# A Roadmap Towards Resilient Internet of Things for Cyber-Physical Systems

Denise Ratasich [1], Faiq Khalid [1], Florian Geissler [2], Radu Grosu [1], Muhammad Shafique [1], Ezio Bartocci [1]
[1] Institute of Computer Engineering, TU Wien, Austria
(e-mail: firstname.lastname@tuwien.ac.at)
[2] CPS Dependability Research Lab, Intel Corporation, Germany
(e-mail: florian.geissler@intel.com)

**Abstract**

The Internet of Things (IoT) is an ubiquitous system connecting many different devices – the *things* – which can be accessed from the distance. With the possibility to monitor and control the physical environment from the distance, that is the IoT contains cyber-physical systems (CPS), the two concepts of dependability and security get deeply intertwined. The increasing level of dynamicity, heterogeneity, and complexity adds to the system's vulnerability, and challenges its ability to react to faults. This paper summarizes state-of-the-art of existing surveys on anomaly detection, fault-tolerance and self-healing and adds a number of other methods applicable to achieve resilience in an IoT. We particularly focus on non-intrusive methods ensuring data integrity in the network. Furthermore, this paper presents the main challenges to build a resilient IoT for CPS. It further summarizes our solutions, work-in-progress and future work to this topic for the project "Trustworthy IoT for CPS". Eventually, this framework is applied to the selected use case of a smart sensor infrastructure in the transport domain.

## 1 Introduction

*Cyber-physical systems* (CPS) [1–4] are the new emerging smart information and communications technology (ICT) that are deeply affecting our society in several application domains. Examples include unmanned aerial vehicles (UAV), wireless sensor networks, (semi-) autonomous cars [5], vehicular networks [3] and a new generation of sophisticated life-critical and networked medical devices [6].

CPS consist of collaborative computational entities that are tightly interacting with physical components through sensors and actuators. They are usually federated as a system-of-systems communicating with each other and with the humans over the *Internet of Things* (IoT), a network infrastructure enabling the interoperability of these embedded devices.

As the advent of *Internet* has revolutionized the communication between humans, so the CPS and IoT are reshaping the way in which we perceive and interact with our physical world. This comes at a price: these systems are becoming so pervasive in our daily life that failures and security vulnerabilities can be the cause of fatal accidents, undermining their trustworthiness in the public eye. Fig. 1 depicts the evolution of embedded systems, their goals and requirements.

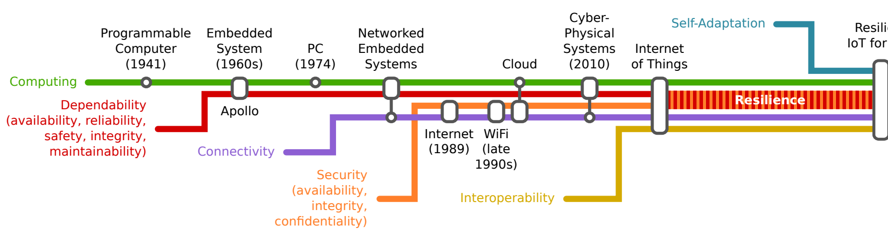


Figure 1: A brief history of computer systems and its roadmap towards resilient IoT for CPS.

Over the last years, popular mainstream newspapers have published several articles about CPS that are recalled from the market due to software and/or hardware bugs. For example in 2015, *The New York Times* published the news [7] about the finding of a software bug in Boeing 787 that could cause "the plane power control units to shut down power generators if they were powered without interruption for 248 days". *The Washington Post* has recently published an article [8] about Fiat Chrysler Automobiles NV recalling over 4.8 million U.S. vehicles for a defect that prevents drivers from shutting off cruise control, placing them in a potential hazard. The recent accident of Uber's self-driving vehicle killing a pedestrian held a world wide appeal in the press [9], raising several concerns about the safety and trustworthiness of this technology.

With the connection to the Internet, security becomes another crucial factor that is intertwined with safety ("if it's not secure it's not safe" [10]). The tight interaction between the software and the physical components in CPS enables cyber-attacks to have catastrophic physical consequences. *The Guardian* reported last year [11] that over half a million pacemakers has been recalled by the American Food and Drug Administration due to fears that hackers could exploit cyber security flaws to deplete their batteries or to alter the patient's heartbeat. In 2015 the BBC announced [12] that the black-out of the Ukraine power grid was the consequence of a malware installed on computer systems at power generation firms, enabling the hackers to get remote access to these computers. In the same year two hackers have proved in front of the media [13] that they could hijack a Jeep over the internet.

The rise of IoT, that is forecast to grow to 75 billions of devices in 2025 (Fig. 2), is exacerbating the problem, by providing an incredibly powerful platform to amplify these cyber-attacks. An example is the MIRAI botnet that in 2016 have exploited more than 400000 devices connected through the IoT as a vehicle to launch some of the most potent distributed denial-of-service (DDoS) in history [14].

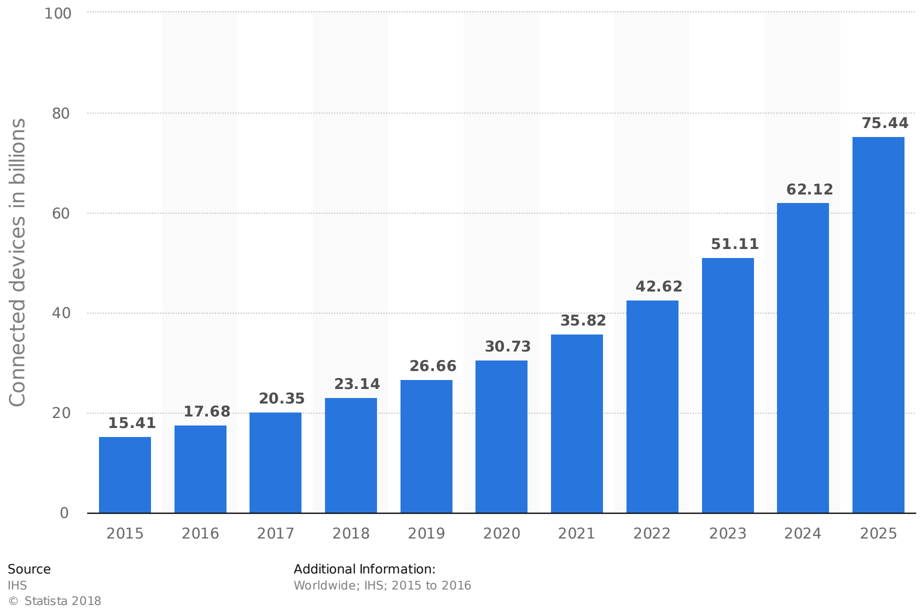


Figure 2: Internet of Things (IoT) connected devices installed worldwide from 2015 to 2025 (in billions) [15].

Managing and monitoring such ultra large scale system is becoming extremely challenging. A desired property to achieve/enforce for this, is to be *resilient*: the service delivery (or functionality) that can justifiably be trusted persists, when facing changes [16]. In other words, the system shall remain safe and secure in the advent of faults and threats (see Fig. 3 for some examples in the automotive domain). Furthermore the resilience mechanisms have to evolve and adapt with the system, meaning the system shall remain resilient in the presence of faults probably not even considered during design time [10, 16].

This paper gives an overview on the state-of-the-art (SotA) to resilience for the IoT. Methods from surveys are summarized and complemented with recent publications. Due to the expected heterogeneous architecture we specifically target non-intrusive methods which reason and act in the communication network or at the interfaces of the IoT devices. In particular, we want to focus on the possibilities to fulfill following requirements regarding resilience:

- **R1: Detection of faulty, attacked or failed components during runtime in an IoT.** Faulty or already failed components shall be detected to be able to maintain or recover to a healthy system state providing correct system services.
- **R2: Maintain reliability and integrity of information in an IoT.**

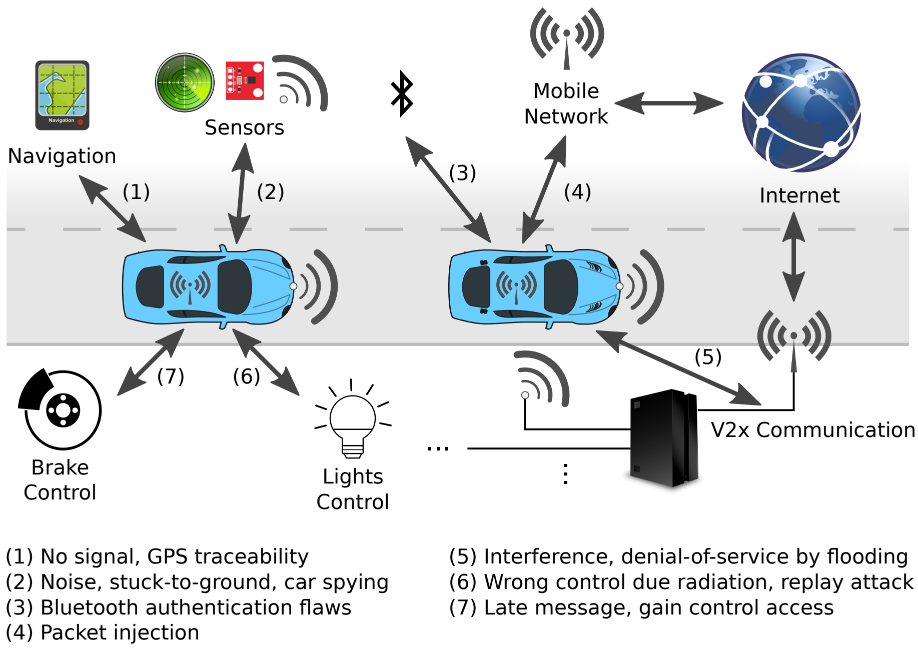


Figure 3: Exemplary faults and threats in a connected vehicle.

> Ensure the *reliability (continuity of correct service)* and *integrity (absence of improper system alterations)* [17] of the communicated information in a dynamic and heterogeneous system.

We further demonstrate our solutions to ensure resilience for the information collected and employed by the IoT.

The rest of the paper is organized as follows. Section 2 gives an overview of related roadmaps and surveys used for the SotA summary. Section 3 introduces resilience, fault types and fault examples. Section 4 collects SotA methods and techniques for failure detection and recovery. Section 5 discusses challenges to resilience in IoT. Section 6 presents our solutions to this topic. Section 7 finally concludes the paper with a discussion of our solution and future work.

