Towards a Robust, Self-Organizing IoT Platform for Secure and Dependable Service Execution

Philipp Eichhammer[*]
Christian Berger
Hans P. Reiser
University of Passau
Germany

Jörg Domaschka
Franz J. Hauck
Gerhard Habiger
Ulm University
Germany

Frank Griesinger
Jakob Pietron
Ulm University
Germany

ABSTRACT

In the IoT, resilience capabilities increasingly gain traction for applications, as IoT systems tend to play a bigger role for both the proper functioning of our society and the survivability of companies. However, hardening IoT service execution against a variety of possible faults and attacks becomes increasingly difficult as the complexity, size and heterogeneity of IoT infrastructures tend to grow further and further. Moreover, many existing solutions only regard either specific faults or security issues instead of following a unifying approach.

In this position paper, we present our research project called SORRIR, which essentially is an approach to develop a self-organizing IoT platform for dependable and secure service execution. One of our main ambitions is to support developers by separating application development (app logic) from resilience properties, so that developers can configure a desired resilience degree without proper knowledge of underlying technical, implementation-level details of employed resilience mechanisms. Further, we consider security requirements and properties as an integral component of our platform.

CCS CONCEPTS

• **Computer systems organization → Dependable and fault-tolerant systems and networks**; Embedded systems; Redundancy;
• **General and reference →** *Reliability*;

KEYWORDS

Internet-of-Things, resilience, middleware, dependability, security, self-organization

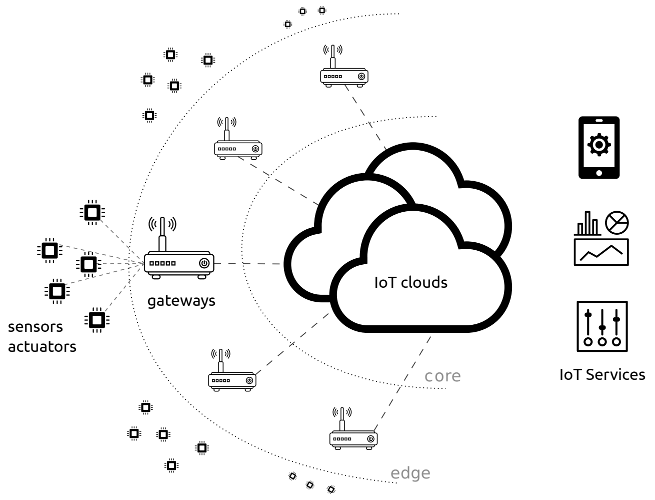

Figure 1: Architecture of IoT infrastructures.

1 INTRODUCTION

In recent years the deployment of Internet-of-Things (IoT) systems not only gained in popularity within the general public by smart homes and smart devices but became also part of critical infrastructures of the industrial sector. Thereby, all IoT installations feature the same basic system architecture as shown in Figure 1. Application specific sensors and actuators are attached to things and living beings where sensors provide data about their environment while actuators interact with their surrounding. Both are either connected by wire or wireless with one or more gateway nodes located in the edge of the Internet. Utilizing those gateways, sensors can gather and distribute data while actuators can receive commands. Gateways are connected to the core network, often referred to as backend, via the Internet. The usually limited computational capacity of those edge devices leads to computationally intensive tasks, e.g., analysis of time series being offloaded to components within the core network, often based on public cloud providers.

However, this described situation increasingly changes and becomes more complex due to a series of developments [2, 4, 10, 12]: (i) IoT systems are not only becoming more and more widespread in everyday life, but are also becoming part of critical infrastructures in Industry 4.0, product service systems or distributed control systems such as autonomous driving support. The spread of IoT systems also increases in the eHealth sector. This creates an increasing social dependence on the fail-safe operation of these systems. (ii) IoT systems are also becoming significantly larger and become especially complex when part of a critical infrastructure. (iii) Latency-critical applications increasingly require that parts of the functionality of IoT systems are offloaded to edge devices. (iv) The development cycle for software in general, but also for IoT software, is getting shorter and shorter while we (v) experience a diversification of the sensor and actuator landscape at the same time.

