A Survey on Resilience in the IoT: Taxonomy, Classification and Discussion of Resilience Mechanisms

CHRISTIAN BERGER[*], University of Passau
PHILIPP EICHHAMMER, University of Passau
HANS P. REISER, University of Passau
JÖRG DOMASCHKA, Ulm University
FRANZ J. HAUCK, Ulm University
GERHARD HABIGER, Ulm University

Internet-of-Things (IoT) ecosystems tend to grow both in scale and complexity as they consist of a variety of heterogeneous devices, which span over multiple architectural IoT layers (e.g., cloud, edge, sensors). Further, IoT systems increasingly demand the resilient operability of services as they become part of critical infrastructures. This leads to a broad variety of research works that aim to increase the resilience of these systems. In this paper, we create a systematization of knowledge about existing scientific efforts of making IoT systems resilient. In particular, we first discuss the taxonomy and classification of resilience and resilience mechanisms and subsequently survey state-of-the-art resilience mechanisms that have been proposed by research work and are applicable to IoT. As part of the survey, we also discuss questions that focus on the practical aspects of resilience, e.g., which constraints resilience mechanisms impose on developers when designing resilient systems by incorporating a specific mechanism into IoT systems.

CCS Concepts: • **General and reference** → **Surveys and overviews**; **Reliability**; • **Computer systems organization** → *Reliability*; *Availability*; *Maintainability and maintenance*; • **Software and its engineering** → *Software fault tolerance*; • **Security and privacy** → *Access control*; *Authorization*; **Intrusion/anomaly detection and malware mitigation**.

Additional Key Words and Phrases: Internet of Things, resilience, dependability, security

ACM Reference Format:
Christian Berger, Philipp Eichhammer, Hans P. Reiser, Jörg Domaschka, Franz J. Hauck, and Gerhard Habiger. 2021. A Survey on Resilience in the IoT: Taxonomy, Classification and Discussion of Resilience Mechanisms. *ACM Comput. Surv.* 1, 1, Article 1 (January 2021), 37 pages. https://doi.org/10.1145/3462513

1 INTRODUCTION

From a technical view, Internet-of-Things (IoT) systems can be clearly distinguished from other systems by a set of technical characteristics. First, they tend to display a layered architecture where system components spread out across different layers, e.g., *sensor landscape*, *edge*, and

[*] Responding author

Authors' addresses: Christian Berger, cb@sec.uni-passau.de, University of Passau, Innstraße 41, Passau, Germany, 94032; Philipp Eichhammer, pe@sec.uni-passau.de, University of Passau, Innstraße 41, Passau, Germany, 94032; Hans P. Reiser, 0000-0002-2815-5747, University of Passau, Innstraße 41, Passau, Germany, 89081; Jörg Domaschka, joerg.domaschka@uni-ulm.de, Ulm University, Albert-Einstein-Allee 43, Ulm, Germany, 89081; Franz J. Hauck, franz.hauck@uni-ulm.de, Ulm University, Albert-Einstein-Allee 11, Ulm, Germany, 89081; Gerhard Habiger, gerhard.habiger@uni-ulm.de, Ulm University, Albert-Einstein-Allee 11, Ulm, Germany, 89081.

0360-0300/2021/1-ART1
https://doi.org/10.1145/3462513

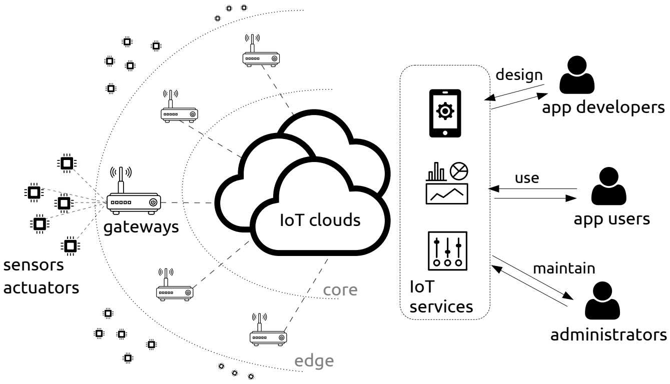

Fig. 1. Layered IoT architecture [44].

cloud [133] (see Figure 1). Furthermore, IoT systems consist of a broad variety of heterogeneous system components. Typically, IoT components can be somehow classified, e.g., by computational power, capabilities, or locality to fit within one of the previously mentioned architectural layers.

IoT systems extend the world of computing to the domain of physical objects, which now become part of the overall system, e.g., by being equipped with sensors and actuators. Ideally, system operators want to ensure the longevity of their IoT systems and thus want their systems to preserve security and dependability properties by implementing concrete resilience mechanisms. These mechanisms allow a system to display behavioral stability (e.g., absorption, adaption, or recovery actions) when facing changes (such as disruptions like attacks or accidental faults). In particular, a fault occurring in some individual component within one of the IoT layers should not lead to the system becoming unable to match its behavioral requirements. System operators seek solutions and are assisted by a variety of efforts that have been made to support the resilience of IoT systems.

The goal of this paper is to first investigate whether there is a general view and a common understanding of resilience across academic literature, including the definition of resilience, its properties, and classifications. Subsequently, we shift the focus towards resilience in the IoT domain and outline which particular challenges exist here. We also discuss the practical aspects of resilience and review a set of resilience mechanisms for their applicability in IoT.

1.1 Motivation

A recent research roadmap has shown the need for research on resilience in IoT infrastructures [140]. The overall goal is often to secure and harden IoT systems against both hardware and software failures [1, 33, 53, 60, 137, 154] including sensor nodes, computers, communication links, and cloud services, and to protect against malicious attacks targeted at the system. Other works also address the challenge of scaling resilience with a growing number of possibly heterogeneous devices [112], and of matching requirements specified by system operators.

In this document, we first want to examine if there is a common understanding of the *definition of resilience*, and, in particular, which *properties* of an IoT system are encompassed by the term resilience. Subsequently, we study *means* for assuring these characteristics in more detail. We are using the term *resilience mechanisms* for these means.

With resilience fitting the field of dependable and secure system design, the seminal work of Avižienis et al. marks a good starting point [13]. It gives precise definitions characterizing concepts of dependability and security of computing and communication systems. Building on their work, this survey shifts the focus to challenges of IoT systems and hence, generates a broader view on resilience covering topics that have not been sufficiently addressed by Avižienis et al.; particularly:

(1) We categorize and elaborate on *security mechanisms* beyond the set of intended security goals defined by Avižienis et al. who also disregard concerns such as privacy and identity management [13]. For IoT systems, which extend to the physical world and where components may collect or process personal data, privacy is a highly important matter: Consequently, neither software nor hardware components should leak information uncontrolled towards the outside world; if they do they cannot be considered to work as intended. Identity management and authorization services also play an important role in the IoT, since nowadays systems are more open and of a larger scale than systems were over a decade ago. Further, current systems tend to include more heterogeneity and are more dynamic (rapidly evolving), making it even harder to keep track of all the components involved.

(2) We explicitly address IoT architectures which typically display a *multi-layer architecture* divided into core, edge, and device landscape. IoT systems face various limitations and face heterogeneous requirements both of which complicate the implementation of resilience. We also provide more technical details on mechanisms discussing them from a practical and implementation-level view that is augmented with different examples of IoT systems.

(3) We extend their *binary stance* on attributes to a continuum reflecting more conditions of the real world. For instance, in contrast to assuming that fault tolerance encapsulates mechanisms that can fully preserve the system functionality by compensating or removing a fault, we acknowledge that in practical settings such system should continue to function, probably in a degraded form, even when faults can no longer be compensated. IoT systems tend to explore new approaches for resilience by leveraging mechanisms that may work towards providing a “best-effort” resilience.

1.2 Research Questions

This paper addresses research questions about the taxonomy and classification of resilience in general, and in particular by identifying resilience mechanisms that can be leveraged as building blocks to design resilient IoT systems. Overall, the questions we want to answer are the following:

① **Definitions of resilience and resilience mechanism:** Does a consistent understanding of resilience in distributed systems, and in particular in IoT, exist? What is a common understanding of resilience, e.g., which concepts and attributes does resilience encompass? Which mechanisms can be labeled as resilience mechanisms?

② **Measurability, adjustability, and best-effort resilience:** Which approaches exist for quantifying resilience? What are relevant dimensions that need to be considered for quantification? How can system operators align systems to satisfy resilience requirements? Can we define degrees of preservation of system functionality? Rather than only distinguishing between correct service execution and service failure, how can a system be designed to deliver its service as a best effort, e.g., if faults cannot be fully tolerated (*graceful degradation*)?

③ **Classification of resilience mechanisms:** Which aspects can be used for a taxonomy of resilience mechanisms in the IoT? More specifically:
- Depending on technical properties of IoT infrastructures, in which different layers (e.g., sensor landscape, edge, cloud) do resilience mechanisms operate in?
- What requirements must applications fulfill to support resilience mechanisms? Which considerations need developers take into account when designing resilient IoT applications?

1.3 Outline

The remainder of this paper is outlined as follows: Section 2 summarizes the methodology employed when creating this survey. Subsequently, Section 3 provides a more general taxonomy on resilience, including its definitions, properties, and classifications. Then, we shift our focus more towards resilience in the IoT. Further, we discuss the measurability and adjustability of resilience in Section 4. In Section 5, we analyze and discuss a broad range of resilience mechanisms from a more practical, implementation-level view, i.e., in respect to the constraints they may impose on applications. Section 6 gives an overview of related work. In Section 7, we draw our conclusions.

2 METHODOLOGY

To create this survey, we employed a basis of research papers that we initially knew and which covered a traditional view on resilience, by focusing on achieving security and dependability in distributed systems. A strong inspiration has been the survey of Avižienis et al. [13]. Building on this, we performed a systematic search that focused on resilience in the IoT. To increase transparency and reproducibility, we briefly present our search methodology to gather and select a large body of academic literature in the field of resilience for IoT systems. Our main ambitions are the following:

- Find distinct approaches in literature for defining the term *resilience* to reason about the question whether a consistent understanding of this term exists
- Gather different approaches for measuring resilience aspects (which might be either quantitative or by ascertaining a specific quality of service can be provided)
- Identify and classify resilience mechanisms to create a broad taxonomy for resilience mechanisms in the IoT

Search strategy. For literature research we used the ACM Digital Library with currently 609,508 records. Further, we employed the following search queries without filters on publication date. We combined the results of these queries and sanitized them for duplicates.

> [[Publication Title: (iot)] OR [Publication Title: "internet of things"]] AND [[Full Text: resilience] OR [Full Text: resilient]] (339 Results)
>
> [[Full Text: (iot)] OR [Full Text: "internet of things"]] AND [[Publication Title: resilience] OR [Publication Title: resilient]] (88 Results)

Selection criteria. We used a list of selection criteria that can be applied to determine whether a paper found by the search queries should be selected to be included in the survey. These criteria also encompass conditions used to assess the quality of a found paper.

Inclusion criteria – A paper is included if one of the following criteria is satisfied:

- The paper presents an own approach for defining, describing, or classifying the term resilience;
- it elaborates metrics to reason about or measure resilience,
- or presents a specific resilience mechanism that can be applied in at least one IoT layer.

Exclusion criteria – The exclusion process is applied after inclusion. A paper is excluded if only one of the following criteria applies:

- The paper is not peer-reviewed;
- the paper presents its own view on resilience but belongs to another field (other than distributed systems);
- the presented paper proposes a resilience mechanism but does not present a careful experimental evaluation showing how it can improve resilience;
- the paper lacks relevance: the paper presents a resilience mechanism but the presented mechanism is an incremental refinement of an earlier proposed mechanism.

Selection procedure and results. Our selection procedure is the following: The first phase consists of a fast scan, where each paper is examined by a single assessor to check if the paper could be of interest. In this phase, only papers that obviously do not qualify are sorted out, e.g., if the field is not computer science, the topic of the paper is not related to resilience, or the paper is

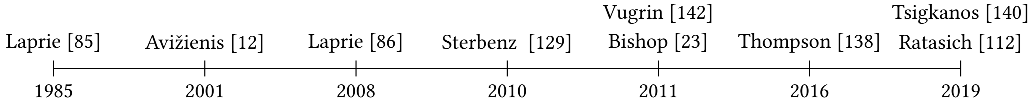

Fig. 2. Timeline for definitions related to resilience.

not a research paper but a short abstract or workshop invitation. In the second round, all remaining papers are examined by two different reviewers to check for their relevance: A paper is relevant if at least one inclusion criterion applies and none of the exclusion criteria applies. Subsequently, all papers are being tagged either *relevant* or *not relevant.* In a final phase, conflicts among assessors are resolved by including a third assessor and discussing the disagreement. In the end, 41 papers have been selected for inclusion in our survey.

3 TAXONOMY

In order to clarify the taxonomy, we first give a broad overview of the usage and meaning of the term *resilience* across different domains. We investigate whether there is a consistent understanding of this term in general and in the context of IoT. From these findings, we derive a definition for *resilience* as well as for *resilience mechanism* that fits the IoT domain (Section 3.1). Further, we investigate the different attributes and concepts resilience encompasses in general, i.e., in the broader field of distributed systems, to establish an understanding of this subject (Section 3.2). Subsequently, we concretize and align our taxonomy of resilience mechanisms with the field of IoT by considering IoT-specific technical system properties and challenges (Section 3.3).

3.1 Definitions

In academic literature there is a multitude of definitions for the term resilience, mainly depending on the research domain. In addition to that, the attributes of resilient systems are often also lacking a consistent notion in literature. Hence, in this chapter, we try to find a common understanding of the term *resilience.* Figure 2 shows a brief timeline of selected definitions.

A general view on resilience. Thompson et al. collected general definitions of resilience from various research fields, e.g., in the context of nature and environment, the public sector, and others [138]. One of it is from the US government giving an organizational view on resilience as

> *the ability to adapt to changing conditions and prepare for, withstand, and rapidly recover from disruption* [114].

This definition is covering a general view on resilience and outlines that resilient systems need to prepare for (possibly unpredictable) changes, and then be capable of re-entering a functional state again (thus "bouncing back"). This general notion of resilience describes the capacity of a system to maintain behavioral stability, which is also being elaborated by different concepts in the work of Vugrin et al. who describe resilience by three general concepts [142]: *absorption* (withstand disruptions), *adaption* (self-organize to, e.g., regain performance), and *recovery* (ability of a system to be repaired easily). Bishop et al. understand resilience as the property that a system may degrade in its *quality of service* under stress over time, but is able to recover [23]:

> *It is important to note that such capability implies that the system must not cease to exist—that is, it must survive at some capacity, in order to autonomously recover. In the cyber domain, a resilient system continues to provide essential functionality, even under duress or in an impaired state* [23].

From a technical perspective, Thompson et al. provide a definition of security-related resilience:

> *resilience is the maintenance of the nominated state of security* [138].

Here, the *nominated state* is defined as a specific condition determined through a governance process that assesses the intrinsic value of the resource that is designated as requiring security [138]. Security is violated when the nominated state changed. This concludes to resilience being the ability to prevent or resolve that change, thus maintaining the nominated state and withstanding disruptions. In the field of computer networks, Sterbenz et al. define resilience as

> *the ability of the network to provide and maintain an acceptable level of service in the face of various faults and challenges to normal operation* [129].

Further, in the field of dependable systems, among oft-cited definitions are the ones given by Laprie et al. in their work *From dependability to resilience* [86]. They define *resilience* as

> *the persistence of service delivery that can justifiably be trusted, when facing changes* [86].

This definition extends the definition of *dependability* that has been used by Laprie in earlier works:

> *dependability is the quality of the delivered service such that reliance can justifiably be placed on this service* [85].

Avižienis et al. also follow this view and define dependability as

> *the ability to deliver service that can justifiably be trusted* [12].

Note that the definition of resilience from Laprie et al. reflects the ambition to preserve *dependability* in the face of (possibly unforeseen) *changes*. In particular, resilience means maintaining dependability even if unexpected things happen. Laprie et al. also conclude this by providing a shorthand, alternate definition:

> *the persistence of dependability when facing change* [86].

Changes and disruptions. So far, these definitions employ the terminology of *disruptions* and *changes*. A disruption is a very specific form of change. According to Tsigkanos et al., *disruption is an adverse change to system stability* and it can fundamentally affect system requirements [140]. Adverse changes can be either *external* to the system, e.g., caused by the environment the system operates in, or *internal*, such as when some internal fault occurs. Such changes might push the system into an unforeseen and possibly unwanted state. On the other hand, changes can be intended and desired as well, e.g., to improve or evolve the system. Planned changes are part of a systematic and subsequent development of the system, such as adding additional resources or novel application components to a system.