## 2 Related Work

Recent surveys on the IoT review definitions, state IoT and research challenges or discuss technologies to enable interoperability and management of the IoT [18–21]. Wollschlaeger *et al.* [20] address IoT SotA and challenges focusing on the communication network and its management. They argue that the IoT is build using consumer electronics using the current Ethernet standard whereas industrial IoT (IIoT) needs real-time, reliable and efficient communication which is still under development (Ethernet TSN, 5G networks). Ray [21] surveys IoT enabling technologies and states the desired functionality of an IoT including self-adaptation and intelligent decision making. However, the survey only goes into details of available architectures with a focus on applications and security. Sfar *et al.* [22] recently published a roadmap to security in IoT. It surveys technologies to ensure privacy, trust and authentication.

Resilience has been denoted and discussed as a challenge in IoT [10,18,20,23], however, has mostly been studied in general or in other areas of computer science [16, 17, 24–30].

Notably, Avizienis *et al.* [17] published an early and very detailed classification of fault-tolerant techniques and defines the attributes of resilience. The terminology used within this paper is adopted from Avizienis *et al.* [17], Laprie [16] and Ceccarelli *et al.* [4].

Section 4.1 ("reason") mainly summarizes the techniques given in surveys on anomaly detection [31, 32]. However, the majority of surveys to "anomaly detection" tackle intrusion detection [33, 34] (e.g., based on machine learning / data mining [32] or computational intelligence [35]). We therefore add other methods to reason about failures: runtime verification [36, 37] and fault localization [38–41]. Chandola *et al.* [31] provides an essential survey of anomaly detection which studies the types of anomalies, the used methods from the applications perspective and elaborate on the different techniques (models, assumptions, complexity, advantages, limitations). Isermann [42] surveys fault detection and fault-tolerance methods from control theory (e.g., parameter estimation of process and signal models). Surveys on intrusion detection [33, 34] contain beside anomaly detection also signature-based or misuse detection (that is the detection of bad behavior encoded in signatures) and specification-based detection (cf. runtime monitoring). The self-healing surveys [25,43] also include an overview and techniques to fault detection and diagnosis.

The software engineering community provides two main roadmaps on self-adaptation [24,27]. These discuss the different aspects of self-adaptation and its research challenges, e.g., requirements engineering, design, models or life-cycles. Section 4.2 ("act") collects SotA from [17,25,28,29,43,44]. The authors in [25,43] provide a thorough survey on self-healing and classifications of the techniques. Papp and Exarchakos [28] focus on the design and testing for reconfigurable networked embedded systems, however, include an overview of methods and types of runtime network reconfiguration. Ghosh *et al.* [43] provide a broad overview about fault-tolerance, self-healing and health maintenance (fault-prevention). The authors additionally include health maintenance beside detection and recovery. We focus on fault-tolerance or self-healing approaches applicable for resilient CPS in IoT. The authors in [44] describe simple fault-tolerance methods (e.g., check-pointing, process migration or restart/replication) for the grid (but can be applied for some CPS in general). Weyns [29] guides the reader through the evolution of self-adaptation. He gives an overview to architectures, runtime models and basic approaches of self-adaptive systems, including adaptation considering goals, requirements and uncertainties.

In this paper we focus on runtime fault detection and recovery, however, fault prevention and health maintenance is another branch of self-healing [43].

Similar detection and recovery methods play a significant role to keep the system healthy, e.g., preemptive anomaly detection [45], redundancy, diversity and fault containment [26, 43].

# 3 Background

CPS are susceptible to failures due to the non-determinism of a complex environment, uncertain and/or incomplete observations, unreliable hardware, design errors or inappropriate usage. Additionally, the components of an IoT are vulnerable by distant attackers due to the devices' connection to the Internet.

## 3.1 Failures and Threats

A *failure* is an event that occurs when a system deviates from its intended behavior. The failure manifests due to an unintended state - the error - of one or more components of the system. The cause of an error is called the fault [17].

A failure manifests in a wrong content or timing (early, late or no message at all) of the intended service. Components may contain an error detection mechanism and additionally suppress wrong outputs. Such components are called *fail-silent*. Some components may automatically stop their execution on failures or halt crash, so-called *fail-stop* components. However, an erroneous component may provide wrong outputs, i.e., the service is erratic (e.g., babbling) which can cause other services to fail. In the worst case the behavior/output of the failed component is inconsistent to different observers (Byzantine failure) [17, 26].

Faults can be distinguished regarding different aspects [17, 26]. The source of a fault may be internal or external. Internal faults may be of physical nature (e.g., broken component connector) or introduced by the design (software/hardware bug). External faults originate from the environment (e.g., noise, radiation) or inputs (e.g., wrong usage of the system, intrusion). Faults can be classified into transient and permanent faults. Although a transient fault manifests only for a short period of time, it can cause an error and might lead to a permanent failure. Physical faults (internal/environmental) and inputs may be transient or permanent. Design faults are always permanent.

Security has been a topic since the beginning of computer networks identifying vulnerabilities (that is an internal fault enabling an attacker to alter the system [17]) and avoiding or mitigating malicious attacks in devices. However, in CPS additional vulnerabilities arise given the connection to the physical domain and the uncertain behavior of the physical environment [46]. A possible attack scenario (that is a malicious external fault) is often referred to as *threat*.

Table 1 summarizes the main types of faults and gives examples of possible faults and threats.

Table 1: **Main classifications of faults with examples.**

| Fault Type | Examples |
|---|---|
| Physical | Broken connector, radiation, noise, interference, power transients, power-down or short generated by an attacker, material theft (e.g., copper), denial-of-service by jamming / signal interference |
| Development | Hardware production defect, hardware design error ("errata"), software bug in program or data (memory leaks, accumulation of round-off errors, wrong set of parameters), unforeseen circumstances (of the system and/or its environment), vulnerabilities |
| Interaction | Input mistake, message collision, spoofing (obscure identity), modify information with a Trojan horse, no or late message delivery (e.g., by replay attack), denial-of-service by flooding (e.g., bomb of connection requests), hacked sensor producing inaccurate or false data causing incorrect control decisions and actuator actions |
| Permanent | Design faults, broken connector, noise, stuck-at ground voltage due to a short, logic bomb carried by a virus slowing down or crashing the system |
| Transient | Radiation, power transients, input mistake, intrusion attempt (via vulnerabilities, e.g., heating the RAM to trigger memory errors) |

## 3.2 From Robustness to Resilience

Given the vast possibilities of faults (Table 1) we desire the IoT for CPS to be dependable and secure throughout its life-cycle. Avizienis [17] defines the system's property *dependability* to be the combination of following attributes: *availability (readiness for correct service), reliability (continuity of correct service), safety (absence of catastrophic consequences), integrity (absence of improper system alterations), maintainability (ability to undergo modifications and repairs)*. Security includes availability, integrity and *confidentiality (the absence of unauthorized disclosure of information)*.

Robustness can be considered as another attribute of dependability. It has its roots in the control theory or CPS where a system is called robust if it continues to function properly under faults of stochastic nature (e.g., noise). In recent work on the concepts of cyber-physical systems-of-systems (CPSoS) [4] robustness is extended to consider also security issues in CPS as well: *Robustness is the dependability with respect to external faults (including malicious external actions)*.

A *fault-tolerant* system recovers from faults to ensure the ongoing service [17], i.e., achieving dependability and robustness of a system.

The term *resilience* is often used by the security community to describe the resistance to attacks (malicious faults). Laprie [16] defines resilience for an ubiquitous, large-scale, evolving system: Resilience is *"The persistence of service delivery that can justifiably be trusted, when facing changes."*. He builds upon the definition of dependability by giving the following short definition of resilience *"The persistence of dependability when facing changes."*.

An ubiquitous, heterogeneous, complex system-of-systems will typically evolve over time raising the need for the dependability and security established during design time to scale up. We therefore find the definition of resilience from Laprie [16] the best fit to express the needs of an IoT for CPS. A resilient IoT ensures the functionality when facing also unexpected failures. Moreover, it scales dependability and security when it comes to functional, environmental and technological changes [47].

### 3.3 Ensure Resilience - a Process Perspective

We believe that the IoT needs self-adaptation techniques to achieve resilience in an evolving system.

Many adaptation techniques consider a (control) loop of monitor, analyze, plan and execute using a knowledge base (MAPE-K) [48]. The software engineering community [24] uses similar phases: *collect* information about the environment and derive internal system properties, *analyze* the observations, *decide* how to adapt to reach a desired state and finally *act*. Psaier and Dustdar [25] define following three phases for self-healing: the *detection* of an alternation in the normal behavior, the *diagnosis* analyzes the root cause and plans a recovery strategy and finally the *recovery*. Kounev *et al.* [30] summarize self-awareness as *learn*, *reason* and *act*. We agree that learning is an essential part specifically in an evolving systems (e.g., IoT) and therefore use the phases *reason* and *act* to distinguish the building blocks enabling a resilient IoT.

## 4 Methods

There are various on- and offline approaches, methods or algorithms to achieve resilience in a system. Developers may try to prevent faults (e.g., by an appropriate design, encryption or consensus), tolerate faults (e.g., by switching to a redundant component or another pre-defined configuration), remove/mitigate faults (e.g., isolate faulty components to avoid the propagation of faults) or forecast faults (e.g., to estimate the severity or consequences of a fault) [17]. We focus on techniques typically applied during runtime in an automatic way.

Apart from traditional fault-tolerance like backup hardware/software components or checkpointing and restarting, *self-healing* is a promising approach which is related to self-adaptation and self-awareness. Self-aware systems learn the models of the system itself and its environment to reason and act (e.g., self-healing) in accordance to higher-level goals (e.g., availability) [30]. The key feature of self-* techniques is learning which is also performed during runtime to evolve the models (for better performance or to cope with system changes).

The following two sections give an overview about fault-tolerance methods split into reason (cf. detect, diagnose) and act (cf. recover) upon failures. They summarize background and terminology, highly-cited surveys (≥100 citations according to Google Scholar), recent surveys (≥2015), recent approaches not part of surveys / additional work, and examples (see distribution in Table 3) given the keywords in Table 2. Note that we tried to cite original publications and no derivations of basic fault-tolerant techniques.

