Inv : L \rightarrow \mathcal{I}(Cl)$ is a state invariant mapping.

For a transition $tr = (l, f, g, ca, u, R, l') \in Tr$, l and l' are respectively the source and target location of tr; f and g are respectively the data guard and time guard of tr, i.e., enabling conditions on data variables and clocks; ca is the communication action of tr; u is an update function through which data variables are updated when tr is executed; and $R \subseteq Cl$ is the set of clocks to be reset.

[3] Given a function $h : A \rightarrow B$, a subset $X \subset A$, the function $h' = h[x \rightarrow y\,,\ x \in X]$ is defined as follows: $h'(z) = y$ if $z \in X$ otherwise $h'(z) = h(z)$. In case X is a singleton of the form $\{x\}$, we denote $h' = h[x \rightarrow y]$ in short.

The semantics of an XTA is a labeled transition system where states s are triples (l, v, w) where l is a location, v and w are data and clock valuations respectively. The transition relation is defined as follows:

- delay transition: $(l, v, w) \xrightarrow{d} (l, v, w')$ where for all $clk \in Cl$, $w'(clk) = w(clk) + d$ with some $d \in T$ such that $v \oplus w' \models Inv(l)$
- action transition: $(l, v, w) \xrightarrow{a} (l', v', w')$ if and only if there exists a transition $(l, f, g, ca, u, R, l') \in Tr$:
  - $v \oplus w \models f \wedge g$,
  - $a = v(ca)$
  - $v' \in [\![u]\!](v)$
  - $w' = w[clk \rightarrow 0\,,\ clk \in R]$.

In the following, we introduce a network of XTA which exchange data using broadcast communication.

A network of extended timed automata. A network of XTA denoted by $\mathcal{A} = ((\mathcal{A}_i)_{i\in\{1,\ldots,n\}}, \mathcal{K})$ is defined as follows:

- $(\mathcal{A}_i)_{i\in\{1,\ldots,n\}}$ a family of XTA $\mathcal{A}_i = (L_i, l_0^i, V_i, Cl_i, P_i, Tr_i, Inv_i)$ which do not share variables and clocks, i.e., for all $i, j \leq n$ we have that $V_i \cap V_j = \emptyset$, $Cl_i \cap Cl_j = \emptyset$, and $P_i \cap P_j = \emptyset$,
- a total function $\mathcal{K} : P_{\mathcal{N}} \rightarrow 2^{P_{\mathcal{N}}}$ specifying connections between ports.

The set of all ports of $\mathcal{N}$ is denoted by $P_{\mathcal{N}} = \bigcup_{i\leq n} P_i$. Besides, the set of all variables (resp. clock variables) of $\mathcal{N}$ is denoted by $V_{\mathcal{N}} = \bigcup_{i\leq n} V_i$ (resp. $Cl_{\mathcal{N}} = \bigcup_{i\leq n} Cl_i$).
We make the hypothesis that latent data issued (potentially by different sources) and targeting some internal port in delivered on that port in the order they were sent, i.e., we will implement this using queues with a policy of first-in-first-out (fifo). In the following, we denote by the function $q : P_{\mathcal{N}} \rightarrow D^*$ the pending data in the network as being the content of queues associated with receiving ports.
The semantics of a network of XTA is a labeled transition system in which: states are tuples of the form $S = ((l_1, v_1, w_1), \ldots, (l_n, v_n, w_n), q)$ with initial states S_0 verify for all $i \leq n$, $l_i = l_0^i$; and transitions are defined as follows:

- delay transition:

  $$((l_1, v_1, w_1), \ldots, (l_n, v_n, w_n), q) \xrightarrow{d} ((l_1, v_1, w_1'), \ldots, (l_n, v_n, w_n'), q)$$

  iff for all $i \leq n$ there exists $(l_i, v_i, w_i) \xrightarrow{d} (l_i, v_i, w_i')$
- internal output transition:
  $((l_1, v_1, w_1), \ldots, (l_i, v_i, w_i), \ldots, (l_n, v_n, w_n), q) \xrightarrow{a}$

  $$((l_1, v_1, w_1), \ldots, (l_i', v_i', w_i'), \ldots, (l_n, v_n, w_n), q')$$

  iff there exists $(l_i, v_i, w_i) \xrightarrow{a} (l_i', v_i', w_i')$ with $a = p!m$, q' is such that for all port p_1 either $p_1 \in \mathcal{K}(p)$ then $q'(p_1) = q(p_1).m$ otherwise $q'(p_1) = q(p_1)$,

– internal input transition:
$((l_1, v_1, w_1), \ldots, (l_i, v_i, w_i), \ldots, (l_n, v_n, w_n), q) \xrightarrow{a}$

$$((l_1, v_1, w_1), \ldots, (l'_i, v'_i, w'_i), \ldots, (l_n, v_n, w_n), q')$$

iff there exists $(l_i, v_i, w_i) \xrightarrow{a} (l'_i, v'_i, w'_i)$ with $a = p?m$, $q(p)$ is not empty and is of the form $q(p) = m.q_1$, and q' is such that $q'(p) = q_1$ and for all $p_1 \neq p$ we have $q'(p_1) = q(p_1)$,
– silent or external action transition:
$((l_1, v_1, w_1), \ldots, (l_i, v_i, w_i), \ldots, (l_n, v_n, w_n), q) \xrightarrow{a}$

$$((l_1, v_1, w_1), \ldots, (l'_i, v'_i, w'_i), \ldots, (l_n, v_n, w_n), q)$$

iff there exists $(l_i, v_i, w_i) \xrightarrow{a} (l'_i, v'_i, w'_i)$ with $a = \epsilon$ or $a = p!m$ (resp. $a = p?m$) such that $\mathcal{K}(p) = \emptyset$ (resp. for all p_1, $p \notin \mathcal{K}(p_1)$).