We believe that these developments can be addressed by employing a holistic approach: the complexity, magnitude and diversification of the sensor landscape aggravates the programmability of an IoT system. The distributed programming model of an IoT system should allow a developer to focus on the application logic instead of bothering with non-functional properties such as scalability or

[*]Corresponding author: pe@sec.uni-passau.de

fault-tolerance. A suitable middleware that provides resilience capabilities against faults and attacks can reduce the complexity from the developer's perspective. Both the longevity of IoT systems and faster development cycles require higher flexibility: it should be possible to customize existing IoT installations to match current needs and conditions, e.g., by adding, removing, moving or updating components. However, due to the size and complexity of the systems, a manual design of a valid system is an immense challenge, so that the developer must be supported by the use of tools.

We address the increasing need for reliability and automation both during the development and operation of IoT systems, as well as emerging development and deployment issues. We outline key ideas of SORRIR which bundles mechanisms that support a self-organizing, resilient planning and execution environment for IoT services. It assists the design of intrusion-tolerant, reliable and available IoT services, allowing the simple and flexible deployment of these services through the composition of IoT components, and ultimately ensuring their secure and reliable performance.

The remainder of this paper is structured as follows: Section 2 sketches the state-of-the-art for IoT architecture, fault-tolerance in IoT and security in IoT and incorporates related work. In Section 3 the SORRIR system model, goals, and architecture are presented before going into more detail on the resilience mechanisms, security aspects and monitoring of our approach. Finally, we outline future work and draw a conclusion in Section 4.

2 BACKGROUND AND RELATED WORK

This section captures the state-of-the-art in IoT architecture, fault-tolerance and security as well as elaborating on related work in these areas.

2.1 IoT architectures

IoT systems spread out over a large physical area and involve multiple different technologies starting from sensors and actors in the field over network components used to transfer collected data and control signals from one participant to another using various protocols such as MQTT and CoAP. Data are processed and stored, mixed and aggregated on their way from the edges of the network to the core.

The ISO/IEC 30141 standard defines an Internet-of-Things reference architecture [7] on a high level basis. According to this standard, the distinctive features of IoT systems are the high degree of distribution within the system, the physical separation and remote connection of sub-systems (sites). Each site consists of a heterogeneous network and can also incorporate proximity networks. It comprises the IoT devices, IoT gateways and services as the core technical building blocks of an IoT landscape. Particularly, an IoT device is defined as a *digital entity that bridges real-world [...] and digital* [7]. Further, IoT gateways are entities that *connect [...] IoT devices to a wide-are network* [7]. Applications are formed from services with well-defined interfaces. Devices, services, and gateways expose network-based endpoints to connect to other entities in the system. The standard also introduces various domains that encapsulate different aspects of an IoT system: The sensing and controlling domain comprises devices and gateways; the application service domain provides user access to the IoT system; the operation and management domain provides the necessary means to maintain the overall health of the system including monitoring, system optimization, and deployment of application components. Besides proximity networks and wide-area networks, the standard defines intranets connecting parts of the service landscape.

Guth et al. compare different IoT platforms based on their reference architecture [5]. The used architecture comprises devices hosting sensors and actuators; gateways connecting devices with WAN, translating commands and data, as well as preliminary data processing; and IoT middleware that forms an integration point for different devices and applications. The IoT application represents business logic on top of the IoT middleware. The paper also shows that this architecture is mappable to many different real-world systems including AWS IoT, OpenMTC, and FIWARE.

Carrez et al. [3] establish a reference architecture for federated IoT platforms and introduce several additional entities that ensure privacy and data integrity between participants. They use raw-data providers, service providers, knowledge producers and experimenters as roles in their framework. Those roles correlate to IoT devices, IoT gateways, execution sites and applications respectively.

Regarding IoT devices, the NIST Network of Things [16] basically distinguishes between sensors and aggregators. Aggregators have the task to transform raw data from multiple sensors to aggregated data. Sensors can be grouped into movable clusters. Different clusters can share sensors and the mapping from sensors to clusters can change at run-time. Aggregators may communicate with other aggregators and external utilities can consume data from the aggregators. The paper stresses that reliability is important for all layers of such a system including the communication channels.

