Journal of Internet Services and Applications

RESEARCH Open Access

Self-adaptive architectures in IoT systems: a systematic literature review

Iván Alfonso[1,2*], Kelly Garcés[1], Harold Castro[1] and Jordi Cabot[2,3]

*Correspondence: id.alfonso@uniandes.edu.co
[1]Department of Systems and Computing Engineering, Universidad de los Andes, Bogotá, Colombia
[2]Universitat Oberta de Catalunya, Barcelona, Spain
Full list of author information is available at the end of the article

Abstract

Over the past few years, the relevance of the Internet of Things (IoT) has grown significantly and is now a key component of many industrial processes and even a transparent participant in various activities performed in our daily life. IoT systems are subjected to changes in the dynamic environments they operate in. These changes (e.g. variations in bandwidth consumption or new devices joining/leaving) may impact the Quality of Service (QoS) of the IoT system. A number of self-adaptation strategies for IoT architectures to better deal with these changes have been proposed in the literature. Nevertheless, they focus on isolated types of changes. We lack a comprehensive view of the trade-offs of each proposal and how they could be combined to cope with simultaneous events of different types.

In this paper, we identify, analyze, and interpret relevant studies related to IoT adaptation and develop a comprehensive and holistic view of the interplay of different dynamic events, their consequences on QoS, and the alternatives for the adaptation. To do so, we have conducted a systematic literature review of existing scientific proposals and defined a research agenda for the near future based on the findings and weaknesses identified in the literature.

Keywords: Internet of things, Dynamic architecture, Dynamic environment, Self-adaptation

1 Introduction

The Internet of Things (IoT) represents a global environment that interconnects the internet with a large number of (cyber-)physical objects such as sensors, vehicles, cell phones, household appliances, cameras, and machines [1]. IoT aims to facilitate communication and exchange of information to enable new forms of interaction between things and people [2]. The IoT has transformed and improved the activities we carry out on a daily basis in various aspects such as transport, agriculture, healthcare, industrial automation, and emergency response.

In most IoT systems, it is critical to guarantee the quality of service (QoS) to the users, according to the requirements of the application domain. For example, in continuous monitoring systems, a decrease in the quality could generate wrong or late alerts stemming from the monitored system (imagine the effects of late alerts in monitoring systems in hospitals). Several metrics to measure the quality of service of IoT systems have been proposed. Singh and Baranwal [3] classify these into three groups: (1) QoS of communication to measure the quality of network services with metrics such as jitter, bandwidth, performance and efficiency, and network connection time; (2) QoS of things with metrics such as availability, reliability, response time, and security; and (3) QoS of computation to measure computational performance with metrics such as scalability, dynamic availability, and response time.

Satisfying these commitments is challenging due to the dynamic nature of the environment surrounding the IoT system. All types of unexpected events (such as unstable signal strength, growth in the number of connected devices, and software and hardware aging [4, 5]) can happen at any time, posing a risk to the QoS. This unpredictable behavior of the system can be addressed at runtime by self-adaptive systems. A self-adaptive system modifies its behavior at runtime in response to changes in the system or its environment to ensure a certain quality level [6].

Self-adaptive systems have been studied for several decades. Wong et. al. [7] classify the evolution of self-adaptive systems into stages. In the first stage (1990-2002), the authors proposed a theoretical model of self-adaptive systems [8, 9]. The second stage (2003-2005) was dominated by studies proposing novel perspectives but without concrete implementations. In particular, Rainbow [10] was released as one of the foundations of self-adaptive systems. Rainbow, used as a standard, is a framework that involves principles of self-configuration, self-optimization, self-healing, and self-protection in computer systems. In the third stage (2006-2010), research was focused on autonomous and self-adaptive web services. Runtime solutions predominated over design time solutions. For example, in modesl@runtime [11] (the use of software models for adaptive mechanisms to manage complexity in runtime environments) was introduced. The last stage (2011-2020) shows a transition between the domains of research interest. The adaptability of IoT systems and Infrastructure as a Code (IaaS) become the focus. However, the exponential increase and variability of IoT devices, and the unpredictable behavior of the environment introduces self-adaptation challenges to maintain quality levels.

This evolution show an increase in the complexity of IoT architectures. Traditional IoT systems consisted of two main layers: (1) the device layer, or physical layer, composed by devices (sensors, actuators, and network devices) that generate and send data to the cloud for processing; and (2) the cloud layer made up of servers that store the application logic and process the data generated in the physical layer. This architecture has been used for several years. An advantage of such an architecture is that it reduces maintenance costs and application development efforts [5]. However, new requirements are pushing changes in the design of IoT systems. Centralized cloud-based architectures cannot properly support the constraints of certain IoT applications. The two main reasons for this are: (1) bandwidth limitations and excessive data transmission costs from system devices to the cloud; and (2) communication delay between devices and the cloud, mainly in applications that require real-time data analysis [12]. These limitations prevent the development of IoT systems that require low-latency responses for large volumes of data to be processed.

Recently, fog and edge computing have emerged as architectures to address some of the challenges posed by the centralized architecture of cloud-based IoT systems. These architectures are implemented as a layer called edge/fog in the system architecture (Fig. 1) between the physical layer and the cloud layer. Fog computing is performed on

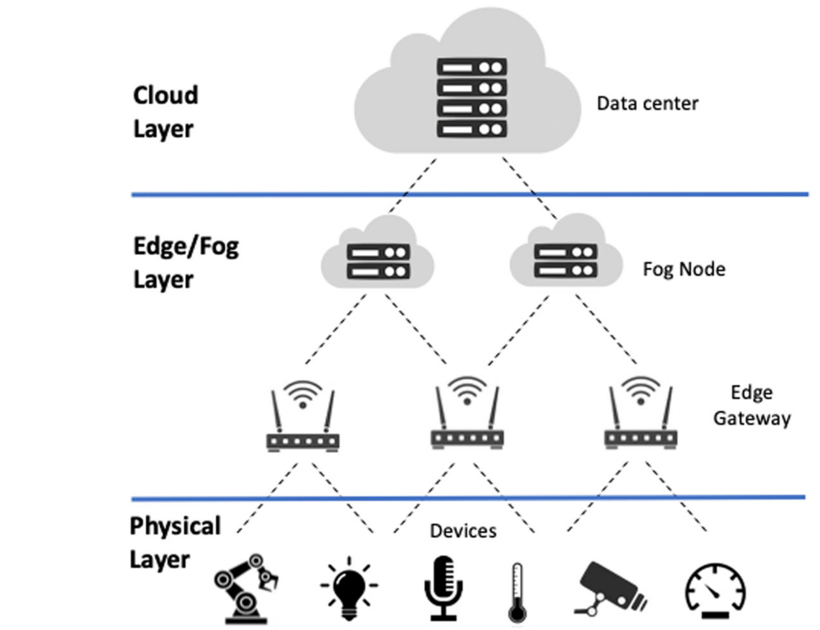

Fig. 1 Decentralized architecture for IoT systems

the system's fog nodes while edge computing is performed on edge devices, e.g. gateways. Nevertheless, both fog and edge computing have a similar purpose: to reduce data travel, latency, and bandwidth consumption, moving computation, storage, communication, control, and decision making closer to the network edge where data is being generated; i.e., in the physical layer [13]. Many authors claim that edge and fog are the same, while other authors indicate that fog is a part of edge computing [14]. Nevertheless, the combination of edge, fog, and cloud computing makes it possible to build a distributed architecture to guarantee QoS compliance in IoT applications. Fog and edge computing offer advantages mainly in terms of latency and bandwidth usage, since they allow data processing at the edge rather than in the cloud.

Despite the advantages of this more flexible architecture, QoS is still impaired by the dynamic events mentioned above. We will use an illustrating example to give a better understanding of the dynamic events and implications: an IoT system for monitoring and controlling ventilation in underground mines, consisting of hundreds or thousands of sensors and actuators that monitor the mining atmosphere, primarily to ensure the safety of workers. The system monitors the location of workers within the mine, and monitors physical variables such as the concentration of toxic and explosive gases, temperature, and oxygen. It also generates alerts and controls mine ventilation with actuators such as fans and alarms to prevent accidents like poisoning or explosions. However, in daily operation, this system could present problems due to the dynamic environment in the following cases:

- The frequency of monitoring toxic and explosive gases may change at certain times, depending on the amount of work and people in the mine. Sensors increase the frequency of monitoring and sending data in the areas of the mine where activity is recorded, which increases bandwidth consumption. On the other hand, when the system detects that the concentration of a hazardous gas exceeds the permitted thresholds, the sensing devices increase their monitoring frequency in the area or throughout the mine. Bandwidth consumption can increase significantly, causing system failures that impact QoS.

- Some devices in the physical layer of the system (sensors and actuators) change their location within the mine. For example, a sensor can be moved from an abandoned area to work areas where new excavations are made in the mine. However, the system should have the ability to configure the network semi-automatically and provide resources to ensure the mobility of system devices.
- Finally, wireless devices suffer from aging software [15] that sometimes induces problems in the system. For example, it is common to update service or application software to improve security, solve bugs, or improve application performance. Updating device software is not a trivial task when it comes to an IoT system with hundreds or thousands of devices. In addition, applications and services running on the edge/fog and cloud layers also need to be updated by developers.

These are just a few of the problems that can be caused by the dynamic environment of an IoT system in the mining domain. However, for other domains, there may be more dynamic environmental events that impact QoS even for systems with distributed architectures. In later sections of this paper, we will discuss dynamic environmental events in other domains.

Most studies found in the literature individually address particular dynamic events in IoT systems and propose specific strategies to ensure QoS. However, each proposal provides only a partial view (and solution) to the self-adaptation problem. It is necessary to have a comprehensive view of all the different kinds of events (for example, environmental) addressed in the literature. Indeed, we need: (1) a classification of the dynamic events that impact the QoS; (2) a classification of the self-adaptation strategies of the IoT system architecture; and (3) some gaps and challenges in the proposed strategies and their relationships.

In this sense, this paper aims to provide such comprehensive overview of the current state of the art in IoT adaptation. To do so, we conducted a systematic literature review of the proposals in this domain.

The remainder of the paper is organized as follows: Section 2 presents the literature review method, including the research questions, search and selection process, inclusion and exclusion criteria, quality assessment, data extraction, and data analysis. Sections 3 and 4 address research questions and limitations of current research. Section 5 presents threats to the validity of our literature review work. We present the summary and future directions opportunities in Section 6. In Section 7, we analyze related work. Finally, Section 8 presents the conclusions of the review.