Table 2: **Keywords used to find research to reason and act upon faults.**

| Reason | Act |
|---|---|
| anomaly detection, fault detection, security in CPS, intrusion detection, runtime monitoring, runtime verification, self-awareness | self-healing, self-adaptation, software adaptation, runtime reconfiguration, fault-tolerance, fault recovery, dependability, resilience |

Table 3: **Collection and distribution of references per publication type (ordered by publication year, ascending).**

| Type | Reason | Act |
|---|---|---|
| Background / Terminology | [17] [42] [31] [36] [37] | [17] [16] [24] [26] [4] [30] [29] |
| Highly-cited surveys | [42] [31] [34] [33] | [48] [43] [24] [25] |
| Recent surveys | [37] [32] [46] | [28] [49] [50] [51] |
| Additional | [52] [36] [53] [37] [54] [37] | [55] [56] [57] [58] [59] |
| Exemplary | [60] [61] [62] [63] [64] [65] [66] [67] [68] [69] [70] [71] [72] [73] [74] [75] [76] [77] [78] [79] [80] [81] [82] [83] [84] [85] [86] [87] [88] [89] | [90] [91] [92] [93] [94] [95] [96] [97] [98] [99] [100] [101] [102] [103] [104] [105] [106] [107] [108] [109] [110] [111] [112] |

## 4.1 Reason

Anomaly detection is the process to identify an abnormal behavior or pattern. The abnormal behavior or service failure (e.g., wrong state, wrong message content) is caused by a fault [17], e.g., a random failure, a design error or an intruder. Though this definition probably complies with all fault detection mechanisms listed in this section, the various communities use different keywords depending on the application or type of the mechanism. The related term *monitoring* is used in the field of runtime verification to refer to the *act of observing and evaluating temporal behaviors* [37]. In the security domain the phrase *intrusion detection* is used for reasoning about threats.

The detection method can be roughly separated w.r.t. the fault behavior to tackle and knowledge used to compare to the actual behavior (Fig. 4). Halting

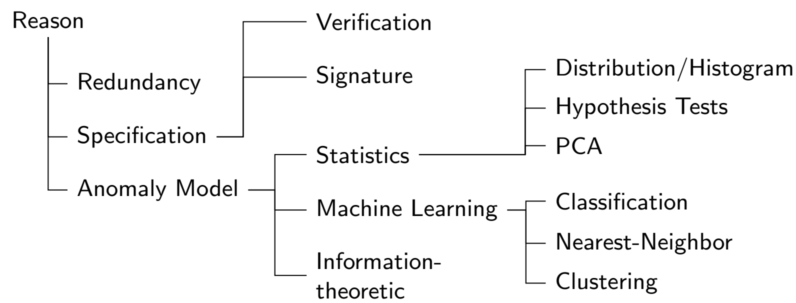


Figure 4: A taxonomy of methods for fault detection.

failures (fail-stop or fail-silent behavior) can be detected by simple methods like watchdogs or timeouts. Faults that manifest in erratic or inconsistent values need a behavior specification, model or replica to compare against (we therefore focus on these methods).

The expected or faulty behavior is represented either via formal models or specifications (runtime verification) [36,37], signatures describing attack behaviors [33,34], learned models (classification, statistics) [31,33,34,42], clusters or the data instances itself (nearest-neighbor) [31,34].

Another field of reasoning about failures is the root cause analysis or *fault localization* which identifies the reason why a fault occurs (e.g., a vulnerability of the system or the first failed component which caused other components to fail due to fault propagation).

### 4.1.1 Redundancy

Additional information sources can detect many types of attacks [85]. A simple method to verify a message's content or intermediate result is plausibility checking or majority voting [26], e.g., by comparing a received message's content against redundant information sources (see also "agreement" in Sec. 4.2). Nevertheless, redundancy is typically the last resort to increase the resilience or to ensure a specific level of dependability because it is costly when it is added explicitly (e.g., triple modular redundancy often deployed in the avionics [26]).

In hardware, fault detection by redundancy is also known as lockstep execution where typically two computational units run the same operations in parallel to detect faults [63]. When three replicas are used, the fault can be masked by majority voting (under the assumption that only one component can fail at the same time), see also TMR in Section 4.2.

However, some techniques exploit implicit or functional redundancy that is already available in the system. For instance, [53] combines anomaly detection with sensor fusion. Their approach uses a particle filter fusing data of different sensors and simultaneously calculating a value of trust of the information sources derived from the normalization factor, i.e., the sum of weights of the particles. When the weights of the particles are high, the information source match the prediction and are rated trustworthy. The authors in [87] propose to use hardwired local data of an automotive ECU to check the plausibility of a received control input. Our method presented in Section 6 is based and relies upon implicit (and explicit) redundancy too.

### 4.1.2 Specification

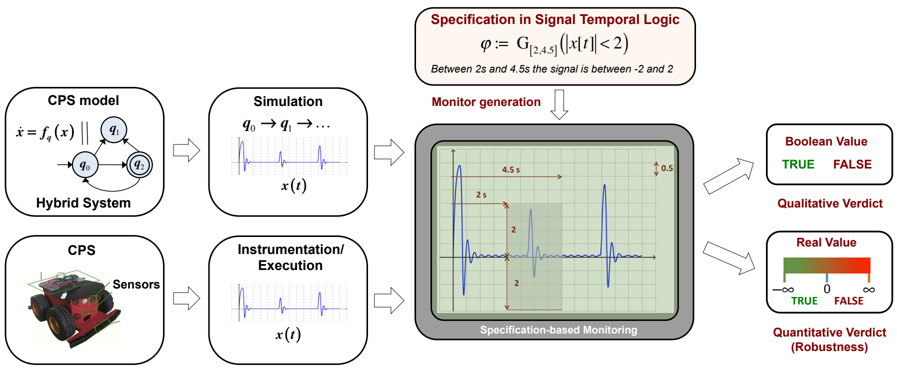


Figure 5: Specification-based monitoring can be employed either during the CPS execution or at design-time during the CPS model simulation.

**Verification of Safety Properties** The IoT generally consists of spatially distributed and networked CPS. At design time, the CPS behavior can be modeled using hybrid systems, a mathematical framework that combines discrete transition systems capturing the computational behavior of the software component with continuous (often stochastic and nonlinear) ordinary differential equations (ODEs) describing the behavior of the physical substratum with which the software component is deeply intertwined.

Although there has been a great effort in literature to provide efficient computational techniques and tools [65, 67, 72, 75, 76, 78–80] to analyze safety properties in hybrid systems, the exhaustive verification (i.e., model checking) is in general undecidable [52]. The approaches currently available to check safety properties are based on generating conservative over-approximations of the state variables dynamics called *flow pipes* [89] and on checking whether those intersect the unsafe regions of interest. However, these methods are generally limited to small scale CPS models. This limitation becomes more evident when we want to study more complex emergent behaviors, which result from the interactions among system components and that can be observed only by taking in consideration large scale CPS.

Hybrid systems are approximation models of the real CPS behavior and so their analysis may be not always faithful due to inevitable approximations errors (especially of the physical behavior) in the modeling phase. Furthermore, CPS models are not always available for intellectual property issues and indeed CPS need to be studied as black box systems where we are not able to observe the internal behavior.

**Runtime Verification** A complementary approach to exhaustive verification is to equip CPS with monitors that verify the correctness of their execution. Monitoring consists in observing the evolution of the discrete and continuous variables characterizing the CPS behavior and deciding whether the observed trace of values is good or bad. As Fig 5 illustrates, these traces can be obtained by simulating the CPS design or can be observed during the CPS execution through the instrumentation of the system under test (SUT) (more details concerning instrumentation techniques can be found in [113]).

Runtime verification (RV) [37] is a *specification-based* monitoring technique that decides whether an observed trace of a SUT conforms to rigorous requirements written in a formal specification language. The main idea of RV consists in providing efficient techniques and tools that enable the automatically generation of software- or hardware-based monitor [82, 88] from a requirement. RV can provide useful information about the behavior of the monitored system, at the price of a limited execution coverage.

RV is nowadays a very well-established technique widely employed in both academia and industry both before system deployment, for testing, verification, and after deployment to ensure reliability, safety, robustness and security.

A typical example of formal specification language is the Linear Temporal Logic (LTL) introduced by Pnueli in [61]. LTL provides a very concise and elegant logic-based language to specify sequences of Boolean propositions and their relations at different points in time. LTL considers only the temporal order of the events and not the actual point in time at which they really occur. For example, it is not possible to specify that a property should hold after one unit of time and before three and a half units of time.

Real-time temporal logics [62] overcome these limits by embedding a continuous time interval in the *until* temporal operator. Signal Temporal Logic [66,74] is a popular example of a real-time temporal logic suitable to reason about real-time requirements for CPS which has been proposed for detection of threats [77].

**Falsification-based analysis and Parameter synthesis** As illustrated in Fig.5, the Boolean semantics of STL decides whether a signal is correct or not with respect to a given specification. However, this answer is not always informative enough to reason about the CPS behavior, since the continuous dynamics of these systems are expected to be tolerant with respect to the value of certain parameters, the initial conditions and the external inputs.

Several researchers have proposed [69, 70] to address this issue by defining a *quantitative* semantics for STL. This semantics replaces the binary satisfaction relation with a quantitative *robustness degree* function that returns a real value (see Fig.5) indicating how far is a signal from satisfying or violating a specification. The positive and negative sign of the robustness value indicates whether the formula is respectively satisfied or violated.

The notion of STL robustness was exploited in several tools [71, 73] for falsification analysis [81] and parameter synthesis [68, 86] of CPS models. On one hand, trying to minimize the robustness [73] is suitable to search counterexamples in the input space that violates (falsifies) the specification. On the other hand, maximizing the robustness [71] can be used to tune the parameters of the system to improve its resilience. To this end, a global optimization engine is employed to systematically guide the search.

**Signature-based Intrusion Detection** Signature-based intrusion detection compares pre-defined behavior (known as golden behavior or signature) to identify the the abnormal event during runtime [33]. Though these techniques effectively identify the intrusion with a small number of false positives they require a precisely calibrated signature [54]. Therefore, such techniques are not feasible if designers and IP providers are not trusted. Such misuse-based intrusion detection typically cannot handle zero-day attacks that are new unknown attacks. It is therefore often combined with anomaly detection (e.g., in [83]).

### 4.1.3 Anomaly-based

**Statistics** In statistical anomaly detection the data is fit into a statistical model. If a test instance occurs in the low probability region of the model, i.e., it is unlikely to be generated by the model, then it is claimed to be an anomaly. Statistical models can be specified with parameters when the underlying distribution is known (e.g., is Gaussian). The parameters are trained by machine learning algorithms [31] or estimation [42] describing the correct behavior of the system. The inverse of the test instance's probability to be generated can directly be used as anomaly score. Statistical tests can also be used to label or score a test instance (e.g., box plot rule).