Naik has identified MQTT, CoAP, AMQP, and HTTP as primary IoT communication protocols outside proximity networks [11]. All of them are located on top of IP.

2.2 Fault-Tolerance in the IoT

Contrary to the common approach of achieving fault-tolerance by using triple modular redundancy and majority voting, in IoT systems the implementation of a threefold hardware installation is often neither practical nor (financially) feasible according to Terry et al. [13]. As a solution the authors argue that, on the one hand, active replication can be relinquished since IoT devices (sensors and actuators) mostly fail-stop. As a consequence of that, the duplication of devices is sufficient for achieving a fault-tolerant operation. On the other hand, devices capable of similar events or actions can be used to fulfill the tasks of faulty ones, which obviates the need for duplication of devices. This can be realized by enabling the IoT platform to discover and select nearby devices with similar events and actions such that a fault free operation of the IoT application can be provided [13]. Zhou et al [18] follow a similar approach by providing fault-tolerance in service-oriented IoT architectures by utilizing the concept of virtual services. Here, the data from more than one sensor device is incorporated into a virtual service, such that an actual service on a faulty device [18] can be compensated. Tsigkanos et al. [14] introduce a roadmap with state-of-the-art techniques to establish resilience in the face of disruption as well as future directions in that area, e.g., abstracting business logic management from infrastructure capabilities and autonomous control and self-healing. Moreover, fault-tolerance on the communication side can be achieved by constructing, recovering and selecting k-disjoint routes that guarantee connectivity even after the failure of up to k-1 paths [6].

However, there are also open challenges and issues which have to be addressed in order for fault-tolerance in IoT to work. Tsigkanos et al. [14] identify challenges for engineering resilience in IoT which comprise e.g., the growing heterogeneity in software stacks and the lack of mobility of software components between diverse administrative domains and locations. Other issues are concerning the automation of switching devices in case of failures and how best to incorporate application-specific requirements and user preferences into an IoT fault-tolerance scheme [13].

2.3 Security in the IoT

Due to the growing size, raising complexity and implementation in critical infrastructures of IoT systems the security aspect is gaining more and more importance. Vashi et al. [15] introduce security challenges of IoT on three different layers (perception layer, network layer and application layer) and describe countermeasures. The layers correspond to IoT devices at the perception layer, IoT gateways and execution sites at the network layer, and the application itself at the application layer. Each layer faces different security problems starting from physical attacks at the perception layer over Denial-of-Service (DoS) attacks at the network layer to malicious code injection at the application layer. According to the authors the countermeasures to those problems comprise encryption, authentication, authorization, confidentiality, certification and access control.

Mahmoud et al. [9] extend those findings by introducing security challenges for each layer and general properties of security measures which are specific to an IoT system like *Lightweight Solutions* and *Heterogeneity* relating to the fact that IoT devices have limited computational capabilities hence require specialized algorithms. The countermeasures listed in this paper relate to the ones from [15].

3 SORRIR

This section focuses on our approach of realizing a robust and self-organizing IoT execution platform by introducing the system model and terminology followed by our five primary goals forming the building blocks of SORRIR. Moreover, details are given regarding the architecture, the resilience mechanisms, security measures, and monitoring and self-organization of SORRIR.

3.1 System Model and Terminology

The system model and terminology of SORRIR are based on ISO/IEC 30141 [7], whereat the terminology is extended with terminology of cloud and edge computing. Regarding the infrastructure, we differentiate between IoT devices, IoT gateways, and execution sites. The communication model allows for horizontally as well as vertically message flow, e.g., from gateway to gateway and from device to gateway respectively.

IoT devices comprise sensors and actuators which form the perception layer, hence, reside in the field. Their computational and power capabilities are typically low and they exist in a variety of different hardware and software configurations. The communication to IoT gateways is usually performed via a direct WAN or (W)LAN connection or via a mesh network to other IoT devices.

IoT gateways either distribute the messages received from the IoT devices to the execution sites or, depending on the computational capabilities, preprocess and aggregate data and interact with other gateways.