In a nutshell, the above definition shows that time advances in the same way for all clocks of XTA forming the network. Besides, an internal emission $p!m$ on a port p has the effect of filling all the fifo associated to the ports of $\mathcal{K}(p)$ and an internal reception $p?m$ on port p consumes the first message stored in its fifo. When a silent action or an external action (reception or an emission) occurs on a port which is not connected to other XTA ports, it is executed with no effect on fifo queues, since it is assumed be connected to some implicit environment.
A run of the network is derived from the labelled transition system as a path starting in an S_0 and alternating delay transitions and action transitions. The property we are interested in is the reachability of states S. A state S is reachable iff there exists a run in which S occurs. In practice, such states are those which satisfy some user-specified formula $\phi = f \wedge g$ on data and clocks.
Let us denote by tr-seq(r) the sequence of (syntactic) transitions in $\bigcup_{i \leq n} Tr_i$ covered by a run r. We recall that such sequence intertwines transitions of different automata composing the network based on induced fifo-communications causalities discussed above.

Two runs r_1 and r_2 are said to be *coverage equivalent* if and only if tr-seq(r_1) = tr-seq(r_2), i.e., they cover the same (syntactic) transitions sequence.
The equivalence classes characterized by this relation guide the symbolic execution to search for all runs of a given class (as a symbolic path together with its path condition). On the other hand, model-checking will be used to compute some representative runs of the class that satisfy a reachability property ϕ. In practice, from the latter, we extract the transition sequence that guides the exploration performed by the symbolic execution.

3 Trickle models

Trickle node behavior. We propose the XTA $(L, Init, V, Cl, P, Tr, Inv)$ which specifies Trickle behaviour of each node in the network, the automaton is depicted in Figure 1. The XTA has 5 locations $L = \{Init, Listen_1, Listen_2, Check_1, Check_2\}$ in which $Init$ is the initial location. The clocks set is a singleton $Cl = \{clk\}$

containing one clock used to implement the Trickle timer. The set of variables contains 5 variables $V = \{I, t, c, myv, rcv\}$: the former three variables I, t and c are Trickle variables which respectively represent the value of the current interval, the instant of transmission and the counter value (whereas I_{min}, I_{max}, k are Trickle constants). Without loss of generality, in this automaton, Trickle is used to maintain consistency of version number across the network, the variable myv stores the most recent version the node holds and rcv is used to store received versions from the neighbourhood.