The model can be expressed by the data itself, e.g., in a histogram, by kernel functions or particles, which is typically used when the distribution of the data is unknown. The test instances or samples may be evaluated by statistical hypothesis tests. For instance, the Wilcoxon signed-rank test [60] compares two related samples to determine if they have the same underlying distribution (which is unknown and does not have to be the normal distribution).

The principal component analysis (PCA) is used to project the data to lower dimensions, i.e., it reduces the dimensionality of the data to a set of uncorrelated variables. A test instance can be marked anomalous when the projection on the components result in a high variance meaning that the test instance does not fit the typical correlation of the data.

However, simple tests, Gaussian models and histograms are nowadays mostly replaced by (deep) neural networks which stand out handling multivariate and non-linear data.

**Machine Learning or Data Mining** Typical anomaly detection techniques based on machine learning can be used with data where no domain knowledge is available (e.g., black-box components like IP cores). The models may be updated during operation. When the desired behavior is known it can be expressed as formal model (specification-based monitoring).

Classification-based anomaly detection learns a model (SVM, neural network, Bayesian networks, rules or decision trees) given labeled training data (e.g., states and observations of the system) to cluster the test data into normal classes and anomalies or outliers [31]. Instead of labeling a test instance to a class, one may use scores representing the likelihood of a test instance being an anomaly. For instance, the authors in [84] use recurrent neural networks to detect anomalies in real-time data. The network models short and long term patterns of time series and serves as a prediction model of the data. The error between predicted and actual value serves as an anomaly score.

Nearest-neighbor-based detection techniques measure the distance from a data instance under test to $k$ neighbors to identify anomalies. Different metrics (e.g., euclidean distance) are applied to specify an anomaly score - that is the likelihood of a data instance to be an anomaly. Another approach is to measure the density that is the number of instances in the area specified by the data instance under test given a radius. The Nearest-Neighbor's complexity increases with the power of two of the number of data instances. Unsupervised.

Data instances are first distributed into clusters (by clustering algorithms, e.g., expectation maximization, k-means, self-organizing maps, many of which use distance or density measures). An anomaly is a data instance that does not fit into any cluster.

**Information-theoretic** By investigating the information content described by, e.g., the entropy of the information, one may draw conclusion about anomalies in the data (for information-theoretic measures characterizing regularity in data see [64]). When the entropy exceeds a threshold the test instance is marked as anomaly. The threshold is defined by the set of anomalies. In highly irregular data the gap between threshold and maximum entropy may be low (the set of true anomalies is small).

#### 4.1.4 Fault-Localization

When the fault detection only gives us the information about a failure happened in a subsystem, we need means to identify the exclusive part causing the failure.

This is often performed by root cause analysis [114] or fault-localization [38–41, 115–117]. In the software engineering community there is a considerable amount of literature about (semi-)automatic techniques assisting the developer to localize and to explain program bugs (for a comprehensive survey we refer the work in [41]). A well-established statistical approach, is the *spectrum-based fault-localization* (SFL) [39], a technique that provides a ranking of the program components that are most likely responsible for the observed fault.

This approach has been employed recently also to localize faults in Simulink/Stateflow CPS models [38, 40, 115–117], displaying a similar accuracy with the same method applied to software systems [116]. Although the classical SFL is agnostic to the nature of the oracle and only requires to know whether the system passes or not a specific test case, in [38], the authors have introduced a novel approach where the oracle is a specification-based monitor. This enables to leverage the trace diagnostic method proposed in [118] and to obtain more information (for example the segment of time where the fault occurred) about the failed tests improving the fault-localization.

Often this approach is only applied offline for debugging processes, however, it can be used to isolate a failed HW/SW component from the system to avoid fault propagation or trigger its recovery.

## 4.2 Act

Broadly speaking, a system can be adapted by changing the parameters or the structure (architecture) of the system [24,30]. Following four action types of possible re-configurations are defined by [28] (splitting structural adaptation into further classes): *re-parameterization* to change the parameters of a component, *re-instantiation* to create and remove components, *rewiring* to redirect connections between components or *relocation* to migrate functionality to another platform. The latter three action types require redundancy to some extent. We extend and refine these types below (Fig. 6).

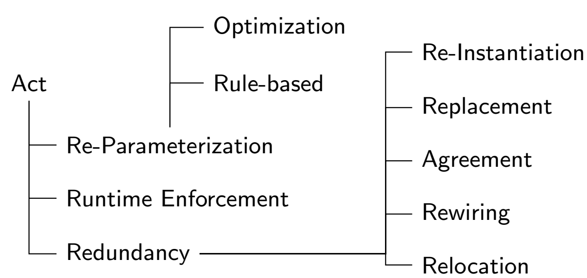


Figure 6: A taxonomy of methods for recovery.

The adaptation can be applied on different architectural levels of the system. For instance, the change of the clock speed or other hardware parameters is the re-parameterization on the physical level of a device. Changing the sender or receiver of a task is rewiring on the task level.

### 4.2.1 Re-Parameterization

In general, a re-parameterization (or reconfiguration) switches to another configuration of one or more components that is typically no longer the optimal setting, i.e., the quality of service is decreased (graceful degradation). Adaptation of parameters requires knowledge about the underlying algorithm of the erroneous component and is therefore typically performed by the component itself or within a subsystem. The configuration can be selected by optimization [94], or a reasoner based on a set of rules, an ontology or a logic program [28]. Approaches from the control theory use state observers or estimators to derive parameters to mitigate stochastic faults [42]. For instance, an adaptive Kalman filter (AKF) [90] changes its filter parameters during runtime based on the inputs. For instance, the measurement covariance can be increased when an input signal gets worse or even permanently fails (cf.: a traditional KF or state estimator mitigates noise including transient failures only).

### 4.2.2 Runtime Enforcement

Runtime enforcement [56,59] merges runtime verification with adaptation. This powerful technique ensures that a program conforms to its specification. A so-called enforcer acts on the interface of a component changing inputs or outputs to comply with a set of formal properties. The enforcer uses an automaton and/or rules to correct the IO in case of faults. This approach has been pioneered by the work of Schneider [92] on security automata which halt the program whenever it deviates from a safety requirement. Since then, there has been a great effort in the RV community to define new enforcement mechanisms with primitives [95,98,99,105,108] or that support more expressive specifications [101, 102, 110].

### 4.2.3 Redundancy

Redundant components ensure availability (passive) and increase reliability (active). Failed components can be re-instantiated, replaced by spares, mitigated by voting or fusion, rewired or relocated [25, 28].

**Re-Instantiation or Restart** A straightforward fault-tolerance method is to restart a failed software component. The tasks or the system typically saves checkpoints or output messages of components on a periodic basis to roll back to a healthy state [103]. The restart might be combined with a re-parameterization. Checkpointing/restart techniques are well studied for operating systems [119] and may be applied on fog nodes or cloud servers. The primary/backup approach activates a typically aperiodic backup task if the primary task fails [91].

**Replacement or Cold/Hot Spares** The simplex architecture [93] considers two redundant subsystems. The high-dependable subsystem jumps in when the high-performance subsystem fails. Triple modular redundancy (TMR) replicates HW and/or SW components to mask failures (through a voter, i.e., includes detection). The replicates are in the best (but most costly) case diverse w.r.t. their design such that also design and input errors can be masked [26]. Such hardware redundancy is typically added during design time and used in closed, non-elastic systems. However, an IoT orchestrator can maintain a directory of available services and redirect resource requests if necessary.

Implicit redundancy like related observations in a system (in contrast to traditional redundancy that is the explicit replication of components) can be exploited by structural adaptation. A substitute component is instantiated to replace the failed component which includes also rewiring and possibly also a relocation [104, 111] (see Sec. 6 for an example).

**Agreement / Voting or Fusion** Byzantine failures (inconsistent failures to different observers) typically caused by malicious attacks can be detected and tolerated using replicas (here: redundant services on different nodes of a distributed system) by agreement or consensus on the outputs [55]. The output of redundant components can be combined or fused, e.g., via filters or fuzzy logic [49]. However, through recent implementations and usage in cryptocurrencies [96, 100] the attention is shifted towards smart contracts and blockchains which ensure authentication and integrity of data [50, 51, 57, 58]. Basically, a blockchain is a series of data records each attached by a cryptographically secure hash function which makes it computationally infeasible to alter the blockchain. However, blockchains suffer from complexity, energy consumption and latency and therefore currently cannot be used for real-time anomaly detection or applied by simple nodes with low computational power [58]. However, it is already examined to manage access to data (authorization), purchase devices or computing power or manage public-key infrastructure in the IoT [57, 109, 112].

**Rewiring or Redirection** Broken links in mesh networks are typically reconfigured using graph theory considering node properties and application requirements [106]. A software component may route the task flow to a recovery routine [25].

**Relocation** Migration of software components or tasks are studied in the field of resource optimization, utilization and dynamic scheduling on (virtual) machines. Optimization algorithms [94], multi-agent systems [97] or reinforcement learning [107] find a new task configuration utilizing resources in case of a platform failure. Tasks may also be migrated in advance when the health state decreases [103]. Cloud applications boost and emerge new technologies like containerization, resource-centric architectures and microservices which ease service orchestration in complex and elastic systems. Dragoni *et al.* [120] prognoses increased dependability using microservices which focus on small, independent and scalable function units (cf. fault containment units in Kopetz [26]), however, security remains a concern.

# 5 Challenges

Self-healing and IoT have their own challenges. For instance, critical issues in IoT are: interoperability, resource limitations, security and privacy or power consumption [18, 23, 121, 122]. In self-healing, development of behavior prediction models, data/log storage, validation of the recovery or latency are challenging [25, 43, 123].

However, in comparison to intrusion detection in cloud applications or reconfiguration in CPS, the techniques enabling resilience for IoT will have to cope with the following main challenges (among others):

- **C1: Resource Limitations.** The majority of IoT components are resource-constrained devices. The developer often has to trade off power, time and costs against resilience. Typical small IoT devices like commercial off-the-shelf (COTS) microcontrollers may provide insufficient capabilities.

  Some technologies might therefore need hardware implementations (e.g., RV monitor) or should be designed as a lightweight and fully distributed, layered, or clustered service (e.g., a monitor per subsystem).

- **C2: Interoperability and Complexity.** The IoT is a large dynamic network of heterogeneous components.

  For instance, COTS or components protected by intellectual property (IP) may not provide a proper specification of its behavior for some of the detection and adaptation methods. Furthermore, new devices or subsystems may introduce unknown interfaces (here: unknown to the resilience-enabling technologies).