2 Method

A systematic literature review (SLR) is a methodology used for the identification, analysis, and interpretation of relevant studies to address specific research questions [16]. Given the complexity of the IoT domain, we decided that an SLR was the best way to systematically reach a comprehensive and fair assessment of this topic. Our SLR consists of six main steps and is based on the methodology proposed by Kitcheham et al. [17]. The steps followed for this SLR are illustrated in Fig. 2 and documented below.

2.1 Research questions

Our goal is to identify the dynamic environmental events in the physical and edge/fog layers of an IoT system that could impact its QoS and therefore require the trigger of self-adaptations of the system. In addition, we classify the strategies to achieve this self-adaptation. For this purpose, our SLR addresses the following two research questions:

- RQ1. Which dynamic events present in the edge/fog and physical layers are the main causes for triggering adaptations in an IoT system?

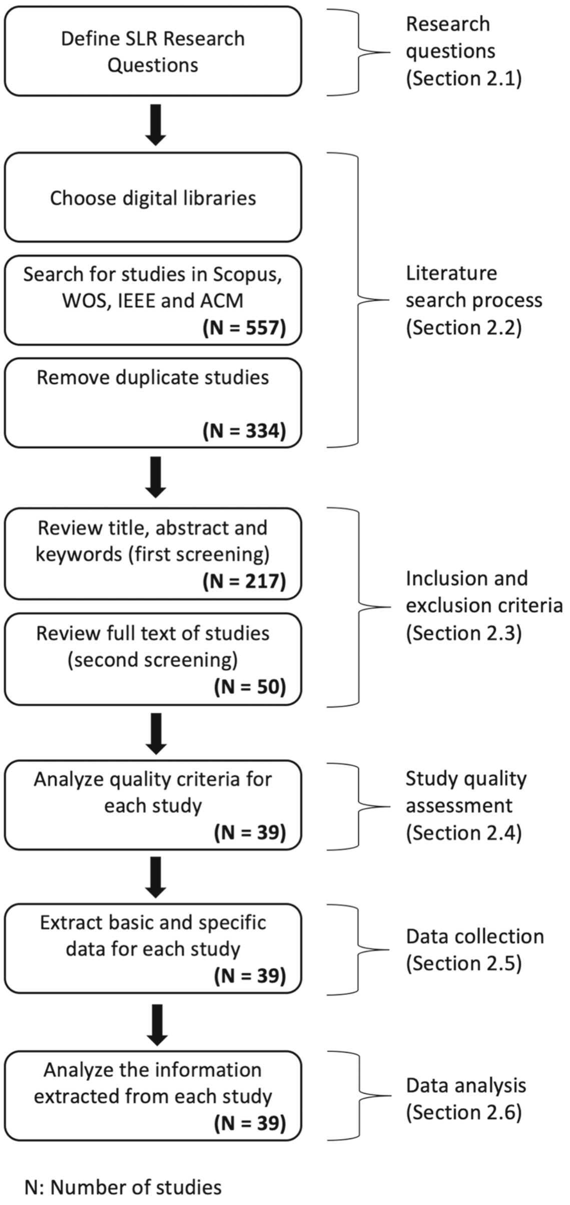
Define SLR Research Questions
Research questions (Section 2.1)
Choose digital libraries
Search for studies in Scopus, WOS, IEEE and ACM (N = 557)
Remove duplicate studies (N = 334)
Literature search process (Section 2.2)
Review title, abstract and keywords (first screening) (N = 217)
Review full text of studies (second screening) (N = 50)
Inclusion and exclusion criteria (Section 2.3)
Analyze quality criteria for each study (N = 39)
Study quality assessment (Section 2.4)
Extract basic and specific data for each study (N = 39)
Data collection (Section 2.5)
Analyze the information extracted from each study (N = 39)
Data analysis (Section 2.6)
N: Number of studies

Fig. 2 Summary of the SLR protocol

Table 1 Search queries

	Search query
SQ1	("fog" OR "edge" OR "osmotic") AND ("IoT" OR "internet of things" OR "cyber-physical") AND ("architecture") AND ("adapt*" OR "self-adapt*")
SQ2	"fog" AND "adapt*" AND "architecture" AND "orchestration"
SQ3	("orchestration" OR "choreography") AND "fog" AND "architecture" AND "dynamic"

- RQ2. How do existing solutions adapt their internal behavior and architecture in response to dynamic environmental events in the edge/fog and physical layers to ensure compliance with its requirements?

2.2 Literature search process

The search process step had three phases [17]: first, we selected the digital libraries; next, we defined the search queries; and finally, we carried out the search and discarded the repeated studies. This section details these phases.

2.2.1 Digital libraries

We chose four digital libraries for our search: Scopus, Web of Science (WOS), IEEE Explore, and ACM. These libraries are frequently updated and contain a large number of studies in the area of this research.

2.2.2 Search queries

As shown in Table 1, we defined three search queries. Keywords such as *IoT* and *cyber-physical* were used to focus the search towards these types of systems; *adapt* or variations of this word (e.g., *adaptation*) and *self-adapt* are keywords to identify adaptation strategies (RQ2) and the events that trigger them (RQ1); *architecture* and *dynamic* keywords to identify studies focused on adaptations on system architecture; *fog*, *edge*, and *osmotic* keywords to retrieve studies using distributed architectures with fog, edge, and osmotic computing (a paradigm to support the efficient execution of IoT services and applications at the network edge [18]); and *orchestration* or *choreography* keywords, two resource management techniques at the fog layer of a system. We searched for matches of the aforementioned words in the title, abstract and keywords of the articles.

2.2.3 Search results

Table 2 shows the search results; we obtained 557 studies after applying the three search queries, out of which 223 were duplicates, for a total of 334 studies.

Table 2 Studies per digital library

Digital library	Studies found
Scopus	229
Web of Science (WOS)	120
IEEE	176
ACM	32
Total	557
Total without duplicates	334

2.3 Inclusion and exclusion criteria

To screen and obtain the primary studies that address the research questions, we defined inclusion and exclusion criteria. We applied two screening phases: in the first screening of the titles, abstracts and keywords, we used three exclusion criteria, to exclude 117 out of the 334 studies. Then, in the second filter we analyzed the full texts, and we discarded 170 additional studies. Finally, using Snowballing to check the list of study citations we included three additional studies, for a total of 50 studies (see Fig. 2). The inclusion and exclusion criteria for each screening phase are presented below.

First screening:

- (Exclusion) It is not a primary study. Literature reviews are discarded.
- (Exclusion) It is not a journal, conference or workshop paper.
- (Exclusion) The paper is written in a language other than English

Second screening:

- (Inclusion) The study addresses a dynamic event in IoT systems that impacts QoS.
- (Inclusion) The study proposes, takes advantage or analyzes a strategy of self-adaptation of architecture for IoT systems.

2.4 Quality assessment

The quality assessment step consists of reading the studies in detail, and answering the assessment questions to get a quality score for each study. We have defined 5 quality assessment questions as follows:

- QA1. Are the aims clearly stated? (Yes) the purpose and objectives of the research are clear; (Partly) the aims of the research are stated, but they are not clear; (No) The aims of the research are not stated, and these are not clearer to identify.
- QA2. Is the research compared to related work? (Yes) the related work is presented and compared to the proposed research; (Partly) the related work is presented, but the contribution of the current research is not differentiated; (No) the related work is not presented.
- QA3. Is there a clear statement of findings and do they have theoretical support? (Yes) the findings are explained clearly and concisely, and are supported by a theoretical foundation; (Partly) the findings are clearly explained, but they lack theoretical support; (No) findings are not clear and have no foundation or theoretical support.
- QA4. Do the researchers explain future implications? (Yes) the author presents future work; (No) future work is not presented.
- QA5. Has the proposed solution been tested in real scenarios? (Yes) The solution is tested in a real scenario; (Partly) the solution is tested in a particular test bed; (No) the solution is not tested in any scenario.

The score given to each answer was: Yes = 1, Partly = 0.5, and No = 0. We calculated the quality score for each study and excluded those that scored less than 3, to select the primary studies that would be used for data extraction and analysis. We analyzed 50 studies and excluded eleven because they obtained a quality score of less than three. In total, we have obtained 39 primary studies for the remaining steps of this SLR, and the quality scores for each is presented in Table 4.

2.5 Data collection

The extracted information was stored in an Excel spreadsheet. Table 3 shows the Data Collected (DC) for each study and the research question addressed. First, we extracted standard information such as title, authors, and year of publication (DC1 to DC4). Second, we extracted relevant information to address the research questions defined in Section 2.1. DC5 records the environmental event addressed by the study, and this information is used to address research question RQ1. DC6 to DC10 are data collected about proposed solutions and strategies to achieve self-adaptations in the IoT system, and this information is used to address research question RQ2.

2.6 Data analysis

Table 4 presents the list of the 39 studies relevant to this SLR, with the following information: the assigned identification number (ID), the author, the type of publication, the year of publication, the answers to the quality questions, and the quality score obtained. In the following sections, we will refer to primary studies by the assigned ID code.

From the standard information extracted from the papers, we can note that the relevant publications for this SLR are relatively recent. The largest number of studies were published in recent years: 12 studies from 2019, 16 studies from 2018, 7 studies from 2017, 3 studies from 2016, and one study from 2015 (see Fig. 3). As for the type of publication, 25 are conference publications, 10 are journal publications, and 4 are workshop publications.

3 RQ1: which dynamic events present in the edge/fog and physical layers are the Main causes for triggering adaptations in an ioT system?

In this section, we present the information and results obtained from the analysis of the data extracted to address research question RQ1. The dynamic events that force an adaptation in the IoT system are presented and discussed in Section 3.1. Finally, we discuss monitored QoS metrics for detecting dynamic events in Section 3.2.

3.1 Dynamic events

Table 5 provides an overview of the answer to research question RQ1. The table presents a classification of the dynamic environmental events present in the edge/fog and physical layers and the studies that addressed that event. We propose this list of events that we have obtained from the detailed analysis of the studies. We then classify each study according to the event it addresses. Strategies for adapting the system in response to these events are discussed in Section 4.

