Computer Science Review 50 (2023) 100598

Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

IoT systems modeling and performance evaluation

Alem Čolaković

University of Sarajevo, Faculty of Traffic and Communications, Sarajevo, Bosnia and Herzegovina

ARTICLE INFO

Keywords:
Internet of Things (IoT)
Cloud computing
Fog computing
Edge computing
System modeling
Performance evaluation
Resource allocation
Quality of services (QoS)

ABSTRACT

The continuous increase of IoT applications leads to a vast amount of data that needs to be transmitted, stored, and processed. Many IoT applications rely on the Cloud infrastructure to handle these specific application demands. However, the integration of IoT and Cloud poses challenges such as network delays, throughput, energy consumption, reliability, etc. Therefore, a new computing concept is required to support emerging IoT applications. These new concepts include fog computing, edge computing, mobile edge computing, mobile cloud computing, and cloudlets. They use various approaches to distribute resources, processes, and services among IoT system architecture layers. The challenge is to decide which offloading system is the best for a specific use case that emphasizes the IoT system modeling issue. In this paper, a model for the formal description of IoT systems is presented. In addition, an analytical evaluation method was proposed to design these systems using the corresponding architecture, technologies, protocols, and integration model to optimize performance. The proposed approach facilitates and simplifies the selection of the corresponding model for the system architecture. This approach enables an efficient method for performance optimization based on offloading processes (load balancing). Also, this paper provides some insights into specific emerging issues and ideas to be addressed by future research.

1. Introduction

The Internet of Things (IoT) is an internetworking paradigm enabled by a technology stack that provides seamless connectivity between physical and virtual objects to facilitate the development of intelligent services. The capabilities of IoT-enabling technologies have improved over the past few years. This improvement has enabled new large-scale applications. However, IoT devices are typically resource-constrained and limited in terms of computation, communication, and other capabilities. Different computing systems can overcome this issue by offloading services from IoT devices. Incorporating computational offloading into IoT system modeling enables the optimization of resource utilization while meeting the system's performance requirements. IoT system modeling helps simulate different allocation strategies and selects the one that minimizes the overall cost or maximizes the system's performance.

The context of IoT system modeling considers computation offloading as a decision-making problem, where the objective is to find the optimal allocation of computation tasks to available resources. Offloading is used to transfer computationally intensive tasks from resource-limited devices to various platforms or resource-rich infrastructures to improve the system performance [1]. This process allows resource-constrained devices to conserve resources and extend their operational lifetime. As the number of IoT devices increases, offloading enables a more efficient use of computing resources, enabling IoT systems to scale up without overburdening individual devices. Offloading also improves the performance of IoT systems by allowing tasks to be executed using more powerful computing resources, reducing energy consumption, and improving response times. Overall, the offloading strategy is critical for improving the efficiency, scalability, security, and performance of IoT systems, particularly as the number of devices and data generated by IoT systems continues to increase. Therefore, the full potential of IoT can be achieved by integrating IoT and the corresponding options for computing systems, such as cloud computing, fog computing, edge computing, mobile edge computing (MEC), mobile cloud computing (MCC), and cloudlets. Many authors use these terms interchangeably, as they bring intelligence and processing capabilities closer to the data source. However, there are some similarities and differences between them [2–4], including the infrastructure deployment location, hardware type, infrastructure ownership, network technology and architecture, etc. All of these computing systems imply resource and services allocation outside IoT devices. Efficient resource management strategies and mechanisms are required to enable the implementation of IoT applications with better Quality of Service (QoS)

E-mail address: alem.colakovic@fsk.unsa.ba.

https://doi.org/10.1016/j.cosrev.2023.100598
Received 29 October 2021; Received in revised form 11 June 2023; Accepted 17 October 2023
Available online 31 October 2023

by offloading computation tasks in a computation-intensive environment [4–10]. These strategies are based on different metrics such as response time, throughput, energy consumption, and utilization [11].

Cloud computing outperforms the storage and computing capabilities of IoT devices and the infrastructure at the edge of the network. However, cloud-supported IoT is not an efficient solution for many applications owing to inherent problems such as unacceptable network delay, lack of location awareness, security and privacy issues, etc. On the other hand, edge computing reduces latencies but does not guarantee a bounded end-to-end response time, which is more determined by the computation time [12]. Therefore, a corresponding computing paradigm is required to support emerging IoT application requirements [13]. Some authors believe that we are already in the post-cloud era [14,15]. Computing paradigms such as edge computing, fog computing, MEC, MCC, and cloudlets help overcome some of these issues by providing basic computation functions (e.g., filtering and simple data processing) closer to the IoT devices (source of data). They enable real-time computations without sending data to the cloud. This is a way to reduce network delay compared to the traditional IoT-Cloud architecture [16]. If there is a need for more advanced processing and storage of a large amount of data, the filtered and summarized data can be transferred to the cloud. However, there are some bottlenecks for each offloading model, depending on the specific IoT application demands and system resource state.