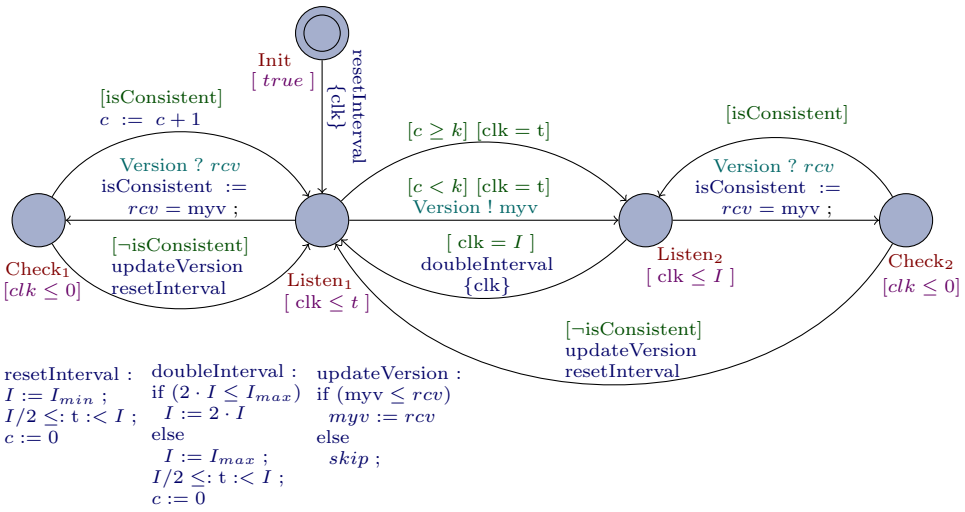

Fig. 1. An extended timed automaton (XTA) of a Trickle node behavior

The automaton has 9 transitions composing the set Tr, that we will overview next together with meaning associated with state invariants defined by mapping Inv. A node can be started at any time; this is captured by state invariant $Inv(Init) : true$ in location l_0, which means that any duration can elapse in this location. The transition $Init \rightarrow Listen_1$ is fired to start the Trickle behaviour. It sets I to I_{min}, assigns counter c and clk with 0, and finally chooses a the transmission time $I/2 \leq: t :< I$ within the second-half of the interval current interval I. The state invariant $Inv(Listen_1) : clk \leq t$ constrain time elapsing to be bounded by t. When clk reaches t, there two possible behaviors: either the transmission occurs given $c < k$ is fulfilled (horizontal transition $Listen_1 \rightarrow Listen_2$ with action $Version?rcv$), otherwise the transmission is suppressed (curved transition $Listen_1 \rightarrow Listen_2$ with action ϵ). A first reception handling is defined by transition $Listen_1 \rightarrow Check_1$. Once started, the automaton satisfies the input enableless property: in every state ($Listen_1$ or $Listen_2$), it is possible to receive every input (action $Version?rcv$). Each time, a version is received, it is compared to the current version of the node: in case of consistency

(same version $rcv = myv$), the counter c is incremented ($c := c + 1$), we recall that the latter counts redundant versions; otherwise (case of inconsistency) a new interval is started, and the node updated its version if it is older.

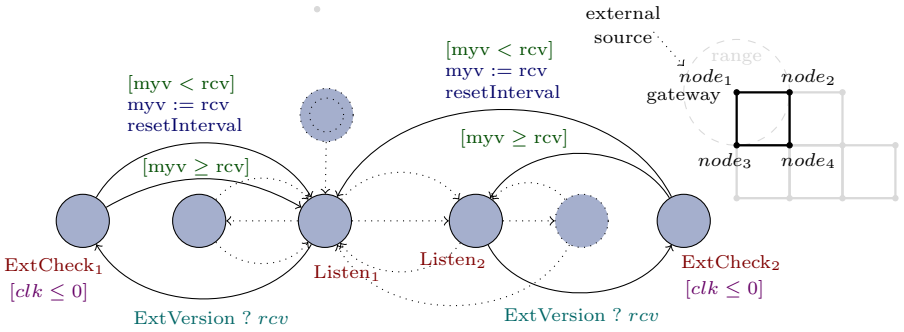

Fig. 2. Setting a network of XTA for a Trickle dissemination

State invariant $Inv(Listen_2) : clk \leq I$ constrains time elapsing to be at most of the value denoted by current interval length I. Subsequently, a new interval is started by doubling I (until I_{max}) and a new t is chosen as previous (transition $Listen_2 \rightarrow Listen_1$). Similar reception handling as in location $Listen_1$ is defined.

Network modelling. A transmission can be associated with several receivers, which are exactly those situated within the broadcast range of the node, i.e., they are its neighbours. A typical Trickle topology contains some gateway node $node_1$ in Figure 2, which can receive versions to be disseminated across the network from an external source. A network of XTA one per node can be naturally designed for such topology. Among those, XTA of gateway nodes are extended with extra transitions which allows the reception of new versions from the external source (transitions $Listen_1 \rightarrow ExtCheck_1$ and $Listen_2 \rightarrow ExtCheck_2$ with input $ExtVersion?myv$). The connections between XTA ports, defined by function $\mathcal{K}$, are inferred from topology connections: e.g., $\mathcal{K}(node_1.Version) = \{noed_2.Version, node_3.Version\}$ for the four-nodes topology depicted by the bidirectional graph in Figure 2. Note that ports of gateway nodes are implicitly connected to external source and can receive any value.

Illustration of network runs. Figure 3 depicts a simple sequence diagram together with a run of a simple two-nodes network of XTA. The run shows that both nodes exchange a version of value 0, that they both initially hold (see data valuations in initial state S_0). As the redundancy constant k is set to 1, the receiver node gets its counter saturated, i.e., c reaches k. Therefore, it suppresses its transmission. This is a typical trickle behaviour which reduces the number of transmissions (gossip) when the neighbourhood is up-to-date.