Resilience in IoT. One concrete goal of our literature research was to find out which approaches for defining resilience in the IoT domain exist and whether the understanding of this term is consistent. After applying our search methodology and investigating which understanding IoT papers have under the term resilience, we made a few observations. First, a large body of papers uses the adjective *resilient* rather nonchalantly to characterize their systems or solutions without explicitly providing an own definition or referencing an existing definition. The attribute *resilient* is just added without further explanation and its interpretation is left to the reader. Second, in other papers, *resilient* is used in *conjunction*, e.g., a developed approach is resilient *against* specific attacks, faults, perturbations, or overloads. In this type of papers (e.g., [18, 20, 72, 151]), the understanding of resilience is implicitly made clear as withstanding specific system alterations (e.g., attacks, faults, or overload) to maintain a functional system state. Third, a few IoT papers explicitly define their own resilience definition or employ existing definitions. We discuss these different approaches to defining resilience in the following.

Delic et al. define resilience by referring to a system's *behavioral* requirements of being capable of self-stabilizing its state during and after change:

> *The resiliency of a system is defined by its capability (1) to resist external perturbances and internal failures; (2) to recover and enter stable state(s); and (3) to adapt its structure and behavior to constant change* [38].

In the context of smart cities, Modarresi et al. adopt the definition of Sterbenz et al. [129] and give a *goal-oriented* view on resilience, that is, maintaining an acceptable level of service during changes:

> *We define resilience as the ability of the system to provide and maintain an acceptable level of service in the face of various faults and challenges to normal operation* [95].

Pradhan et al. present a goal-driven orchestration middleware for resilient IoT systems. They employ their own explanation of resilience. In their view resilience is goal-oriented and has behavioral requirements:

> *Each application deployed for a mission has specific goal(s) that must be satisfied at all times. IoT systems should therefore be equipped with mechanisms that ensure all critical goals are satisfied for as long as possible, i.e., they must be resilient by facilitating failure avoidance, failure management, and operations management to support incremental hardware and software changes over time* [110].

Further, Khan et al. use a resilience definition for routing in the IoT [75] that is borrowed from earlier work [45]. This definition reasons about both the behavior (absorption capability) and goal (continuation of delivery) of resilience:

> *the ability of a network to absorb the performance degradation under some failure pattern (random or intentional) and to continue delivering messages with an increasing number of k compromised nodes* [45].

Another definition focusing on service continuation despite changes is given by Witti et al.:

> *Device resilience refers [to] the ability of a component to maintain service with alteration in the system environment* [146].

Since IoT systems are systems that tend to evolve in larger scale and have considerable dynamics, complexity, and heterogeneity of involved components, these circumstances may be regarded in a resilience definition as well: Ratasich et al. extend Laprie's definition and argue that resilience is a property which, in the context of IoT systems, should also *scale* dependability and security, e.g., when dealing with environmental or technological changes, which they refer to as *long-term dependability and security* [112]. Here, the definition of dependability is taken from Avižienis et al. who define dependability as *the ability to deliver service that can justifiably be trusted* [12]. Avižienis also follows the definition of Laprie et al. in his more recent work [11]. Tsigkanos et al. summarize a vision, challenges and research directions road-map for IoT systems, and they define resilience as *the persistence of reliable requirements satisfaction when facing change* [140], thus largely following the view of Laprie, too.

Summary and take-away message. Summarized, we see that the definitions of resilience require the system to be capable of, *in the face of changes*, preserving some correct state and functionality ("withstanding changes") or to provide means to recover and go back to an intended state ("bouncing back"). Being able to maintain behavioral stability of a system even during disruptions is a core characteristic of resilience. We conclude that, while the terminology used to define resilience is diverse in literature, the understanding of resilience is substantially consistent: If the goal of a resilient system is to be capable of delivering service that can justifiably be trusted even when facing changes, then it needs to provide mechanisms for ensuring behavioral stability, that is, absorbing disruptions, temporarily degrading its state but subsequently recovering (bouncing back) to its nominated state later on. Moreover, in the IoT domain, we found that the definitions of Laprie and Sterbenz seem to be rather popular as they have both been employed (either one-to-one or in

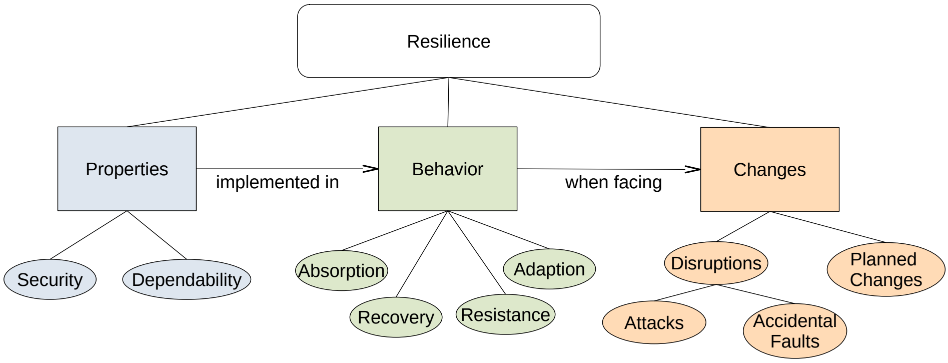

Fig. 3. Resilience terms and their relation. Resilience can be understood as the ambition to preserve dependability and security properties when facing changes, which, in turn, manifests in a specific behavior of the system that facilitates stability despite these changes.

an adapted form) in several IoT papers [85, 129]. In Figure 3, we show a diagram that illustrates how terms used to define resilience are connected.

For our own definition of resilience we strongly identify with the ones of Laprie et al., hence, also largely with Avižienis et al. and Ratasich et al.. Further, we also distill security as a main property to resilience besides dependability. In addition to outlining the properties dependability and security, we also *emphasize the expected behavior* of resilient systems, that is, being able to withstand changes or recover. In summary, our definitions of resilience and resilience mechanisms are formulated as:

> *Resilience* is the property of preserving the dependability and security of a system when the system encounters changes, thus withstanding or recovering from impairments. *Resilience mechanisms* are all means that work towards achieving this property.

We emphasize our understanding of *resilience* as an umbrella term, which considers both dependability and security properties, thus asserting that some well-defined correct state can be recovered or preserved (possibly temporarily at some degraded level) even if changes, such as (potentially malicious) faults and attacks, occur in the system. Consequently, mechanisms that ensure the persistence of dependability and security when facing changes are *resilience mechanisms*.

Fault, error, and failure. When we speak about faults in the remainder of this paper, we stick to the terminology introduced by Avižienis et al., in particular, we employ the *fault→error→failure* chain [13]: *Faults* can be dormant or active. Once a fault activates, it produces an *error*, hence a fault is the cause of some error, while an error is the manifestation of some preceding fault, e.g., in form of an incorrect state. If an error leads to another error, we call this *error propagation*. Further, if some error leads to a deviation of system behavior from its intended service delivery, e.g., a system produces and returns a wrong result because of its own erroneous state, then we call it a *failure*. Note that this chain may propagate recursively, because the failure of some component A may appear as an external fault to another component B, if the correct execution of B depends on receiving correct results from A. For a more specific classification of fault types, we refer to the extensive taxonomy of Avižienis et al. [13].

Relation between resilience and security. Often, research works covering the topic of resilience mainly focus on dependability mechanisms, e.g., fault tolerance and fault prevention.

Sometimes, resilience and security are even treated as two separate, partly distinct subjects. From the view of a system operator, however, shifting the focus between dependability and security might depend on knowledge about the environment a system runs in and the capability of making reasonable assumptions about the behavior and interactions of the system with its users. If available, these assumptions can be used for creating models (e.g., a fault model or an attacker model) and to subsequently select appropriate mechanisms that work for the selected model. Resilience consists of dependability and security. Both are often closely related as they share a common set of system properties, such as ensuring availability and integrity [13]. Thus, some mechanisms can also fit into both categories. An example of this is intrusion-tolerant replication [22], where the integrity and availability of a service is preserved even if a bounded amount f of replicas becomes faulty or intruded[1] by an attacker within a bounded amount of time T, assuming the system employs proactive recovery [128]. Our definition of resilience emphasizes that resilience also needs to preserve the security of a system in the face of harmful conditions like attacks, thus resilience includes security.

3.2 Attributes and Basic Classifications

Section 3.1 shows that there is a broad variety of definitions for resilience. In consequence, the properties and concepts used to describe a resilient system are varying as well and moreover sometimes become overloaded with different meanings. In this section, we seek to find a common set of well-defined attributes, which are both applicable for practical IoT systems and also commonly agreed upon across different research works. Due to the fact that definitions of resilience are often linked to the properties *dependability* and *security*, we begin by considering resilience attributes as the set of attributes that are the conjunction of the security and dependability attributes. As a starting point, we refer to the well-established taxonomy of Avižienis et al., who characterize *dependability* as consisting of the following attributes [13]:

- **availability** *(readiness for correct service)*
- **integrity** *(absence of improper system alterations)*
- **reliability** *(continuity of correct service)*
- **safety** *(absence of catastrophic consequences)*
- **maintainability** *(ability to undergo modifications and repairs)*

Furthermore, *security* includes **availability**, **integrity**, and

- **confidentiality** *(absence of unauthorized system disclosure of information)*

The relation of system attributes as described by Avižienis et al. is largely agreed upon and has been adopted by a broad variety of academic literature in the field of resilient systems [23, 32, 112, 124, 129, 141].

Yet, in this paper, differing from Avižienis, we consider *availability* to be an attribute encompassing both maintainability and reliability. For instance, the probability for a system or component to keep working successfully up to a specified amount of time can describe the property *reliability* among other possible metrics. Generally, reliability can be modeled through the use of stochastic means and metrics, e.g., mean time to failure. Further, maintainability is a characteristic that expresses how easy (and time consuming) it is to repair a system in case of failures and thus to make it available again. Both attributes compose the availability of a system. Maintainability also considers planned changes to the system, e.g., by adding or removing components as the system evolves over time. Our approach of classifying availability matches the current IEC standard IEC 60050-192:2015

[1] Generally, attacks like an intrusion could be viewed as malicious, external faults.

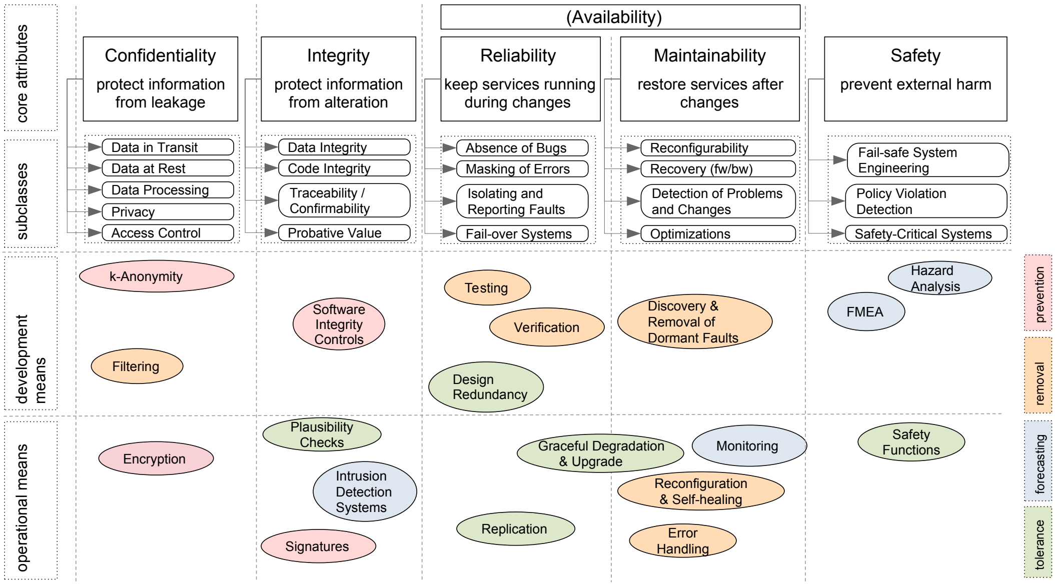

Fig. 4. Attributes of resilience and means.

which notes that "*[a]vailability depends upon the combined characteristics of the reliability, recoverability, and maintainability of the item, and the maintenance support performance*" [65]. The minor difference here is only that we use "maintainability" as the generic umbrella term that encompasses the recoverability, the maintainability, and the maintenance support from the IEC definition.

Subclasses of resilience core attributes. The core attributes can be further decomposed into different subclasses as shown in Figure 4. The attribute *confidentiality* consists of different subclasses that have the common goal of protecting information from leakage. This includes protecting data in transit, at rest, and while processing as well as controlling the legitimized access to data. When applications collect and process data related to individuals, e.g., a person's location, then privacy, which is the ability of an individual or group to express themselves selectively, also becomes a crucial concern.

Further, *integrity* is a property that predicates whether information or resources are protected from unauthorized modification. This includes the integrity of data, regardless of whether it is stored or in transit (e.g., in form of messages) as well as the integrity of applications (which process data), e.g., by ascertaining code integrity. In respect to users interacting with a system or with other users, it is often necessary to guarantee traceability, i.e., being able to trace specific actions back to users as well as the attached probative value, e.g., being able to prove the judicial liability of a specific user for some specific action.

Reliability is a property that aims to make a promise about the system's correct behavior over time, such as "the mean time for this component to fail is X days" or "the probability of a component failing before time T is reached is X". The attribute *reliability* encompasses, for instance, the absence of bugs, masking of errors, or isolating or reporting faults. The common goal among these is to keep the service running, despite of considering different paths of how this can be achieved.

Maintainability is an attribute which is often regarded in a broader sense. Overall, it means to be capable of restoring services after changes or problems occur. Firstly, if we consider *adverse* changes, such as faults or attacks, we refer to the system's capability to react to these changes. In this respect, it encompasses the detection of such changes and problems and also the recovery of the system as well as its reconfiguration. Secondly, if we consider planned changes, such as adding new system components to provide additional functionality, or scaling the system size to increase its performance, maintainability may largely refer to how easy such an intended reconfiguration can be implemented by administrators.

Finally, *safety* is a property that states that the system should prevent external harm to its environment or persons. As subclasses it encompasses fail-safe system engineering, policy violation detection, and safety-critical systems.

Implementing resilience into practical systems. There is a variety of means to implement resilience attributes into practical systems (some examples are shown in Figure 4). These means can be categorized into *operational means*, which function during the operation of the system, and *developmental means*, which are applied during the design phase of the system [13]. Moreover, means can be also classified further in respect to how they approach faults, into *removal*, *prevention*, *tolerance*, and *forecasting* types [13]. Fault-removal means are employed to reduce faults, e.g., *software testing* can discover bugs, which are then subsequently fixed. Moreover, fault-prevention means try to prevent the occurrence of faults, e.g., *anonymization* techniques like k-anonymity [131] can protect a service from privacy-related faults. Fault-tolerance means prevent service failures despite occurring faults, for example by masking the presence of faults using a replication protocol. A typical example of this are replication protocols.

Finally, fault-forecasting means have the ambition to estimate both the present and future occurrences of faults and their consequences [13]. We show for some chosen resilience means how they fit into these categorizations in Figure 4. It is noteworthy that a single resilience means can *preserve multiple resilience attributes*, which is not sufficiently covered by Figure 4. For instance, *replication* is aimed to preserve the availability of a service, e.g., by managing redundant copies of an application state, but could also protect integrity, by letting replicas redundantly execute requests with a majority agreeing on what is the correct copy of the application state.

While the previously presented core attributes are very commonly agreed upon across literature, there are also works that refine or extend this basic set. Ratasich et al. [112] unify the concepts of dependability and security as defined by Avižienis under the term *resilience*. They also define the property *scalable resilience* [112], which encompasses both *resilience* (with its usual attributes) and *evolvability*, which expresses the need of a heterogeneous and complex system to be capable of evolving over time. This requires dependability and security attributes that are considered during design time to scale up with actual IoT system requirements. Sterbenz et al. divide resilience properties into two groups [129]: *trustworthiness resilience* encompasses common *security* and *dependability* attributes along with *performability*, which contains quality of service metrics, while the other group called *challenge tolerance disciplines* consists of survivability (e.g., in case of correlated failures or disasters), disruption tolerance (e.g., caused by the environment) and traffic tolerance (such as DDos attacks).