Table 3 Data collection

#	Field	RQ
DC1	Author	N/A
DC2	Title	N/A
DC3	Year	N/A
DC4	Publication venue	N/A
DC5	Environmental event addressed by the solution	RQ1
DC6	Favored quality attributes	RQ2
DC7	Adaptation strategies and techniques	RQ2
DC8	Architecture description	RQ2
DC9	Architectural styles and patterns	RQ2
DC10	Key responsibilities of architectural components	RQ2

Table 4 Studies

ID	Author	Type	Year	QA1	QA2	QA3	QA4	QA5	QA Score
S1	Young, R. et al. [19]	Conference	2018	Y	Y	Y	Y	P	4.5
S2	Wang, J. et al. [20]	Workshop	2017	Y	Y	Y	N	P	3.5
S3	Muñoz, R. et al. [21]	Article	2018	Y	Y	Y	N	P	3.5
S4	Cheng, B. et al. [22]	Conference	2015	Y	Y	Y	Y	N	4
S5	Kimovski, D. et al. [23]	Conference	2018	Y	Y	Y	Y	P	4.5
S6	Young, R. et al. [24]	Conference	2018	Y	Y	Y	Y	P	4.5
S7	Tseng, C. et al. [25]	Conference	2018	Y	N	Y	Y	P	3.5
S8	Peros, S. et al. [26]	Conference	2018	Y	Y	Y	Y	P	4.5
S9	Rausch, T. et al. [27]	Conference	2018	Y	Y	Y	N	Y	4
S10	Pahl, C. et al. [28]	Conference	2018	Y	Y	Y	N	P	3.5
S11	Lorenzo, B. et al. [29]	Article	2018	Y	Y	Y	Y	N	4
S12	Prabavathy, S. et al. [30]	Article	2018	Y	Y	Y	Y	P	4.5
S13	Yigitoglu, E. et al. [31]	Conference	2017	Y	Y	Y	Y	P	4.5
S14	Morabito, R. et al. [32]	Workshop	2017	P	P	Y	Y	P	3.5
S15	Desikan, K. S. et al. [33]	Workshop	2017	Y	Y	Y	N	P	3.5
S16	de Brito, M. S. et al. [34]	Conference	2017	Y	Y	Y	Y	N	4
S17	Velasquez, K. et al. [35]	Conference	2017	Y	P	Y	Y	P	4
S18	Flores, H. et al. [36]	Conference	2017	Y	N	Y	Y	P	3.5
S19	Pizzolli, D. et al. [37]	Conference	2016	Y	N	P	Y	P	3
S20	Montero, D. et al. [38]	Conference	2016	Y	Y	Y	Y	P	4.5
S21	Chen, L. et al. [39]	Article	2018	Y	Y	Y	Y	Y	5
S22	Mass, J. et al. [40]	Conference	2018	Y	Y	Y	Y	Y	5
S23	Li, X. et al. [41]	Article	2018	Y	Y	Y	N	Y	4
S24	Suganuma, T. et al. [42]	Article	2018	Y	Y	Y	Y	Y	5
S25	Deng, G. et al. [43]	Conference	2018	Y	Y	Y	N	Y	4
S26	Sami, H. et al. [44]	Conference	2018	Y	Y	Y	Y	Y	5
S27	Wu, D. et al. [45]	Conference	2019	Y	Y	Y	N	Y	4
S28	Skarlat, O. et al. [46]	Conference	2019	Y	Y	Y	Y	Y	5
S29	Mechalikh, C. et al. [47]	Conference	2019	Y	Y	Y	Y	Y	5
S30	Castillo, E. et al. [48]	Conference	2019	Y	P	Y	N	Y	3.5
S31	Breitbach, M. et al. [49]	Conference	2019	Y	Y	Y	Y	Y	5
S32	Torres Neto, J. et al. [50]	Article	2019	Y	Y	Y	Y	Y	5
S33	Theodorou, V. et al. [51]	Workshop	2019	Y	N	Y	Y	P	3.5
S34	Guntha, R. [52]	Conference	2019	Y	Y	Y	N	P	3.5
S35	Jutila, M. [53]	Article	2016	Y	Y	Y	Y	Y	5
S36	Cui, K. et al. [54]	Conference	2019	Y	Y	Y	Y	Y	5
S37	Bedhief, I. et al. [55]	Conference	2019	Y	Y	Y	P	N	3.5
S38	Asif-Ur-Rahman, Md et al. [56]	Article	2019	Y	Y	Y	Y	Y	5
S39	Yousefpour, A. et al. [57]	Article	2019	Y	Y	Y	Y	Y	5

3.1.1 Client mobility

Mobile devices such as cell phones or automobiles produce events in the physical layer of the IoT system, causing challenges to ensure QoS. When the system devices change their location, it is necessary to make network reconfigurations, storage synchronizations, and rescheduling of processes among the edge/fog nodes based on available resources. For example, client mobility is one of the dynamic events present in the example illustrated in Section 1 because it is necessary to move sensors through the tunnels and work areas in the mine.

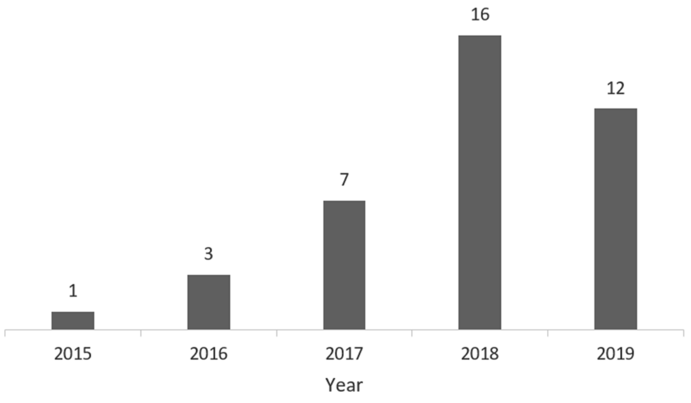

Fig. 3 Number of studies published per year

Eight studies included in this SLR (S5, S9, S10, S17, S19, S20, S22, and S29) address the mobility of clients in the IoT system, which, through different techniques, seek to provide the resources and services at the edge/fog layer to efficiently manage mobility. For example, in S19 a case study of client mobility is addressed. This case consists of patients wearing a device (without 3G/4G connection) to monitor health parameters such as temperature and heart pressure. When the patient leaves his/her home and moves to the hospital, a gateway in the hospital automatically discovers the wearable device and assigns or associates a monitoring service to it.

Client mobility is an event or requirement of IoT systems that poses challenges due to the constant movement of devices, the heterogeneity of communication technologies, and resources, which can be requested on demand simultaneously by multiple devices in different locations [58]. When a device changes location, a series of steps are performed: (1) this change has to be automatically detected; (2) the availability of resources must be guaranteed to deploy the service in the edge/fog nodes in order to manage that device; and (3) in case the device changes location again, it is evaluated if it must be connected to other edge/fog nodes that are closer to obtain better latency.

Monitoring and detecting client mobility depend on the communication protocol between the client (devices) and the edge/fog layer nodes. In scenarios with devices that use low level communication protocols (e.g. Bluetooth and WiFi), it is possible to discover the mobile device in a coverage area and automatically associate it to an IoT gateway, as suggested in S19.

S5 proposes a method of client mobility discovery using MQTT, a standard messaging protocol. MQTT is frequently used in IoT systems due to the advantages of the publication/subscription pattern regarding scalability, asynchronism and decoupling between

Table 5 Dynamic environmental events

ID	Dynamic event	Studies
E1	Client mobility	S5, S9, S10, S17, S19, S20, S22, S29
E2	Dynamic data transfer rate	S3, S6, S7, S11, S15, S18, S19, S21, S26, S32, S39
E3	Important event detected by sensors	S1, S2, S8, S24, S27, S31, S36, S38
E4	Failures and software aging	S4, S13, S14, S16, S28
E5	Network connectivity	S1, S23, S25, S30, S33, S34, S35, S37
E6	Cyber-attacks in IoT applications	S12

clients. MQTT architecture uses one or several nodes called Brokers to manage the network, to receive messages from publishers, and to send messages to subscribers. S9 proposes a distributed QoS-aware MQTT middleware for edge computing, addressing the mobility of clients. When new clients join the system network, a controller searches and maps the broker that offers less latency to the client. In S9, the mobility of clients is detected through the brokers. Every time a client joins the IoT system, it subscribes to one of the MQTT broker's topics. The broker has the capacity to monitor different factors, such as the number of subscribed clients, and detect when a new client subscribes.

3.1.2 Dynamic data transfer rate

The data transmission rate of the devices is another dynamic event that significantly influences the system's QoS. In IoT systems, the data transmission rate from the physical layer to the edge/fog layer may vary depending on the circumstances, objects, or conditions in which the devices are surrounded. The system devices may increase or decrease the frequency of data transmission due to different stimuli. For example, to reduce power consumption in an office building, the sensor devices of the IoT temperature monitoring system are scheduled to raise the data transmission rate during business hours and lower the monitoring frequency and transmission rate during unmanned hours. *Dynamic data transfer rate* event, which is the most discussed topic in the literature, is addressed by eleven studies (S3, S6, S7, S11, S15, S18, S19, S21, S26, S32, and S39) analyzed in this SLR. These studies analyze scenarios with great variability in the data collected and data transfer rates of IoT devices.

The consequences of this dynamic event in IoT systems commonly lead to increased latency and unavailability of system services because increased data volume could congest the network and generate bottlenecks. In addition, this dynamic event implies increase in data to be analyzed or processed by the edge devices, which will likely have limited computer resources. Therefore, the edge nodes could be overloaded with processing work resulting in delays, downtimes, or unavailability.

Two monitoring techniques are used to identify changes in the data transfer rate. The first technique is to watch the computational resources consumed by the edge/fog nodes. The %CPU used is the most commonly used metric by studies (S6, S7, S18, and S32) to identify when an edge/fog node is overloaded due to the increased data it must process. Although the %CPU does not accurately measure the data transfer rate, it is used to detect an increase or decrease in the amount of data to be processed by the node. Increasing the amount of data that arrives at a node to be attended to or analyzed also increases the processing tasks and the %CPU used. The second technique to detect data transfer rate variations is to monitor the network, as proposed by S3, S11, S15, S19, S21. For example, in S3, IoT Flow Monitors are deployed to supervise the average bandwidth of the aggregated IoT traffic and to detect IoT-traffic congestion. Wireshark[1] is one of the tools used to monitor network IoT traffic.

3.1.3 Important event detected by sensors

When an alert or alarm is generated by sensor data in an IoT monitoring system, a set of tasks is triggered to inform the end user and/or control the emergency. These tasks may increase network, processing, and storage consumption at some layer of the sys-

[1]https://www.wireshark.org

tem architecture (physical, edge/fog, and cloud). For example, in a smart city when a vehicular accident is detected by video surveillance cameras, the accident is immediately communicated to authorities (e.g., the police) and medical centers. New processing tasks begin to run in edge/fog nodes or cloud servers: 1) there are increases in the processing and storage of video taken by surveillance cameras; 2) visual alerts are generated to other drivers on the road; 3) tasks are executed to synchronize streetlights to address the emergency and reduce vehicle traffic. Studies S1, S2, S8, S24, S27, S31, S36, and S38 detect important events from sensors. For example, in study S1, changes in weather conditions are detected (e.g., when it starts raining). In study S2, emergencies are detected through a video surveillance system. In study S8, events are detected when one of the IoT devices breaks a rule configured by the user (e.g. when a motion sensor is activated). In study S24, emergency situations such as sudden illness of a group of athletes is detected by wearable vital sensors. Studies such as S31 and S36 do not monitor specific events in the physical layer but propose solutions to address these types of events that commonly require the deployment of additional services at the edge and fog nodes.