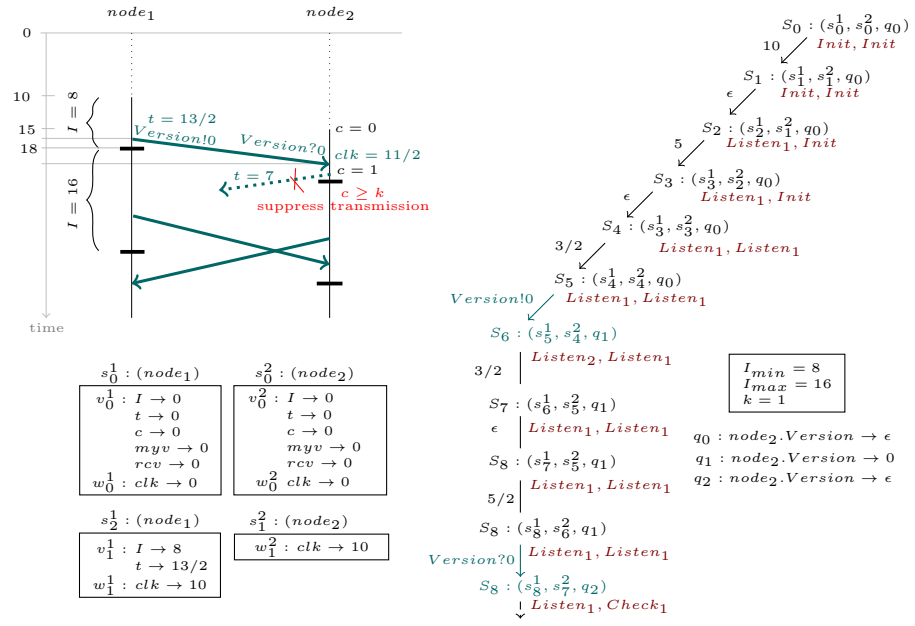

Fig. 3. A run of a two-nodes network of XTA.

4 Exploring Trickle with UPPAAL and DIVERSITY

UPPAAL model. We have created a model in UPPAAL, which corresponds to the network of XTA presented in Section 3. The UPPAAL model has a very similar structure in terms of states and transitions. Locations $Check_1$ and $Check_2$ have been declared as *Committed* (marked with a "C") which means that time cannot elapse in this location as intended in the original model. Also, such locations have a higher priority to be taken than non-committed ones. This reduces interleaving between automata if the latter are not executing on their turn transitions from committed locations. To implement asynchronous communication actions, we have created c-like functions in UPPAAL which implement fifo operations on queues. Unlike XTA which uses unbounded queues, those are arrays of fixed parameter size QUEUE_SIZE.

In UPPAAL, clocks are only compared to integer expressions, and clock guards are essentially conjunctions. This does not allow the specification of guards of the form $clk \leq t$ where t can take any random value in the dense interval $[I/2, I[$. In XTA, those values are (isomorphic to) positive rationals ($\mathbb{Q}_+$). In Figure 4, we propose a UPPAAL pattern so that values assigned with t are positive integers ($\mathbb{Z}_+$). This is compliant with the nature

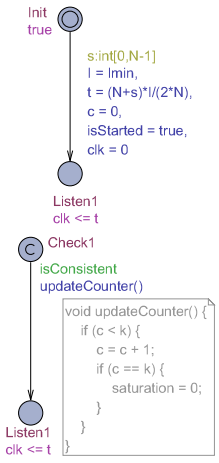

Fig. 4. UPPAAL transitions.

of clock constraints supported by UPPAAL, i.e., clocks in guards are bounded by integer expressions. On the other hand, UPPAAL provides a *select* statement $s : [L, U]$ on transition which selects a random value for an integer variable s within a specified integer interval. Interval bounds L and U are necessarily constants. It is equivalent to an update $L \leq: s :\leq U$ in XTA. The pattern allows the selection of at most N values for t within the current interval $[I/2, I[$: first an integer s is selected in the interval $[0, N-1]$, s is then used to assign t with a value denoted by the expression $e = (N+s) \cdot I/(2 \cdot N)$, however t is an integer variable, so t will be assigned exactly by the greatest integer less than or equal to the valuation of e. For instance, for $I = 8$ (resp. for $I = 16$) and $N = 4$, t can be assigned with the four integers 4, 5, 6 and 7 (resp. 8, 10, 12 and 14). The variable t can have an infinite number of possible values within the second half of I; the pattern allows exploring with UPPAAL just a few (at most N integers, we experiment with small values). But since we are interested in reachability, if a solution exists for those, the verification concludes.

Model exploration in UPPAAL. The tool uses the notion of *zone* to represent the set of valuations of clocks symbolically. A zone is defined by the conjunction of difference constraints of the form $clk \prec s$ or $clk - clk' \prec s$ where s is an integer. The simulation graph in UPPAAL is composed of nodes of the form $((l_1, v_1), \ldots, (l_n, v_n), Z, v)$ where l_i is the location (resp. v_i is the data valuation) for the i^{th} automaton, Z is a zone over clocks of the n involved automata, and v is the valuation of global or shared data variables. An example of such node is $((Listen_1, v_0^1), (Init, v_0^2), node_1.clk \leq 4, v_0)$ with notation of Figure 3, v_0 associates Trickle constants to their values, it sets queues to empty at start. The exploration of the graph uses inclusion on zones which checks whether a zone of a successor node in the graph is already covered by some zones of previously explored nodes. In case of inclusion, data valuations must coincide in order to prune the search. This helps master the search when infinite cycles exist. Intuitively, a typical cycle is when all nodes reach I_{max}, share the same version and the content of the queues coincides, then same behaviours will start over again. To enable detect this situation: i) we choose small values for I_{max} and ii) we stop increment the counter c once it reaches k (see transition $Check_1 \rightarrow Listen_1$ in Figure 4). Note that *saturation* is an extra clock of the node that will be discussed later; obviously, this has no effect on Trickle behaviour since the decision to suppress transmission depends only on reaching exactly k. In fact, after exchanging k or many more redundant versions is similar concerning subsequent behaviours. Otherwise, counter c will be assigned differently depending on the number of received versions. In which case, matching data valuations fails despite zone inclusion, and the cycle never exit. UPPAAL provides classical search strategies Depth-First Search (DFS) and Breadth-First Search (BFS), as well as Random Depth-First Search (RDFS). As we consider reachability, so one solution is wanted, DFS or RDFS are typically the most efficient option according to

the tool documentation. When applying a DFS (or RDFS), inclusion on zones is of practical use as is it avoids getting lost in an infinite cycle.