Summary. The *resilience attributes* of IoT systems cover the preservation of *integrity*, *confidentiality*, *availability*, *reliability*, *maintainability* and *safety* of these systems. Resilience mechanisms target the implementation of these properties in practical systems during development and operation. IoT systems tend to face new and unforeseen challenges as they grow in size, heterogeneity, and complexity over time.

3.3 IoT-specific Challenges for Resilience Mechanisms

After defining the term resilience and identifying its corresponding core attributes, we here shift the focus towards resilience in IoT, investigate challenges that exist for the design of resilience mechanisms, and further touch on possible solutions. We go into more detail on these in Section 5.

The main challenges we identified for incorporating resilience mechanisms into IoT systems are *architectural* challenges (e.g., device constraints, heterogeneity, and scalability), the challenge of limiting *dependencies across layers*, and *administrative* challenges (e.g., the dispersion of system components into different administrative domains).

Architectural challenges of IoT systems. IoT systems have a broad heterogeneity of physical components distributed across different layers within IoT infrastructures, namely sensor landscape (device level), edge, and cloud. The heterogeneity of components results in a distinct variation in terms of available hardware resources such as computing capabilities, power constraints, and available memory. These also impose limitations on the applicability of certain resilience mechanisms (we will discuss this in more detail in Section 5). Further, IoT system developers are also faced with the question of how the overall system can be designed resiliently when spanning different layers, in contrast to just making individual components within a single layer resilient. Initially, among sensor landscape, edge, and cloud, different resilience mechanisms might be employed individually to preserve overall resilience properties for the system. For example, resilience techniques that preserve *integrity* in the sensor landscape might include a design that uses redundancy among sensors [137] (and ensures correctness by voting on measurement values in the edge, e.g., taking the median of values as measured by distinct sensors), plausibility checks [82] (e.g., does the reported value match some pre-defined expression, pattern, or range), and physical protection of sensors so that they can not be tampered with by malicious attackers [7] (e.g., holding a lighter towards a temperature sensor makes it produce wrong results). More sophisticated integrity-preserving mechanisms could be deployed in edge and cloud, e.g., intrusion-detection systems are often rather placed on edge nodes such as gateways (e.g., because this allows observing the network flow) or cloud servers (where data is often stored and processed), than on resource-constrained devices. Yet, some lightweight IDS solutions exist that enable IDS functionality even on small IoT devices [87, 103]. In this context, a technique that proves to be especially useful in the cloud to build IDS is *virtual machine introspection* [51]. Summarized, we need to carefully regard the technical limitations dictated by the IoT architecture when selecting and incorporating resilience mechanisms into system components.

Resilience mechanisms should work across IoT layers. We see that it is possible to place resilience mechanisms into different layers and subsequently achieve certain resilience properties within some layer; however, we also need to be concerned with the dependencies across layers, and hence also design techniques that work across layers. For instance, IoT systems might face connectivity disruptions between edge and cloud. Ideally, a resilient system should try to continue delivering its service functionality on a best-effort basis. By employing *partition-tolerant redundancy* we could allow data that is usually stored and processed on cloud nodes to be also maintained (i.e., cached and processed) locally by edge nodes [69]. As long as the connection between edge and cloud works, they can synchronize and keep data consistent. If the connectivity breaks, the system could degrade gracefully and fall back to letting the edge service operate under restricted performance or functionality. Here, the consistency between edge and cloud could temporarily be sacrificed in favor of keeping the service available. For instance, parts of the application state can be buffered in the edge and updated to the cloud at a later time. Several current research works provide fault-tolerance and availability-enhancing solutions between edge and cloud [30, 69, 70, 147]. Apart from replication and redundancy, there are other resilience mechanisms that function across several

IoT layers, such as monitoring mechanisms. In general, the goal of monitoring is, in most cases, the observation and analysis of as much information as possible to gain as detailed insights into an infrastructure as possible. In terms of IoT infrastructures, this means that data collection is performed on all possible devices across all layers. The data aggregation and analysis process is then carried out, depending on the monitoring mechanism, on various layers. This distribution of mechanisms automatically enables fallback to local execution in case of failures. As an example, let us assume a smart city operates several parking garages all collecting and analyzing monitoring data. In the event of one garage losing connection to the central monitoring instance, it can still perform its local monitoring tasks, e.g., IDS within its infrastructure. Yet, incidents cannot be reported to other garages. Several current research efforts are focusing on distributing monitoring mechanisms from the cloud to the edge and device layers [24, 63].

We conclude that resilient IoT systems, and in particular the employed resilience mechanisms, should be designed in a way that they are agnostic of operating across layers, minimize component dependencies to adjacent layers and account for unpredictable faults occurring in components located in other layers.

IoT systems can have multiple, different administrative domains. For IoT systems, it is also important to distinguish between the technical and the administrative view of the system. For instance, an IoT system can consist of several administrative domains with disjunct administrators being in charge of specific IoT components and devices. According to Tsigkanos et al., one important challenge for IoT is *addressing mobility and distribution of software components between diverse administrative domains and locales* [140]. As an illustrative scenario, consider a smart city where available parking space is managed by an application that steers car drivers to a nearby parking garage with available space. This application might span several parking garages, and every garage runs its own local management services. Apart from that, there is a navigation service that maintains the global perspective of available parking space across the city and offers navigation services to drivers. This requires *coordination*, e.g., with a central cloud-based service provided and maintained by the city administration. Overall, these garages might be managed by different operators and administrators and the hierarchically superordinate coordination service also lies in some distinct administrative domain. Every administrative domain might have its own resilience requirements and the need to harden itself for failures of components located in other domains—a centralized coordination service might represent a single point of failure in the worst case. Tsigkanos et al. argue that in order to increase resilience in IoT, it is necessary for software components to coordinate in a decentralized way and be capable of self-adaption when disruptions occur at runtime [140]. Overall, multiple and diverse administrative domains induce different challenges that need to be addressed by technical solutions and also need to be considered when designing resilience mechanisms. For instance, in the context of *maintainability*, again the question arises how available objects in IoT can be efficiently coordinated and managed across different administrative domains [83]. In particular, diverse trust assumptions and identities may have to be considered and managed. Further, *privacy* concerns emerge, e.g., when data is forwarded to possibly unknown administrative domains where privacy-preserving policies are neither known nor enforced.

4 MEASURABILITY AND ADJUSTABILITY OF RESILIENCE

In this section, we first investigate the question of how resilience can be measured and quantified, i.e., what different types of *resilience metrics* exist and how they can be applied to practical IoT systems in Section 4.1. Closely related to the question of quantifying resilience, and arguing about qualitative properties of a system, is the question of how *adjustability* can be accomplished from the perspective of a system operator who wants to align his system to match specific resilience requirements. We discuss the latter question in Section 4.2.

4.1 Resilience Metrics

In the following, we identify several *different* approaches to *specifying or measuring* resilience. The reasons for this are mainly, that (i) a broad variety of resilience mechanisms exists which have different goals and limitations and (ii) mechanism-specific metrics are applied to individual resilience-enabling approaches or a family of resilience mechanisms to subsequently argue about, e.g., their cost or effectiveness. We argue that resilience itself can not be treated as a single aspect and measured using some specific function. Rather, resilience encompasses a broad variety of solutions, often domain specific, that pursue a common goal. Resilience can best be understood as a multi-dimensional property of a system and thus multiple different metrics can be applied for resilience mechanisms, too. The selection of these metrics also depends on the resilience properties, which a resilience mechanism preserves and which should be quantified.

Different goals mean employing different metrics. Overall, the goal is often to reason about the *effect* that a resilience mechanism has once applied to some system while using an underlying model to capture the granularity of fault types or malicious misbehavior that a system can then endure, e.g., the proportion of faulty components. For instance, if we were to argue about the *effectiveness* of a resilience mechanism that should preserve the reliability of a system, then we could employ the *mean time to failure* as a measure. However, system engineers are usually concerned with preserving several, if not all, resilience properties, which in turn, requires having a multi-dimensional view of resilience and thus employing a toolbox of different metrics. In the following we review general and established *concepts* for measuring resilience.

Coverage. Avižienis et al. define the *measure of effectiveness* of some specific fault-tolerance technique as the *coverage* or *fault-tolerance coverage* [13]. This concept encompasses (i) the *error- and fault-handling coverage*, which is a measure to capture how many of the actually occurring faults are detected and treated by the fault-tolerance mechanism since its capabilities to handle actual faults might be restricted, e.g., due to development faults, and (ii) the *fault-assumption coverage* which is a measure for congruence between model and reality. In particular faulty components can display behavior different from what is assumed by the fault-tolerance mechanisms (e.g., Byzantine instead of crash faults, correlated faults instead of independent ones). Hence, these faults would not be successfully treated. This measure indicates that the design of the fault-tolerance mechanism does not match reality fairly enough. An often made assumption in distributed systems and, in particular, replicated systems, is that individual components are assumed to fail independently. In practice, this needs to be regarded with caution as common bugs in these components can lead to correlated failures. N-version programming [10, 31] could approach common software bugs but is often viewed as impractical. However, reasonable models and reliability metrics [37] to assess the overall system's reliability for these approaches exist. For replicated systems, an interesting measure is the risk of having common weaknesses or vulnerabilities among replicas [50] and thus measure the configuration risk of a system which is useful to reason about the diversity of a state-machine replication (SMR) system [36, 50]

Degree of replication and fail-over time. A commonly used resilience mechanism that allows a system to tolerate crashing components is the *primary-backup approach* [26], providing a number of available and consistently updated backup replicas. In case the primary fails, a backup can replace the primary and continue service operation. Budhiraja et al. use the term *degree of replication* [26] mainly as a cost metric for counting redundancy, e.g., the concrete number of servers that are necessary to implement a resilience mechanism into a service. The authors also introduce a metric for limitations such as the *fail-over time*, which is defined as *worst case period during which requests can be lost because there is no primary* [26]. This is a metric that upper-bounds the effort (as measured in time) of the resilience mechanism to restore the functional state of the service and hence allows to give certain quality-of-service level guarantees to applications (or developers building applications). The degree of replication is also present in active replication protocols [118]. Replication protocols compensate the behavior or individual faulty components and usually specify and *denote the number of faulty replicas* that can be tolerated as f out of n total replicas. It is noteworthy that the expenses of replication then also depend on the concrete fault model which is employed. A fault model defines the assumptions with respect to which faults are considered in the system. For instance, in state-machine replication, the *Byzantine fault model* [84], which assumes arbitrary behavior of faulty components, usually requires the BFT protocol to employ at least $n = 3f + 1$ replicas, while it is only $2f + 1$ in a crash fault-tolerant (CFT) SMR protocol. If more than f replicas fail at once it will bring the system to a state in which it can not continue to operate any longer.

General metrics. Strigini et al. investigate general metrics and assessments of resilience in the field of engineering [130]. They list several metrics of *tolerable disturbances*, which capture different ambitions of quantifying the resilience of a system, such as (i) *how far the object of interest can be pushed without losing its ability to rebound or recover*, e.g., how many replica failures could be masked in some replication protocol, (ii) *how quickly it will rebound*, thus measuring the time needed to re-obtain the ability to deliver a certain service, also referred to as *failover time* [26] by Budhiraja, or (iii) *how closely its state after rebounding will resemble the state before the disturbance*, which is of particular interest if the resilience mechanism implements some sort of *service degradation*.

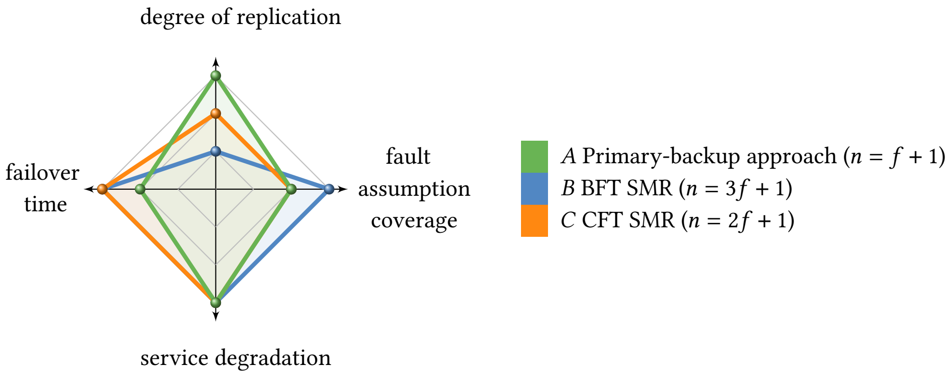

Fig. 5. Radar diagram comparing the resilience of replicated systems using different resilience measures. Increased distance from the center is better, e.g., a failover-time result is further from the center when it is shorter, a service-degradation result when it does not or only marginally degrade service functionality.

Note that even a high-level comparative statement on two systems in regard to resilience, such as "system A is more resilient than system B" is very problematic because of the multi-dimensionality of objectives of resilience. This calls for the application of different resilience metrics at once.

Example. Imagine we want to ensure the availability of some specific service using a replication technique, and have to choose among three different possible systems A, B, and C. The first system A uses the primary backup approach to tolerate f out of $f + 1$ faulty components. It assumes crash faults and experiences some fail-over time if the primary crashes, so there can be some period of time in which requests cannot be processed. Further, it does not degrade in service functionality. The second system B uses BFT SMR, employs $n = 3f + 1$ nodes to tolerate f faulty nodes, and can withstand Byzantine faults. A faulty node does not lead to a failover time, because requests are actively processed by all nodes and also service functionality does not degrade. The third system, C, also uses SMR but assumes only crash faults and thus only requires $2f + 1$ nodes. If we compare

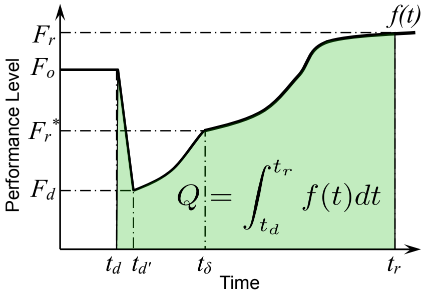

Fig. 6. A performance curve shows the relation between time and functionality during a disruption [142].

these systems using the different resilience metrics, *degree of replication*, *fault assumption coverage*, *service degradation*, and *failover time*, we will notice that none of the systems is pareto-optimal (see Figure 5). While A offers the lowest replication overhead and can thus tolerate the proportionally largest amount of faulty components (f out of $f + 1$), system B is optimal with respect to fault assumption coverage as it can withstand fault classes that neither A or C consider. Meanwhile, C lies in a sweet spot, where it is as good as B when it comes to failover time but it can tolerate a proportionally higher amount of faulty components. However, this comes at the price of making restrictions on what kind of faults can be considered. Now, if we want to argue about which of these systems offers the best resilience for some specific service, we need to make trade-off decisions, depending on which resilience metrics are the most important to our specific scenario and environment.

Quality of service and degrees of preservation. Bishop et al. mean by resilience that a system may degrade in its *quality of service* (QoS) under stress over time, but is able to recover [23]. Since resilience is associated with requirements of concrete services, the authors argue that QoS metrics can be defined and applied to quantify resilience in respect to an application. They explain that there are *degrees of preservation* [23] which are basically discrete levels that are used to define how much certain quality aspects of a system can be preserved, e.g., availability and confidentiality. These metrics are easier to define for availability than confidentiality, because we can use performance metrics. However, even for confidentiality and integrity, discrete approaches for quantifying resilience exist, e.g., a QoS measure for confidentiality could be the time needed to brute-force a key of the specific length that is used by the service. Data integrity can be at least specified binarily: whether it has been violated or not [23], although approaches for using integrity constraints probabilistically exist as well [76]. In recent works dealing with resilience in networked systems, resilience can be viewed and measured as the integral of maintaining critical functionality of the system, e.g., reflecting the state over time, such as the proportion of functioning system nodes or functionality that is still working [19, 49].

Absorption capacity, adaptive capacity, and recovery capacity. Vugrin et al. describe resilience more generally by defining three general concepts of a system as *resilience capacities*, namely the *absorptive capacity*, the *adaptive capacity*, and the *recovery capacity* [142]. The *absorptive capacity* is the degree to which the system can withstand (thus absorb) disruptions/impacts and prevent or minimize consequences to the system with little effort. It can be viewed as an inherent property of the system. While this is a very general concept, in the context of distributed systems, this resembles the category of *compensation* (or more generally, *error handling*). Further, the *adaptive capacity* means the degree to which a system can self-organize and recover its performance in response to an impact, e.g., by considering internal aspects that manifest over time after a disruption happened. This may require the system to take a lot of internal effort to change (adapt to a disruption) during a recovery period [142]. Lastly, the *recovery capacity* defines the ability of a system to be repaired easily. Figure 6 illustrates how these abstract concepts can be quantified: Let F_d be the system functionality in terms of performance that could be preserved after a disruption (which happens at time t_d in the diagram) and F_0 the performance level of the initial, unimpaired system. Then, the proportion of F_d/F_0 measures the absorptive capacity of the system while the proportion F_r/F_0 defines the *adaptive capacity* with F_r being the system's performance after it has fully recovered (thus adapted to the disruption).

