Parametric verification to ensure safe behaviour of connected devices

Mathias Ramparison

Parametric verification to ensure safe behaviour of connected devices

Mathias Ramparison[0000−0001−6764−1214]

Univ. Grenoble Alpes, CNRS, Grenoble INP†, VERIMAG, UMR 5104, 38000, Grenoble, France

Abstract. With the surge of IoT devices, these systems must provide reliable services and make trustworthy decisions based on fast and accurate data analytics. Effective data analysis is crucial for IoT systems to make rapid decisions, gain insights, uncover hidden patterns, and interact with users and other systems efficiently. This work in progress aims to model connected devices as Parametric Timed Automata, allow them to share operating data on a blockchain and verify its good behaviour with this data. Operating data are seen as unknown constraints in Parametric Timed Automata and can be used to verify or improve models of connected devices, along with providing safety. We propose a framework to model simple connected devices, upload and download parameter through smart contracts and verify the safety of the produced behaviour.

1 Introduction

Experts predict that smart devices will generate a very large volume of data globally, potentially reaching 80 ZB by 2025 [Cor21] generated by 55.7 billion connected IoT devices. This raises questions about the future of this automatically generated data: who will store it, where, and for what purpose? In the data-driven economy, manufacturers of connected devices aim to collect as much user data as possible. The generated data has become more valuable than the device itself, serving to enhance the product's functionality or to be sold to other manufacturers. Companies' marketing departments will utilize this data to promote related products, such as applications or energy-related items: keep private data under control remains a tough task.

Connected devices face significant security challenges, particularly in terms of security and formal specifications [Sin21]. As these devices can be considered embedded systems, formal specifications are crucial. Flaws in embedded system design have historically led to project failures, such as the Ariane 5 rocket explosion in 1996* due to a simple but hard-to-detect conversion error [Den]. To ensure safety and security in modern connected devices a more secure, design-oriented philosophy is required. This involves formally model a system and its

†Institute of Engineering Univ. Grenoble Alpes

*exploded 36.7 seconds after its launch, due to a bad 64 to 16 bits conversion (an easily correctable error but humanly hard to spot).

functionalities before coding and ensuring that the written code adheres to the initial model. On top of that, energy consumption optimization is a key point of smart devices while many of these devices use proprietary software and hardware, limiting users' ability to customize them freely.

In [CAR22] the authors propose a formal framework for IoT architectures based on Discrete Event System Specification to enhance resource management and energy efficiency in edge computing applications for centralized architectures. Besides, [HKB17] focuses on the formal verification of the communication protocol between connected objects.

Outline. This work aims to help users build models of connected device which can share, using the blockchain, data from a specific usage. Devices are allowed to download new data and update their connected devices with global settings computed by similar connected devices. In Section 2 we present formal verification through the Parametric Timed Automata formalism. In Section 3 we exemplify the interest of sharing verified data to optimize the use of connected devices. In Section 4 we discuss a framework to upload and download data from connected devices in order to create a collaboration process through a blockchain. Finally in Section 5, we expose the future scientific directions brought by this work and conclude in Section 6.

2 Parametric Timed Automata to model IoT devices

Formal methods are a powerful tool to prove the good behaviour of embedded systems. Precisely, Timed Automata theory is a mature research field that has been under studies for more than 20 years. Timed Automata theory is an excellent way to model and analyze timed systems [AD94] *i. e.,* systems that evolve according to time. With this formalism it is possible to model a system and prove whether a given timed property is respected. It can be described a tuple composed of a set of clocks that model time elapsing, locations that represent physical states of a system, and transitions between locations, possibly augmented with guards which are time constraints involving clocks that must be true to take the transition. A more powerful method to model timed systems where not all timed constants are known is the use of Parametric Timed Automata (PTA). In these models, constraints involving clocks can contain unknown variables [AHV93] called *parameters.*