Reachability properties in UPPAAL. The tool supports a subset of Computation Tree Logic (CTL) [9]. As we are interested in reachability, we propose to use CTL formulae of the form $E\Diamond\phi$ where $\phi = f \wedge g$ is a formula on data variables and clocks. The satisfaction of such formula is defined on the tree with root an S_0 extracted from the labelled transition system of the network (see Section 2). The operator E quantifies over paths (or runs) of such a tree: it checks if there exists a path (with root S_0) in the tree satisfying the sub formula $\Diamond\phi$, the latter on the other hand is satisfied by that path if a state S satisfying ϕ occurs in the path. A simple property is $E\Diamond(node_2.c \geq k \wedge node_2.clk = node_2.I)$. The property is satisfied by the run discussed in Figure 3, in which the node $node_2$ suppresses its transmission. Let us discuss the following two properties (expressed for the four-nodes topology of Figure 2.

$$\textit{Updated}: E\Diamond((node_1.myv = NEW) \wedge (node_2.myv = NEW) \wedge (node_3.myv = NEW) \wedge (node_4.myv = NEW))$$

$$\textit{Outdated}: E\Diamond((node_4.isStarted) \wedge (node_4.myv = OLD) \wedge (node_2.myv = NEW) \wedge (n_2.c \geq k) \wedge (node_3.myv = NEW) \wedge (n_3.c \geq k)) \wedge ((n_2.saturation \geq D) \wedge (n_3.saturation \geq D))$$

The first property states that it is possible that all the nodes are updated. The second property states that there exists a node which is still outdated (hols an old version) while its neighbours are all updated, yet they have suppressed their transmissions. We use an extra clock *saturation* per node which is reset when the counter c reaches k (see Figure 4). The clock measures the delay elapsed since then, and the formula requires that such delay is bounded by a parameter $D > 0$. This situation is not desirable, especially after having observed numerous exchanges of messages in the networks.

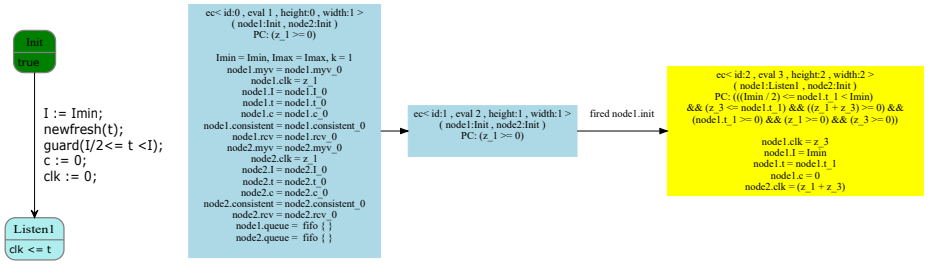

Fig. 5. Symbolic execution of a DIVERSITY transition.

DIVERSITY models and Symbolic execution. DIVERSITY [25] tools provides symbolic execution for state-based models(e.g., [4]) involving data expressions, data and clock guards. Input parameters or fresh symbols (they are used only once) can substitute uninitialized variables, reception variables used in communication with the external environment or any variable using a dedicated explicit

newfresh statement. The latter is of the form newfresh(x); it associates the variable x with a new fresh symbol. We have developed a DIVERSITY model of the XTA network for Trickle, as the UPPAAL model it has a similar structure in terms of states and transitions. DIVERSITY provides communication over unbounded fifo queues. We declare a single queue per node as each node owns only one port for internal communications. Those are initially empty, and their size is automatically adjusted by the tool as communication actions are evaluated. Figure 5 depicts a DIVERSITY transition ($Init \rightarrow Listen_1$). It suggests a pattern in DIVERSITY which allows to assign the variable t with a random value within current interval $[I/2, I[$: newfresh(t) associates with t with a new fresh symbol, the latter is constrained by subsequent *guard statement* $I/2 \leq t < I$. It is possible in DIVERSITY to declare I_{min}, I_{max}, I and t to be typed as a positive rational numbers ($\mathbb{Q}_+$) so as to be compatible with clock clk. DIVERSITY computes the so-called symbolic tree in which nodes are called *execution contexts ec*: they store pieces of information about the execution including the current location, about the transition which allows reaching the context, and importantly about Path Conditions and Substitutions of data variables, clocks, queue places, ... by arithmetical expressions over input parameters. Figure 5 depicts the symbolic execution step of the previous transition (being of $node_1$) from ec_1. The context ec_2 is reached, the transmission variable t is substitutes by a new symbol $node_1.t_1$ (substituted by t_0 in initial context ec_0), t_1 is constrained by the PC sub-formula $((I_{min}/2) \leq node_1.t_1 < I_{min})$. Note that sub-formula $(z_3 \leq node1.t_1)$ shows that time elapsing in ec_2 (denoted by duration symbol z_3) is bounded by the value denoted by $node1.t_1$. This condition corresponds to the evaluation of clock invariant $clk \leq t$ in location $Init$ for $node_1$ ($node_1.clk$ is substituted by z_3 in ec_2). Symbolic execution techniques characterize all intended runs. DIVERSITY provides different classical search strategies which can be used to unfold the symbolic tree from the initial context ec_0 up to criteria on tree size (depth, width or number of nodes). Naturally, this results in a huge tree. DIVERSITY provides heuristic search [25] guided by a user-specified sequence of transitions, possibly non-consecutive as it is difficult in general to guess strict sequencing when it comes to automata network. We use UPPAAL to find such a sequence corresponding to some runs satisfying a user-specified property. Let us now overview the connection between both tools.