Execution sites provide computational resources for IoT gateways and are accessible via network. Dependent on the platform maintainers configuration those sites can comprise regions of public cloud providers, private clouds and computing clusters. Their higher computing capabilities allow gateways to outsource resource intensive tasks to them and their usually well-defined APIs allow for better roll-out of new functionality.

Applications and components are encapsulating the business logic whereat an application is composed of a set of instantiable components. Those can be directly instantiated on IoT gateways and execution sites and may define communication dependencies between each other.

3.2 Goals

SORRIR comprises the following set of ambitions, each corresponding to a feature that acts as a building block within our platform architecture.

3.2.1 Design of a New Programming Model for Distributed, Resilient IoT-Components. The programming model should decouple the realization of the component logic from resilience properties and thus enable the developer to focus on implementing the application logic while utilizing SORRIR programming interfaces and design patterns. Depending on which interfaces have been implemented (at design-time), the platform can automatically identify which selection of resilience techniques the application can be configured to employ (at run-time).

3.2.2 Detection of Non-Defined, Non-Coherent and Faulty States. Monitoring the infrastructure (edge, core, network) and components should diagnose faulty, inconsistent and undefined states. Information on this should be accessible and easy to understand for both the infrastructure manager and the operator of the application. In addition, it provides the decision-making basis for the SORRIR execution environment to choose the right reaction in respect to self-healing and optimization mechanisms.

3.2.3 Realization of a Configuration Environment. A configuration environment should provide the capability of (i) executing components developed with the given programming model and (ii) flexibly combining them. Additional mechanisms provide the IoT application operator with an easy configuration of the desired reliability, both at the individual component level and at the level of the entire IoT application.

3.2.4 Realization of a Resilient Execution Environment. The resilient execution environment provides resources to run the application created with the configuration environment at the desired resilience level. Depending on the configuration, the execution environment might provide other services besides the specified

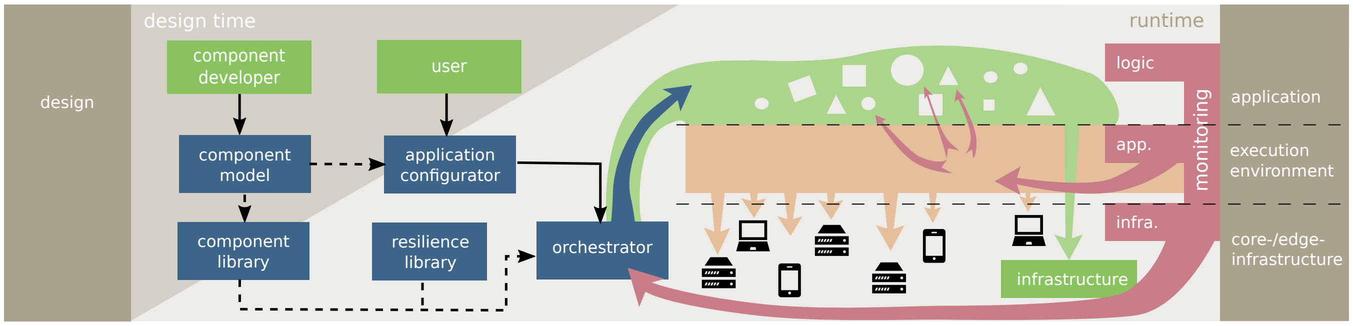

Figure 2: Overview of SORRIR components and their interplay.

components to ensure resilience. E.g., healing processes are initiated as a reaction to error diagnosis to ensure resilience in the long term. Cloud solutions of already established, large providers as well as private clouds with special properties can be flexibly used, as long as they offer the mechanisms required by the execution environment to implement the resilience properties. Our execution environment also covers the edge layer, so that, e.g,. in case of particularly sensitive data or especially time-critical applications, the communication with the cloud is not a necessity.

3.2.5 Vertical Fault-Tolerance. We further want to explore the benefits of novel resilience mechanisms, which leverage the full availability of the IoT platform across different levels - especially the gateway and the cloud - compared to mechanisms that operate within a single layer. For example, functionality could be dynamically migrated between the cloud and gateways as needed, depending on the current state of the network or the available processing power on the gateways.