2.1 Clocks, parameters and parametric clock constraints

We assume a set $\mathbb{X} = \{x_1, \ldots, x_H\}$ of *clocks*, *i. e.,* real-valued variables that evolve at the same rate. A clock valuation is a function $v : \mathbb{X} \to \mathbb{R}_+$. We identify a clock valuation v with the point $(v(x_1), \ldots, v(x_H))$ of $\mathbb{R}_+^H$. We write $\vec{0}$ for the clock valuation assigning 0 to all clocks. Given $d \in \mathbb{R}_+$, $v + d$ denotes the valuation s.t. $(v + d)(x) = v(x) + d$, for all $x \in \mathbb{X}$. Given $R \subseteq \mathbb{X}$, we define the *reset* of a valuation v, denoted by $[v]_R$, as follows: $[v]_R(x) = 0$ if $x \in R$, and $[v]_R(x) = v(x)$ otherwise.

We assume a set $\mathbb{P} = \{p_1, \ldots, p_M\}$ of *parameters*, *i. e.,* unknown constants. A *parameter valuation* pv is a function $pv : \mathbb{P} \to \mathbb{Q}_+$.

We assume $\bowtie \in \{<, \leq, =, \geq, >\}$ and $\triangleleft \in \{<, \leq\}$. A *parametric clock constraint* pcc is a constraint over $\mathbb{X} \cup \mathbb{P}$ defined by a set of inequalities of the form $x \bowtie \sum_{1 \leq i \leq M} \alpha_i p_i + d$, with $\alpha_i \in \{0, 1\}$ and $d \in \mathbb{Z}$. Given pcc, we write $v \models pv(pcc)$ if the expression obtained by replacing each x with $v(x)$ and each p with $pv(p)$ in pcc evaluates to true.

2.2 Parametric timed automata

We recall PTAs [AHV93].

Definition 1. *A PTA $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, l_0, \mathbb{X}, \mathbb{P}, I, E)$, where: i) Σ is a finite set of actions, ii) L is a finite set of locations, iii) $l_0 \in L$ is the initial location, iv) $\mathbb{X}$ is a finite set of clocks, v) $\mathbb{P}$ is a finite set of parameters, vi) I is the invariant, assigning to every $l \in L$ a parametric clock constraint $I(l)$, vii) E is a finite set of edges (or transitions) $e = (l, g, a, R, l')$ where $l, l' \in L$ are the source and target locations, $a \in \Sigma$, $R \subseteq \mathbb{X}$ is a set of clocks to be reset, and the guard g is a parametric clock constraint.*

Given a parameter valuation pv, we denote by $pv(\mathcal{A})$ the non-parametric structure where all occurrences of a parameter p have been replaced by $pv(p)$. We denote as a *timed automaton* any structure $pv(\mathcal{A})$.[†]

Let us recall the concrete semantics of TAs.

Definition 2 (Concrete semantics of a TA). *Given a PTA $\mathcal{A} = (\Sigma, L, l_0, \mathbb{X}, \mathbb{P}, I, E)$, and a parameter valuation pv, the concrete semantics of $pv(\mathcal{A})$ is given by the timed transition system $(S, s_0, \to)$, with*

- $S = \{(l, v) \in L \times \mathbb{R}_+^H \mid v \models pv(I(l))\}$,
- $s_0 = (l_0, \vec{0})$
- $\to$ *consists of the discrete and (continuous) delay transition relations:*
  - *discrete transitions:* $(l, v) \overset{e}{\mapsto} (l', v')$, *if* $(l, v), (l', v') \in S$, *there exists* $e = (l, g, a, R, l') \in E$, $v' = [v]_R$, *and* $v \models pv(g)$.
  - *delay transitions:* $(l, v) \overset{d}{\mapsto} (l, v + d)$, *with* $d \in \mathbb{R}_+$, *if* $\forall d' \in [0, d], (l, v + d') \in S$.

Moreover we write $(l, v) \xrightarrow{e} (l', v')$ for a combination of a delay and discrete transition where $((l, v), e, (l', v')) \in \to$ if $\exists d, v'' : (l, v) \overset{d}{\mapsto} (l, v'') \overset{e}{\mapsto} (l', v')$.