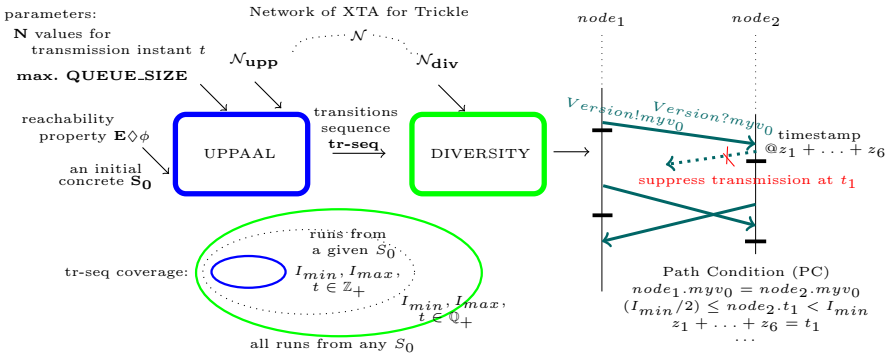

Fig. 6. Workflow UPPAAL-DIVERSITY.

Tools workflow and experiments. The workflow is given in Figure 6. UPPAAL takes as input a user-specified property $E\phi$. The aim is to check the property from an initial state S_0. Parameters on the number N of transmission instants t to be selected within the Trickle interval have to be given together with a bound QUEUE_SIZE on communication queues. The property is assessed on the UPPAAL model $\mathcal{N}_{upp}$. In case of the property is verified, i.e., there exists at least a run from S_0 which satisfies ϕ (such runs are schematically depicted by a blue ellipse), then the sequence of covering transitions tr-seq is derived from the UPPAAL solution. The idea now is to compute the equivalence classes composed by all runs from any sates S_0 which are covered by the sequence tr-seq. For this, a symbolic exploration guided by the sequence is conducted on the DIVERSITY model $\mathcal{N}_{div}$, a symbolic path together with its Path Condition (PC) is obtained then. The path is naturally feasible (satisfiability of its PC is assessed with SMT solvers). It identifies all runs covered by tr-seq with dense domain for I_{min}, I_{max} and t as being positive rationals (those are schematically depicted by a green ellipse). Since the path often represents pairwise communication actions from different automata, it can be depicted in a natural manner as a Sequence Diagram (SD). Figure 7 depicts an SD which highlights an outdated node situation, that of $node_4$. It has been computed by DIVERSITY for the four-nodes grid topology given in Figure 2. The situation is atypical: neighbours of $node_4$, that is $node_2$ and $node_3$ are first updated by gateway $node_1$ with a new version (green messages), they hence reset their interval to I_{min}; right away, $node_4$ gets them to reset their interval again by transmitting its old version (blue messages), their transmissions are postponed; this somehow gives $node_1$ time to retransmit the new version and saturate them (orange messages); therefore they suppress their transmissions for $node_4$. This unfortunate circumstance for $node_4$ can be prevented by increasing the redundancy constant so that its neighbours, $node_2$ and $node_3$, can still transmit, yet this comes at a cost, the number of messages increases for the entire nodes lifetime. Or $node_4$ can still be updated later because it together with $node_1$ will get their intervals doubled, and their transmission instants are at least dephased of I_{min} of those of $node_2$ and $node_3$

which leaves them time to update $node_4$. This is in favour of using Trickle, even if a node is in such outdated situation, it will not remain for a long time, thanks to the dynamic interval adjustment which avoids flood the network with messages. We have experimented with more nodes in the grid (up to 9), Table 1 reports on those. *Updated* and *Outdated* properties were successfully checked, which is satisfactory given the non-trivial kind of interactions in case of outdated nodes.