3.3 SORRIR platform architecture

In order to achieve the goals listed in Section 3.2, the SORRIR platform follows a holistic approach that combines a novel design process with further building blocks such as library containing resilience mechanisms (which allows components to compensate for faults at run-time) or an orchestrator that associates design-time aspects with run-time artifacts.

We distinguish between two roles called *component developer* and *application operator* (or: user of the application configurator). The *component developer* pursues the goal of implementing *business logic*, and thus to create an IoT application that encapsulates this logic in a simple and straightforward way. Further, this developer should not need to have a technical background on non-functional properties of the system, e.g., fault-tolerance aspects. Also, he should not be burdened with the task of implementing a set of fault-tolerance or security mechanisms. However, he can be obliged into implementing a set of interfaces of our IoT programming model. The *design-time* encompasses the process of developing these components. The *application operator* deploys and maintains the platform. He uses the *application configurator* tool to select the desired resilience level. Moreover, he supervises and controls the platform. Overall, our architecture comprises the following building blocks as shown in Figure 2:

- **component model**: we follow the idea of developing IoT applications as components to foster maintainability and re-usability aspects. Components are instantiable. IoT applications can be composed of a set of such components, and developers can focus on developing specific IoT components, hence decreasing complexity.
- **component library:** this library bundles the available components. At execution sites, components will be instantiated and at a specific site, the scale out factor of a component defines the overall number of its instances.
- **application configurator**: a tool that is used by the application operator to fine-granulary configure the resilience degree of each component of the application. The tool outputs an application configuration that determines application structure and communication topology.
- **resilience library:** a library that bundles a broad variety of resilience mechanisms (they match resilience properties which can be configured by the application configurator) and are automatically interwoven into the application components by the orchestrator.
- **orchestrator:** a tool that combines design-time and runtime aspects of the platform. It uses the application configuration to allocate resources on execution sites, incorporate resilience mechanisms that match requirements specified by the operator into the components, and deploys these components on the IoT infrastructure.

Utilizing a new programming model, the SORRIR platforms aims at a separation between the specification of resilience requirements by the developer and the implementation of resilience mechanisms in a distributed run-time platform. Depending on the developer's usage of interfaces and specific design patterns, base classes and APIs provided by the SORRIR programming model, components become capable of supporting distinct fault models. The support for a selection of these models is automatically identified by the application configurator and can be initally selected by the application operator at *design-time* and later adjusted if needed at *run-time*.

In the remaining part of this section we want to briefly summarize our ideas regarding resilience mechanisms that can be employed (§3.4), security mechanisms to pursue specific security goals for our platform (§3.5) and a monitoring module that resports failures and security-related incidents within our infrastructure (§3.6).

3.4 Resilience Mechanisms

Resilience aims at providing a sufficient and comprehensive functionality even in case of faulty system components. Techniques for ensuring resilience are capable of not only tolerating the complete failure of specific subsystems but can also compensate faulty and purposely harmful behavior of components [1]. The SORRIR platform is, as explained in 3.2.4, targeting resilience by incorporating state-of-the-art safety and dependability techniques featuring enhanced fault-tolerance mechanisms, e.g., for tolerating Byzantine faults.

In general those techniques can be categorized into static and dynamic resilience mechanisms. Regarding static mechanisms common techniques are static replication e.g., on the basis of snapshots and backups or redundant execution. However, preventing the impairment of the overall system by the repetitive occurrence of faults, dynamic replication mechanisms, recognizing inconsistencies at run-time and restoring the degree of fault-tolerance on demand, are required. For this purpose, SORRIR supports not only crash fault tolerance (CFT) focusing on the outage of communication channels and nodes but also Byzantine fault tolerance (BFT) known as a practicable method for tolerating unexpected system behavior and attacks. In conjunction with proactive recovery, which will also be provided by SORRIR, this ensures state-of-the-art dynamic replication.

3.5 Security Measures

IoT platforms offer a variety of different security challenges from physical attacks on IoT devices to distributed Denial-of-Service (DDoS) attacks against virtual machines in the cloud. The heterogeneity of IoT systems' device types and software components, their scale and dispersion across different layers tend to increase the attack surface. To counteract on threats, we need to specify security measures to be incorporated into the SORRIR platform.