Given a TA $pv(\mathcal{A})$ with concrete semantics $(S, s_0, \to)$, we refer to the states of S as the *concrete states* of $pv(\mathcal{A})$. A *run* of $pv(\mathcal{A})$ is a possibly infinite alternating sequence of states of $pv(\mathcal{A})$ and edges starting from the initial state s_0 of the form $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{m-1}} s_m \xrightarrow{e_m} \cdots$, such that for all $i = 0, 1, \ldots$, $e_i \in E$, and $(s_i, e_i, s_{i+1}) \in \to$.

[†]We should use a rescaling of the constants to avoid comparisons of clocks with rationals: by multiplying all constants in $pv(\mathcal{A})$ by the least common multiple of their denominators, we obtain an equivalent (integer-valued) TA, as defined in [AD94].

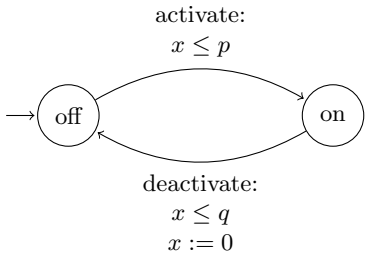

Fig. 1: PTA that models an electric radiator

Given a state $s = (l, v)$, we say that s is reachable if s appears in a run of $pv(\mathcal{A})$, or simply that l is reachable in $pv(\mathcal{A})$, if there exists a state (l, v) that is reachable.

Example 1. Consider an electric radiator that turns on for a specific period of time. Fig. 1 models the behaviour of the radiator depending on time with:

1. L : {on, off}, $\mathbb{X}$: $\{x\}$, Σ : {activate, deactivate}
2. guard $g(activate) = \{x \leq p\}$, $g(deactivate) = \{x \leq q\}$
3. reset $R(deactivate) = \{x\}$ as $x := 0$.

In the initial state the radiator is off, the clock x starts from 0 and its value increases monotonically. It might activate itself within p units of time, p being an unknown constant, called a *parameter*. Once it is activated, it might deactivate itself within $q - p$ units of time, with q a parameter as well. This transition is possible if and only if $p \leq q$. Finally, x is reset to 0 when the radiator is turned off. Questions such as, given values for p and q, "is it possible my radiator never turn off?" can be answered using this model. For more complex and general systems (*i. e.,* involving more states, clocks and parameters) such questions remain undecidable [AHV93].

The formula "EF" expresses *reachability* [AD94], that is EF-emptiness asks whether the set of parameter valuations for which a given location is reachable for at least one run is *empty* or not. We focus notably on EF-synthesis that asks to *synthesize* these parameter valuations, *i. e.,* given a PTA $\mathcal{A}$ and a location l, synthesize the set $\{pv \mid \exists s_0 \xrightarrow{e_0} (l_1, v_1) \xrightarrow{e_1} \cdots \xrightarrow{e_{m-1}} (l, v)$ a run of $pv(\mathcal{A})\}$. In our example Fig. 1, we consider reachability of the location *off*.

3 Improve models using collected data resulting from collaboration

From daily data sent by every connected devices to the blockchain, we try to improve parameterized models in order to optimize the device operations and behaviour.

We take again as an example our connected radiator, but this time working along with a thermometer. We depict this scenario in Fig. 2. Using the basic model with timed automaton of Fig. 1, we valuate the parameters p and q with a parameter valuation pv, say $pv(p) = 60$ minutes and $pv(q) = 90$ minutes. With these values for parameters, the radiator can estimate how much time it is in an active state. Combining this with its known hourly electricity consumption it can estimate its daily consumption in order to keep a room at a given temperature.

Given another radiator at different places, say one has a lower daily electricity consumption to keep the same average temperature. It can upload its settings (*i. e.,* its values for parameters p and q) to the blockchain so other connected radiators can download these new parameter values and try to apply them both, or one of them only, or even try to produce more accurate values for the model by using approached values of $pv(p)$ and $pv(q)$.