  So to reliably monitor and heal the IoT over time, the mechanisms shall be itself self-adaptive.

  The devices of a CPS are specified during design time having a specific application in mind. Things of an IoT will most likely be shared between applications while different fog/cloud applications might request different QoS of the devices, e.g., regarding dependability.

  The methods therefore must also consider and combine the requirements of different applications and the value of trust of the information (e.g., used to derive actions).

  Due to the vast size of an IoT a central mechanism most likely will not be able to cope with all the input data necessary to achieve resilience (considering memory and time constraints).

- **C3: Real-Time and Scalability.** One major shift from sensor networks to the IoT is the control and manipulation of actuators from the distance, i.e., the IoT comprises a cyber-physical system. The CPS typically has to satisfy *time constraints* (rates, deadlines) in order to function correctly.

  In such real-time applications the probing of information by a monitor or changes in the system (e.g., connection of new things, updates, recovery) shall not influence the timing behavior of the CPS. Furthermore, the timeliness to detect and react to critical failures has to be considered.

However, the complexity and dynamicity of the network will leave the door ajar for some faults, e.g., physical faults, design errors or zero-day malware. Therefore a proper *never-give-up* strategy [26] to cope with unconsidered failures has to be developed.

# 6 Case Study: Resilient Smart Mobility

We present the current status and future directions of our solutions on an automotive use case. The IoT will most likely contain many heterogeneous components with different capabilities of resilience. We therefore want to focus on the integrity and reliability of the information communicated between the components (things, fog, cloud).

Let's consider vehicles driving on a highway (Fig. 7). Radar sensors are mounted along the street and form a collaborative sensor field. In order to improve object detection and classification, a multi-object tracking scheme is employed, which uses subsequent sensor measurements in the form of prediction and update cycles to estimate vehicle locations. The tracking data can be used for, e.g., traffic congestion forecast or accident investigations. A set of radar sensors is connected to a fog node, that is a computing unit and IoT gateway in the near area of the sensors. The tracker - a software component running on a fog node - tracks the vehicles on the road segment covered by the associated radars. Some vehicles (e.g., autonomous cars) are equipped with distance sensors like radar, lidar or depth cameras. The fog node(s) of these cars can connect to near fog nodes of the street (directly over a vehicular network called VANET, or via the mobile network over the cloud).

We assume the IoT infrastructure (things, fog, cloud, network) is given and propose methods to increase the resilience of the IoT.

Failures of the radar sensors in our example will lead to inaccurate or even unusable tracking results. Failure scenarios like communication crashes and dead batteries (fail-silent, fail-stop) are relatively easy to handle (e.g., watchdog/timeout). However, the sensor measurements received by the tracker running in the fog node may be erroneous due to noise (e.g., communication line, aging), environmental influences (e.g., dirtying of the radar) or a security breach (e.g., hacked fog node that collects data of a group of sensors). To detect a failure of the sensor one has to create particular failure models for each possible hazard (c.f., aging, dirtying and a security breach). A simple method detecting a faulty sensor value in different failure scenarios is to check against other information sources, i.e., exploit redundancy. However, explicit redundancy that is replicating observation components is costly.

Self-healing can be applied to react also to failures not specifically considered during design-time. A very promising way of achieving self-healing is through structural adaptation (SHSA), by replacing a failed component with a substitute component by exploiting implicit redundancy (or functional and temporal redundancy) [124]. We use a knowledge base [104, 111] modeling relationships among system variables given that certain implicit redundancy exists in the system and extract a substitute from that knowledge base using guided search (Sec. 6.2). The knowledge base can also be used to monitor the system by comparing the information of variables against each other, i.e., to detect failures (Sec. 6.1).

SHSA can be encapsulated in separate components listening and acting on the communication network of the IoT, e.g., as tasks *monitor*, *diagnose* and

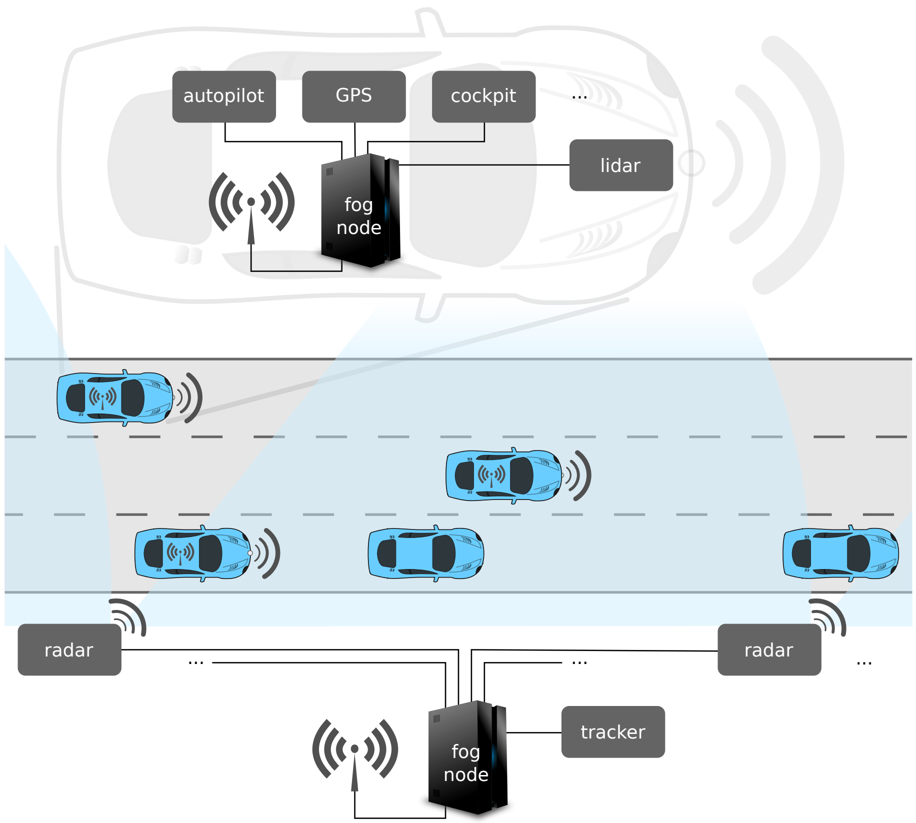


Figure 7: Visualization of the use case.

*recover* running on a fog node (Fig. 8).

SHSA monitors the information communicated between components (typically the sensor measurements or filtered/estimated observations), identifies the failed component and replaces messages of the failed component delivering an erroneous output by spawning a substitute software component. SHSA considers the currently available information in the network, i.e., can be applied in dynamic systems like the IoT (components may be added and removed during runtime). The knowledge base, in particular the relationships between the communicated information, can be defined by the application's domain expert or learned (approximated by, e.g., neural networks, SVMs or polynomial functions, see also [124]).

Alternatively, the monitor and diagnose task may be installed in the cloud analyzing the logged tracks to trigger maintenance of radar sensors. The requirements needed by SHSA regarding the architecture of the system (e.g., communication network) and a reference implementation of SHSA can be found in [124].

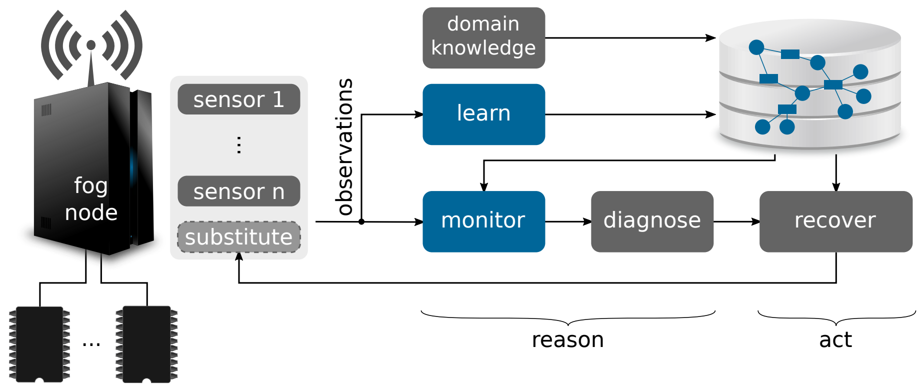


Figure 8: Overview of the self-healing components and proposed integration into a fog node.

## 6.1 Reason

In our future work we want to use the SHSA knowledge base described below to perform plausibility checks upon related information.

As our focus is the adaptation in the software cyber-part of a CPS (cf. dynamic reconfiguration of an FPGA), we assume that each physical component comprises at least one software component (e.g., the driver of the radar in the vehicle) and henceforth consider software components only. The CPS implements some functionality – a desired service (e.g., collision avoidance). The subset of components implementing the CPS' objectives are called controllers.

### 6.1.1 SHSA Knowledge Base [111]

A system can be characterized by properties referred to as *variables* (e.g., the position and velocity of a tracked vehicle). The values of system variables are communicated between components typically via message-based interfaces. Such transmitted data that is associated to a variable, we denote as information atom, short *itom* [125]. A variable can be provided by different components simultaneously (e.g., two radars with overlapping field of view). Each software component executes a program that uses input itoms and provides output itoms. An itom is *needed*, when it is input of a controller. A variable is *provided* when at least one corresponding itom can be received.

Variables are related to each other. A relation is a function or program (e.g., math, pseudo code or executable python code) to evaluate an output variable from a set of input variables.

The knowledge base is a bipartite directed graph (which may also contain cycles) with independent sets of variables and relations of a CPS. Variables and relations are the nodes of the graph. Edges specify the input/output interface of a relation. For instance, Fig. 9 models the relationships between the variables in the tracking use case (only relevant nodes, relationships and edge directions

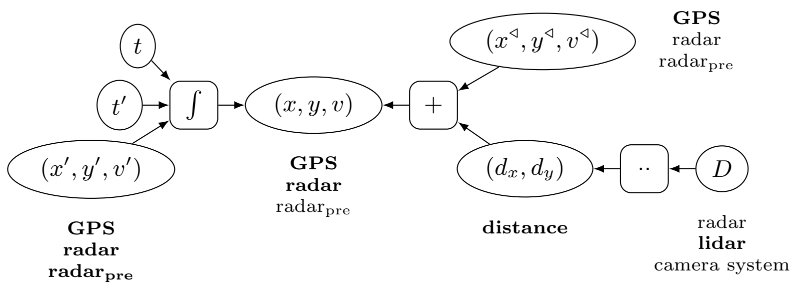


Figure 9: Knowledge base. Ellipses are variables, boxes are relationships (functions). The variables are annotated with possible itoms. Bold itoms are available in the scenario in Fig. 10.