3.5.1 Authentication. Authentication measures are an essential part of securing IoT systems and are necessary for all participants, e.g., IoT devices, IoT gateways and execution sites. However, the realization can be rather challenging due to the heterogeneity of IoT comprising multiple entities (e.g., devices, humans, services, etc.). As a solution the SORRIR platform is, on the one hand, using already established authentication measures like OpenID where applicable. On the other hand, new authentication measures especially designed for the computationally limited nature of IoT devices should be supported [8, 17].

3.5.2 Authorization. Authorization and authentication go hand in hand in securing IoT since devices and services shall only access certain data intended for them. Hence, SORRIR is incorporating specialized IoT authorization techniques similar to OAuth but not limited to certain communication protocols.

3.5.3 Confidentiality, Integrity and Availability (CIA). Encryption is a fundamental security measure when confidentiality is to be maintained. However, IoT systems comprise multiple heterogeneous devices with different computational resources leading to a heterogeneous set of possible cryptography algorithms, in particular, lightweight primitives need to be considered for constrained IoT device classes. SORRIR is taking those requirements into account and aims to offer a variety of different cryptographic primitives suited for a broad hardware spectrum.

Furthermore, we want to support industrial standards, e.g., to ensure integrity as well as confidentiality our approach includes, among others, using TLS since it provides end-to-end encryption for confidential message exchange and message authentication codes (MACs) for safeguarding the integrity of messages and origin authentication.

Besides ensuring confidentiality and integrity, availability is also a concern since the deployment of IoT systems in critical infrastructures requires IoT devices and services to deliver a certain quality of service. Hence, the SORRIR platform aims to promote availability by providing a set of mechanisms such as active replication or traffic filtering. This way, we increase an attacker's necessary efforts (e.g., if the attacker carries out attacks like DDOS, which are a common threat to availability) to harm the system.

3.6 Monitoring and Self-Organization

On top of resilience and security measures, a monitoring component, providing information for self-organization, is incorporated into the SORRIR approach. The main purpose of this component is to monitor the overall IoT system for both security relevant incidents and resilience related events. Here, we follow a holistic approach which improves the monitoring capabilities in heterogeneous distributed IoT infrastructures (edge and cloud) by the integration of a set of highly diverse possibilities for introspection techniques for IoT systems. On top of that our components provide an evaluation of the monitoring data in order to recognize unexpected and inconsistent states. Contrary to the common state-of-the-art SORRIR is aiming at advanced techniques for adaption and self-organization to enable a proper reaction to component- and communication-loss as well as attacks.

4 CONCLUSIONS AND FUTURE WORK

IoT infrastructures tend to grow both in scale and complexity while the heterogeneity of devices increases and software development cycles are shrinking. As IoT systems become part of critical infrastructures such as factories of the future or eHealth applications, requirements regarding the resilient and secure execution of services on these platforms emerge.

In this position paper we outlined ideas to address this problem and support the development and operation of secure and resilient IoT services by providing a middleware tailored to separate the fulfillment of resilience properties from the development of the actual business logic. We outlined the goals of our project and then sketched the architecture of our platform, SORRIR, which is a self-organizing IoT platform for dependable and secure service execution composed of several building blocks that capture the relation between design-time and runtime.

We note, that there is still much work left until our approach is completely realized. Starting with the elaboration of all details regarding the different aspects of our platform over the realization of the individual components to the finished implementation. Thereby, special effort has to be invested into coping with challenges related to IoT especially regarding the heterogeneity and hardware limitations of devices and gateways.

ACKNOWLEDGMENTS

The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany.