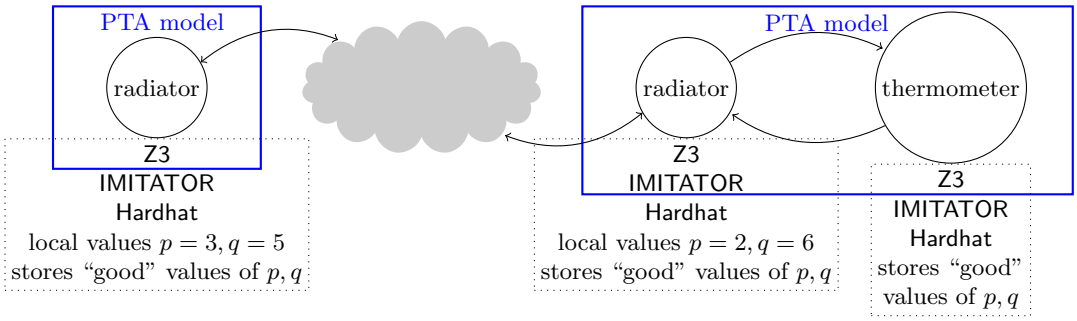

Fig. 2: Scenario of a pair radiator/thermometer connected to a distant radiator

In the following we propose to build a formal verification framework that is portable on low resources devices. We use a smart contract to store values that can be used in our model, *e. g.,* maximum and minimum time values that allow our radiator to turn on and off. Suppose a connected device wants to upload data. It can download a PTA model that represents its behaviour. It downloads from to the blockchain the latest parameter values that are suited for its environment and tasks *e. g.,* from other connected devices of the same type (we can use categories of devices, say connected radiator with connected radiators, connected fridge with connected fridges...) or create new values and verify these values do not contradict the current constraints used in the model with the parametric model checker IMITATOR [AFKS12] and the constraint solver Z3 [dMB08]. The output is whether the model guarantees the "good" behaviour of the connected devices for these parameter values. Finally, if data is not compromising the behaviour of the device category, the parameter values are validated and new values are stored on the blockchain.

4 Communication and offline parametric verification

We implemented the following smart contracts, fully portable on the main Ethereum blockchain, to allow connected devices to:

1. upload data on their behaviour;
2. download data in order to modify their behaviour;
3. run offline a model checking algorithm to verify stored data.

In this work in progress, we do not automatically collect data from the connected devices but use a set of predefined values to illustrate the behaviour of our models.

We describe the components we use in the following, which are Hardhat, Z3 and IMITATOR.

Hardhat is a comprehensive development environment and task runner for Ethereum smart contract development, designed to streamline the process of building, testing, and deploying smart contracts. It offers features such as a local Ethereum network for testing, a built-in testing framework, Solidity compilation, and a plugin system for extended functionality. Hardhat's focus on developer experience, coupled with its robust toolset including debugging capabilities, network management, and Typescript support, has made it a popular choice among Ethereum developers.

Z3 [dMB08] is a powerful theorem prover and SMT solver developed by Microsoft Research, designed to determine the satisfiability of logical formulas over various theories. It's widely used in software verification, program analysis, and formal methods, helping developers and researchers prove properties about programs, find bugs, and solve complex logical and mathematical problems. Z3 is known for its high performance and ability to handle large and complex formulas efficiently, making it valuable in both academic research and industrial applications. The solver provides APIs for several programming languages, facilitating its integration into various tools and applications across different domains of computer science and software engineering.

IMITATOR [AFKS12] is an open-source model checker specifically designed for the analysis of parametric timed systems, using PTA as its underlying formalism. It allows for the modeling, verification, and parameter synthesis of real-time systems where timing constants may be unknown or variable. IMITATOR can perform various tasks including parameter synthesis (finding parameter values that satisfy given properties), robustness analysis, and model checking of parametric systems. The tool's ability to handle systems with parametric constraints sets it apart from traditional model checkers, enabling more flexible and comprehensive analysis of time-dependent systems. IMITATOR natively supports operations on rational parametric fixed variable to model *e. g.,* costs or weights.