System tasks generated by alarms or alerts commonly require additional network, processing, or storage resources. Some systems react to these types of events by increasing monitoring frequencies. Other systems react by deploying or running new tasks on the edge/fog nodes. These tasks and system reactions can affect network connectivity due to increased bandwidth consumption and increased processing at the edge/fog nodes.

The events detected by the system sensors depend on the domain of the application and the IoT system. For some systems, it is important to monitor weather conditions because they significantly influence the performance of wireless networks such as Wi-Fi [59]. Other systems only need to monitor the processes of the domain itself. These events are detected by analyzing the data coming from the devices of the physical layer of the system, and a component of the system architecture is responsible for analyzing the data to detect the event. This component commonly checks that the data sensors are within an expected range. For example, the approach proposed in S1 focuses on a connected vehicle use case where weather conditions are monitored. The vehicle sends data frequently to a central controller to predict driver alertness. When rain is detected, it is assumed that the quality of the communication between the vehicle and the node decreases. The size of the subset of data sent to the central controller is then reduced, selecting only the critical data for driver alertness prediction.

3.1.4 Failures and software aging

The software embedded in the devices, nodes, and servers of an IoT system needs to be updated and redeployed by developers to fix service errors, improve application performance, improve system security, etc. Some upgrades or deployments of system services and application software may involve adaptations to the layers of the system architecture. First, when new services are deployed at edge/fog nodes, it may be necessary to adapt the bindings (e.g., service registry, network topology) established between the services deployed in the nodes and the components that consume said services to ensure the communication. Second, software upgrades are sometimes unsuccessful due to storage, hardware, or connectivity failures. In these cases, the system should detect the problem and fix it. Third, the physical layer and edge/fog devices have limited processing capabilities that may bring risks to successful software upgrades. This implies increased latency and, in some cases, unresponsive services.

S14 proposes a flexible architecture that allows for the network to be dynamically adapted and for the containers to be routed in the fog nodes every time new software deployments could change the operational chain. S28 propose FogFrame, a framework that reacts dynamically to failures or overloads in fog nodes. FogFrame redeploys or redistributes software to reduce node overload or to ensure availability when a fog node fails. S4, S13, and S16 provide frameworks for automating software deployments in fog layer nodes and dynamically adapting the location of containers and the network topology. For example, S13 proposes Foggy, a framework to facilitate and automate the deployment of software in fog nodes of an IoT system. The deployment of applications in fog nodes of the system and the management of resources is handled by an orchestration server. Foggy was based on the use of Docker[2] containers and deployment rules to facilitate dynamic resource provisioning. Container allocation decisions are defined by the developer through deployment rules. For example, the developer can create a rule to deploy a software version to all fog nodes with RAM greater than 2GB. These deployment rules give the developer control over deployment location decisions. The Orchestration Server monitors the use of resources and dynamically adapts the placement of the containers in the nodes. However, Foggy does not detect faults in the containers at runtime, nor does it perform system adaptations such as rollback, redeployment, or movement of software versions.

To detect software failures and aging, it is necessary to constantly monitor the availability of services, nodes, and the status of software containers and/or virtual machines. Two techniques are commonly used to detect these types of failures: (1) ping/echo is an asynchronous request/reply message to determine reachability and the round-trip delay, but this technique is only for nodes interconnected via IP; and (2) heartbeat is a fault detection mechanism that consists of exchanging messages periodically between the node and a monitoring component [60].

3.1.5 Network connectivity

According to S3, the main network requirements for IoT services are low latency, high-speed traffic, large capacity traffic, and massive connections. Although these requirements depend on the domain of the IoT application, most systems require the fulfillment of at least one of these. IoT systems constantly present variations in network connectivity characteristics that make it difficult to meet network requirements. These variations, mainly present in wireless communications, can generate negative effects on the transmission and reception of data between the system's devices, nodes, and cloud servers: (1) out-of-date information due to communication delays; (2) incomplete information due to intermittent or interrupted communication; (3) unavailability of services or system applications due to lost or broken communication.

Network characteristics may be affected by changes in weather [59], variations in system power voltage, wireless signals that interrupt communication, high bandwidth consumption, or other external factors that are normally ignored. To detect deterioration in the quality of communications between system devices and the edge/fog nodes, it is necessary to monitor the network. For example, in S1, a component is designed

[2] https://www.docker.com

to monitor and detect changes in network connectivity. When the monitor detects that the network connectivity is less than 50%, the amount of data sent by the devices from the physical layer to the edge node is decreased and prioritized. To monitor and detect changes in the quality of communication, it is necessary to monitor network metrics such as latency, bandwidth, and lost packets. In S33, S34, and S35 the communication latency between the devices and the nodes or servers that process the data is monitored. When the latency exceeds a predefined threshold, adaptations of data flow reconfiguration and task offloading are performed. In S34, the state of the connection to the cloud is monitored. If the network connection to the cloud is lost, sensor data analysis tasks hosted in the cloud are offloaded to the fog layer nodes.

3.1.6 Cyber-attacks in IoT applications

Although the security topic was not intentionally addressed in this study, we found the work of Prabavathy et al. (S12), which proposes a strategy based on the use of fog computing to detect cyber-attacks. The threats that come from the data of the physical layer devices towards the edge/fog layers and cloud are events induced by attackers that violate the confidentiality, integrity, and availability of the system. In an IoT system, sensors and actuator devices frequently capture and share personal data from our daily life, detect critical physical variables in industrial processes, and control the vehicular flow in a city. The impact of an attack on the devices in any of the layers of the architecture can cause loss of critical information, disasters in the processes that control the system, and unavailability of the system, among others. Therefore, it is essential to ensure the security of the IoT system by designing self-adaptation techniques to defend against attacks.

Bass et al. [60] propose four techniques for detecting attacks on software systems: (1) *detect intrusion* consists of comparing network traffic patterns with known malicious behavior patterns stored in a database; (2) *detect service denial* consists of comparing network traffic entering a system with historical profiles of known denial of service attacks; (3) *verify message integrity* consists of using checksum or hash values to validate the integrity of message information; and (4) *detect message delay* consists of detecting potential man-in-the-middle attacks. These strategies can be adapted to detect attacks on IoT systems. In particular, S12 detects intrusion by implementing an Extreme Learning Machine (ELM) algorithm on the system's fog nodes. ELM is a fast learning algorithm for a hidden single-layer neural network [61], it is suitable for real-time applications due to the low performance of problem solving. In S12, each data packet that is sent from the physical layer to the fog layer is analyzed by the algorithm. This algorithm can detect attacks from different categories including denial of service, user to root, probe-response, and remote to local.

3.2 QoS metrics monitored

Monitoring is an important task to detect dynamic events in the IoT systems. These events are detected by analyzing metrics about node resource consumption (such as CPU, memory, and energy consumption), network behavior (such as bandwidth consumption and communication latency), and availability. Table 6 presents the monitored metrics to detect the dynamic events for each study. The resource consumption in the edge/fog nodes is the most monitored feature to detect events. In particular, CPU and memory consumption are used to detect three of the dynamic events: *Client mobility*, *Dynamic*

Table 6 Monitored metrics

Event	Study	CPU	Memory	Storage	Bandwidth	Availability	Latency	Sensor data
E1	S5	X	X					
E1	S9						X	
E1	S17	X	X	X				
E1/E2	S19			X		X		
E2	S3			X				
E2	S6	X						
E2	S7	X						
E2	S11				X			
E2	S15						X	
E2	S18	X	X	X				
E2	S21					X		
E2	S32	X						
E3	S1				X			X
E3	S2							X
E3	S8							X
E3	S24							X
E3	S27							X
E3	S38							X
E4	S4	X	X	X	X	X	X	
E4	S13	X	X	X	X			
E4	S14	X	X	X				
E4	S16	X	X	X				
E4	S28	X	X	X				
E5	S1				X			X
E5	S23						X	
E5	S25				X		X	
E5	S30						X	
E5	S33						X	
E5	S34				X			
E5	S35						X	
E5	S37				X			
E5	S39						X	
E6	S12							X

data transfer, and *Failures and software aging*. Sensor data (column 9) is not a QoS metric, but its analysis is used to detect the dynamic events *Important event detected by sensors*, *Network connectivity*, and *Cyber-attacks in IoT applications*. Availability and Latency are seldom monitored metrics to detect dynamic events. However, ensuring low latency is one of the important requirements for real-time applications. Similarly, ensuring the availability of services and applications in IoT systems is also a common requirement. S10, S20, S22, and S29 are not included in Table 6 because they do not monitor any QoS metrics. These four studies address the dynamic event client mobility, which they detect by identifying new clients joining or leaving the system. Studies S31 and S36 do not focus on the detection of the dynamic event, instead they cover the architectural adaptations to cope with the event. For this reason, these two studies are not included in Table 6.

The QoS metrics monitoring conducted by the studies is carried out in the edge/fog and cloud layers of the system, but not in the physical layer devices. The World Wide Web Consortium [62] proposes an ontology of non-functional properties for IoT devices including accuracy, sensitivity, response time, drift, and frequency. The monitoring of these properties is not trivial due to the heterogeneity of IoT devices, difference in firmwares, and variability of communication protocols. However, constantly monitoring these properties at the physical layer of the system would allow for improved QoS. For example, it is possible to avoid device failures by scheduling preventive maintenance after analyzing the information on the accuracy and sensitivity of the sensors.

4 RQ2: how do existing solutions adapt their internal behavior and architecture in response to dynamic environmental events in the edge/fog and physical layers to ensure compliance with its requirements?

In this section, we address the research question RQ2. In Section 4.1, we discuss the adaptive strategies used by the studies to support the dynamic events outlined in the previous section. We then discuss the relationship between dynamic events and adaptive strategies in Section 4.2.