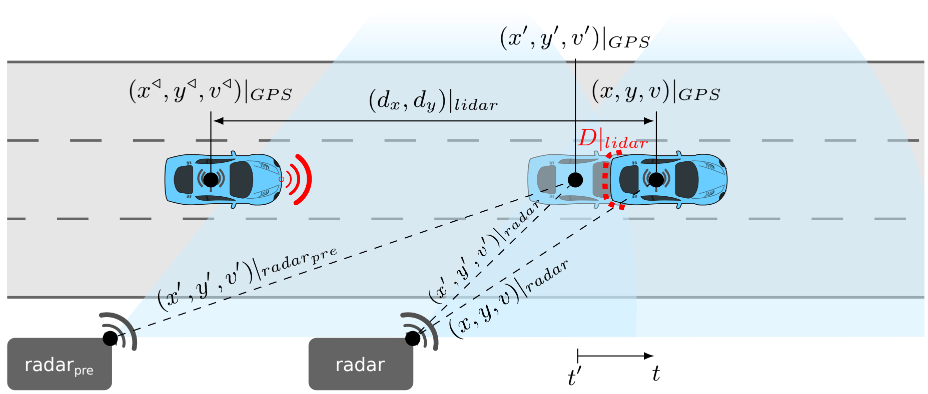


Figure 10: An exemplary scenario from the use case. Visualization of itoms $variable|_{itom}$ from the knowledge base in Fig. 9.

for the scenario in Fig. 10 are shown).

A proper data association identifies which itoms or measurements represent the same variable, e.g., links the different position itoms $(x, y, v)|_*$ to each other. For instance, the GPS position $(x, y, v)|_{GPS}$ of a vehicle (transmitted by the vehicle itself) has to be linked to the corresponding radar track $(x, y, v)|_{radar}$ (provided by the radar).

Subsequently, the redundant itoms can be used, e.g., to monitor a radar sensor, to substitute a failed radar or to increase the accuracy of a tracking application by sensor fusion.

### 6.1.2 Fault Detection by Redundancy (work-in-progress)

An itom has failed, when the itom deviates from the specification. Our monitor uses the knowledge base to perform a plausibility check in every time step to identify a failed itom. The automatic setup of a runtime monitor follows successive procedure:

- Select the variable to be monitored (typically the corresponding variable to the itom under test), e.g., the position of a vehicle.
- Collect the provided itoms (e.g., simply subscribe to all available messages). Note, the availability of variables may change from time to time which should trigger a new setup of the monitor.
- Extract relations of the monitored variable and available variables from the knowledge base (similar to the search of valid substitutions in Sec. 6.2).

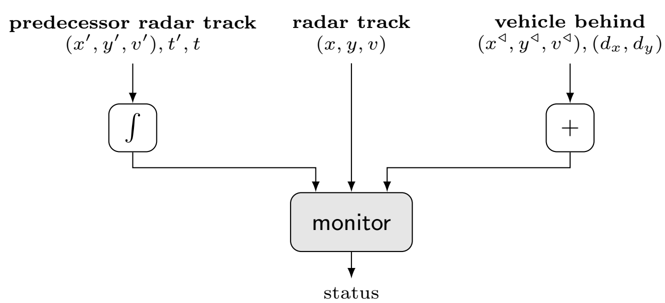


Figure 11: A monitor checking the position of a vehicle using different itoms. The itoms are first transferred into the common domain (here: position of the vehicle $(x, y, v)$) and compared against each other.

The instantiated monitor for the position of a vehicle is depicted in Fig. 11. At each time step the relations are executed to bring the available itoms (provided variables) into the common domain (variable to be monitored) where the values are compared against each other. The monitor returns the fault status or a confidence / health / trust value for each itom used in the plausibility check.

The confidence may be expressed by a distance metric or error between the itoms in the common domain. The trust or confidence of a radar may be accumulated from the individual confidence values of the tracked vehicles, i.e., the vehicles in the field of view of the radar. As soon as the confidence falls below a specific threshold for a specific amount of time the status of the respective itom is classified as failed.

The monitor can identify failed itoms in the common domain, however, when the output of a relation mismatches in the common domain, all inputs of the relation are marked faulty. To avoid additional monitors (a monitor for each input variable is necessary to identify the failed itom) a fault localization can be performed.

### 6.1.3 Fault Localization [38]

The engineers often design CPS using the MathWorks$^{tm}$ Simulink toolset to model their functionalities. These models are generally complex hybrid systems that are often impossible to analyze only by using the reachability analysis techniques described before. A popular technique to find bugs in Simulink/Stateflow models is falsification-based testing [71,73,126]. This approach consists in monitoring an STL property over traces produced by systematically simulating the CPS design using different set of test cases. For each generated trace the monitor returns a real-value that provides an indication as how far the trace is from violation. This information can be used to guide the test case generation to find an input sequence that would falsify the specification. However, this approach does not provide any information concerning which is the failed component and the precise moment in time that is responsible for the observed violation. To overcome this shortcoming, in [38] Bartocci *et al.* have recently introduced a new procedure that aids designers in debugging Simulink/Stateflow hybrid system models, guided by STL specifications. This approach combines a trace diagnostics [118] technique that localizes time segments and interface variables contributing to the property violations, a slicing method [127] that maps these time segments to the internal states and transitions of the model and a spectrum-based fault-localization method [39] that produces a ranking of the internal states and/or transitions that are most likely to explain the fault.

## 6.2 Act

A failed itom can be replaced by a function of related itoms. To this end, the knowledge base is searched for relationships using provided variables and spawns a substitute.

### 6.2.1 Replacement [111]

The substitute search algorithm traverses the knowledge base (Fig. 9) from the failed but needed information as root to find a valid substitution.

A substitution of a variable is a connected acyclic sub-graph of the knowledge base with following properties: i) The output variable is the only sink of the substitution. ii) Each variable has zero or one relationship as predecessor. iii) All input variables of a relation must be included (it follows that the sources of the substitution graph are variables only).

A substitution is valid if all sources are provided, otherwise the substitution is invalid (Fig. 12). Only a valid substitution can be instantiated (to a substitute) by concatenating the relationships which take the selected itoms as input (e.g., best itoms of the source variables).

Substitutions can be found by depth-first search of the knowledge base with the failed variable as root. The search may stop as soon as all unprovided variables are substituted [104]. In [111] we present a guided search approach using a performance measure for substitutions.

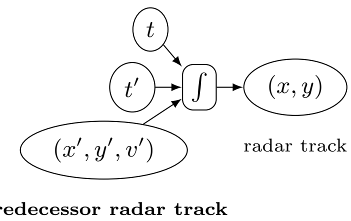


Figure 12: A valid substitution for the failed street radar. Old data from the predecessor radar is used to forward estimate the position of the vehicles.

The result of the search - the substitution - is instantiated in a substitute [124]. In particular, the substitute subscribes to the input itoms and concatenates the functions or programs from the relationships. The substitute then periodically publishes the output. To avoid inconsistencies and fault propagation, the failed component (probably publishing erratic messages) should be shut down as soon as possible.

# 7 Conclusion

This paper summarizes the state-of-the-art of anomaly detection / runtime monitoring and adaptation to react to failures in IoT for CPS, presents the main challenges these methods have to cope with and proposes a solution on an automotive example. The SHSA knowledge base presented in Section 6 describes implicit and explicit redundancy in a communication network. It can therefore be exploited to monitor, replace or fuse information.

Because our approach is based on redundancy it can handle various fault scenarios. Especially permanent faults in the IoT can be detected and recovered given some redundancy exists. As long as the failed components can be isolated and replaced by redundant information the methods can handle physical, development or interaction faults manifested as failures at the components' interfaces.

The monitor tackles the requirement on fault detection (R1) by voting over redundant information. An additional fault localization identifies and triggers a disconnection of the failed component to avoid fault propagation. The substitution replaces failed information with redundant one. Thus, fault localization and substitution recovers from the faulty behavior achieving reliability and integrity of the communicated information (R2).

The individual IoT devices might not have the resources to implement self-healing (C1) nor a common understanding of the information or access to relevant redundancy (C2). Under some constraints (bounded or static SHSA knowledge base, estimation of the worst-case execution time of relationships) our approach is suitable for real-time applications [104]. Furthermore, solutions to increase scalability have to be investigated (C3). In future work we therefore want to focus on a distributed approach of the mechanism (e.g., by splitting the knowledge base for subsystems, or monitor in a distributed fashion like agreement protocols do).

## Acknowledgement

The research leading to these results has received funding from the IoT4CPS project partially funded by the “ICT of the Future” Program of the FFG and the BMVIT. The authors acknowledge the TU Wien University Library for financial support through its Open Access Funding Programme.

## References

[1] Edward A. Lee and Sanjit A. Seshia. An introductory textbook on cyber-physical systems. In *Proceedings of the 2010 Workshop on Embedded Systems Education*, WESE '10, pages 1:1–1:6, New York, NY, USA, 2010. ACM.

[2] Ragunathan (Raj) Rajkumar, Insup Lee, Lui Sha, and John Stankovic. Cyber-physical systems: The next computing revolution. In *Proc. of DAC '10: the 47th Design Automation Conference*, pages 731–736, New York, NY, USA, 2010. ACM.

[3] Ragunathan Rajkumar. A cyber-physical future. *Proceedings of the IEEE*, 100(Special Centennial Issue):1309–1312, 2012.

[4] Andrea Ceccarelli, Andrea Bondavalli, Bernhard Froemel, Oliver Hoeftberger, and Hermann Kopetz. *Basic Concepts on Systems of Systems*, pages 1–39. Springer International Publishing, Cham, 2016.

[5] Daniel J. Fagnant and Kara Kockelman. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77:167–181, 2015.

[6] Insup Lee, Oleg Sokolsky, Sanjian Chen, John Hatcliff, Eunkyoung Jee, BaekGyu Kim, Andrew King, Margaret Mullen-Fortino, Soojin Park, Alexander Roederer, and Krishna K. Venkatasubramanian. Challenges and research directions in medical cyber-physical systems. *Proceedings of the IEEE*, 100(1):75–90, 2012.

[7] Jad Mouawad. F.A.A. Orders Fix for Possible Power Loss in Boeing 787. New York Times, May 1, 2015.