Runtime metrics. Andersson et al. describe a set of runtime metrics that can be used to quantify system resilience [6]: (i) metrics that measure *the continuity of correct service*, e.g., the reliability of a system such as the mean time to failure, (ii) metrics that can quantify the readiness of a correct service to measure the availability (in particular maintainability) of the service, e.g., the ratio of time the system spends in acceptable states with respect to the total observation interval and (iii) metrics that measure the overall, accumulated quality Q of delivered service, which can be leveraged to assess the degradability of the system, e.g., in respect to some performance level which can be defined by the integral of observed performance quality over some time interval (see Figure 6).

Cost metrics: resilience has a price tag. Making a system resilient is often associated with mechanism-specific costs and limitations. Redundancy-based mechanisms that mask faults usually demand the provision of additional resources (computing components, space, time) and raise costs for system operators, thus making resource-efficiency an important design goal for resilience mechanisms [73]. In IoT systems, a new approach could consist of exploiting the natural redundancy of functionality across devices to compensate for failures while decreasing costs. For example, different devices in a smart home context can report similar events, e.g., a motion sensor and a video camera [137]. Further, an incorporated resilience mechanism may cause some overhead in the system (e.g., additional computational steps or messaging, use of cryptographic primitives) which can lead to decreased performance during fault-free execution. As an example, in replication protocols, the replicated service is usually benchmarked against the non-replicated service as a baseline to reason about the incurred performance overhead. Metrics that cover these aspects could be motivated by either an economic view (e.g., monetary costs that occur for a system operator) or QoS perspective (impairment of service performance due to resilience mechanism execution).

4.2 Adjustability

Given the observations from the previous section, matching specific resilience requirements can have different implications, e.g., with regard to providing additional hardware resources or the selection of resilience mechanisms that might be necessary for a system operator to align their system to satisfy these requirements. We summarize our findings as follows:

- Fault-model assumptions (e.g., the *fault-tolerance coverage* [13]) have an impact on the proportion of faulty components that can be tolerated in the system and thus define how much redundancy (i.e., the *degree of replication* [26]) is needed. For instance, more redundancy is needed to tolerate Byzantine faults compared to crash faults. This also means that, given the same number of provided components, we might be able to tolerate more faulty components under the crash-fault model than under the Byzantine fault model.
- Fault-model assumptions might not reflect reality accurately enough [13]. Failures that are assumed to occur independently might actually be correlated, such as when redundant components are attached to the same power supply. This may result in requiring more additional hardware resources than initially planned.

- Different views towards how a resilient system should behave might be derived from application-specific requirements. For these, *quality-of-service metrics* may help to define at which degrees of preservation [23] a system can operate when under disruption.
- Graceful degradation as a resilience mechanism is strongly related to the application and its domain. It is thus virtually impossible to adjust resilience in this regard without adapting the application itself.
- Adjusting security levels might be hard because security is a property that generally cannot easily be quantified. Still, it can make sense to enforce the use of specific cryptographic primitives or to use adequate key lengths [23].
- Resilience properties can be of either qualitative type (a certain quality of service level can be delivered or not, e.g., confidentiality: cryptographic primitives are in place or not) or quantitative (like availability where metrics like uptime or an upper bound of tolerable faulty components exist). This differentiation is not strict. Some “rather qualitative” attributes, like confidentiality or integrity, can be viewed quantitatively by measuring the difficulty of breaking underlying cryptographic primitives (e.g., measuring key length to reason about brute force difficulty). Typical quantitative properties like reliability and maintainability come with reasonable metrics that can help system operators to specify desired values.
- The specification and implementation of desired resilience properties might vary across different IoT architectural layers. For instance, confidentiality requirements might differ between clouds, where demanding AES-256 encryption might be a reasonable choice, and sensor landscape where the use of lightweight block ciphers (with smaller key length specified) is a good trade-off to incorporate security for resource-constrained devices with limited power.

5 RESILIENCE MECHANISMS AND THEIR APPLICATION REQUIREMENTS

In this section, we analyze and discuss concrete resilience mechanisms from a more practical, implementation-level view, e.g., in respect to the constraints they impose on the application. This discussion includes a broader range of redundancy mechanisms (Section 5.1), monitoring mechanisms (Section 5.2), protection mechanisms (Section 5.3), and recovery mechanisms (Section 5.4).

When it comes to resilience in IoT services, we also distinguish different roles. While *users* access the functionality of IoT services and demand their resilience, *administrators* maintain the system. They rely on the autonomous functioning of resilience mechanisms within the system, e.g., when it comes to monitoring components, compensating faulty components, or initiating the recovery of components to preserve system functionality. On the other hand, *developers* create concrete IoT applications (or application components) and while doing so, might need to make some considerations at design-time to support their resilience.

Assembling and incorporating a set of resilience mechanisms into an IoT platform leads to a design challenge: Every resilience mechanism may have its own *constraints*, such as interfaces that need to be implemented by the application developer, and also *constraints*, e.g., regarding the execution of the application or its programming model. Ideally, a configurable and resilient IoT service execution platform should ascertain the availability of specific interfaces, as far as they are provided by an application, and automatically derive which resilience mechanisms can be applied. The constraints specific to a resilience mechanism may complicate (i) the design of an execution platform that can leverage all these constraints, as well as (ii) the porting of existing IoT applications to match this platform. In this section, we aim to conduct a survey to discuss the constraints of a selection of resilience mechanisms and present an overview of our findings in Table 1.

Resilience Mechanisms	*Resilience Goal*	*Fault Model*	*Constraints for the Application*

redundancy mechanisms			
auto-scaling	adapt capacity to workload	overload	application needs to provide metrics as decision basis, architecture needs to be scalable
state-machine replication	tolerate up to f faulty replicas out of n replicas ($n > 2f$ for crash faults)	Byzantine faults, crash faults	deterministic service execution, interfaces for state transfer, execution and client-server interaction model
primary-backup replication	tolerate up to f faulty replicas out of n replicas ($n > f$ for crash faults)	crash faults	support for state updates and request logging
partition-tolerant redundancy	tolerate connectivity loss to cloud, preserve functionality in edge	network connectivity loss between edge and cloud	interfaces to define data to be cached in edge, fallback functions
data redundancy	tolerate faulty nodes	up to f faulty nodes	has to deal with inconsistencies
redundant network links	tolerate failure of network links	up to f broken links	— (often handled by underlying protocols or middleware)
mechanisms related to monitoring			
monitoring	general surveillance and anomaly detection of component behavior	malicious intrusions and faults	component provides interface to report its own health status to the outside world
intrusion detection systems	detection of intrusions for mitigation purposes	malicious intrusions	application provides support for plausibility checks on data
introspection	monitor a virtual machine or container state dynamically from outside, assert specific policies	malicious alterations	application needs to run in some VM or container
honeypots	analysing attacking patterns and strategies; distracting attackers	malicious intruders	if honeypots for components (e.g., sensors) should be automatically deployable, application needs to provide a description for execution platform
plausibility checks	detection of implausible sensor data, hence, prevention of failures in service execution	malicious modifications	either integrated in some IDS (anomaly detection) or handled by application itself (application-specific)
protection mechanisms			
encryption	protect the confidentiality of data in transit at rest or while processing	malicious eavesdroppers	key and identity management, support for standard protocols like TLS
signatures	protect the integrity of data	malicious modifications	key and identity management, support for standard protocols like TLS
verification	detect and remove software bugs. assert correctness of applications	development faults	e.g. creating a model for model-checking an application (along with its interactions)
privacy filters	protect the privacy of users	unwanted leakage of personal information	domain-specific knowledge on which and how data needs to be masked or aggregated
identity management, authentication and authorization services	protect against forging or spoofing identities and/or attackers trying to access resources without permission	Sybil / spoofing attackers, unauthorized resource consumption	— (often handled by underlying protocols or middleware)
sensor fusion	prevention of distribution of faulty sensor data	malicious intruders	handled by application itself due to application-specific knowledge requirements

recovery mechanisms			
rollback	periodically perform checkpoints and if necessary, e.g., restart some component and restore state from checkpoint	unpredictable errors occurring in the system	provide interface for checkpointing and for restoring application state from checkpoints
rollforward	periodically perform checkpoints and upon error detection perform fault diagnosis *concurrently*; trans-from a faulty component's state into a new state without errors	unpredictable errors occurring in the system	provide interface for checkpointing and for restoring application state from checkpoints
graceful degradation	continue service delivery on best efforts basis, e.g., under a restricted service level	faults that can not be fully tolerated and need to be circumvented	define QoS requirements, specify service levels and application-specific behavior as well as transitions between service levels
graceful upgrade	recover original service level after graceful degradation		

Table 1. Overview over application requirements of resilience mechanisms.

5.1 Redundancy Mechanisms

To make systems more resilient by design, we can anticipate the presence of faults within the system and employ redundancy among hardware, software, or data to overcome possible faults. *Redundancy mechanisms* typically try to mask a specific proportion of faults which are then tolerated and do not lead to a failure. Thus, these mechanisms are a type of *fault tolerance*, in particular *compensation* [13] or sometimes referred to by the term *absorption* [142]. Further, redundancy mechanisms can be classified depending on how they work e.g., what is being provided (and performed) redundantly such as processing, data storage, or networking.

Auto-scaling. Scalability describes the capability of a software system to address a growing workload by using more resources. Accordingly, horizontal scalability denotes the capability of a distributed application to increase its performance by increasing the number of resource units (such as physical servers or virtual machines) available to the application [121]. Applications that support *Elastic Scaling* support the action of *scaling* (out or in) while the application is running, hence without downtime and ideally without impacting application performance [121]. Finally, *auto-scaling* provides means to automatically adapt the scale-out factor of an elastically scalable application depending on current or even predicted workload.

According to [111] supporting auto-scaling requires workload monitoring, the existence of an application performance model, a controller matching actual (or predicted) workload against the model, and deciding on the application's scale-out factor. Moreover, an elastic infrastructure is required to dynamically acquire and release resource units as well as internal or external mechanisms to balance the workload among available resource units.

Under these circumstances, auto-scaling can be used to increase the resilience of an application by adapting capacity to workload and hence, mitigating possible unavailability due to overload situations. On the downside, naive auto-scaling makes an application vulnerable to denial of service attacks that provoke unnecessary scaling actions and leads to high costs. Auto-scaling has been an active area of research in the cloud era and many different approaches exist focusing on heuristics [15, 64] and language-level [2]. The latter approach is also used by auto-scaling features offered by Amazon's AutoScaling[2] and Kubernetes[3].

[2] https://aws.amazon.com/de/autoscaling/

[3] https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

State-machine Replication. *State-machine Replication (SMR)* [118], also known as active replication, is an approach for tolerating faults by abstracting the service functionality (i.e., the processing of client requests) as a deterministic *state machine* which is then replicated on multiple independent server replicas. These replicas maintain a *consistent state* by applying the same sequence of state transitions. In practice, this can be achieved by letting replicas deterministically process client requests in the same order. The requests can be ordered by utilizing a form of *totally-ordered or atomic broadcast* between clients and the replica group. Typically for resilience, such a broadcast mechanism is implemented by a multi-consensus protocol, whereas in recent years the effectively same functionality has also been provided by blockchains [143]. Whereas the broadcast can be easily hidden from the application developer, the deterministic execution cannot. If there are means to convert code unaware of determinism into deterministic code then developers still have to avoid certain operations, library functions, or language concepts as they cannot be converted into a deterministic variant. An example of such means is DJ, a deterministic run-time for Java [41]. However, mostly such means are not widely available or employed. Examples for non-deterministic application aspects include *multi-threading*, *random number generation* or any sources of randomness, *time measurements*, *external calls*, or any kind of interaction with some external service. Further, some sources of non-determinism might appear unintuitive to the developer, e.g., when it comes to the usage of internal libraries. For instance, when iterating over the keys of a Java `HashMap`, there is no explicit guarantee given on a deterministic order. Client-replica interactions, like broadcasting a request to the replicas or having the client wait for a specific number of matching responses are also automatically coordinated by the SMR protocol [118]. Practical Byzantine Fault-Tolerance (PBFT) [27] is a prominent example. A more recent, still maintained, and feature-rich replication library is BFT-SMaRt [21]. In BFT-SMaRt, the replica set is dynamically scalable which is helpful, e.g., to add more replicas at run-time or also to remove replicas if desired (e.g., to re-integrate replicas after completed maintenance).

A modular architecture helps to separate the library into exchangeable components for different concerns, e.g., *state transfer*, *reconfiguration*, or *consensus*. The library provides interfaces that can be used by application developers to implement execution behavior, customize state transfer or employ specific client-server interaction models such as publish-subscribe.

Passive Replication. In passive replication, also called *primary-backup* approach [26], a single server or system, the *primary*, is responsible for processing client requests and distributes *state updates* to the backups. Only when the primary fails, a failover occurs, and one of the backup replicas takes over as the new primary. In order to be able to take over, the new primary has to start from the latest known state and re-execute requests that have occurred since then. Thus, these requests have to be retained in some way, e.g., by logging to stable storage or by broadcasting to all servers. The longer the update interval, the more involved the recovery becomes. Additionally, clients using the system have to be made aware of a changing primary, which becomes rather more difficult in the context of IoT, where e.g., resource-constrained devices in the field need to switch to a different edge device. Hokeun et al. present a solution to this problem of migrating IoT devices to fail-over backup systems [77]. Very short update intervals need more communication bandwidth than SMR and are usually avoided. A typical misconception is that passive replication does not require deterministic execution. If there is multi-threading in the primary or at least some of the request processing produces side effects in other services, the execution must be deterministic, too. Additionally, the replication system has to retrieve the application state for updates. Sometimes, this is not easily possible without involvement of the application developer, as (i) due to multithreading a consistent state has to be coordinated with request processing, or (ii) not all state is actually relevant for updates. In these cases, state has to be compiled by the application. In order to save communication costs, the replication system may then compute incremental or compressed state updates by preserving previous versions. More general limitations of this technique are, that—in contrast to state-machine replication—passive replication does not support the Byzantine fault model, and the response time of the passively replicated system to a client request is much higher during recovery time after a primary failure.

A recent example of this is shown in Ambrosia [55], which utilizes checkpointing and application logging to resume execution on fail-over secondary servers when a primary fails. Fail-over time, however, was measured to be roughly 3 orders of magnitudes higher than normal case latencies. CESSNA [59] presents a solution to the problem of proper checkpointing, request logging, and replay in a distributed edge computing scenario where information about the order and status of requests are dispersed in the edge, as is often the case in IoT systems.

Partition-tolerant Data Redundancy. By partition-tolerant redundancy we refer to techniques allowing storing data redundantly so that in case of partitions every partition can still access the data. This would be helpful in an IoT system where the edge gets disconnected from cloud services. An edge service could redundantly maintain some data that is usually stored in the cloud. Edge services synchronize the data with the cloud on best effort. In principle, this is very similar to caching because the goal is to benefit from proximity to data for faster access. The availability of the edge service increases by preserving functionality of the edge even if the connection to the cloud is disrupted. If connectivity between edge and cloud is broken, services running in the edge can still process (important) events by working on their local data. Mechanisms that address partition-tolerant data redundancy often work inside architectures where data is being viewed either as a stream or as a state. According to the CAP theorem, however, such mechanisms have to sacrifice consistency [25].

In a *data-as-state* architecture, as soon as data is changed in both partitions, e.g., in edge and cloud, inconsistencies occur and have to be reconciled. Reconciliation can be application-specific and, in the worst case, even need user interaction. The usage of CRDTs [122] (conflict-free replicated data types) can allow for automatic reconciliation. In this approach, replicas employ the eventually strong consistency model where all replicas eventually converge when no further updates are issued. In IoT systems, CRDTs can be used to provide a reliable storage solution at the edge [81].