4.1 Adaptive strategies

Table 7, which presents a classification of the strategies used by each study to support specific dynamic events, provides a preliminary answer to research question RQ2. Similar to the classification of dynamic events (3.1), we propose this list of adaptations after analyzing the studies in detail. Study S12 is not included in Table 7 because it does not address an adaptation strategy. This study monitors, detects, and classifies attacks coming from the physical layer devices of the IoT system, but the system is not adapted to these attacks. Although the topic of this SLR is not the security of the IoT system and study S12 does not perform any adaptation strategy, we include it in this SLR because it is the only one that addresses dynamic event E6 in our list of studies, and it contributes to solve research question RQ1.

The adaptive strategies are described below.

4.1.1 Data flow reconfiguration

The routing of data traveling from the physical layer to the Edge/Fog or cloud layer is modified mainly to improve latency. The direction of the data flow and the devices involved in communication, such as gateways and messaging servers, are strategically selected to carry the data to the nodes that perform the processing.

The data flow reconfiguration strategy is used to address three dynamic events: *Client mobility*, *Dynamic data transfer rate*, *Important event detected by sensors*, and *Network connectivity*. Studies addressing client mobility (S5, S9, S10, S17, S19, and S20) use this technique to control communication between physical layer devices and edge/fog and cloud layer nodes. When a physical layer device changes location, it is necessary to identify the edge/fog nodes with the most optimal position to provide the lowest communication latency with the device. Additionally, it is necessary to ensure that these edge/fog nodes have the resources available to support the new device load.

In S9, an edge-enabled publish-subscribe middleware is proposed to address client mobility challenges. The data flow between clients and brokers, i.e., the servers that implement the MQTT server protocol, is reconfigured to optimize communication

Table 7 Adaptations

ID	Adaptation	Studies
A1	Data flow reconfiguration	S3, S5, S8, S9, S10, S11, S14, S15, S17, S19, S20, S23, S25, S35, S37, S38
A2	Auto scaling of services and applications	S2, S7, S17, S18, S19, S22, S31, S39
A3	Software deployment and upgrade	S4, S13, S16, S28
A4	Offloading tasks	S1, S6, S21, S26, S27, S29, S30, S32, S33, S34, S36

latency. When there is client mobility, the least communication latency with the MQTT broker is guaranteed. However, the availability of resources of the edge/fog node that hosts the broker is not monitored. If a group of clients is assigned to a broker that has little processing capacity, the broker could be overloaded and could fail. Additionally, there are no mechanisms to autonomously auto-scale the brokers in the edge nodes and register them in the system. Auto-scaling (4.1.2) is another strategy used to address some dynamic events, and this strategy is complemented by data flow reconfiguration.

To adapt the IoT system to cope with the dynamic events *Dynamic data transfer rate*, *Important event detected by sensors*, and *Network connectivity*, some authors propose to reconfigure the data flow with the aim of balancing the load between the edge/fog nodes, or to redirect the data flow to the node with the best conditions (resource availability and lower response latency). For example, S8 proposes a framework that enables the developer to specify dynamic QoS rules. A rule is made up of a source device (e.g. a video camera), a target device (e.g., a web server), a rule activation event (e.g. when a system sensor detects motion), and a QoS requirement that must be guaranteed (e.g. 200ms communication latency between source and target). When the event configured in the rule is triggered, the path of the data flow between the source and the destination is reconfigured to establish the optimal path through a set of switches. This architecture assumes that there are several switches that enable communication between the physical layer devices and the cloud layer. However, the edge/fog layer is not included to do edge processing, which could improve system QoS by lowering latency and bandwidth. The system architecture proposed in S8 assumes that the edge/fog layer is composed of devices that only serve the function of relaying the data, but the data processing capacity in the edge devices is ignored. Additionally, it is necessary to consider using the MQTT protocol and broker for communication which offers lower power consumption and low latency due to its very small message header and packet message size (approximately 2 bytes) [63].

The software-defined network (SDN) is a network management technology commonly adopted by studies that propose the strategy of adaptation (*Data flow reconfiguration*). S14, S23, S25, S37, and S38, deploy SDN to flexibly manage network resources according to changing system conditions. The SDN controller has the functionalities to configure the data flows through the system devices. Several algorithms have been proposed to optimize data flows and ensure QoS. For example, S25 proposes a routing algorithm that finds the lowest cost routes based on latency, available bandwidth, and lost data packets. S23 proposes a routing algorithm to optimize data traffic by considering QoS metrics such as latency and power consumption. Dynamic management of network resources and dynamic flow is possible through the SDN controller.

4.1.2 Auto scaling of services and applications

This strategy consists of automatically deploying or terminating replicated services and applications on the system's edge/fog nodes or cloud servers. Auto-scaling is used to ensure stable application performance, and it is one of the most widely used techniques in web applications deployed in the cloud. Auto-scaling is also used in IoT systems but with additional concerns to address. A challenge of auto-scaling at the edge/fog layer is related to the selection of the best node for the deployment and execution of the service or application. While the cloud layer has a large amount of network, processing, and storage resources, the edge/fog layer has limited resources. For this reason, when scaling an application at the edge/fog layer, it is necessary to strategically select the node that has availability of the necessary computing resources and that offers the greatest communication latency benefits with the physical layer devices.

According to Table 8, auto-scaling of services and applications is a technique used to address three dynamic events in IoT systems: (1) for the *Client mobility* event, the applications are auto-scaled to attend the requests of new customers that connect a cluster of edge/fog nodes; (2) for the *Dynamic data transfer rate* event, it is necessary to auto-scale the applications and services in the edge/fog nodes to support the growth of data that are processed; (3) in some domains, the *Important event detected by sensors* requires auto-scaling of applications to support the growth of monitoring frequencies in cases of system emergency.

In S2, an auto-scaling method is proposed for a distributed intelligent urban surveillance system. The proposed architecture has three layers: video cameras in the physical layer, desktops in the edge layer to analyze the video information, and cloud servers that host the web application for the end user. When the video cameras detect an emergency, the frame rates of video capturing increase and image analysis for some objects turn to high-priority tasks. The system then scales the data analysis application by deploying virtual machines to the edge nodes closest to the emergency site. However, deploying the application at the node closest to the physical layer device does not always guarantee the best performance. Other factors such as network latency and node bandwidth consumption should be considered for application allocation decisions. Additionally, the use of virtual machines has limitations given the resource scarcity that characterizes edge nodes. Other virtualization technologies such as containers have advantages for deploying applications to edge/fog layer nodes. In particular, the reduced size of the images and the low startup time are advantages that make containers suitable for IoT systems.

4.1.3 Software deployment and upgrade

The process of deploying and updating software in a semi-automatic way is one of the strategies used to solve problems, correct software issues, improve application performance, and improve system security.

S4, S13, S16, S28, and S39 perform software deployment and movement of services remotely in the edge/fog nodes of the system. Containerization is one of the most used technologies that facilitates the semi-automatic deployment of software, given the reduced size of images and the low start time compared to virtual machines. These three studies (S4, S13, S28, and S39) use docker technology to package and run the software versions in containers on Fog nodes. S13 proposes Foggy, a framework for continuous

Table 8 Connection between events and strategies

Dynamic event	Data flow reconfiguration	Auto scaling of services and applications	Software deployment and upgrade	Offloading tasks
Client mobility	X	X		X
Dynamic data transfer rate	X	X		X
Important event detected by sensors	X	X		X
Network connectivity				X
Failures and software aging	X		X	

automated deployment in fog nodes. Foggy allows for the definition of four software allocation rules in Fog nodes: (1) software deployment on a specific node; (2) software deployment on a specific number of nodes that match a hardware feature (i.e., three nodes that have 2GB of memory); (3) software deployment on a group of nodes that comply with a hardware feature (i.e., all nodes that have 4GB of memory); and (4) software deployment on all nodes in the system. Foggy's architecture is based on an orchestration server responsible for monitoring the resources in the nodes and dynamically adapting the software allocation according to the rules defined by the user. However, Foggy's software allocation rules can only be configured according to fixed hardware characteristics of the nodes, i.e., node selection does not depend on dynamic system metrics such as latency, bandwidth consumption, and power consumption. These QoS factors should also be considered for software allocation decisions in fog nodes. Additionally, Foggy does not monitor the state of the running docker containers to detect and fix failures through actions such as rollback to the previous stable version or redeployment of the software container.

4.1.4 Offloading tasks

The processing tasks executed at the edge/fog nodes can be classified according to their importance and their response time required. While there are system tasks that do not require immediate processing, other tasks such as real-time data analysis are critical to the system and require low response latency. It is necessary to guarantee low latency for these critical tasks, but it is not trivial to achieve this when dynamic events occur in the system such as increased data flow from the physical layer. The adaptation strategy *Offloading tasks* addresses this problem in the following way: to guarantee low response latency for critical processing tasks performed by the edge/fog nodes, non-critical tasks are offloaded to the cloud servers to free up capacity in the edge/fog nodes. However, it is necessary to establish when it is really necessary to offload tasks to the cloud servers. In contrast, offloading tasks from cloud servers to edge/fog nodes is also possible. This is done to improve QoS such as latency, as long as the edge/fog nodes have the necessary resources to execute the task.

S6 proposes an architecture that coordinates data processing tasks between an edge node and the cloud servers. The edge node performs data processing tasks with the data collected by devices in the physical layer of the system. A monitoring component frequently checks the CPU usage of the edge node, and every time the value exceeds a usage limit (75%) one of the non-critical tasks executed by the node is offloaded to a cloud server. This frees up resources on the fog node for processing tasks that require low latency. However, offloading of tasks between edge/fog nodes is not considered. Before moving tasks to cloud servers, the offload tasks between neighboring edge/fog nodes that have the necessary resources available should be considered to take advantage of edge and fog computing. In particular, response latency is lower for tasks that can be executed in the edge/fog layer rather than in the cloud layer. Additionally, decisions to move tasks from one node to another node or to a cloud server could be determined by other factors such as latency, RAM usage, power consumption, and battery level (if the node is battery powered). These factors must be monitored and analyzed to make intelligent offloading decisions according to the QoS requirements of the system.

S27 proposes a fog computing framework for cognitive portable ground penetrating radars (GPRs). An offloading policy decides where to execute the sensor data analysis tasks (in mobile nodes of the physical layer or in fog nodes). The offloading policy is based on two metrics: (1) power limitations for mobile nodes that are battery powered, and (2) the efficiency of the node to perform the task. However, cloud servers are not evaluated by the offloading policy.