The development constraints were the following:

– IMITATOR synthesizes constraints on an infix form *e. g.,* $2 + 2 * 4 \leq 11$ read from left to right, and is not meant to check that a conjunction of a set of constraints is satisfiable; it must run on a low resources device;

– Hardhat uses Typescript/Javascript; it must run on a low resources device;
– Z3 most developed APIs are for Java and Python. The one for Typescript/-Javascript runs with constraint of similar form, *e. g.,* $2.add(2.mul(4)).le(11)$ but not with an infix form *e. g.,* $\leq +2 * 2\,4\,11$ which is easier to parse.

The developed solution[‡], all in Typescript working with Hardhat, processes as follows:

We model *e. g.,* a radiator as shown in Fig. 1 in the IMITATOR format, and set up IMITATOR and a Hardhat node on a low resources device such as a Raspberry Pi. This is one node, the left one shown in Fig. 2. Using IMITATOR we output the set of values for the device to run as expected *i. e.,* synthesize the parameter values s.t. "off" is reachable[§] and this set is $0 \leq q - p$. We store "good" parameter values p and q (*e. g.,* $pv(p) = 3$ and $pv(q) = 5$) on the Hardhat node as the initialization. We can also decide to use our own parameter values. Recall that the parameter values that are given in the example are predefined values that are used to illustrate the behaviour of our model.

We fetch two parameters values p and q and along with the constraints (*e. g.,* $0 \leq q - p$) given by IMITATOR we:

– create a Z3 solver;
– transform the constraints in prefix notation (*e. g.,* $\leq - q\,p\,0$);
– replace parameters by their values (*e. g.,* $\leq - 5\,3\,0$);
– transform the constraints into a well-parenthesized expression (*e. g.,* $(0 \leq (5 - 3))$) and in Z3 notation (*e. g.,* $0.le(5.sub(3))$);
– check whether the constraint is satisfiable.

Now if the constraint is satisfiable, we consider that the parameter valuation s.t. $pv(p) = 3$ and $pv(q) = 5$ ensures the good behaviour of our system.

As we have used model-checking techniques, the good behaviour is proven under the given parameter valuations. However, a model checker needs computing resources that may not be available on some IoT hardware. We prove the good behaviour of the model of our system and assume the model is as close as possible to the actual device.

5 Future directions

Formally model a network of connected devices and prove that security and safety constraints hold is a complex collaborative project.

Automatic connection. The next development step includes the ability for connected devices to verifies on-the-fly that safety and security properties hold each time a new device is added to or removed of the network. Indeed, it would allow connected device to interact together with respect to some security constraints previously required, resulting in a win-win scenario. This can be done without the

[‡]Code, examples and technical environment can be found at https://gricad-gitlab.univ-grenoble-alpes.fr/MRprojects/imitatortoz3hardhat/

[§]Note that the radiator can always stay off which is considered a safe behaviour

need of a human third-party: the decentralized network becomes autonomous, especially in the case of nearby devices.

Light model checkers. Model checkers tend to be greedy in resources. We also would like to develop light versions of model checkers *e. g.,* Uppaal for Timed Automata [BLL+95] and Romeo for Petri Nets [GLMR05]. By compiling a light model checker version that runs on other architectures such as Arduino (ours runs on ARM Raspberry Pi 32bits), we could increase the number of architectures and therefore devices that support our framework. In that case, we could split the model checking and the blockchain communication on separated devices if the hardware lacks resources.

Security and safety models. In order to enlarge the capabilities of our models, we could use attack-fault trees as a mean to model safety failures. Such models use simplified semantic to model possible events combinations that may lead to a system failure or security breach. Attack-fault trees can be used along with model checkers to improve their modeling power (see *e. g.,* [KS17,ALRS21].