In *data-as-a-stream* architectures, ideally data flows from producers to consumers along individual processing stages which form a pipeline. Such a pipeline can be distributed (and moved) between edge and cloud by deciding where to place these processing components while data is streamed through the pipeline and buffered at each system site. A recent approach is presented by CEFIoT [69], which describes a crash fault-tolerant architecture that allows placing processing components in both edge and cloud. CEFIoT uses a data transport layer that employs a pub/sub messaging framework (Apache Kafka), so data streams can be buffered and replicated across a cluster. Using Apache Kafka, the data pipeline maintains network fault tolerance by allowing data buffering locally at the edge during periods in which Internet connectivity is being temporarily disrupted. Multiple pub/sub topics are used to buffer data streams, hence making data available for the cluster even during a reconfiguration of individual processing stages. Data is buffered in topics, which are also replicated to other Kafka instances in case some instance becomes unresponsive.

Redundant Network Links. Redundancy can also be applied when it comes to networking. While there are protocol-layer–specific solutions to establish reliability over unreliable network links (such as cyclic redundancy codes or re-transmission of lost packages), the failure of individual network links or network devices can also be tolerated by employing redundancy among these resources. For instance, Hasan et al. propose a method for fault-tolerant routing in the IoT between a larger number of interconnected devices, which is implemented by constructing, recovering, and selecting k-disjoint multipath routes that guarantee connectivity even after the failure of up to $k-1$ paths [60]. *Frontier* [104], which is a distributed and resilient edge processing platform for IoT devices, uses network path diversity and selective network-aware replay to recover from transient network failures. Redundancy-based models have also been developed and implemented [125], which allow assessing reliability aspects. Since applications are usually not aware of how reliability in networking is handled, there are no concrete constraints for the application developers.

5.2 Monitoring and Analysis Mechanisms

Monitoring and analysis encompass a variety of different mechanisms usable for the detection of changes and harmful conditions of a system. These mechanisms can be divided into different groups like performance, security, and system status, whereas performance monitoring is primarily used for detecting changes in hardware-related performance metrics, e.g., detecting insufficient hardware components that may lead to failures in the availability of services. Security monitoring mechanisms usually focus on actively detecting attacks, such as intrusions, based on different detection strategies [153]. Furthermore, monitoring and analysis mechanisms can be divided into different phases. These encompass monitoring, which is the actual collection and storage of data, aggregation, which encloses filtering and aggregation of data, and alerting, which analyzes the data and triggers alerts if necessary. In general, monitoring systems provide the essential functionality of detecting changes and harmful conditions enabling timely manual or automated intervention.

System Monitoring. System monitoring focuses on retrieving and analyzing system status information. Thereby, the monitored objects reach from single components to complete infrastructures. Status information is retrieved using various tools, e.g., by utilizing the SNMP protocol or custom system information. Their analysis can be rather diverse, reaching from rule-based systems where some thresholds are manually defined (e.g., “storage medium below 10 Gb of free space”) to more sophisticated ones like anomaly detection using machine learning techniques. Due to the focus on system status information these monitoring mechanisms do not impose any constraints on the application. However, the runtime environment, operating systems, or physical components have to provide such system information. As an example, DARPA [63] proposes the integration of cheap, reliable read-only hardware clocks into IoT devices, to enable a secure heartbeat protocol for monitoring an IoT network, in order to mitigate attacks on individual devices.

Intrusion Detection Systems. Intrusion detection systems (IDS) are a widely evaluated field of research with early efforts dating back to the 1980s [5]. IDS are designed to detect malicious actions, called intrusions, within an infrastructure, and to report those either to a human or to other services capable of taking appropriate action, hence, ensuring resilience [153]. In order to detect such intrusions there exist a multitude of different approaches starting with the simplest one, called specification based, where rules of a specific behavior of a system are manually specified and every violation of one of those rules is categorized as an intrusion. This method is very time-consuming and error-prone, since missing a rule or changing the system’s behavior always requires human input. A second approach is to monitor if the behavior of a system matches an attack signature previously recorded and stored in a database. Effort-wise this approach is less time demanding than the previous, since the signature only has to be captured once. However, there is a delayed response time when the system is confronted with new attacks. Anomaly based systems are coping with this challenge by first observing the normal behavior of the monitored system and afterwards detecting deviations of this normal behavior which are then classified as intrusions. The main advantage of this technique is its self-adaption to changing normal behavior and the automatic detection of new attacks. As an example, Parker et al. [107] implemented an anomaly detection strategy utilizing deep extraction and mutual information selection, which increases the interpretability of machine learning models and is optimized for resource-constrained and dynamic IoT environments.

However, the IoT with its different infrastructure in comparison to traditional ones, holds new challenges for IDSs. On the one hand, IoT devices are usually restrained regarding their computational power and with that partly incapable of executing IDS services. On the other hand, there is a multitude of heterogeneous devices, gateways, etc. increasing the number of, e.g., placement strategies of IDS systems and the variety of used communication protocols [153]. There are currently several research studies to improve the status quo in this field [3, 8, 24, 113], like Breitenbacher et al. [24], which present a host-based IDS with low performance overhead, thereby extending the deployment capabilities of IDS on IoT devices. The authors utilize a whitelist approach to identify malicious processes running on the IoT devices themselves. Other mechanisms include a behavior monitor for blockchains running on IoT devices classifying their behavior regarding maliciousness [3]. A further category of the attack surface prominent in IoT infrastructures is physical attacks, which open up new strategies for the detection of such attacks. An example in this category would once again be the work of Ibrahim et al. [63], which proposed a distributed heartbeat protocol capable of detecting the absence of devices. In this case, the protocol assumes an attack and initiates mitigation strategies. Other approaches observing the physical domain of IoT devices are proposed by Yasaei et al. [150], which detect anomalies in sensor data and, based on that, predict possible malicious actions. However, many questions remain to be answered [153].

Introspection. IDS systems have often been placed directly in a host, e.g., using host-based agents for monitoring, or they have been placed on routers to monitor the network traffic. However, network-based IDS can not capture details about what exactly is going on within a host. Introspection techniques monitor virtual machines or containers at *run-time and from the outside*. For instance, *virtual machine introspection* (VMI) is a technique that can be employed as the basis to build an intrusion detection architecture on top of it [51]. This gives some distinct advantages: The IDS can achieve an excellent view of the host system while being isolated and thus more protected from manipulations than in-host agents (if the host becomes compromised). In fact, the monitored virtual machine is not even aware that it is being monitored. Within the context of IoT systems, the applicability of VMI is a bit narrowed: it is particularly applicable in the cloud layer of an IoT infrastructure. There is a range of already existing VMI tools [68] that are suitable for monitoring a VM for possible intrusions, such as the detection of root kits. VMI works application-agnostic and does not impose concrete constraints on an application (except running in a VM environment).

Honeypots. Honeypots aim to trick attackers into thinking they are interacting with a real system instead of a purposely set up mockup. The purpose is often to either study the attacking behavior and patterns, or to distract attackers from attacking real systems. Recent research work [42, 57, 92, 106] discusses how the use of honeypots in the IoT landscape (e.g., mimicking IoT devices) can identify novel IoT-related threats and attack patterns. As an example, Tanabe et al. [134] identified the common IoT botnet infrastructure and its behavior. Their results showed that IoT botnets, while being less sophisticated in their architecture compared to traditional infrastructure botnets, are disposable, meaning they are only used once. This dynamic nature allows them to be resistant against countermeasures like blacklisting. To integrate honeypots into an IoT execution environment and deploy those automatically, the concept of shadow honeypots is very promising. Shadow honeypots exist alongside "real" IoT components/applications and mock these by instantiating the same component/application in a secured environment. Afterwards, intruders either actively access those shadow honeypots or get redirected to them once they are found out [4].

Plausibility Checks. Plausibility checks are a very important instrument in detecting faulty and even malicious components [82, 116]. Here, the reported values of, e.g., a monitoring component are compared to either the known environment boundaries of the monitored service or values of other monitoring components or both, to check whether the reported values are plausible. For instance, based on knowledge we can tell that a temperature sensor located in a living area reporting 200°C is most likely wrong. Apart from this simple example there are also more sophisticated methods, like machine learning value prediction and anomaly detection, ensuring the plausibility of values. In respect to IoT built around the concept of sensors measuring data these checks can be an essential part of ensuring resilience. Checks can be performed on different architectural layers, such as cloud or edge layer, depending on the quality of data and requirements of application-specific information. However, many of these checks can only be realized with application-specific knowledge, meaning that only the application logic itself can decide whether a given value is plausible or not. Hence, the application developer needs to implement this resilience mechanism and, simultaneously, appropriate interfaces for reporting the processed data.

5.3 Protection Mechanisms

A variety of mechanisms are especially designed to shield a system from external and possibly malicious harm, to prevent (malicious) faults from occurring. We call these *protection mechanisms.*

Encryption. Encryption is a security mechanism that aims at ensuring the confidentiality of data while in transit, at rest, or in processing to avoid that data is disclosed to some unauthorized entity, e.g., a malicious eavesdropper. As IoT systems consist of heterogeneous device classes that come with different computational resources, a variety of cryptographic primitives and protocols are utilized to match application-specific requirements. Lightweight cryptographic primitives [126, 149] should be considered to be used with *power-constrained* devices, e.g., lightweight block ciphers might come with smaller block and key sizes than commonly employed. As for asymmetric cryptography, elliptic curve cryptography (ECC) [94] is a better fit than, e.g., the RSA [115] algorithm, because RSA employs rather large key sizes. For encryption schemes, the overall goal in IoT is finding a practical trade-off: ensure a reasonable degree of security with the least amount of overhead while saving computational power, battery life, and bandwidth since they are sparse resources. As an example, Ferretti et al. [47] introduced a symmetric proxy re-encryption scheme ensuring encryption among IoT devices and edge computing instances even in case of a failed cloud connection. Another approach is presented by Shi et al. [123], which introduces a new ultra-light weight encryption scheme capable of protecting confidentiality of data of white-box attack scenarios. They implemented this scheme so that the main memory requirement decreases to a fraction of comparable approaches, enabling its utilization within the IoT. Similarly, Gu and Potkonjak [56] presented a secure multistage physically unclonable function (PUF) design which allows for key distribution, key storage and rekeying for purposes of IoT device authentication with ultra low power consumption characteristics. From an application-level view, the process of encryption is mostly transparent, so that encryption schemes can be easily implemented and applied on a broad variety of devices. However key management can become a challenge in larger sized systems.

Signatures. Generally, signatures can guarantee essential security properties, in particular, integrity, data origin authentication, and non-repudiation. They can be applied e.g., to verify data and application integrity. Often, signatures are used as authentication devices on messages. In the field of IoT, a security goal could be to protect the integrity of sensor data. It has been shown that integrity protection, e.g., with the use of ECC signatures is in principle possible nowadays even on constrained devices but with some restrictions [16, 97, 108].

Verification. Verification has the ambition of guaranteeing that an application satisfies some specified properties, e.g., by generating correctness proofs or finding counterexamples. This allows the detection and removal of internal development faults within the system during its design phase. In the IoT, software bugs can have implications on the safety of systems as unintended interactions within the physical spaces may lead to unsafe or insecure environments [29], which motivates code verification. Verification efforts have contributed to building secure, safe, and dependable systems: for instance, model checking based approaches [28, 102] exist that allow for detecting interaction-level flaws between IoT components possibly causing unsafe and dangerous physical states and can also help to validate safety properties. Other approaches suggest the use of code-level analyses by symbolic execution [132] in contrast to model checking for its faster analysis, better detection rate, and fewer false positives. In the same vein, approaches like CASCA [39] aim to improve the process of designing IoT hardware by creating domain-specific hardware modeling languages which can reason about e.g., side-channel attack-resilience in the circuit design phase of IoT silicon. Gatouillat and Badr suggest using linear logic to automatically compose applications of components following their smart object model [52]. Particularly, a smart object formally describes an artifact structure and behavior. Also, a learning approach for model-based testing of communication in the IoT, such as implementations of different MQTT brokers [136] has been proposed, which can identify differences in various implementations that adhere to the same MQTT specification (hence indicating possible bugs), using models that are automatically learned from the implementations.

Privacy Filters & Privacy Preserving Techniques. Due to the nature of IoT and cyber-physical systems, a lot of sensitive data of users (e.g., related to location, health, or daily activities) can be generated and aggregated, forwarded, and processed between IoT architecture layers. This raises the demand for techniques to preserve privacy, e.g., by enforcing specific policies to protect the privacy of users of IoT applications (which can also be implemented in a network-centric approach [127]). An approach could implement local differential privacy obfuscation to leverage IoT data analytics at the edge, so that data is aggregated and distilled without disclosing users' sensitive data before sending it to the cloud [148]. Other approaches ensure privacy by applying blockchains and using smart contracts to control and enforce connection rights [91].

Identity Management, Authentication and Authorization Services. The scale, growth and heterogeneity of IoT systems also impact the manageability of identities leading to novel challenges when designing authentication and authorization services. Those new challenges encompass not only the external attackers on the cloud and edge layer, but also on the sensor layer, e.g., by placing malicious sensors among the regular ones with the goal of connecting them to the infrastructure. Since IoT extends into the physical world, system components might be deployed in adverse environments. Generally, the goal of identity management is to protect the system against an attacker forging identities (spoofing attacks) or spawning fake identities (Sybil attacks). Identities are often used with authentication and authorization services to prevent an attacker from accessing information or consuming resources unauthorized. OnboardICNg is an example of a protocol that prevents malicious IoT devices from joining a trusted information-centric network (ICN), and prevents mislead honest devices to become part of a malicious ICN [34]. Because IoT devices may collect sensitive data or impact the safety of their environment, a reasonable choice is to realize the principle of least privilege. Recent research work has shown that embedding and process awareness of IoT devices impose a natural sequencing of accesses, which can be used to enforce history-based access control policies [135]. Further, identity management as well as authentication and authorization services are often implemented by middleware and protocols below the application level, so that they can be used with little interference to the application logic and do not have to be considered as a distinct problem by an application developer [61]. The IoT introduces new challenges: the availability of authentication and authorization services deployed on edge computers needs to be preserved even under, e.g., network failures or attacks. Recent research work proposes a resilient authentication and authorization framework for IoT which allows an IoT device to securely migrate to another trusted edge computer if its own local authorization service becomes unavailable (e.g., due to a Denial-of-Service attack) [78]. A further approach relies on controlling the flow of information for event-based IoT systems at the brokers to restrict which devices may communicate with each other [48]. The consideration of *home-limited channels* [71] that can be accessed only within a house yet remain inaccessible for attackers outside the house also provides an interesting application for authenticating devices in a smart home [71].

Lastly, securely pairing new IoT devices can be a problem in many public IoT systems – such as in hospitals, factories or laboratories – where a multitude of different devices provide critical services and share private data in unsafe environments with potentially many close-by attackers. This, as well as the lack of sophisticated input options on many small IoT devices, can preclude the usage of usual pairing techniques based on, e.g., proximity or entering pairing codes. The works of Li et. al present solutions for these scenarios ([88], [89]). They utilize sensing and matching of semi-random user input sequences like petting a device or using universal input sensors like buttons or knobs to securely pair devices in the presence of multiple malicious attackers.

Sensor Fusion. Sensor Fusion describes the mechanism of fusing the values of different measurement points (sensors) in order to get results of higher quality or better plausibility [137]. There are various methods for reaching these goals, starting with simple aggregation of data from homogeneous sensors and calculating the median value, for example. A more elaborate approach is to aggregate data from heterogeneous sources to get a better impression of an environment, hence preventing possible faults on one sensor by comparing to the environment. Another idea is to interpolate or predict possible values of single sensors by using information of surrounding sensors, thus checking future values for correctness. Linked to these approaches, there are also sensor data quality analyses trying to quantify the quality of received data [82]. This mechanism, however, is highly application specific, meaning that the application developer is in charge of implementing it.

5.4 Recovery Mechanisms

Another pillar of resilient system design is anticipating faults and errors that occur, and making the system capable of detecting and recovering from them. *Recovery mechanisms* aim to steer a system back to its intended, functional state. Unlike redundancy mechanisms which mask present errors and faults, recovery transforms a system state that contains errors and faults into a new system state in which errors do not exist and faults will preferably not activate again [13].