4.2 Relationship between events and adaptations

Table 8 presents the relationship between the dynamic events and the adaptation strategies to address them. The most used strategies are *Data flow reconfiguration* (A1) and *Offloading tasks* (A4). Strategy A1 is used to ensure system QoS for five dynamic events: (1) *Client mobility* because data traffic from devices joining the system can be routed to assign services to attend them; (2) *Dynamic data transfer rate* because it is possible to dynamically balance the variable data flows and distribute processing workloads among the edge/fog nodes; (3) the adaptation for *Important event detected by sensors* depends on domain requirements, for example, it is possible to redirect sensor data to the nearest nodes to reduce latency (as in S8); (4) *Network connectivity* because it is possible to dynamically control data flow to ensure QoS for variations in network connectivity; and (5) *Failures and software aging* because it is possible to reroute data traffic when failures are detected on a node.

The adaptation strategy *Auto Scaling of services and applications* (A2) enables automatic adjustment of system capacity to ensure performance. This strategy enables the system to support variations in the workloads of edge/fog nodes and cloud servers. Therefore, the A2 strategy is used to guarantee system QoS for dynamic events that involve an increase in data processing at nodes such as *Client mobility*, *Dynamic data transfer rate*, and *Important event detected by sensors*.

The strategy *Offloading tasks* (A4) is used to guarantee QoS at least in the critical tasks of the system. This strategy frees up processing resources on the edge/fog nodes to address critical tasks. Meanwhile, low critical tasks can be offloaded to neighboring edge/fog nodes or to cloud servers. In the literature, the A4 strategy is used to ensure the QoS of critical tasks in dynamic events such as *Client mobility*, *Dynamic data transfer rate*, *Important event detected by sensors*, and *Network connectivity*.

Finally, the strategy of adaptation *Software deployment and upgrade* is only used by the studies that deal with the event *Failures and software aging*. The deployment of updates or new software versions in the edge/fog nodes allows for the repair of problems detected in the software. Operations such as rollback, whose function is to release the previous stable version, can also alleviate problems due to software failures in the upgrade process.

5 Threats to validity

According to Kitchenham and Brereton [64], there is no way to completely avoid personal bias in a literature review. Although some stages of the SLR were developed by a single researcher, we looked for methods to ensure the quality of the results. These methods are described below.

The main threats come from the screening and data collection stages. In particular, the first and second screening of inclusion and exclusion criteria are to a certain extent subjective processes that may lead to misclassification of studies. The process of filtering the studies was carried out by the first author of this paper. To ensure the quality of the filtering process, another expert researcher in the field (not a co-author of this study) carried out the same process with the same inclusion and exclusion criteria. This researcher took 25% of the initial studies and applied the inclusion and exclusion criteria. The agreement of results between the first author and the external researcher was 85% of the studies analyzed; the small disagreement difference is due in part to the fact that the external researcher discarded studies that addressed overly specific problems of a domain. We had a face-to-face meeting with the external researcher to understand the differences in the classification of 15% of the papers and make final decisions.

Another threat to validity is the subjectivity in extracting and analyzing information from studies (Sections 2.5 and 2.6), which can result in misunderstandings. One researcher extracted the data from the studies, and the rest of the authors of this SLR checked the extraction. We scheduled regular meetings to analyze the information; particularly, the data that answered the research questions. In these meetings we discussed disagreements and when necessary, we studied the text and diagrams of the conflicting study.

The list of events and adaptations we suggest is related to another threat to validity. As we did not have a unified preliminary list of events and adaptations, we defined them based on the findings of the studies. To mitigate the risk of omitting any event or adaptation, we apply an iterative content analysis method to continuously evolve the list of events and adaptations found in the studies. All the authors met together to discuss the studies that contributed a new event or adaptation to our classification. In this way we ensure that the events and adaptations addressed by the SLR studies conform to the proposed categorization.

6 Summary and future directions

In this section, we synthesize the findings and then elaborate on research topic that deserve further exploration.

6.1 Summary of the results

Beyond the detailed analysis provided in the previous section we would also like to outline some overall conclusions.

- To detect dynamic events during system operation, different QoS metrics and resource consumption are constantly monitored. The monitoring mechanisms proposed by the literature mainly focus on measuring resource consumption (CPU, memory, and storage) [18]. However, other factors should also be monitored and taken into account to accurately detect dynamic events. For example, power consumption is an important metric for edge nodes that are powered by batteries.
- The monitored QoS metrics could be analyzed at a later stage to make system improvement decisions. This is one of the most important stages (known as Feedback) suggested by the DevOps practices [65, 66]. Surprisingly, none of the studies analyzed in this review propose a method for recording and storing the data in a storage system so that they can be consulted later.
- One of the technologies that is gaining strength for software deployment is containerization [67, 68]. Most studies use docker containers to package and deploy

software on the edge/fog nodes [69]. One challenge for orchestrating containers in IoT systems is related to efficient allocation decisions between the edge and fog layer nodes [70]. The selection of the nodes to deploy the containers can impact the system on factors such as performance and availability.
- Finally, to control the deployment of new software versions decreasing the risk of process failures and increasing reliability, software deployment patterns (such as rolling deployment, blue-green, and canary) has been proposed [71]. While these patterns have been widely used in cloud-based applications, in IoT systems with distributed architectures it have been seldom explored.

6.2 Future directions

The design of IoT systems involves coping with several challenges to ensure a good QoS even when considering the dynamic nature of the IoT environment. Some specific challenges were pointed out by the studies analyzed in this paper. Indeed, the conclusions above suggest already some areas that are not yet fully developed even if some works start to appear that address them.

Nevertheless, in this section, we want to highlight additional significant open challenges we believe need to be addressed to improve current adaptation strategies.

We classified the issues and challenges as follows.

6.2.1 Monitoring and logging the dynamic events themselves

Monitoring the system infrastructure is a key process in the design of a self-adaptive architecture. However, designing a continuous, scalable, resilient, and non-intrusive monitoring system for IoT systems is a challenge. In the literature, efforts are focused on designing strategies to adapt the IoT system at run time. But self-adaptations for system monitoring components also require attention. For example, according to the state of the infrastructure, the monitoring system must self-adapt to the characteristics of the heterogeneity of devices (e.g. gateways, servers, switches, and user devices), heterogeneity according to virtualization (e.g. virtual machines, containers, and pods), and scalability (join and leave of devices).

In addition to detecting events at runtime, monitoring data is also important for analyzing historical data and making decisions to improve its architecture. One of the critical stages of DevOps is monitoring and feedback. This stage involves constantly monitoring the system, storing data, and then analyzing data to provide feedback to developers. Through the analysis of the data, improvement decisions are made for the next DevOps iteration. However, it is necessary to effectively monitor and store the data for historical queries and analysis to identify system improvements. Logging of monitoring data implies the design of a domain model that abstracts the main concepts of self-adaptive and distributed IoT architectures. For example, concepts such as the different types of devices (including their resources, location, and hardware characteristics), QoS metrics, dynamic events, and adaptations. In addition, to persist time series, events, and metrics, it is necessary to select an appropriate scalable storage system such as InfluxDB[3] or TimescaleDB[4].

[3]https://www.influxdata.com
[4]https://www.timescale.com

6.2.2 Software deployment on heterogeneous devices

Some adaptation strategies such as service auto-scaling, software deployment, and upgrades involve the deployment of new software versions in the different layers of the system architecture. Container-based virtualization and hypervisor-based virtualization are widely used in the edge/fog and cloud layers for software deployment. Container-based virtualization improves performance and efficiency when compared to hypervisor-based virtualization [72]. While a virtual machine requires a complete installation of the operating system, a software container shares the host OS kernel, binaries, and libraries. Additionally, containers are faster to create or migrate when compared to virtual machines. Therefore, containers are preferred at the edge/fog layer of the system, and there are tools such as Kubernetes[5], K3S[6], and Docker Swarm[7] to manage and orchestrate software deployments in containers. However, there are challenges for software deployment and migration in the edge/fog layer nodes of IoT systems. One of the present challenges is related to making intelligent allocation decisions to guarantee QoS. When deploying or moving an application in the system, it is necessary to select the edge/fog nodes that have enough resources for the operation of the application, and to offer the appropriate QoS. Tools like Kubernetes provide functionality through the scheduler component to select the appropriate node that will host the container with the new software version, but this component only checks the resources requested (CPU and RAM) by the container. Other factors such as energy consumption, network latency, reliability, and bandwidth usage should be considered when making allocation decisions.

Another challenge relates to the managing of software deployments on physical layer devices. The heterogeneity of sensor and actuator devices makes it difficult to deploy software at this layer due to the variety of communication protocols and software languages supported. Some physical layer devices do not support remote upgrades, nor virtualization, or containerization.

6.2.3 Machine learning for self-adaptable systems

Machine learning systems can automatically identify normal and abnormal patterns and alert a client or third parties when things deviate from observed standards, without requiring prior configuration by human operators. For IoT systems, learning algorithms can also help to prevent disruptive events affecting system availability and QoS. For example, to predict when a hardware component might fail to take action and avoid system downtime. Predictive maintenance of physical devices is one of the tasks that can be efficiently forecasted by algorithms learning. For example, manufacturing systems require maintenance of their machinery to avoid system failures and downtimes. While there are traditional challenges for the design of a learning algorithm such as the selection of the efficient model, the amount of data, and data cleaning, there are also other problems related to the technologies and processes to obtain the data or features. For example, the monitoring of non-functional properties such as accuracy, frequency, sensitivity, and drift is one of the challenges due to the heterogeneity of IoT devices in the physical layer.

[5]https://kubernetes.io
[6]https://k3s.io
[7]https://docs.docker.com/engine/swarm/

6.2.4 Global self-adaptive architecture

The studies included in this SLR propose techniques and strategies to address at most two of the dynamic events. However, in some scenarios or domains, it is necessary to propose solutions to support various/simultaneous dynamic events. For example, in the scenario illustrated in Section 1, the IoT ventilation monitoring and control system for underground mines can present several types of dynamic events. Another example is a smart city system, which synchronizes the basic functions of a city based on seven key components, including natural resources and energy, transport and mobility, buildings, life, government, economy, and people [73]. Due to the large number of IoT devices considered, a smart city system can experience all the dynamic events that we have identified in Table 5.

Therefore, it is necessary to design a general architecture for IoT systems with components to monitor, detect events, and self-adapt the system: an architecture with the ability to adapt to various dynamic events. For example, a system that can detect failures in software updates and perform operations such as software rollback, while supporting new devices being added to the system. This same system could also support other types of events such as dynamic data transfer rate and network connectivity failures.

For designing this general self-adaptive architecture for IoT systems, some base technologies are especially promising. For example, the MQTT communication protocol is ideal for IoT applications since it presents advantages concerning scalability, asynchronism, decoupling between clients, low bandwidth, and power consumption. Regarding virtualization technology, containerization offers several advantages for software deployment in IoT systems. In particular, it is possible to deploy containers on various types of hardware and operating systems, something very useful considering the heterogeneity of nodes in the edge/fog layer. For example, it is possible to deploy a container with an application on both a RaspberryPI[8] and a Linux server.