Hybrid models. We are yet limited by the expressiveness of our models: PTAs can model timing constraints that evolve at the same rate and its extensions additionally model costs and weight values (see *e. g.,* Priced/Weighted Timed Automata [BFH+01,ALP04] and their parametric extensions [ALR21]). Hybrid Automata [HKPV98] offer the most suitable model as variables can evolve at different rates: this offers the ability to model more environmental parameters such as energy consumption and temperature. However, model checking in that case is more complex [Fre08,AFKS12].

Machine Learning. Machine learning can be valuable for classifying types of connected devices and enhancing their safety, such as by detecting maintenance needs early [SK23].

6 Conclusion

This on-going work offers a framework using TypeScript and Hardhat to model connected devices, such as radiators, using Parametric Timed Automata. We can upload and download operating data as parameters using smart contracts written in Solidity to share them with other connected devices on a blockchain to preserve integrity of the data. The process involves using IMITATOR to generate and verify parameter values that ensure the device operates as expected. These values are stored on a Hardhat node running on a low-resource device like a Raspberry Pi. The solution also uses the Z3 solver to check the satisfiability of constraints derived from IMITATOR when new parameter values are introduced. Finally, if the the behaviour of the model is not compromised by these new parameter values, they are stored on the blockchain and can be verified and used by other connected devices. Note that the model-checker is developed for classic desktop computer systems: it needs an IoT device with powerful computing resources and also depends on the accuracy of the model compared to the actual device. In future versions, we plan to optimize the model-checker for low resources devices and the models.

References

AD94. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.

AFKS12. Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer, 2012.

AHV93. Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601, New York, NY, USA, 1993. ACM.

ALP04. Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. *Theoretical Computer Science*, 318(3):297–322, 2004.

ALR21. Étienne André, Didier Lime, and Mathias Ramparison. Parametric updates in parametric timed automata. *Log. Methods Comput. Sci.*, 17(2), 2021.

ALRS21. Étienne André, Didier Lime, Mathias Ramparison, and Mariëlle Stoelinga. Parametric analyses of attack-fault trees. *Fundam. Informaticae*, 182(1):69–94, 2021.

BFH+01. Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In *HSCC*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.

BLL+95. Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer–Verlag, October 1995.

CAR22. Román Cárdenas, Patricia Arroba, and José Luis Risco-Martín. Bringing AI to the edge: a formal m&s specification to deploy effective iot architectures. *J. Simulation*, 16(5):494–511, 2022.

Cor21. International Data Corporation. Future of industry ecosystems: Shared data and insight. https://blogs.idc.com/2021/01/06/future-of-industry-ecosystems-shared-data-and-insights/, 2021. Accessed: 2024-06-30.

Den. E. Denney. Formal methods for the certification of auto-generated flight code. *NASA Ames Research Center*.

dMB08. Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In *TACAS, held as part of ETAPS,*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

Fre08. Goran Frehse. Phaver: algorithmic verification of hybrid systems past hytech. *Int. J. Softw. Tools Technol. Transf.*, 10(3):263–279, 2008.

GLMR05. Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier H. Roux. Romeo: A tool for analyzing time petri nets. In *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 418–423. Springer, 2005.

HKB17. Manel Houimli, Laid Kahloul, and Sihem Benaoun. Formal specification, verification and evaluation of the mqtt protocol in the internet of things. In *2017 International Conference on Mathematics and Information Technology (ICMIT)*, pages 214–221, 2017.

HKPV98. Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.

KS17. Rajesh Kumar and Mariëlle Stoelinga. Quantitative security and safety analysis with attack-fault trees. In *HASE*, pages 25–32. IEEE, 2017.

Sin21. Satyajit Sinha. State of iot 2024: Number of connected iot devices growing 13% to 18.8 billion globally. https://iot-analytics.com/number-connected-iot-devices/, 2021. Accessed: 2024-06-30.

SK23. Malik Abdul Sami and Tamim Ahmed Khan. Forecasting failure rate of iot devices: A deep learning way to predictive maintenance. *Comput. Electr. Eng.*, 110:108829, 2023.