Recovery: Checkpointing, Rollbacks, and Roll-forwards. Since unpredictable errors can occur within system components, it might be reasonable to anticipate this and provide resilience mechanisms that, once a failure is detected, can restart a component and recover its application state to continue execution. A basic approach is to let applications periodically create checkpoints [80, 139] which might require the provision of interfaces for acquiring and persisting state and for restoring the application from that state. The latter is typically called a rollback. In rarer cases, e.g., with some sort of transactions involved, application state could also be rolled forward to a new and error-free state. One example is the completion of NTFS transactions according to log entries belonging to a particular transaction [101]. Depending on the application, checkpoints can become large. Solutions to create smaller entities are incremental checkpointing or the deployment of techniques like event sourcing, which record state changes at a fine granularity level. Ambrosia [55] is a recent example of a system that provides resilience by logging requests (and periodically performing checkpoints) to replicated storage prior to execution and guaranteeing correct state reconstruction during replay-based recovery. Its key features include high performance, support for non-determinism as well as support for language and machine heterogeneity, which also makes it a suitable choice for IoT applications. In IoT systems, applications might be deployed across heterogeneous devices which also involve resource-constrained devices. A recent approach works at design time to automate an optimized rollback-recovery for constrained scenarios by employing a co-design of firmware, runtime and compiler transformations to create multiple resilient variants of an application which can reduce the checkpointing overhead [14].

Compared to rollbacks, roll-forwards also purge detected errors from the system state but without retrying from the last correct checkpoint. For instance, in a modular redundant system (e.g., Duplex system), a roll-forward lets the execution of tasks continue while the fault diagnosis and recovery functions are performed concurrently (by an external spare component). In particular, on both the correct and the faulty component the task execution continues *forward*, beyond the last checkpoint where disagreement occurred [109]. After identifying which component is faulty by employing the spare component parallel to continuing task execution on both of the other components, the state of the correct component can be copied to the faulty one, hence transforming its erroneous state to a state without errors, from which it can continue execution.

In event-driven IoT applications, an approach for data repair consists of recording events and replaying computations that depend on corrupted data. *SANS-SOUCI* [90] is a recent example that combines a functions-as-a-service architecture for the event-driven system, append-only data structures to persist application state durably, and distributed, causal dependency tracking for efficient replay of dependent computations across multi-tiered (device, edge, cloud) IoT deployments.

Graceful Degradation and Upgrade. *Graceful degradation* means that an IoT system, in case of faults, does not entirely fail but tries to uphold as much functionality as possible. Graceful degradation is a recovery mechanism that enters the scene when faults can no longer be tolerated by other resilience mechanisms. For instance, the loss of connectivity between cloud and edge may leave the edge without access to resource-consuming and enabling services provided by the cloud layer. In this case, a degradation mechanism could switch to less resource-consuming alternatives in the edge by sacrificing some of the original quality, e.g., precision, accuracy, efficiency, throughput, or consistency. The degraded service could even abstain from executing certain tasks and providing specific functionality at all. Thus, the degradation can affect *function*, *quality*, *performance*, or any combination thereof. Graceful degradation approaches are well known for the design and operation of distributed embedded systems [54, 79, 100], but have recently also been introduced for IoT scenarios, e.g., for a video surveillance application [30], a smart-office case study [145], and a drone application [152]. Degradation often depends on defining a set of different *service levels* that determine the graduations of quality of service (QoS) a system can operate at, where different levels might require a different set of functioning IoT components. Further, degradation as a resilience mechanism is application specific, as only the application domain can define service levels and interpret them by deploying different implementation alternatives.

IoT is not restricted to a certain application domain as it can reach from private home environments to even globally distributed networks. Also, degradation decisions depend on faults that no longer can be tolerated by other resilience mechanisms. Thus, there is a need to combine system support with application-specific degradation mechanisms. Application developers should be guided on how to define service levels and how to integrate degradation with system-provided support mechanisms, e.g., fault-tolerance means, monitoring of failure events, etc. One approach is to use a product-family approach to define service levels [99]. Besides, the switch between service levels should also be supported by framework or system functions in order to reduce tangled application-, service-level- and system-specific code fragments in IoT applications.

A degraded system may automatically switch back to the originally desired service level, or at least to a better level, if the cause of the degradation has disappeared or changed. This procedure is called *graceful upgrade.* Ideally, an automatic upgrade procedure exists, e.g., the system monitors the availability and functionality of needed components and re-decides about its next service level. If not, a system will need manual reconfiguration to redeploy at the original service level.

5.5 Combined Mechanisms

So far, we have analyzed and discussed a broad range of resilience mechanisms. In practice, these mechanisms can and will be combined in one way or the other. For instance, several of the mechanisms listed inherently depend on other mechanisms. Further, the architectural characteristics of IoT systems imply that different mechanisms should be used on different layers of the architecture.

Inherently combined mechanisms. Several mechanisms rely on other mechanisms on a lower level. First and foremost self-monitoring plays a crucial role for many mechanisms. The fact that the system has some degree of self-awareness allows it to detect when to apply countermeasures. Similarly, monitoring is necessary to identify bottlenecks in the system, detect failures and attacks, and thus creates a decision basis for other mechanisms to activate. Recovery is a fundamental building block enabling individual IoT components to join back into the system. It is therefore the basis of a more general and automated procedure of handling failures. For instance, primary-backup replication applies rollback or roll-forward when a backup replica takes over the primary role. Finally, replication protocols aiming at Byzantine fault tolerance usually apply signatures, while identity management, authentication, and authorization services make use of encryption.

Combinations for enhanced resilience. Mechanisms can also be combined to enhance the overall resilience. This includes for instance a mutual support of IDS and honeypots as presented by Baykara et al. [17] and anomaly detection mechanisms integrated with shadow honeypots [4] both aiming at the reduction of false positives. The first approach focuses on zero-day attack detection of signature-based IDS, where honeypots enable the autonomous signature generation of such attacks. Otherwise those types of IDS would not be able to detect zero-day attacks. The second combination enhances the detection accuracy of the single mechanisms, by having the honeypot verifying positive tagged network traffic by the anomaly detection mechanism.

Moreover, a broad variety of mechanisms should be considered at design time to help withstanding and absorbing changes. These include code validation mechanisms, but also protection mechanisms like cryptographic primitives. The latter help shielding a system from attacks while redundancy mechanisms can be used to compensate for a specific number of faulty components in the system.

Control-loops and self-healing. The paradigm of self-healing has been designed in the early 2000s [74] and are representative of systems utilizing different mechanisms to achieve reconfiguration with the aim of satisfying their requirements in a changed environment or possibly degrade gracefully. These mechanisms always include some form of continuous monitoring, essential for detecting unforeseen changes that might hinder the system's resilience, in combination with mechanisms capable of executing appropriate counteractions that can maintain the requirement satisfaction, e.g., graceful degradation. The general procedure can be described by a MAPE loop [74, 140]: Constant (M)onitoring of the environment and reflecting changes in a model, (A)nalyzing the model for requirement violations, (P)lanning appropriate countermeasures, and finally (E)xecuting those and updating the model. A MAPE-K loop enhances MAPE with a knowledge base [9, 117], while MAPE-SAC introduces security aspects in the control loop [67]. Auto-scaling approaches are usually realized via a MAPE(-K) loop.

Tsigkanos et al. see self-adaptation capabilities as a core feature of future autonomous resilient IoT systems [140]. Muccini et al. stress that IoT systems may require more than one control loop and that different control loops may be overlapping [98]. Ozeer et al. present a failure management protocol for stateful IoT applications in a dynamic environment [105]. It applies a control loop combining recovery techniques (using checkpointing) and monitoring as well as failure notification and reconfiguration. Current research in this area includes the work of Seeger et al. who introduce an approach for failure detection and mitigation strategies of IoT (edge) devices for complex software tasks [119]. The mitigation strategies include an optimal task allocation strategy for distributed task execution on IoT devices. Dias et al. introduce a pattern language for specifying self-healing strategies for IoT systems [40]. A major challenge that has not received much attention is the design of a resilient monitoring system and control loop respectively able to take decisions from incomplete information and being able to deal with failures in the monitoring or execution chain.

Architectural combinations. Due to their size and geographical distribution as well as the large attack surface, IoT systems are prone to failures even more than traditional distributed systems.

Consequently, resilience mechanisms need to be applied on all layers of an IoT architecture. Here, a particular challenge for enabling resilient IoT systems their massive heterogeneity. Network capacities and latency in different layers may vary by six and more orders of magnitude. Similar holds for the computational and storage capacities. These differences and the fact that not each device is alike imposes constraints on the use of resilience mechanisms and the orchestration of compute tasks. For instance, mechanisms such as state-machine replication are only ill-suited for the sensor and IoT edge layer due to the rather weak network connectivity, large network latency, and limited computational capabilities. On the other hand, the use of wireless communication technology inherently creates redundancy provided that multiple gateway nodes are placed in the range of a sensor. Hence, the installation of an IoT system limits resilience possibilities. This dependency on the actual geo-distribution of devices furthermore requires that mechanisms such as graceful degradation need to be tailored individually to each IoT system.

6 RELATED WORK

Various research studies and analyzes the current status of incorporating resilience in IoT and distrusted systems, thus presenting a broad variety of emerging challenges and solutions in dependability and security [1, 33, 53, 60, 96, 137, 140, 154].

Related systematization of knowledge papers. To begin with, the extensive taxonomy of Avižienis et al. for dependable and secure computing is helpful for establishing a common understanding of key concepts in the broader field of distributed systems [13]. Ratasich et al. present a roadmap towards a more resilient IoT for cyber-physical systems [112]. They focus on summarizing presented techniques, e.g., on fault tolerance, anomaly detection, or self-healing, and outline present and future challenges for dependability and security in such systems. Moghaddam et al. present a systematic mapping study in which they identify and categorize a set of methods for achieving fault tolerance in the IoT [96]. The authors focus more on a statistical comparison of how current research employs existing techniques, e.g., for which different architectural styles these are employed, which attributes are aimed to be improved, and which research trends are emerging. The work of Tsigkanos et al. proposes a research roadmap for resilient IoT system design and future challenges [140]. The authors identify high-level resilience mechanisms and the need to deal with disruptions, and sketch future directions for engineering resilient IoT systems. For instance, they propose having the edge infrastructure consumed as a full-fledged utility, abstracting business logic management from the infrastructure, or autonomous control and self-healing capabilities. Welsh et al. survey resilience techniques for cloud environments [144]. The authors first analyze the state-of-the-art of techniques for different cloud components and subsequently discuss current limitations and challenges. Further, Sequeiros et al. survey existing attack and system modeling techniques that are applicable to IoT and cloud computing [120].

IoT research trends with resilience focus. Terry et al. discuss fault-tolerance techniques, such as *compensation*, in the context of IoT to make IoT systems more resilient [137]. The authors propose to seek novel and lightweight solutions. For instance, devices like sensors or actuators are often fail-stop, and hence application-level fault tolerance can be provided without employing state-machine replication but instead with simple fail-overs to additional, correct components. An interesting trick for IoT systems could be to leverage pre-existing redundancy, e.g., by letting IoT devices discover and select nearby devices that can report similar events, such as motion or presence, or that support similar actions, like turning on a light. Additionally, devices could automatically connect to nearby operating hubs and then switch hubs as soon as failures occur [137]. This can be utilized to reduce replication costs. Further, concepts such as *virtual services* [154] that consolidate data from more than one physical sensor can also be considered for resilient design in service-oriented IoT architectures. The use of blockchains and smart contracts to build reliable IoT ecosystems is discussed in recent works [33, 43, 46, 58]. Blockchains can provide distributed, trust-less, and resilient service execution and can thus be employed, e.g., as a billing layer for marketplaces of services between devices. Abreu et al. propose a modular IoT middleware for smart cities that comprises a distinct component called *Resilience Manager*, which is in charge of supervising activities by accessing a *Monitor* module [1]. Additionally, it employs a *Protection and Recovery* module that can perform actions upon detection of faults, e.g., relying on other modules such as topology control as well as placement and migration modules. Gia et al. study resilience in the domain of *e-Health* systems and present an approach for increasing reliability on top of a 6LoWPAN communication infrastructure [53]. The authors use backup routing between nodes and advanced service mechanisms to maintain connectivity in case of failing connections between system components. Martín et al. introduce a multi-level platform architecture for resilient IoT applications [93]. Their platform employs containerization of components, abstracts functionality of devices behind so-called shadow devices, and achieves fault tolerance through the replication of physical devices and automatic reconfigurations. The authors show their approach can be practically used, e.g., in the field of health monitoring. Javed et al. propose CEFIoT as a fault-tolerant architecture for the IoT [69]. CEFIoT employs the Apache Kafka platform for data replication in both edge and cloud and uses Kubernetes for fault-tolerant management and automatic reconfiguration of the processing pipeline in case of, e.g., connectivity failures. Costa et al. present a voting mechanism for edge computing that lets the edge network validate computations that were performed in a multi-cloud environment [35].

Resilience engineering and resilience techniques. Hukerikar et al. present a catalog of resilience design patterns [62], which is aligned to the field of high performance computing but is also applicable to other computer engineering domains since reoccurring problems and their solutions are described on a very abstract level. Also, their work is highly didactic and tries to guide computer scientists to incorporating resilience-enhancing ideas into concrete system designs. Jackson et al. present a survey on a set of abstract, top-level resilience principles for engineered systems, such as *absorption*, physical and functional *redundancy*, and *layered defense* [66]. Dias et al. present a pattern-language for self-healing IoT systems [40]. In their work, the authors describe a catalog of patterns (building blocks), which can be efficiently combined to craft self-healing systems. These patterns can be divided into two main classes: *Error detection* and *recovery and maintenance of health.* They also explain how different health pattern actions can be sequenced to achieve service restorations towards normal state, thus enabling resilience for a system.

7 CONCLUSIONS

Resilience is often being referred to in different contexts and with different ambitions in mind. Generally, resilience has the meaning to preserve different system properties such as availability, confidentiality, integrity, reliability, maintainability, or safety *when the system encounters change.* Colloquially speaking, a system is being expected to "withstand" change or "bounce back" to a functional state. "Change" can be further differentiated, e.g., into planned changes or unplanned disruptions, such as internal or external faults or attacks. On this note, resilience comprises not only dependability but also security, i.e., should be able to deal with attacks.

The IoT has unique architectural and technical properties with specific challenges such as scalability, evolvability, maintainability, heterogeneity, and complexity. In particular, we need to consider that system architectures span over several IoT layers, and resilience mechanisms should therefore also work across layers. Further, we may need to consider that there are multiple administrative domains and diverse views to the system. Resilience requirements may differ between the views of, e.g., an application user, developer, system operator, or IoT hardware provider. As IoT systems change, these requirements might evolve over time, too.

Quantifying resilience is a multi-dimensional approach, which depends on the goals that are considered. Concepts to measure the effectiveness of resilience mechanisms vary. For instance, quality-of-service metrics, fault-tolerance coverage, or cost metrics exist, but have to be carefully chosen to best represent an application's resilience constraints. Further, it is crucial to enable IoT systems' self-adaptation capabilities, since resilience does not only mean "tolerance", but often also an approach to "trying to uphold service under best efforts".

A broad variety of applicable resilience mechanisms exists. We divided them into redundancy, monitoring, protection, and recovery techniques. The goal of these mechanisms is to make a system capable of compensating faults (redundancy), detecting faults or attacks (monitoring), shielding against attacks, preventing the occurrence of harm (protection), and steering the system back to a well-defined functional state (recovery). The incorporation and combination of a well-chosen set of such resilience mechanisms allow an IoT system to self-heal and thus become more resilient, but mechanism-specific constraints need to be respected and considered by application developers at design time and by the execution platform at runtime.

ACKNOWLEDGMENTS

The authors thank the anonymous referees for their valuable comments and helpful suggestions. This work has received financial support by the Federal Ministry of Education and Research of Germany under grant no 01IS18068, SORRIR and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant no 268730775, OptSCORE.

REFERENCES

[1] D. P. Abreu, K. Velasquez, M. Curado, and E. Monteiro. A resilient internet of things architecture for smart cities. *Ann. of Telecomm.*, 72(1):19–30, Feb 2017.

[2] A. P. Achilleos, K. Kritikos, A. Rossini, G. M. Kapitsaki, J. Domaschka, M. Orzechowski, D. Seybold, F. Griesinger, N. Nikolov, D. Romero, and G. A. Papadopoulos. The cloud application modelling and execution language. *J. Cloud Comp.*, 8:20, 2019.

[3] J. Ali, T. Ali, Y. Alsaawy, A. S. Khalid, and S. Musa. Blockchain-based smart-IoT trust zone measurement architecture. In *Int. Conf. on Omni-Layer Intel. Sys. (COINS)*, page 152–157, New York, NY, USA, 2019. ACM.

[4] K. Anagnostakis, S. Sidiroglou, P. Akritidis, M. Polychronakis, A. Keromytis, and E. Markatos. Shadow honeypots. *Int. J. of Comp. and Netw. Sec.*, 01 2010.

[5] J. P. Anderson. Computer security threat monitoring and surveillance. *Techn. Rep., James P. Anderson Company*, 1980.