7 Related work

Literature reviews have been presented in the IoT field, mainly focused on different specific sub-domains such as smart cities, cyber-physical systems, or industrial IoT. For example, [74] presents a review of standards, technologies, and implementations of industrial IoT applications, [75] provides a review of the use of smart home applications and open challenges, and [76] elaborates on key features and applications of the IoT paradigm to support sustainable development of smart cities. These studies focus on describing the current state of IoT applications for a specific domain (industrial IoT, smart homes, and smart cities respectively), and identifying challenges and opportunities for future research in the area.

With respect to literature reviews more closely related to our research we found three studies, each of them is discussed in the following paragraphs and compared with our work.

Mekuria et al. [77] contribute a SLR about smart home reasoning systems (SHRS). The reasoning techniques for these systems are characterized and analyzed to discuss the challenges in smart living environments. The requirements, assumptions, strengths,

[8]https://www.raspberrypi.org

and limitations of SHRSs are discussed. Although SHRSs involve control and adaptations operations, the aim of this study is not to classify system architecture adaptation strategies and dynamic events that impact QoS.

Muccini et al. [78] present an SLR to analyze the adaptation strategies at the level of cyber-physical systems architecture. From the findings, future work opportunities and challenges are identified. The target topic of [78] is complementary to our SLR. However, our study deals with both cyber-physical systems and IoT systems. In addition, we classify and analyze the dynamic events that cause adaptations due to the changing environment.

Nguyen et al. [79] depict a systematic review to identify and analyze the most significant approaches in deployment and orchestration for IoT. This SLR focuses on describing the technical details of the orchestration, reliability aspects (including monitoring, adaptation, and shared access to resources), and the technical challenges for future research. However, the adaptations analyzed are limited to solutions that consider a software orchestrator. In addition, the dynamic events that induce the adaptations are not analyzed.

In a nutshell, to the best of our knowledge, there are no literature reviews that comprehensively target self-adaptation strategies, dynamic events and their impact in the quality attributes of IoT systems. Therefore, the added value of this SLR with respect to the state-of-the-art is to present an overview of self-adaptive IoT systems, the strategies used within, the events that induce the adaptations, and some gaps that need to be addressed.

8 Conclusion

We have conducted a systematic literature review to study the dynamic events that impact the QoS of IoT systems, to analyze the strategies implemented by the literature to address them, and to identify the weaknesses of the approaches found in the state-of-the-art.

We identified six types of dynamic events and four adaptation strategies in response to the events. *Dynamic data transfer rate* is the event most addressed by the literature, while *Data flow reconfiguration* is the strategy most used in the studies.

Monitoring the resource consumption of the edge/fog nodes is one of the most used strategies to detect some dynamic events of the system. In particular, the consumption of CPU and RAM memory are metrics that are frequently monitored to identify events such as *Dynamic data transfer* and *Failures and software aging*. Other factors such as power consumption, availability, latency, throughput, and bandwidth are poorly monitored, but these factors should be considered to accurately detect dynamic events.

As further work, we plan to advance in the opportunities discussed in the previous section. We aim to address the challenges and open gaps related to the software deployment on heterogeneous devices.

Abbreviations

SLR: Systematic literature review; IoT: Internet of things; QoS: Quality of service; SDN: Software-defined networking; MQTT: Message queue telemetry transport; GPR: Ground penetrating radar; SHRS: Smart home reasoning system

Acknowledgements

We would like to thank researcher Wilmer Rubio for contributing in one of the tasks to address threats to validity: to conduct the process of filtering studies independent of our process, to compare them and avoid misclassification.

Authors' contributions

IA contributed to the literature search process, application of inclusion and exclusion criteria, study quality assessment, data collection and analysis. KG, HC, and JC collaborated in the formulation of the problem and research questions, the definition of inclusion and exclusion criteria, and participated in the extraction and analysis of data from the studies. All authors write, read, and approved the final manuscript.

Funding
This work has been partially funded by the Spanish government (LOCOSS project - PID2020-114615RB-I00), the Colombian government (*Becas del Bicentenario* program - Art. 45, law 1942 of 2018), and the ECSEL Joint Undertaking (JU) under grant agreement No 101007260. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Netherlands, Finland, Germany, Poland, Austria, Spain, Belgium, Denmark, Norway.

Availability of data and materials
Please contact authors for data requests.

Declarations

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Department of Systems and Computing Engineering, Universidad de los Andes, Bogotá, Colombia. [2]Universitat Oberta de Catalunya, Barcelona, Spain. [3]ICREA, Barcelona, Spain.

Received: 10 March 2021 Accepted: 29 November 2021
Published online: 27 December 2021

References
1. Gubbi J, Buyya R, Marusic S, Palaniswami M. Internet of things (IoT): A vision, architectural elements, and future directions. Futur Gener Comput Syst. 2013;29(7):1645–60.
2. Cirani S, Davoli L, Ferrari G, Léone R, Medagliani P, Picone M, Veltri L. A scalable and self-configuring architecture for service discovery in the internet of things. IEEE Internet Things J. 2014;1(5):508–21.
3. Singh M, Baranwal G. Quality of service (qos) in internet of things. In: 2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU). Bhimtal: IEEE; 2018. p. 1–6.
4. Buyya R, Srirama SN. Fog and Edge Computing: Principles and Paradigms. Hoboken: Wiley; 2019.
5. Patel P, Ali MI, Sheth A. On using the intelligent edge for IoT analytics. IEEE Intell Syst. 2017;32(5):64–69.
6. De Lemos R, Giese H, Müller HA, Shaw M, Andersson J, Litoiu M, Schmerl B, Tamura G, Villegas NM, Vogel T, et al. Software engineering for self-adaptive systems: A second research roadmap. In: Software Engineering for Self-Adaptive Systems II. Germany: Springer; 2013. p. 1–32.
7. Wong T, Wagner M, Treude C. Self-adaptive systems: A systematic literature review across categories and domains. arXiv preprint arXiv:2101.00125. 2021.
8. Cheng AMK. Self-stabilizing real-time rule-based systems. In: Proceedings 11th Symposium on Reliable Distributed Systems. Houston: IEEE Computer Society; 1992. p. 172–73.
9. Beauquier J, Bérard B, Fribourg L. A new rewrite method for proving convergence of self-stabilizing systems. In: International Symposium on Distributed Computing. Bratislava: Springer; 1999. p. 240–55.
10. Garlan D, Cheng S-W, Huang A-C, Schmerl B, Steenkiste P. Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer. 2004;37(10):46–54.
11. Blair G, Bencomo N, France RB. Models@ run. time. Computer. 2009;42(10):22–27.
12. Jiang Y, Huang Z, Tsang DH. Challenges and solutions in fog computing orchestration. IEEE Netw. 2017;32(3):122–29.
13. Group OCAW, et al. Openfog reference architecture for fog computing. OPFRA001. 2017;20817:162.
14. Satyanarayanan M. The emergence of edge computing. Computer. 2017;50(1):30–39.
15. Wen Z, Yang R, Garraghan P, Lin T, Xu J, Rovatsos M. Fog orchestration for internet of things services. IEEE Internet Comput. 2017;21(2):16–24.
16. Keele S, et al. Guidelines for performing systematic literature reviews in software engineering. Technical report, Ver. 2.3 EBSE Technical Report. EBSE. 2007.
17. Kitchenham B, Brereton OP, Budgen D, Turner M, Bailey J, Linkman S. Systematic literature reviews in software engineering–a systematic literature review. Inf Softw Technol. 2009;51(1):7–15.
18. Villari M, Fazio M, Dustdar S, Rana O, Ranjan R. Osmotic computing: A new paradigm for edge/cloud integration. IEEE Cloud Comput. 2016;3(6):76–83.
19. Young R, Fallon S, Jacob P. A governance architecture for self-adaption & control in IoT applications. In: 2018 5th International Conference on Control, Decision and Information Technologies (CoDIT). Thessaloniki: IEEE; 2018. p. 241–46.
20. Wang J, Pan J, Esposito F. Elastic urban video surveillance system using edge computing. In: Proceedings of the Workshop on Smart Internet of Things. New York: ACM; 2017. p. 7.
21. Muñoz R, Vilalta R, Yoshikane N, Casellas R, Martínez R, Tsuritani T, Morita I. Integration of IoT, transport sdn, and edge/cloud computing for dynamic distribution of IoT analytics and efficient use of network resources. J Lightwave Technol. 2018;36(7):1420–28.
22. Cheng B, Papageorgiou A, Cirillo F, Kovacs E. Geelytics: Geo-distributed edge analytics for large scale IoT systems based on dynamic topology. In: 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT). Milan: IEEE; 2015. p. 565–70.
23. Kimovski D, Ijaz H, Saurabh N, Prodan R. Adaptive nature-inspired fog architecture. In: 2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC). Washington DC: IEEE; 2018. p. 1–8.
24. Young R, Fallon S, Jacob P. Dynamic collaboration of centralized & edge processing for coordinated data management in an IoT paradigm. In: 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA). Krakow: IEEE; 2018. p. 694–701.
25. Tseng C-L, Lin FJ. Extending scalability of IoT/m2m platforms with fog computing. In: 2018 IEEE 4th World Forum on Internet of Things (WF-IoT). Singapore: IEEE; 2018. p. 825–30.