	Updated			Outdated		
Nodes	UPPAAL	DIVERSITY	Messages	UPPAAL	DIVERSITY	Messages
3	1ms	1s542ms	7	12ms	4s156ms	8
4	3ms	2s731ms	16	984ms	9s796ms	12
5	4ms	9s395ms	12	168ms	11s828ms	12
6	7ms	32s750ms	21	7s621ms	1m28s953ms	23
7(2)	6ms	29s453ms	29	5s772ms	2m5s375ms	26
7(3)	5ms	49s375ms	29	3s227ms	1m42s718ms	24
8	8ms	59s640ms	41	8m9s503ms	9m36s375ms	42
9	9ms	2m58s408ms	64	9m19s22ms	15m36s11ms	62

The nodes graph is bidirectional in the form of a grid topology (See Figure 2), 7(2) (resp. 7(3)) denotes that $node_7$ is two blocks (resp. three blocks) from the gateway $node_1$. Results measured on an Intel Core i7-7920HQ processor with RAM 32GB, the redundancy constant k was set to 1 the minimum transmission case in Trickle. DIVERSITY time includes PC check with CVC4. UPPAAL concluded in few more trials inline with the exponential growth in running time.

Table 1.

5 Conclusion

We have developed first models for desynchronized Trickle network. Those models use data to express adjustable transmission intervals and abstract transmitted information. To assess reachability properties, we combine model checking and symbolic execution. If the property is verified on the model in which data is concrete, we derive from the returned solution a sequence of transitions that guides the symbolic execution to compute the corresponding symbolic path. The latter is a compact representation of the equivalence class of behaviours for the transitions coverage and can be depicted in the user-friendly format of sequence diagrams to enable their understanding. For future work, we plan to extract from the model-checking solution more information about structural coverage of the mini-language of updates on transitions to refine the equivalence classes by symbolic execution. We also plan to experiment with other gossip protocols. We believe that our approach can be of practical use to highlight for those non-trivial nodes interactions and give hints on their benefit for transmissions control.

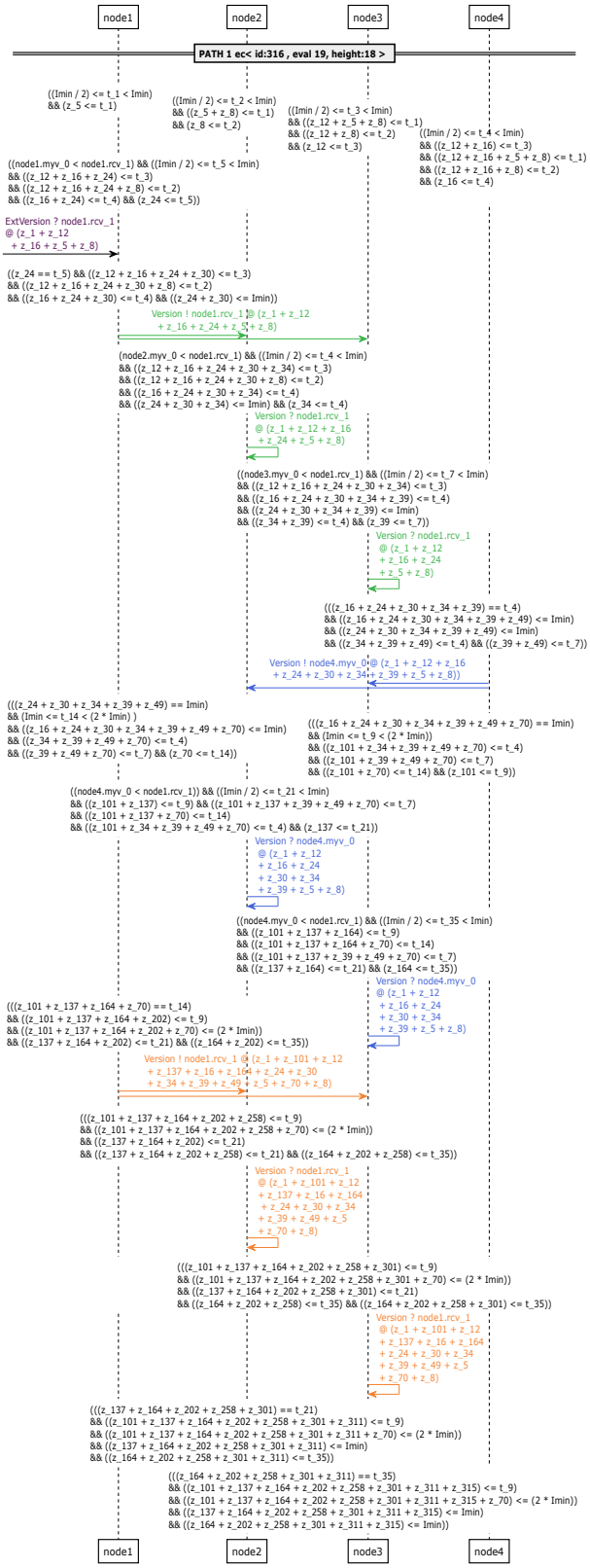

Fig. 7. Outdated node situation - Sequence Diagram generated by DIVERSITY.