REFERENCES

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

[2] N. Benamar, A. Jara, L. Ladid, and D. E. Ouadghiri. 2014. Challenges of the Internet of Things: IPv6 and Network Management. In *8th Int. Conf. on Innov. Mobile and Internet Serv. in Ubiq. Comp.* 328–333. https://doi.org/10.1109/IMIS.2014.43

[3] F. Carrez, T. Elsaleh, D. Gómez, L. Sánchez, J. Lanza, and P. Grace. 2017. A Reference Architecture for federating IoT infrastructures supporting semantic interoperability. In *2017 European Conference on Networks and Communications (EuCNC)*. 1–6. https://doi.org/10.1109/EuCNC.2017.7980765

[4] D. Fuller. 2014. System design challenges for next generation wireless and embedded systems. In *Design, Automation Test in Europe Conf. Exhibition (DATE)*. 1–1. https://doi.org/10.7873/DATE.2014.014

[5] Jasmin Guth, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Lukas Reinfurt. 2016. Comparison of IoT Platform Architectures: A Field Study based on a Reference Architecture. In *2016 Cloudification of the Internet of Things (CIoT)*. IEEE, 1–6. https://doi.org/10.1109/CIOT.2016.7872918

[6] M. Z. Hasan and F. Al-Turjman. 2017. Optimizing Multipath Routing With Guaranteed Fault Tolerance in Internet of Things. *IEEE Sensors Journal* 17, 19 (Oct 2017), 6463–6473. https://doi.org/10.1109/JSEN.2017.2739188

[7] ISO/IEC 130141:2018 2018. *Internet of Things (IoT) – Reference Architecture*. Standard. International Organization for Standardization, Geneva, CH.

[8] Jun-Ya Lee, Wei-Cheng Lin, and Yu-Hung Huang. 2014. A lightweight authentication protocol for internet of things. In *2014 International Symposium on Next-Generation Electronics (ISNE)*. IEEE, 1–2.

[9] Rwan Mahmoud, Tasneem Yousuf, Fadi Aloul, and Imran Zualkernan. 2015. Internet of things (IoT) security: Current status, challenges and prospective measures. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*. IEEE, 336–341.

[10] Subhas Chandra Mukhopadhyay. 2014. *Internet of Things, Challenges and Opportunities*. Smart Sensors, Measurement and Instrumentation, Vol. 9. Springer Int. Publishing.

[11] N. Naik. 2017. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In *2017 IEEE International Systems Engineering Symposium (ISSE)*. 1–7. https://doi.org/10.1109/SysEng.2017.8088251

[12] S. Qanbari, S. Pezeshki, R. Raisi, S. Mahdizadeh, R. Rahimzadeh, N. Behinaein, F. Mahmoudi, S. Ayoubzadeh, P. Fazlali, K. Roshani, A. Yaghini, M. Amiri, A. Farivarmoheb, A. Zamani, and S. Dustdar. 2016. IoT Design Patterns: Computational Constructs to Design, Build and Engineer Edge Applications. In *IEEE 1st Int. Conf. on Internet-of-Things Design and Impl. (IoTDI)*. 277–282. https://doi.org/10.1109/IoTDI.2015.18

[13] D. Terry. 2016. Toward a New Approach to IoT Fault Tolerance. *Computer* 49, 8 (Aug 2016), 80–83. https://doi.org/10.1109/MC.2016.238

[14] Christos Tsigkanos, Stefan Nastic, and Schahram Dustdar. 2019. Towards resilient Internet of Things: Vision, challenges, and research roadmap. In *Proc. 39th IEEE Int. Conf. Distrib. Comput. Syst.(ICDCS)*. 1–11.

[15] S. Vashi, J. Ram, J. Modi, S. Verma, and C. Prakash. 2017. Internet of Things (IoT): A vision, architectural elements, and security issues. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. 492–496. https://doi.org/10.1109/I-SMAC.2017.8058399

[16] Jeffrey Voas. 2016. *Networks of 'Things'*. NIST Special Publication 800-183. National Institutee of Standards and Technology. https://doi.org/nistpubs/SpecialPublications/NIST.SP.800-183.pdf

[17] Quangang Wen, Xinzheng Dong, and Ronggao Zhang. 2012. Application of dynamic variable cipher security certificate in internet of things. In *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, Vol. 3. IEEE, 1062–1066.

[18] S. Zhou, K. Lin, J. Na, C. Chuang, and C. Shih. 2015. Supporting Service Adaptation in Fault Tolerant Internet of Things. In *2015 IEEE 8th International Conference on Service-Oriented Computing and Applications (SOCA)*. 65–72. https://doi.org/10.1109/SOCA.2015.38