[8] Peter Holley. Chrysler Fiat announces recall of nearly 5 million U.S. cars. The Guardian, May 25, 2018.

[9] Edward Helmore. Uber shuts down self-driving operation in Arizona after fatal crash. The Guardian, May 23, 2018.

[10] Robin Bloomfield, Kateryna Netkachova, and Robert Stroud. Security-informed safety: If it's not secure, it's not safe. In Anatoliy Gorbenko, Alexander Romanovsky, and Vyacheslav Kharchenko, editors, *Software Engineering for Resilient Systems*, pages 17–32, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[11] Alex Hern. Hacking risk leads to recall of 500,000 pacemakers due to patient death fears . The Guardian, August 31, 2017.

[12] Rory Cellan-Jones. F.A.A. Orders Fix for Possible Power Loss in Boeing 787. BBC, February 26, 2016.

[13] Andy Greenberg. Hackers Remotely Kill a Jeep on the Highway – With Me in It. Wired, July 1, 2015.

[14] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and other Botnets. *Computer*, 50(7):80–84, 2017.

[15] IHS. Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions). In *Statista - The Statistics Portal.* Accessed September 10, 2018. Available from `https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/`.

[16] Jean-Claude Laprie. From Dependability to Resilience. In *Dependable Systems and Networks (DSN 2008), 38th Annual IEEE/IFIP International Conference*, 2008.

[17] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. on Dependable and Secure Computing*, 1:11–33, 2004.

[18] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[19] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014.

[20] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE Industrial Electronics Magazine*, 11(1):17–27, 2017.

[21] Partha Pratim Ray. A survey on internet of things architectures. *Journal of King Saud University-Computer and Information Sciences*, 30(3):291–319, 2018.

[22] Arbia Riahi Sfar, Enrico Natalizio, Yacine Challal, and Zied Chtourou. A roadmap for security challenges in the internet of things. *Digital Communications and Networks*, 4(2):118–137, 2018.

[23] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Soler Jubert, Margaretha Mazura, Mark Harrison, Markus Eisenhauer, et al. Internet of things strategic research roadmap. *Internet of Things-Global Technological and Societal Trends*, 1(2011):9–52, 2011.

[24] B. H. C. Cheng *et al.* Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems*, pages 1–26. Springer Verlag, Berlin, Heidelberg, 2009.

[25] Harald Psaier and Schahram Dustdar. A survey on self-healing systems: approaches and systems. *Computing*, 91(1):43–73, Jan 2011.

[26] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Springer, New York, 2nd edition, 2011.

[27] R. de Lemos *et al.* *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*, pages 1–32. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[28] Zoltan Papp and George Exarchakos, editors. *Runtime Reconfiguration in Networked Embedded Systems - Design and Testing Practices.* Internet of Things - Technology, Communications and Computing. Springer Science+Business Media Singapore, 2016.

[29] Danny Weyns. Software Engineering of Self-Adaptive Systems: An Organised Tour and Future Challenges. Springer, 2017.

[30] Samuel Kounev, Peter Lewis, Kirstie L. Bellman, Nelly Bencomo, Javier Camara, Ada Diaconescu, Lukas Esterle, Kurt Geihs, Holger Giese, Sebastian Götz, Paola Inverardi, Jeffrey O. Kephart, and Andrea Zisman. *The Notion of Self-aware Computing*, pages 3–16. Springer International Publishing, Cham, 2017.

[31] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly Detection: A Survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[32] A. L. Buczak and E. Guven. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys Tutorials*, 18(2):1153–1176, Secondquarter 2016.

[33] Ismail Butun, Salvatore D Morgera, and Ravi Sankar. A survey of intrusion detection systems in wireless sensor networks. *IEEE communications surveys & tutorials*, 16(1):266–282, 2014.

[34] Robert Mitchell and Ing-Ray Chen. A survey of intrusion detection techniques for cyber-physical systems. *ACM Computing Surveys (CSUR)*, 46(4):55, 2014.

[35] Shelly Xiaonan Wu and Wolfgang Banzhaf. The use of computational intelligence in intrusion detection systems: A review. *Applied Soft Computing*, 10(1):1 – 35, 2010.

[36] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293 – 303, 2009.

[37] Ezio Bartocci and Yliès Falcone, editors. *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*. Springer, 2018.

[38] Ezio Bartocci, Thomas Ferrère, Niveditha Manjunath, and Dejan Nickovic. Localizing faults in simulink/stateflow models with STL. In *Proc. of HSCC 2018: the 21st International Conference on Hybrid Systems: Computation and Control*, pages 197–206, 2018.

[39] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund. On the accuracy of spectrum-based fault localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques*, pages 89–98. IEEE, 2007.

[40] Jyotirmoy V. Deshmukh, Xiaoqing Jin, Rupak Majumdar, and Vinayak S. Prabhu. Parameter optimization in control software using statistical fault localization techniques. In *Proc. of ICCPS 2018: the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 220–231. IEEE / ACM, 2018.

[41] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Trans. Software Eng.*, 42(8):707–740, 2016.

[42] Rolf Isermann. *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer Science & Business Media, 2006.

[43] Debanjan Ghosh, Raj Sharman, H. Raghav Rao, and Shambhu Upadhyaya. Self-healing Systems - Survey and Synthesis. *Decis. Support Syst.*, 42(4):2164–2185, January 2007.

[44] S. Siva Sathya and K. Syam Babu. Survey of fault tolerant techniques for grid. *Computer Science Review*, 4(2):101 – 120, 2010.

[45] Ramin M Hasani, Guodong Wang, and Radu Grosu. An automated auto-encoder correlation-based health-monitoring and prognostic method for machine bearings. *arXiv preprint arXiv:1703.06272*, 2017.

[46] Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. Cyber-physical systems securitya survey. *IEEE Internet of Things Journal*, 4(6):1802–1831, 2017.

[47] Jean-Claude Laprie. Resilience for the scalability of dependability. In *Network Computing and Applications, fourth IEEE international symposium on*, pages 5–6. IEEE, 2005.

[48] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan 2003.

[49] Nancy E ElHady and Julien Provost. A systematic survey on sensor failure detection and fault-tolerance in ambient assisted living. *Sensors*, 18(7):1991, 2018.

[50] Ana Reyna, Cristian Martín, Jaime Chen, Enrique Soler, and Manuel Díaz. On blockchain and its integration with iot. challenges and opportunities. *Future Generation Computer Systems*, 2018.

[51] Minhaj Ahmad Khan and Khaled Salah. Iot security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82:395–411, 2018.

[52] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 278–292. IEEE Computer Society, 1996.

[53] N. Bißmeyer, S. Mauthofer, K. M. Bayarou, and F. Kargl. Assessment of node trustworthiness in VANETs using data plausibility checks with particle filters. In *2012 IEEE Vehicular Networking Conference (VNC)*, pages 78–85, Nov 2012.

[54] Davide Fauri, Daniel Ricardo dos Santos, Elisa Costante, Jerry den Hartog, Sandro Etalle, and Stefano Tonetta. From system specification to anomaly detection (and back). In *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 13–24. ACM, 2017.

[55] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[56] Yliès Falcone. You should better enforce than verify. In Howard Barringer, Yliès Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon J. Pace, Grigore Rosu, Oleg Sokolsky, and Nikolai Tillmann, editors, *Proceedings of the 1st international conference on Runtime verification (RV 2010)*, volume 6418 of *Lecture Notes in Computer Science*, pages 89–105. Springer-Verlag, 2010.

[57] Marco Conoscenti, Antonio Vetro, and Juan Carlos De Martin. Blockchain for the internet of things: A systematic literature review. In *Computer Systems and Applications (AICCSA), 2016 IEEE/ACS 13th International Conference of*, pages 1–6. IEEE, 2016.

[58] W. Meng, E. W. Tischhauser, Q. Wang, Y. Wang, and J. Han. When Intrusion Detection Meets Blockchain Technology: A Review. *IEEE Access*, 6:10179–10188, 2018.

[59] Yliès Falcone, Leonardo Mariani, Antoine Rollet, and Saikat Saha. Runtime failure prevention and reaction. In *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457, pages 103–134. Springer, 2018.

[60] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[61] Amir Pnueli. The temporal logic of programs. In *Proc. of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[62] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *Journal of ACM*, 41(1):181–204, 1994.

[63] Stefan Poledna. *Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism.* Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[64] Wenke Lee and Dong Xiang. Information-theoretic measures for anomaly detection. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 130–143. IEEE, 2001.

[65] Eugene Asarin, Thao Dang, and Oded Maler. The d/dt tool for verification of hybrid systems. In *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *LNCS*, pages 365–370. Springer, 2002.

[66] Oded Maler and Dejan Nickovic. Monitoring Temporal Properties of Continuous Signals. In *Proc. of FORMATS-FTRTFT 2004: the Joint International Conferences on Formal Modeling and Analysis of Timed Systmes and Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 3253 of *LNCS*, pages 152–166. Springer-Verlag, 2004.

[67] Martin Fränzle and Christian Herde. Hysat: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.

[68] Alexandre Donzé, Bruce Krogh, and Akshay Rajhans. Parameter synthesis for hybrid systems with an application to simulink models. In *Proc. of*

*HSCC 2009: the 12th International Conference on Hybrid Systems: Computation and Control*, volume 5469 of *LNCS*, pages 165–179. Springer, 2009.

[69] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, 2009.

[70] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *Proc. of FORMATS'10: the 8th International Conference on Formal Modeling and Analysis of Timed Systems*, volume 6246 of *LNCS*, pages 92–106. Springer, 2010.

[71] Alexandre Donzé. Breach, A toolbox for verification and parameter synthesis of hybrid systems. In *Proc. of CAV 2010: the 22nd International Conference on Computer Aided Verification*, volume 6174 of *LNCS*, pages 167–170. Springer, 2010.

[72] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In Shaz Qadeer Ganesh Gopalakrishnan, editor, *CAV*, LNCS. Springer, 2011.

[73] Yashwanth Annpureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *Proc. of TACAS 2011: the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 6605 of *LNCS*, pages 254–257, 2011.

[74] Alexandre Donzé, Oded Maler, Ezio Bartocci, Dejan Ničković, Radu Grosu, and Scott A. Smolka. On temporal logic and signal processing. In *Proc. of ATVA 2012: the 10th International Symposium on Automated Technology for Verification and Analysis*, volume 7561 of *LNCS*, pages 92–106. Springer, 2012.

[75] Matthias Althoff. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In *Proc. of HSCC '13: the 16th International Conference on Hybrid Systems: Computation and Control*, pages 173–182. ACM, 2013.