[6] J. Andersson, V. Grassi, R. Mirandola, and D. Perez-Palacin. A distilled characterization of resilience and its embraced properties based on state-spaces. In *Int. Worksh. on Softw. Eng. for Res. Sys.*, pages 11–25. Springer, 2019.

[7] I. Andrea, C. Chrysostomou, and G. Hadjichristofi. Internet of things: Security vulnerabilities and challenges. In *IEEE Symp. on Comp. and Comm. (ISCC)*, pages 180–187. IEEE, 2015.

[8] E. Anthi, L. Williams, M. Słowińska, G. Theodorakopoulos, and P. Burnap. A supervised intrusion detection system for smart home iot devices. *IEEE Intern. of Things J.*, 6(5):9042–9053, 2019.

[9] P. Arcaini, E. Riccobene, and P. Scandurra. Modeling and analyzing mape-k feedback loops for self-adaptation. In *IEEE/ACM 10th Int. Symp. on Softw. Eng. for Adapt. and Self-Manag. Sys. (SEAMS)*, pages 13–23, 2015.

[10] A. Avižienis. The n-version approach to fault-tolerant software. *IEEE Trans. on Softw. Eng.*, SE-11(12):1491–1501, 1985.

[11] A. Avižienis. A visit to the jungle of terminology. In *47th Ann. IEEE/IFIP Int. Conf. on Dep. Sys. and Netw. Workshops (DSN-W)*, pages 149–152. IEEE, 2017.

[12] A. Avižienis, J.-C. Laprie, and B. Randell. *Fundamental concepts of dependability.* University of Newcastle upon Tyne, Computing Science, 2001.

[13] A. Avižienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. on Dep. and Sec. Comp.*, 1(1):11–33, Jan 2004.

[14] S. S. Baghsorkhi and C. Margiolas. Automating efficient variable-grained resiliency for low-power IoT systems. In *Int. Symp. on Code Gen. and Opt. (CGO)*, page 38–49, New York, NY, USA, 2018. ACM.

[15] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin, and S. Kounev. Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field. *IEEE Trans. on Par. and Distr. Sys.*, 30(4):800–813, 2019.

[16] J. Bauer, R. C. Staudemeyer, H. C. Pöhls, and A. Fragkiadakis. ECDSA on things: IoT integrity protection in practise. In *Inf. and Comm. Sec. (ICICS)*, volume 9977 of *LNCS*. Springer, Nov. 2016.

[17] M. Baykara and R. Das. A novel honeypot based security approach for real-time intrusion detection and prevention systems. *J. of Inform. Sec. and App.*, 41:103–116, 2018.

[18] J. Beal, M. Viroli, D. Pianini, and F. Damiani. Self-adaptation to device distribution in the internet of things. *ACM Trans. on Auton. and Adapt. Sys. (TAAS)*, 12(3):1–29, 2017.

[19] E. Bellini, F. Bagnoli, A. A. Ganin, and I. Linkov. Cyber resilience in IoT network: Methodology and example of assessment through epidemic spreading approach. In *IEEE World Congr. on Svc. (SERVICES)*, volume 2642, pages 72–77. IEEE, 2019.

[20] K. E. Benson, G. Bouloukakis, C. Grant, V. Issarny, S. Mehrotra, I. Moscholios, and N. Venkatasubramanian. FireDeX: a prioritized IoT data exchange middleware for emergency response. In *19th ACM/IFIP Int. Middlew. Conf.*, pages 279–292, 2018.

[21] A. Bessani, J. Sousa, and E. E. Alchieri. State machine replication for the masses with BFT-SMaRt. In *44th Ann. IEEE/IFIP Int. Conf. on Dep. Sys. and Netw. (DSN)*, pages 355–362. IEEE, 2014.

[22] A. N. Bessani. From Byzantine fault tolerance to intrusion tolerance (a position paper). In *IEEE/IFIP 41st Int. Conf. on Dep. Sys. and Netw. Worksh. (DSN-W)*, pages 15–18. IEEE, 2011.

[23] M. Bishop, M. Carvalho, R. Ford, and L. M. Mayron. Resilience is more than availability. In *New Sec. Parad. Worksh.*, pages 95–104, 2011.

[24] D. Breitenbacher, I. Homoliak, Y. L. Aung, N. O. Tippenhauer, and Y. Elovici. HADES-IoT: A practical host-based anomaly detection system for IoT devices. In *ACM Asia Conf. on Comp. and Comm. Sec. (Asia CCS)*, page 479–484, New York, NY, USA, 2019. ACM.

[25] E. A. Brewer. Towards robust distributed systems (abstract). In *19th Ann. ACM Symp. on Princ. of Distr. Comp. (PODC)*, page 7, New York, NY, USA, 2000. ACM.

[26] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. The primary-backup approach. *Distr. Sys.*, 2:199–216, 1993.

[27] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *3rd Symp. on Oper. Sys. Des. and Impl. (OSDI)*, pages 173–186, 1999.

[28] Z. B. Celik, P. McDaniel, and G. Tan. Soteria: Automated IoT safety and security analysis. In *USENIX Ann. Techn. Conf. (ATC)*, pages 147–158, 2018.

[29] Z. B. Celik, P. McDaniel, G. Tan, L. Babun, and A. S. Uluagac. Verifying internet of things safety and security in physical spaces. *IEEE Sec. & Priv.*, 17(5):30–37, 2019.

[30] H. Chang, A. Hari, S. Mukherjee, and T. Lakshman. Bringing the cloud to the edge. In *IEEE Conf. on Comp. Commu. Worksh. (INFOCOM WKSHPS)*, pages 346–351. IEEE, 2014.

[31] L. Chen and A. Avižienis. N-version programming: A fault-tolerance approach to reliability of software operation. In *8th IEEE Int. Symp. on Fault-Tol. Comp. (FTCS)*, volume 1, pages 3–9, 1978.

[32] J. Cho, P. M. Hurley, and S. Xu. Metrics and measurement of trustworthy systems. In *IEEE Mil, Comm. Conf. (MILCOM)*, pages 1237–1242, 2016.

[33] K. Christidis and M. Devetsikiotis. Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303, 2016.

[34] A. Compagno, M. Conti, and R. Droms. OnboardICNg: A secure protocol for on-boarding IoT devices in ICN. In *3rd ACM Conf. on Inform.-Centric Netw. (ICN)*, page 166–175, New York, NY, USA, 2016. ACM.

[35] P. A. Costa and M. Beko. Dependable and secure voting mechanism in edge comp. *Future Internet*, 11(12):262, 2019.

[36] C. Y. da Silva Costa and E. A. P. Alchieri. Diversity on state machine replication. In *32nd Int. Conf. on Adv. Inform. Netw. and App. (AINA)*, pages 429–436. IEEE, 2018.

[37] Y. Dai, M. Xie, K. Poh, and S. Ng. A model for correlated failures in n-version programming. *IIE Trans.*, 36(12):1183–1192, 2004.

[38] K. A. Delic. On resilience of IoT systems: The internet of things (Ubiquity symposium). *Ubiquity*, 2016, Feb. 2016.

[39] L. Delledonne, V. Zaccaria, R. Susella, G. Bertoni, and F. Melzani. Casca: A design automation approach for designing hardware countermeasures against side-channel attacks. *ACM Trans. Des. Autom. Electron. Syst.*, 23(6), Nov. 2018.

[40] J. a. P. Dias, T. B. Sousa, A. Restivo, and H. S. Ferreira. A pattern-language for self-healing internet-of-things systems. In *Eur. Conf. on Pattern Lang. of Progr. (EuroPLoP)*, New York, NY, USA, 2020. ACM.

[41] J. Domaschka. *A comprehensive approach to transparent and flexible replication of Java services and applications.* PhD thesis, Universität Ulm, 2013.

[42] S. Dowling, M. Schukat, and H. Melvin. A ZigBee honeypot to assess IoT cyberattack behaviour. In *28th Irish Signals and Systems Conf. (ISSC)*, pages 1–6. IEEE, 2017.

[43] A. Durand, P. Gremaud, and J. Pasquier. Resilient, crowd-sourced LPWAN infrastructure using blockchain. In *1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*, pages 25–29, 2018.

[44] P. Eichhammer, C. Berger, H. P. Reiser, J. Domaschka, F. J. Hauck, G. Habiger, F. Griesinger, and J. Pietron. Towards a robust, self-organizing IoT platform for secure and dependable service execution. *Tagungsbd d. FB-SYS Herbsttr.*, 2019.

[45] O. Erdene-Ochir, A. Kountouris, M. Minier, and F. Valois. A new metric to quantify resiliency in networking. *IEEE Comm. Letters*, 16(10):1699–1702, 2012.

[46] T. M. Fernández-Caramés and P. Fraga-Lamas. A review on the use of blockchain for the internet of things. *IEEE Access*, 6:32979–33001, 2018.

[47] L. Ferretti, M. Marchetti, and M. Colajanni. Fog-based secure comm. for low-power IoT devices. *ACM Trans. Internet Technol.*, 19(2), Mar. 2019.

[48] J. C. Fuentes Carranza and P. W. L. Fong. Brokering policies and execution monitors for IoT middleware. In *24th ACM Symp. on Access Contr. Models and Techn. (SACMAT)*, page 49–60, New York, NY, USA, 2019. ACM.

[49] A. A. Ganin, E. Massaro, A. Gutfraind, N. Steen, J. M. Keisler, A. Kott, R. Mangoubi, and I. Linkov. Operational resilience: concepts, design and analysis. *Scientific reports*, 6(1):1–12, 2016.

[50] M. Garcia, A. Bessani, and N. Neves. Lazarus: Automatic management of diversity in bft systems. In *20th ACM/IFIP Int. Middlew. Conf.*, pages 241–254, 2019.

[51] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Netw. and Distr. Sys. Sec. Symp.*, pages 191–206, VA, USA, 2003. Internet Society.

[52] A. Gatouillat and Y. Badr. Verifiable and resource-aware component model for IoT devices. In *9th Int. Conf. on Mgmt of Digital EcoSys. (MEDES)*, page 235–242, New York, NY, USA, 2017. ACM.

[53] T. N. Gia, A. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen. Fault tolerant and scalable IoT-based architecture for health monitoring. In *IEEE Sensors Applications Symp. (SAS)*, pages 1–6, April 2015.

[54] M. Glass, M. Lukasiewycz, C. Haubelt, and J. Teich. Incorporating graceful degradation into embedded system design. In *Des. Autom. Test in Eur. Conf. and Exh. (DATE)*, pages 320–323, 2009.

[55] J. Goldstein, A. Abdelhamid, M. Barnett, S. Burckhardt, B. Chandramouli, D. Gehring, N. Lebeck, C. Meiklejohn, U. F. Minhas, R. Newton, R. G. Peshawaria, T. Zaccai, and I. Zhang. A.m.b.r.o.s.i.a: Providing performant virtual resiliency for distributed applications. *Proc. VLDB Endow.*, 13(5):588–601, Jan. 2020.

[56] H. Gu and M. Potkonjak. Efficient and secure group key management in IoT using multistage interconnected PUF. In *Int. Symp. on Low Power Electr. and Des. (ISPLED)*, New York, NY, USA, 2018. ACM.

[57] J. D. Guarnizo, A. Tambe, S. S. Bhunia, M. Ochoa, N. O. Tippenhauer, A. Shabtai, and Y. Elovici. Siphon: Towards scalable high-interaction physical honeypots. In *3rd ACM Worksh. on Cyber-Physical Syst. Security*, pages 57–68, 2017.

[58] R. Han, V. Gramoli, and X. Xu. Evaluating blockchains for IoT. In *9th IFIP Int. Conf. on New Techn., Mobil. and Sec. (NTMS)*, pages 1–5. IEEE, 2018.

[59] Y. Harchol, A. Mushtaq, V. Fang, J. McCauley, A. Panda, and S. Shenker. Making edge-computing resilient. In *11th ACM Symp. on Cloud Comp. (SoCC)*, page 253–266, New York, NY, USA, 2020. ACM.

[60] M. Z. Hasan and F. Al-Turjman. Optimizing multipath routing with guaranteed fault tolerance in internet of things. *IEEE Sensors J.*, 17(19):6463–6473, Oct 2017.

[61] J. L. Hernandez-Ramos, M. P. Pawlowski, A. J. Jara, A. F. Skarmeta, and L. Ladid. Toward a lightweight authentication and authorization framework for smart objects. *IEEE J. on Selected Areas in Comm.*, 33(4):690–702, 2015.

[62] S. Hukerikar and C. Engelmann. Resilience design patterns: A structured approach to resilience at extreme scale. *arXiv preprint arXiv:1708.07422*, 2017.

[63] A. Ibrahim, A.-R. Sadeghi, G. Tsudik, and S. Zeitouni. Darpa: Device attestation resilient to physical attacks. In *9th ACM Conf. on Sec. & Priv. in Wireless and Mobile Netw. (WiSec)*, page 171–182, New York, NY, USA, 2016. ACM.

[64] A. Ilyushkin, A. Ali-Eldin, N. Herbst, A. Bauer, A. V. Papadopoulos, D. Epema, and A. Iosup. An experimental performance evaluation of autoscalers for complex workflows. *ACM Trans. Model. Perform. Eval. Comp. Syst.*, 3(2), Apr. 2018.

[65] International Electrotechnical Commission. International electrotechnical vocabulary – Part 192: Dependability. Standard IEC 60050-192:2015, International Electrotechnical Commission, 2015.

[66] S. Jackson and T. L. Ferris. Resilience principles for engineered systems. *Sys. Eng.*, 16(2):152–164, 2013.

[67] S. Jahan, I. Riley, C. Walter, R. F. Gamble, M. Pasco, P. K. McKinley, and B. H. Cheng. MAPE-K/MAPE-SAC: An interaction framework for adaptive systems with security assurance cases. *Future Gen. Comp. Sys.*, 109:197–209, 2020.

[68] B. Jain, M. B. Baig, D. Zhang, D. E. Porter, and R. Sion. SoK: Introspections on trust and the semantic gap. In *IEEE Symp. on Sec. and Priv.*, pages 605–620. IEEE, 2014.

[69] A. Javed, K. Heljanko, A. Buda, and K. Främling. CEFIoT: a fault-tolerant IoT architecture for edge and cloud. In *IEEE 4th World Forum on Intern. of Things (WF-IoT)*, pages 813–818. IEEE, 2018.

[70] A. Javed, J. Robert, K. Heljanko, and K. Främling. IoTEF: A federated edge-cloud architecture for fault-tolerant IoT applications. *J. of Grid Comp.*, pages 1–24, 2020.

[71] X. Ji, C. Li, X. Zhou, J. Zhang, Y. Zhang, and W. Xu. Authenticating smart home devices via home limited channels. *ACM Trans. Internet Things*, 1(4), Aug. 2020.

[72] H. Jin and P. Papadimitratos. Resilient privacy protection for location-based services through decentralization. *ACM Trans. Priv. Secur.*, 22(4), Sept. 2019.

[73] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel. Cheapbft: resource-efficient byzantine fault tolerance. In *7th ACM Eur. Conf. on Comp. Sys. (EuroSys)*, pages 295–308, 2012.

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

[75] Z. A. Khan, J. Ullrich, A. G. Voyiatzis, and P. Herrmann. A trust-based resilient routing mechanism for the internet of things. In *12th Int. Conf. on Avail., Rel. and Sec. (ARES)*, New York, NY, USA, 2017. ACM.

[76] N. Khoussainova, M. Balazinska, and D. Suciu. Towards correcting input data errors probabilistically using integrity constraints. In *5th ACM Int. Worksh. on Data Eng. for Wirel. and Mobile Access*, pages 43–50, 2006.

[77] H. Kim, E. Kang, D. Broman, and E. A. Lee. An architectural mechanism for resilient IoT services. In *1st ACM Worksh. on the Intern. of Safe Things (SafeThings)*, page 8–13, New York, NY, USA, 2017. ACM.

[78] H. Kim, E. Kang, D. Broman, and E. A. Lee. Resilient authentication and authorization for the Internet of things (IoT) using edge computing. *ACM Trans. Internet of Things*, 1(1), Mar. 2020.

[79] J. C. Knight and E. A. Strunk. Achieving critical system survivability through software architectures. In R. de Lemos, C. Gacek, and A. Romanovsky, editors, *Architecting Dependable Systems II*, LNCS, pages 51–78. Springer, 2004.

[80] R. Koo and S. Toueg. Checkpointing and rollback-recovery for distributed systems. *IEEE Trans. on Softw. Eng.*, SE-13(1):23–31, 1987.