26. Peros S, Janjua H, Akkermans S, Joosen W, Hughes D. Dynamic qos support for IoT backhaul networks through sdn. In: 2018 Third International Conference on Fog and Mobile Edge Computing (FMEC). Barcelona: IEEE; 2018. p. 187–92.
27. Rausch T, Nastic S, Dustdar S. Emma: distributed qos-aware mqtt middleware for edge computing applications. In: 2018 IEEE International Conference on Cloud Engineering (IC2E). Orlando: IEEE; 2018. p. 191–97.
28. Pahl C, El Ioini N, Helmer S, Lee B. An architecture pattern for trusted orchestration in IoT edge clouds. In: 2018 Third International Conference on Fog and Mobile Edge Computing (FMEC). Barcelona: IEEE; 2018. p. 63–70.
29. Lorenzo B, Garcia-Rois J, Li X, Gonzalez-Castano J, Fang Y. A robust dynamic edge network architecture for the internet of things. IEEE Netw. 2018;32(1):8–15.
30. Prabavathy S, Sundarakantham K, Shalinie SM. Design of cognitive fog computing for intrusion detection in internet of things. J Commun Netw. 2018;20(3):291–98.
31. Yigitoglu E, Mohamed M, Liu L, Ludwig H. Foggy: a framework for continuous automated IoT application deployment in fog computing. In: 2017 IEEE International Conference on AI & Mobile Services (AIMS). Honolulu: IEEE; 2017. p. 38–45.
32. Morabito R, Beijar N. A framework based on sdn and containers for dynamic service chains on IoT gateways. In: Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems. Los Angeles: ACM; 2017. p. 42–47.
33. Desikan K, Srinivasan M, Murthy C. A novel distributed latency-aware data processing in fog computing-enabled IoT networks. In: Proceedings of the ACM Workshop on Distributed Information Processing in Wireless Networks. New York: ACM; 2017. p. 4.
34. de Brito MS, Hoque S, Magedanz T, Steinke R, Willner A, Nehls D, Keils O, Schreiner F. A service orchestration architecture for fog-enabled infrastructures. In: 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC). Valencia: IEEE; 2017. p. 127–32.
35. Velasquez K, Abreu DP, Gonçalves D, Bittencourt L, Curado M, Monteiro E, Madeira E. Service orchestration in fog environments. In: 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud). Prague: IEEE; 2017. p. 329–36.
36. Flores H, Su X, Kostakos V, Ding AY, Nurmi P, Tarkoma S, Hui P, Li Y. Large-scale offloading in the internet of things. In: 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). Kona: IEEE; 2017. p. 479–84.
37. Pizzolli D, Cossu G, Santoro D, Capra L, Dupont C, Charalampos D, De Pellegrini F, Antonelli F, Cretti S. Cloud4IoT: A heterogeneous, distributed and autonomic cloud platform for the IoT. In: 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). Luxembourg: IEEE; 2016. p. 476–79.
38. Montero D, Serral-Gracià R. Offloading personal security applications to the network edge: A mobile user case scenario. In: 2016 International Wireless Communications and Mobile Computing Conference (IWCMC). Paphos: IEEE; 2016. p. 96–101.
39. Chen L, Zhou P, Gao L, Xu J. Adaptive fog configuration for the industrial internet of things. IEEE Trans Ind Inform. 2018;14(10):4656–64.
40. Mass J, Chang C, Srirama SN. Context-aware edge process management for mobile thing-tofog environment. In: Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings. New York: Association for Computing Machinery; 2018. p. 1–7.
41. Li X, Li D, Wan J, Liu C, Imran M. Adaptive transmission optimization in sdn-based industrial internet of things with edge computing. IEEE Internet Things J. 2018;5(3):1351–60.
42. Suganuma T, Oide T, Kitagami S, Sugawara K, Shiratori N. Multiagent-based flexible edge computing architecture for IoT. IEEE Netw. 2018;32(1):16–23.
43. Deng G-C, Wang K. An application-aware qos routing algorithm for sdn-based IoT networking. In: 2018 IEEE Symposium on Computers and Communications (ISCC). Natal: IEEE; 2018. p. 00186–91.
44. Sami H, Mourad A. Towards dynamic on-demand fog computing formation based on containerization technology. In: 2018 International Conference on Computational Science and Computational Intelligence (CSCI). Las Vegas: IEEE; 2018. p. 960–65.
45. Wu D, Omwenga MM, Liang Y, Yang L, Huston D, Xia T. A fog computing framework for cognitive portable ground penetrating radars. In: ICC 2019-2019 IEEE International Conference on Communications (ICC). Shanghai: IEEE; 2019. p. 1–6.
46. Skarlat O, Karagiannis V, Rausch T, Bachmann K, Schulte S. A framework for optimization, service placement, and runtime operation in the fog. In: 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC). Suiza: IEEE; 2018. p. 164–73.
47. Mechalikh C, Taktak H, Moussa F. A scalable and adaptive tasks orchestration platform for IoT. In: 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC). Tangier: IEEE; 2019. p. 1557–63.
48. Castillo EA, Ahmadinia A. Iot-based multi-view machine vision systems. In: 2019 IEEE International Conference on Big Data (Big Data). Los Angeles: IEEE; 2019. p. 5206–12.
49. Breitbach M, Schäfer D, Edinger J, Becker C. Context-aware data and task placement in edge computing environments. In: 2019 IEEE International Conference on Pervasive Computing and Communications (PerCom. Kyoto: IEEE; 2019. p. 1–10.
50. Torres Neto JR, Rocha Filho GP, Mano LY, Villas LA, Ueyama J. Exploiting offloading in IoT-based microfog: experiments with face recognition and fall detection. Wirel Commun Mob Comput. 2019;2019:1–13.
51. Theodorou V, Diamantopoulos N. Glt: Edge gateway elt for data-driven intelligence placement. In: 2019 IEEE/ACM Joint 4th International Workshop on Rapid Continuous Software Engineering and 1st International Workshop on Data-Driven Decisions, Experimentation and Evolution (RCoSE/DDrEE). Montreal: IEEE; 2019. p. 24–27.
52. Guntha R. Iot architectures for noninvasive blood glucose and blood pressure monitoring. In: 2019 9th International Symposium on Embedded Computing and System Design (ISED). Kollam: IEEE; 2019. p. 1–5.
53. Jutila M. An adaptive edge router enabling internet of things. IEEE Internet Things J. 2016;3(6):1061–69.

54. Cui K, Sun W, Sun W. Joint computation offloading and resource management for usvs cluster of fog-cloud computing architecture. In: 2019 IEEE International Conference on Smart Internet of Things (SmartIoT). Tianjin: IEEE; 2019. p. 92–99.
55. Bedhief I, Foschini L, Bellavista P, Kassar M, Aguili T. Toward self-adaptive software defined fog networking architecture for IIoT and industry 4.0. In: 2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD). Limassol: IEEE; 2019. p. 1–5.
56. Asif-Ur-Rahman M, Afsana F, Mahmud M, Kaiser MS, Ahmed MR, Kaiwartya O, James-Taylor A. Toward a heterogeneous mist, fog, and cloud-based framework for the internet of healthcare things. IEEE Internet Things J. 2018;6(3):4049–62.
57. Yousefpour A, Patil A, Ishigaki G, Kim I, Wang X, Cankaya HC, Zhang Q, Xie W, Jue JP. Fogplan: a lightweight qos-aware dynamic fog service provisioning framework. IEEE Internet Things J. 2019;6(3):5080–96.
58. Santos J, Wauters T, Volckaert B, De Turck F. Fog computing: Enabling the management and orchestration of smart city applications in 5g networks. Entropy. 2018;20(1):4.
59. Bri D, Fernández-Diego M, Garcia M, Ramos F, Lloret J. How the weather impacts on the performance of an outdoor wlan. IEEE Commun Lett. 2012;16(8):1184–87.
60. Bass L, Clements P, Kazman R. Software Architecture in Practice. New Jersey: Addison-Wesley Professional; 2003.
61. Huang G, Huang G-B, Song S, You K. Trends in extreme learning machines: A review. Neural Netw. 2015;61:32–48.
62. (W3C) WWWC. Semantic Sensor Network Ontology. 2017. https://www.w3.org/TR/2017/REC-vocab-ssn-20171019/. Accessed 3 Mar 2021.
63. Wukkadada B, Wankhede K, Nambiar R, Nair A. Comparison with http and mqtt in internet of things (IoT). In: 2018 International Conference on Inventive Research in Computing Applications (ICIRCA). Coimbatore: IEEE; 2018. p. 249–53.
64. Kitchenham B, Brereton P. A systematic review of systematic review process research in software engineering. Inf Softw Technol. 2013;55(12):2049–75.
65. Bass L, Weber I, Zhu L. DevOps: A Software Architect's Perspective. New Jersey: Addison-Wesley Professional; 2015.
66. Ebert C, Gallardo G, Hernantes J, Serrano N. Devops. IEEE Softw. 2016;33(3):94–100.
67. Brogi A, Mencagli G, Neri D, Soldani J, Torquati M. Container-based support for autonomic data stream processing through the fog. In: European Conference on Parallel Processing. Santiago de Compostela: Springer; 2017. p. 17–28.
68. Leon D, Miori L, Sanin J, El Ioini N, Helmer S, Pahl C. A lightweight container middleware for edge cloud architectures. In: Rajkumar B, Satish Narayana S, editors. Fog and edge computing: principles and paradigms. Hoboken: John Wiley & Sons; 2019. p. 145–70.
69. Pahl C, Lee B. Containers and clusters for edge cloud architectures–a technology review. In: 2015 3rd International Conference on Future Internet of Things and Cloud. Rome: IEEE; 2015. p. 379–86.
70. Santos J, Wauters T, Volckaert B, De Turck F. Resource provisioning in fog computing: From theory to practice. Sensors. 2019;19(10):2238.
71. Ahmadighohandizi F, Systä K. Application development and deployment for IoT devices. In: European Conference on Service-Oriented and Cloud Computing. Vienna: Springer; 2016. p. 74–85.
72. Singh S, Singh N. Containers & docker: Emerging roles & future of cloud technology. In: 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT). Bangalore: IEEE; 2016. p. 804–07.
73. Colistra J. The evolving architecture of smart cities. In: 2018 IEEE International Smart Cities Conference, ISC2 2018; 2019. https://doi.org/10.1109/ISC2.2018.8656678.
74. Liao Y, Loures E. d. F. R., Deschamps F. Industrial internet of things: A systematic literature review and insights. IEEE Internet Things J. 2018;5(6):4515–25.
75. Alaa M, Zaidan AA, Zaidan BB, Talal M, Kiah MLM. A review of smart home applications based on internet of things. J Netw Comput Appl. 2017;97:48–65.
76. Alavi AH, Jiao P, Buttlar WG, Lajnef N. Internet of things-enabled smart cities: State-of-the-art and future trends. Measurement. 2018;129:589–606.
77. Mekuria DN, Sernani P, Falcionelli N, Dragoni AF. Smart home reasoning systems: a systematic literature review. J Ambient Intell Humanized Comput. 2021;12(4):4485–502.
78. Muccini H, Sharaf M, Weyns D. Self-adaptation for cyber-physical systems: a systematic literature review. In: Proceedings of the 11th international symposium on software engineering for adaptive and self-managing systems. New York: Association for Computing Machinery; 2016. p. 75–81.
79. Nguyen P, Ferry N, Erdogan G, Song H, Lavirotte S, Tigli J-Y, Solberg A. Advances in deployment and orchestration approaches for IoT-a systematic review. In: 2019 IEEE International Congress on Internet of Things (ICIOT). Milan: IEEE; 2019. p. 53–60.