[76] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *CAV*, pages 258–263, 2013.

[77] Austin Jones, Zhaodan Kong, and Calin Belta. Anomaly detection in cyber-physical systems: A formal methods approach. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 848–853. IEEE, 2014.

[78] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. dReach: $\delta$-reachability analysis for hybrid systems. In *TACAS*, pages 200–205, 2015.

[79] Parasara Sridhar Duggirala, Sayan Mitra, Mahesh Viswanathan, and Matthew Potok. C2E2: A verification tool for stateflow models. In *TACAS*, pages 68–82. Springer, 2015.

[80] Rajarshi Ray, Amit Gurung, Binayak Das, Ezio Bartocci, Sergiy Bogomolov, and Radu Grosu. Xspeed: Accelerating reachability analysis on multi-core processors. In *HVC*, volume 9434 of *LNCS*, pages 3–18. Springer, 2015.

[81] Adel Dokhanchi, Aditya Zutshi, Rahul T. Sriniva, Sriram Sankaranarayanan, and Georgios Fainekos. Requirements driven falsification with coverage metrics. In *Proc. of EMSOFT: the 12th International Conference on Embedded Software*, pages 31–40. IEEE, 2015.

[82] Stefan Jaksic, Ezio Bartocci, Radu Grosu, Reinhard Kloibhofer, Thang Nguyen, and Dejan Ničković. From signal temporal logic to FPGA monitors. In *Proc. of MEMOCODE 2015: the 13th ACM/IEEE International Conference on Formal Methods and Models for Codesign*, pages 218–227. IEEE, 2015.

[83] O. E. David and N. S. Netanyahu. Deepsign: Deep learning for automatic malware signature generation and classification. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2015.

[84] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long Short Term Memory Networks for Anomaly Detection in Time Series. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 89–94, April 2015.

[85] J. Petit and S. E. Shladover. Potential Cyberattacks on Automated Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):546–556, April 2015.

[86] Ezio Bartocci, Luca Bortolussi, Laura Nenzi, and Guido Sanguinetti. System design of stochastic models using robustness of temporal properties. *Theor. Comput. Sci.*, 587:3–25, 2015.

[87] Jrgen Drrwang, Marcel Rumez, Johannes Braun, and Reiner Kriesten. Security Hardening with Plausibility Checks for Automotive ECUs. In *VEHICULAR 2017: The Sixth International Conference on Advances in Vehicular Systems, Technologies and Applications*, pages 38–41. IARIA, 07 2017.

[88] Konstantin Selyunin, Stefan Jaksic, Thang Nguyen, Christian Reidl, Udo Hafner, Ezio Bartocci, Dejan Nickovic, and Radu Grosu. Runtime monitoring with recovery of the SENT communication protocol. In *Proc. of CAV 2017: the 29th International Conference on Computer Aided Verification*, volume 10426 of *LNCS*, pages 336–355. Springer, 2017.

[89] Hui Kong, Ezio Bartocci, and Thomas A. Henzinger. Reachable set over-approximation for nonlinear systems using piecewise barrier tubes. In *Proc. of CAV 2018: the 30th International Conference on Computer Aided Verification*, volume 10981 of *LNCS*, pages 449–467. Springer, 2018.

[90] R. Mehra. On the identification of variances and adaptive kalman filtering. *IEEE Transactions on Automatic Control*, 15(2):175–184, April 1970.

[91] S. Ghosh, R. Melhem, and D. Mosse. Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 8(3):272–284, Mar 1997.

[92] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

[93] Lui Sha. Using Simplicity to Control Complexity. *IEEE Software*, 18(4):20–28, July 2001.

[94] R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

[95] Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: enforcement mechanisms for run-time security policies. *Int. J. Inf. Sec.*, 4(1-2):2–16, 2005.

[96] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[97] W. Khamphanchai, S. Pisanupoj, W. Ongsakul, and M. Pipattanasomporn. A multi-agent based power system restoration approach in distributed smart grid. In *2011 International Conference Utility Exhibition on Power and Energy Systems: Issues and Prospects for Asia (ICUE)*, pages 1–7, Sept 2011.

[98] Yliès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods in System Design*, 38(3):223–262, 2011.

[99] Nataliia Bielova and Fabio Massacci. Iterative enforcement by suppression: Towards practical enforcement theories. *Journal of Computer Security*, 20(1):51–79, 2012.

[100] Andrew Miller and Joseph J LaViola Jr. Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin. *Available on line: http://nakamotoinstitute. org/research/anonymous-byzantine-consensus*, 2014.

[101] Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, and Omer Nguena-Timo. Runtime enforcement of timed properties revisited. *Formal Methods in System Design*, 45(3):381–422, 2014.

[102] Yliès Falcone and Hervé Marchand. Enforcement and validation (at runtime) of various notions of opacity. *Discrete Event Dynamic Systems*, 25(4):531–570, 2015.

[103] E. Meneses, X. Ni, G. Zheng, C. L. Mendes, and L. V. Kal. Using Migratable Objects to Enhance Fault Tolerance Schemes in Supercomputers. *IEEE Transactions on Parallel and Distributed Systems*, 26(7):2061–2074, July 2015.

[104] Oliver Höftberger. *Knowledge-based Dynamic Reconfiguration for Embedded Real-Rime Systems*. PhD thesis, Technische Universität Wien, 2015.

[105] Egor Dolzhenko, Jay Ligatti, and Srikar Reddy. Modeling runtime enforcement with mandatory results automata. *Int. J. Inf. Sec.*, 14(1):47–60, 2015.

[106] Cristina Alcaraz and Javier Lopez. Safeguarding Structural Controllability in Cyber-Physical Control Systems. In Ioannis Askoxylakis, Sotiris Ioannidis, Sokratis Katsikas, and Catherine Meadows, editors, *Computer Security – ESORICS 2016*, pages 471–489, Cham, 2016. Springer International Publishing.

[107] Y. Yan, B. Zhang, and J. Guo. An Adaptive Decision Making Approach Based on Reinforcement Learning for Self-Managed Cloud Applications. In *2016 IEEE International Conference on Web Services (ICWS)*, pages 720–723, June 2016.

[108] Yliès Falcone, Thierry Jéron, Hervé Marchand, and Srinivas Pinisetty. Runtime enforcement of regular timed properties by suppressing and delaying events. *Systems & Control Letters*, 123:2–41, 2016.

[109] Ali Dorri, Salil S Kanhere, Raja Jurdak, and Praveen Gauravaram. Blockchain for iot security and privacy: The case study of a smart home. In *Pervasive Computing and Communications Workshops (PerCom Workshops), 2017 IEEE International Conference on*, pages 618–623. IEEE, 2017.

[110] Matthieu Renard, Antoine Rollet, and Yliès Falcone. Runtime enforcement using büchi games. In *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software*, pages 70–79. ACM, 2017.

[111] D. Ratasich, T. Preindl, K. Selyunin, and R. Grosu. Self-Healing by Property-Guided Structural Adaptation. In *2018 IEEE 1st International Conference on Industrial Cyber-Physical Systems (ICPS)*, pages 199–205, May 2018.

[112] Olivier Alphand, Michele Amoretti, Timothy Claeys, Simone Dall'Asta, Andrzej Duda, Gianluigi Ferrari, Franck Rousseau, Bernard Tourancheau,

Luca Veltri, and Francesco Zanichelli. Iotchain: A blockchain security architecture for the internet of things. In *Wireless Communications and Networking Conference (WCNC), 2018 IEEE*, pages 1–6. IEEE, 2018.

[113] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. In *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*, pages 1–33. Springer, 2018.

[114] Paul F Wilson. *Root cause analysis: A tool for total quality management.* ASQ Quality Press, 1993.

[115] Bing Liu, Lucia, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. Localizing multiple faults in simulink models. In *International Conference on Software Analysis, Evolution, and Reengineering*, pages 146–156. IEEE Computer Society, 2016.

[116] Bing Liu, Lucia, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. Simulink fault localization: an iterative statistical debugging approach. *Softw. Test., Verif. Reliab.*, 26(6):431–459, 2016.

[117] Bing Liu, Lucia, Shiva Nejati, and Lionel C. Briand. Improving fault localization for simulink models using search-based testing and prediction models. In *International Conference on Software Analysis, Evolution and Reengineering*, pages 359–370. IEEE Computer Society, 2017.

[118] Thomas Ferrère, Oded Maler, and Dejan Nickovic. Trace diagnostics using temporal implicants. In *International Symposium on Automated Technology for Verification and Analysis*, volume 9364 of *LNCS*, pages 241–258. Springer, 2015.

[119] J. C. Sancho, F. Petrini, K. Davis, R. Gioiosa, and S. Jiang. Current practice and a direction forward in checkpoint/restart implementations for fault tolerance. In *19th IEEE International Parallel and Distributed Processing Symposium*, April 2005.

[120] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*, pages 195–216. Springer, 2017.

[121] Nicola Dragoni, Alberto Giaretta, and Manuel Mazzara. The internet of hackable things. In *Proceedings of 5th International Conference in Software Engineering for Defence Applications*, pages 129–140, Cham, 2018. Springer International Publishing.

[122] Muhammad Shafique, Theocharis Theocharides, Christos-Savvas Bouganis, Muhammad Abdullah Hanif, Faiq Khalid, Rehan Hafiz, and Semeen Rehman. An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the iot era. In *Design,*

*Automation & Test in Europe Conference & Exhibition (DATE), 2018*, pages 827–832. IEEE, 2018.

[123] Michael E Locasto. Self-healing: science, engineering, and fiction. In *Proceedings of the 2007 Workshop on New Security Paradigms*, pages 43–48. ACM, 2008.

[124] D. Ratasich, O. Höftberger, H. Isakovic, M. Shafique, and R. Grosu. A Self-Healing Framework for Building Resilient Cyber-Physical Systems. In *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 133–140, May 2017.

[125] Hermann Kopetz. A Conceptual Model for the Information Transfer in Systems-of-Systems. In *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 17–24, June 2014.

[126] Aditya Zutshi, Sriram Sankaranarayanan, Jyotirmoy V. Deshmukh, James Kapinski, and Xiaoqing Jin. Falsification of safety properties for closed loop control systems. In *International Conference on Hybrid Systems: Computation and Control*, pages 299–300. ACM, 2015.

[127] Robert Reicherdt and Sabine Glesner. Slicing MATLAB simulink models. In *International Conference on Software Engineering*, pages 551–561. IEEE Computer Society, 2012.