References

1. RPL: Ipv6 routing protocol for low-power and lossy networks, request for comments: 6550. Technical report, Cooper Power Systems and Cisco Systems and Stanford University, March 2012.
2. R. Alur and D. Dill. A theory of timed automata. *Journal Theoretical Computer Science*, 1994.
3. R. Alur and D. Dill. A theory of timed automata. *Journal Theoretical Computer Science, Volume 126 Issue 2, April 25, 1994, Pages 183 - 235*, 1994.
4. Bannour B., Escobedo J. P., Gaston C., and Le Gall P. Off-line test case generation for timed symbolic model-based conformance testing. In *Int. conf. ICTSS*. Springer, 2012.
5. Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. Springer, 2011.
6. M. Becker, K. Kuladinithi, and C. Görg. Modelling and simulating the trickle algorithm. In *MONAMI*. Springer.
7. Michael J. Breza and Julie A. McCann. Lessons in implementing bio-inspired algorithms on wireless sensor networks. In *NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2008, Noordwijk, The Netherlands, June 22-25, 2008*, pages 271–276. IEEE Computer Society, 2008.
8. Z. Chen, D. Zhang, R. Zhu, Y. Ma, P. Yin, and F. Xie. A review of automated formal verification of ad hoc routing protocols for wireless sensor networks. *CoRR*, abs/1305.7410, 2013.
9. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *25 Years of Model Checking - History, Achievements, Perspectives*, 2008.
10. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 2001.
11. T. Coladon, M. Vucinic, and B. Tourancheau. Multiple redundancy constants with trickle. In *PIMRC*. IEEE, 2015.
12. L. Mendonça de Moura and N. Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *International Conference, TACAS*. Springer, 2008.
13. D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems, International Workshop*. Springer, 1989.
14. J. Dong, J. Sun, J. Sun, K. Taguchi, and X. Zhang. Specifying and verifying sensor networks: an experiment of formal methods. *International Conference on Formal Engineering Methods*, 2008.
15. A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *IEEE ICLCN*, 2004.
16. J. Hui and R. Kelsey. Multicast protocol for low-power and lossy networks, request for comments: 7731. Technical report, Silicon Labs, February 2016.
17. C.King J. Symbolic execution and program testing. *Communications of the ACM, Volume 19*, July 1976.
18. H. R. Kermajani, C. Gomez, and M. H. Arshad. Modeling the message count of the trickle algorithm in a steady-state, static wireless sensor network. *IEEE Communications Letters*, 2012.

19. M. Z. Kwiatkowska, G. Norman, and D. Parker. Analysis of a gossip protocol in PRISM. *SIGMETRICS Performance Evaluation Review*, 2008.
20. M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. Springer, 2011.
21. B. Lee, H. K. Song, Y. Suh, K. H. Oh, and H. Y. Youn. Energy-efficient gossiping protocol of wsn with realtime streaming data. In *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, 2014.
22. P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. *TinyOS: An Operating System for Sensor Networks*. Springer, 2005.
23. P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Int. Symp. NSDI*. USENIX Association, 2004.
24. Y. Liu, J. Sun, and J. S. Dong. PAT 3: An extensible architecture for building multi-domain model checkers. In T. Dohi and B. Cukic, editors, *IEEE 22nd International Symposium on Software Reliability Engineering, ISSRE 2011, Hiroshima, Japan, November 29 - December 2, 2011*. IEEE Computer Society, 2011.
25. Arnaud M., Bannour B., and Lapitre A. An illustrative use case of the DIVERSITY platform based on UML interaction scenarios. *Electr. Notes Theor. Comput. Sci.*, 2016.
26. T. Meyfroyt. Modeling and analyzing the trickle algorithm. *Master's Thesis, Eindhoven University of Technology, The Netherlands*, 2013.
27. T. Meyfroyt, Sem C. Borst, Onno J. Boxma, and Dee Denteneer. On the scalability and message count of trickle-based broadcasting schemes. *Queueing Syst.*, 2015.
28. T. M. M. Meyfroyt. An analytic evaluation of the trickle algorithm: Towards efficient, fair, fast and reliable data dissemination. In *WoWMoM*. IEEE.
29. X. Nan, M. Fei, and T. Yang. Randomized and efficient time synchronization in dynamic wireless sensor networks: a gossip-consensus-based approach. *Vol 2018 Complexity*, 2018.
30. N. M. T. Nguyen, B. Bannour, A. Lapitre, and P. Le Gall. Behavioral models and scenario selection for testing iot trickle-based lossy multicast networks. In *VVIoT@ICST workshop*. IEEE, 2019.
31. M. Vucinic, M. Król, B. Jonglez, T. Coladon, and B. Tourancheau. Trickle-D: High fairness and low transmission load with dynamic redundancy. *IEEE IoT Journal*, 2017.
32. M. Webster, M. Breza, C. Dixon, M. Fisher, and J. A. McCann. Formal verification of synchronisation, gossip and environmental effects for wireless sensor networks. *ECEASST*, 2018.
33. M. Woehrle, R. Bakhshi, and M. Mousavi. Mechanized extraction of topology anti-patterns in wireless networks. In John Derrick, Stefania Gnesi, Diego Latella, and Helen Treharne, editors, *Integrated Formal Methods*. Springer, 2012.
34. M. Zheng, J. Sun, Yang Liu, Jin Song Dong, and Yu Gu. Towards a model checker for nesc and wireless sensor networks. In Shengchao Qin and Zongyan Qiu, editors, *Formal Methods and Software Engineering*. Springer, 2011.