[81] I. Kopestenski and P. Van Roy. Erlang as an enabling technology for resilient general-purpose applications on edge IoT networks. In *18th ACM SIGPLAN Int. Workshop on Erlang*, pages 1–12, New York, NY, USA, 2019. ACM.

[82] D. Kuemper, T. Iggena, R. Toenjes, and E. Pulvermueller. Valid.IoT: a framework for sensor data quality analysis and interpolation. In *9th ACM Multimedia Systems Conf.*, pages 294–303, 2018.

[83] D. Kyriazis and T. Varvarigou. Smart, autonomous and reliable Internet of Things. *Procedia Comp. Sci.*, 21:442–448, 2013.

[84] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM TOPLAS*, 4(3):382–401, 1982.

[85] J.-C. Laprie. Dependable computing and fault-tolerance. *Digest of Papers FTCS-15*, pages 2–11, 1985.

[86] J.-C. Laprie. From dependability to resilience. In *38th IEEE/IFIP Int. Conf. on Dep. Sys. and Netw. (DSN)*, pages G8–G9, 2008.

[87] J.-Y. Lee, W.-C. Lin, and Y.-H. Huang. A lightweight authentication protocol for internet of things. In *Int. Symp. on Next-Generation Electronics (ISNE)*, pages 1–2. IEEE, 2014.

[88] X. Li, F. Yan, F. Zuo, Q. Zeng, and L. Luo. Touch well before use: Intuitive and secure authentication for IoT devices. In *The 25th Ann. Int. Conf. on Mobile Comp. and Netw. (MobiCom)*, pages 1–17, 2019.

[89] X. Li, Q. Zeng, L. Luo, and T. Luo. T2Pair: Secure and usable pairing for heterogeneous IoT devices. In *ACM SIGSAC Conf. on Comp. and Comm. Sec. (CCS)*, page 309–323, New York, NY, USA, 2020. ACM.

[90] W.-T. Lin, F. Bakir, C. Krintz, R. Wolski, and M. Mock. Data repair for distributed, event-based IoT applications. In *13th ACM Int. Conf. on Distrib. and Event-based Sys. (DEBS)*, pages 139–150, 2019.

[91] F. Loukil, C. Ghedira-Guegan, K. Boukadi, A.-N. Benharkat, and E. Benkhelifa. Data privacy based on IoT device behavior control using blockchain. *ACM Trans. Internet Techn.*, 21(1), Jan. 2021.

[92] T. Luo, Z. Xu, X. Jin, Y. Jia, and X. Ouyang. IoTCandyJar: Towards an intelligent-interaction honeypot for IoT devices. [online] https://www.blackhat.com/docs/us-17/thursday/us-17-Luo-Iotcandyjar-Towards-An-Intelligent-Interaction-Honeypot-For-IoT-Devices-wp.pdf, 2017.

[93] C. Martín, D. Garrido, M. Díaz, and B. Rubio. From the edge to the cloud: Enabling reliable IoT applications. In *7th Int. Conf. on Future Intern. of Things and Cloud (FiCloud)*, pages 17–22. IEEE, 2019.

[94] V. S. Miller. Use of elliptic curves in cryptography. In *Conf. on the Theo. and App. of Crypt. Techn. (EUROCRYPT)*, pages 417–426. Springer, 1985.

[95] A. Modarresi and J. P. Sterbenz. Multilevel IoT model for smart cities resilience. In *12th Int. Conf. on Future Intern. Techn. (CFI)*, New York, NY, USA, 2017. ACM.

[96] M. T. Moghaddam and H. Muccini. Fault-tolerant IoT. In R. Calinescu and F. Di Giandomenico, editors, *Software Engineering for Resilient Systems*, pages 67–84, Cham, 2019. Springer.

[97] M. Mössinger, B. Petschkuhn, J. Bauer, R. C. Staudemeyer, M. Wójcik, and H. C. Pöhls. Towards quantifying the cost of a secure IoT: Overhead and energy consumption of ECC signatures on an ARM-based device. In *IEEE 17th Int. Symp. on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6. IEEE, 2016.

[98] H. Muccini, R. Spalazzese, M. T. Moghaddam, and M. Sharaf. Self-adaptive IoT architectures: An emergency handling case study. In *12th Eur. Conf. on Softw. Arch. (ECSA): Companion Proc.*, New York, NY, USA, 2018. ACM.

[99] W. Nace and P. Koopman. A graceful degradation framework for distributed embedded systems. In *Worksh. on Rel. in Emb. Sys. (in Conj. with SRDS)*, Oct 2001.

[100] W. Nace and P. Koopman. A product family approach to graceful degradation. In B. Kleinjohann, editor, *Architecture and Design of Distributed Embedded Systems: IFIP WG10.3/WG10.4/WG10.5 Int. Worksh. on Distr. and Par. Emb. Sys. (DIPES)*, pages 131–140. Springer, 2001.

[101] R. Nagar. *Windows NT File System Internals: A Developer's Guide.* O'Reilly, 1997.

[102] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. Colbert, and P. McDaniel. IoTSan: Fortifying the safety of IoT systems. In *14th Int. Conf. on emerging Networking EXperiments and Technologies*, pages 191–203, 2018.

[103] D. Oh, D. Kim, and W. W. Ro. A malicious pattern detection engine for embedded security systems in the internet of things. *Sensors*, 14(12):24188–24211, 2014.

[104] D. O'Keeffe, T. Salonidis, and P. Pietzuch. Frontier: Resilient edge processing for the Internet of Things. *Proc. VLDB Endow.*, 11(10):1178–1191, June 2018.

[105] U. Ozeer, X. Etchevers, L. Letondeur, F.-G. Ottogalli, G. Salaün, and J.-M. Vincent. Resilience of stateful IoT applications in a dynamic fog environment. In *15th EAI Int. Conf. on Mobile and Ubiq. Sys.: Comp., Netw. and Svc. (MobiQuitous)*, page 332–341, New York, NY, USA, 2018. ACM.

[106] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow. IoTPOT: A novel honeypot for revealing current IoT threats. *J. of Inform. Proc.*, 24(3):522–533, 2016.

[107] L. R. Parker, P. D. Yoo, T. A. Asyhari, L. Chermak, Y. Jhi, and K. Taha. DEMISe: Interpretable deep extraction and mutual information selection techniques for IoT intrusion detection. In *14th Int. Conf. on Avail., Rel. and Sec. (ARES)*, New York, NY, USA, 2019. ACM.

[108] H. C. Pöhls. JSON sensor signatures (JSS): end-to-end integrity protection from constrained device to IoT application. In *9th Int. Conf. on Innov. Mobile and Intern. Svc. in Ubiq. Comp.*, pages 306–312. IEEE, 2015.

[109] D. K. Pradhan and N. H. Vaidya. Roll-forward checkpointing scheme: A novel fault-tolerant architecture. *IEEE Trans. on Comp.*, 43(10):1163–1174, 1994.

[110] S. Pradhan, A. Dubey, S. Khare, S. Nannapaneni, A. Gokhale, S. Mahadevan, D. C. Schmidt, and M. Lehofer. CHARIOT: Goal-driven orchestration middleware for resilient IoT systems. *ACM Trans. Cyber-Phys. Syst.*, 2(3), June 2018.

[111] C. Qu, R. N. Calheiros, and R. Buyya. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Comp. Surv.*, 51(4), July 2018.

[112] D. Ratasich, F. Khalid, F. Geissler, R. Grosu, M. Shafique, and E. Bartocci. A roadmap toward the resilient internet of things for cyber-physical systems. *IEEE Access*, 7:13260–13283, 2019.

[113] S. Raza, L. Wallgren, and T. Voigt. Svelte: Real-time intrusion detection in the internet of things. *Ad hoc networks*, 11(8):2661–2674, 2013.

[114] Risk Steering Committee. DHS risk lexicon 2010 edition. https://www.dhs.gov/xlibrary/assets/dhs-risk-lexicon-2010.pdf, 2010.

[115] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. of the ACM*, 21(2):120–126, 1978.

[116] M. Rumez, J. Dürrwang, J. Braun, and R. Kriesten. Security hardening of automotive networks through the implementation of attribute-based plausibility checks. *Int. J. on Adv. in Sec.*, 11(1&2):52–59, 2018.

[117] E. Rutten, N. Marchand, and D. Simon. Feedback control as mape-k loop in autonomic computing. In R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese, editors, *Software Engineering for Self-Adaptive Systems III. Assurances*, pages 349–373, Cham, 2017. Springer.

[118] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comp. Surv.*, 22(4):299–319, 1990.

[119] J. Seeger, A. Bröring, and G. Carle. Optimally self-healing IoT choreographies. *ACM Trans. Internet Technol.*, 20(3), July 2020.

[120] J. a. B. F. Sequeiros, F. T. Chimuco, M. G. Samaila, M. M. Freire, and P. R. M. Inácio. Attack and system modeling applied to iot, cloud, and mobile ecosystems: Embedding security by design. *ACM Comp. Surv.*, 53(2), Mar. 2020.

[121] D. Seybold, S. Volpert, S. Wesner, A. Bauer, N. Herbst, and J. Domaschka. Kaa: Evaluating elasticity of cloud-hosted dbms. In *IEEE Int. Conf. on Cloud Comp. Techn. and Sci. (CloudCom)*, pages 54–61, 2019.

[122] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In X. Défago, F. Petit, and V. Villain, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 386–400. Springer, 2011.

[123] Y. Shi, W. Wei, Z. He, and H. Fan. An ultra-lightweight white-box encryption scheme for securing resource-constrained IoT devices. In *32nd Ann. Conf. on Comp. Sec. App. (ACSAC)*, page 16–29, New York, NY, USA, 2016. ACM.

[124] S. N. Shirazi, A. Gouglidis, A. Farshad, and D. Hutchison. The extended cloud: Review and analysis of mobile edge computing and fog from a security and resilience perspective. *IEEE J. on Sel. Areas in Comm.*, 35(11):2586–2595, 2017.

[125] S. Sinche, O. Polo, D. Raposo, M. Femandes, F. Boavida, A. Rodrigues, V. Pereira, and J. S. Silva. Assessing redundancy models for IoT reliability. In *IEEE 19th Int. Symp. on "A World of Wirel., Mobile and Multim. Netw." (WoWMoM)*, pages 14–15. IEEE, 2018.

[126] S. Singh, P. K. Sharma, S. Y. Moon, and J. H. Park. Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions. *J. of Ambient Intel. and Humanized Comp.*, pages 1–18, 2017.

[127] V. Sivaraman, H. H. Gharakheili, A. Vishwanath, R. Boreli, and O. Mehani. Network-level security and privacy control for smart-home IoT devices. In *IEEE 11th Int. Conf. on Wirel. and Mobile Comp., Netw. and Comm. (WiMob)*, pages 163–167. IEEE, 2015.

[128] P. Sousa, A. N. Bessani, M. Correia, N. F. Neves, and P. Verissimo. Highly available intrusion-tolerant services with proactive-reactive recovery. *IEEE Trans. on Par. and Distr. Sys.*, 21(4):452–465, 2009.

[129] J. P. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Comp. Netw.*, 54(8):1245–1265, 2010.

[130] L. Strigini. Fault tolerance and resilience: meanings, measures and assessment. In *Resilience assessment and evaluation of computing systems*, pages 3–24. Springer, 2012.

[131] L. Sweeney. k-anonymity: A model for protecting privacy. *Int. J. of Uncert., Fuzziness and Knowl.-Based Sys.*, 10(05):557–570, 2002.

[132] F. M. Tabrizi and K. Pattabiraman. Design-level and code-level security analysis of IoT devices. *ACM Trans. Emb. Comp. Sys.*, 18(3), May 2019.

[133] A. Taivalsaari and T. Mikkonen. A roadmap to the programmable world: Software challenges in the IoT era. *IEEE Software*, 34(1):72–80, Jan 2017.

[134] R. Tanabe, T. Tamai, A. Fujita, R. Isawa, K. Yoshioka, T. Matsumoto, C. Gañán, and M. van Eeten. Disposable botnets: Examining the anatomy of IoT botnet infrastructure. In *15th Int. Conf. on Avail., Rel. and Sec. (ARES)*, New York, NY, USA, 2020. ACM.

[135] L. Tandon, P. W. L. Fong, and R. Safavi-Naini. HCAP: A history-based capability system for IoT devices. In *23rd ACM Symp. on Access Contr. Models and Techn. (SIGMAT)*, page 247–258, New York, NY, USA, 2018. ACM.

[136] M. Tappler, B. K. Aichernig, and R. Bloem. Model-based testing IoT communication via active automata learning. In *IEEE Int. Conf. on Softw. Testing, Verif. and Valid. (ICST)*, pages 276–287. IEEE, 2017.

[137] D. Terry. Toward a new approach to IoT fault tolerance. *Computer*, 49(8):80–83, Aug 2016.

[138] M. Thompson, M. Ryan, J. Slay, and A. Mclucas. A new resilience taxonomy. *INCOSE Int. Symp.*, 26:1318–1330, 07 2016.

[139] Z. Tong, R. Y. Kain, and W. T. Tsai. A low overhead checkpointing and rollback recovery scheme for distributed systems. In *8th Symp. on Rel. Distr. Sys. (SRDS)*, pages 12–20, 1989.

[140] C. Tsigkanos, S. Nastic, and S. Dustdar. Towards resilient internet of things: Vision, challenges, and research roadmap. In *39th IEEE Int. Conf. Distrib. Comp. Syst. (ICDCS)*, pages 1–11, 2019.

[141] M. Vieira, H. Madeira, K. Sachs, and S. Kounev. Resilience benchmarking. In *Resilience Assessment and Evaluation of Comp. Systems*, pages 283–301. Springer, 2012.

[142] E. D. Vugrin, D. E. Warren, and M. A. Ehlen. A resilience assessment framework for infrastructure and economic systems: Quantitative and qualitative resilience analysis of petrochemical supply chains to a hurricane. *Process Safety Progress*, 30(3):280–290, 2011.

[143] M. Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *Int. Worksh. on Open Probl. in Netw. Sec.*, pages 112–125. Springer, 2015.

[144] T. Welsh and E. Benkhelifa. On resilience in cloud computing: A survey of techniques across the cloud domain. *ACM Comp. Surv.*, 53(3), May 2020.

[145] M. Willocx, I. Bohé, and V. Naessens. QoS-by-design in reconfigurable IoT ecosystems. In *IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 628–632. IEEE, 2019.

[146] M. Witti and D. Konstantas. A secure and privacy-preserving internet of things framework for smart city. In *6th Int. Conf. on Inform. Techn.: IoT and Smart City (ICIT)*, page 145–150, New York, NY, USA, 2018. ACM.

[147] Y. Xiong, Y. Sun, L. Xing, and Y. Huang. Extend cloud to edge with kubeedge. In *IEEE/ACM Symp. on Edge Comp. (SEC)*, pages 373–377. IEEE, 2018.

[148] C. Xu, J. Ren, D. Zhang, and Y. Zhang. Distilling at the edge: A local differential privacy obfuscation framework for IoT data analytics. *IEEE Comm. Mag.*, 56(8):20–25, 2018.

[149] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao. A survey on security and privacy issues in Internet-of-Things. *IEEE Internet of Things J.*, 4(5):1250–1258, 2017.

[150] R. Yasaei, F. Hernandez, and M. A. A. Faruque. IoT-CAD: Context-aware adaptive anomaly detection in IoT systems through sensor association. In *39th Int. Conf. on Comp.-Aided Des. (ICCAD)*, New York, NY, USA, 2020. ACM.

[151] S. Z. Yong, M. Zhu, and E. Frazzoli. Switching and data injection attacks on stochastic cyber-physical systems: Modeling, resilient estimation, and attack mitigation. *ACM Trans. on Cyber-Phys. Sys.*, 2(2):1–2, 2018.

[152] M.-K. Yoon, B. Liu, N. Hovakimyan, and L. Sha. VirtualDrone: Virtual sensing, actuation, and communication for attack-resilient unmanned aerial systems. In *8th Int. Conf. on Cyber-Phys. Sys. (ICCPS)*, page 143–154, New York, NY, USA, 2017. ACM.

[153] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga. A survey of intrusion detection in internet of things. *J. of Netw. and Comp. App.*, 84:25–37, 2017.

[154] S. Zhou, K. Lin, J. Na, C. Chuang, and C. Shih. Supporting service adaptation in fault tolerant Internet of Things. In *IEEE 8th Int. Conf. on Service-Oriented Comp. and App. (SOCA)*, pages 65–72, Oct 2015.