According to the above statements, the following research questions are "which model of integration to choose and which functions to allocate to each layer of the system architecture (where to offload the tasks generated by the IoT devices)," "How many resources are allocated to each layer of the architecture," and "which technologies to deploy for the specific IoT application?" These questions highlight the IoT system modeling issue for finding eligible deployments of different architectures, integration models, technologies, protocols, etc. [14,17–19]. A model of an IoT system can be used to simulate, analyze, and optimize system performance, as well as to identify potential issues and evaluate design choices. IoT system modeling and offloading are closely related because modeling helps to identify which tasks and data should be transferred to other computing resources in an IoT system. By modeling the behavior of the IoT system, researchers can identify tasks that may require more computing power than what is available on the IoT devices themselves. Some tasks can be offloaded to various computing resources such as edge servers, cloud servers, or other devices in the network. For example, a model of an IoT system that includes sensors, actuators, and a cloud server may reveal that the processing and analysis of sensor data require more computational power than what is available on individual devices. By offloading the processing and analysis tasks to the cloud server, IoT devices can conserve their resources and extend their operational lifetime. IoT system modeling can also help determine the optimal offloading strategy for a given IoT system. These procedures are necessary to achieve the desired QoS but also to achieve the concept known as G-IoT (green IoT) [20]. By simulating and analyzing the system under different conditions, researchers can identify which tasks and data should allocate to what components and when offloading should occur. This open issue needs future attention from the research community because there is a lack of quantitative evaluation methods which include multiple metrics. Furthermore, many existing mechanisms for IoT system offloading are based on complex algorithms that drain device resources, which makes them impractical for application in real-case IoT systems. This challenging issue is a key motivation for this study.

The main scope of this paper is to present a model for a formal IoT system description and a method for its evaluation based on computational and communication metrics. This paper introduces a practical offloading mechanism based on multiple metrics to ensure the QoS performance of an IoT system, which does not drain the computation resources of resource-limited devices. The proposed approach is an efficient tool for finding an eligible computation offloading system to improve IoT system performance and optimize resource consumption. This enables the selection of the corresponding IoT architecture and technologies for any part of the IoT system architecture. The proposed evaluation method includes an algorithm whose implementation does not require excessive resources. In addition, this paper provides the necessary understanding to drive the system evaluation and alleviate some system limitations, such as IoT device resource constraints. The proposed model can be used to design and optimize IoT systems for various applications and address the challenges associated with traditional cloud computing for IoT applications. In addition, this study discusses some performance indicators and highlights open research issues. Furthermore, some advantages and disadvantages of IoT integration with computing systems are presented by providing insights into the possibilities of different offloading models.

The remaining of this paper is organized as follows. Section 2 presents a literature review of this area of research. Section 3 introduces IoT-enabling technologies according to their functionalities and describes the possible integration models. In addition, this section provides a model for the formal description of the IoT system based on different integration models of IoT and offloading systems, and various technologies deployed in the system architecture. The key performance indicators (KPI) for IoT system evaluation are presented, discussed, and modeled in Section 4. Section 5 provides a new analytical method for evaluating IoT systems. Section 6 compares the results obtained using the proposed model with those of previously validated approaches in the scientific literature. Finally, Section 7 contains concluding remarks and suggestions for future research.

2. Related works

Research on IoT system modeling and computation offloading is a highly trending topic. This section provides a brief overview of the background information related to IoT system modeling and presents current research related to offloading in an IoT environment.

The IoT system modeling involves developing mathematical and computational models to represent the behavior and dynamics of the system. Modeling aims to gain insights into how the system works, predict its performance, and optimize its design. There is no standard methodology for modeling real-world IoT-based systems. However, there are some contributions in the modeling where authors propose different layers of IoT system architecture with various functional blocks. For example, the paper [21] presents a theoretical modeling approach for fog computing that includes mathematical models for analyzing the performance of the fog computing system. The authors discuss the key parameters that affect the performance of the fog computing system, such as the number of fog nodes, the network bandwidth, and the workload distribution. The paper [22] provides an overview of the task offloading algorithms and presents several optimization approaches and their mathematical problem formulation. The study [23] focuses on the design and performance evaluation of a three-tier Cloud of Things (CoT) architecture. The authors propose a three-tier CoT architecture that consists of the edge tier, the fog tier, and the cloud tier, each responsible for different aspects of the CoT system. Also, the authors in this study mathematically characterize each tier in terms of energy consumption and latency. The paper [24] presents a novel approach for modeling and simulating IoT systems that can help developers and designers to address challenges associated with developing scalable and reliable IoT applications. In this paper, the authors propose a hybrid agent-oriented approach that combines the benefits of modeling and simulating the behavior of individual IoT devices and their interactions. The paper [25] presents a computational model using the concepts of graphs and reflection to address the complexities associated with the visualization, modeling, interaction, analysis, and abstraction of information in the IoT. The articles mentioned above provide a good starting point for exploring the theoretical modeling of IoT systems, and they also provide references to additional research in this area. Most of these contributions involve the following steps:

- Identifying the key components of the IoT system and their interconnections.
- Formulating mathematical equations that describe the behavior of each component and combining these equations to create a system-level model.
- Simulating the model to predict the system's performance under different conditions.
- Validating the model by comparing its predictions to real-world data.

Theoretical modeling of IoT systems helps to design more efficient systems by optimizing the use of resources, reducing costs, and improving reliability. Also, it can help identify potential problems before they occur, allowing preventative measures. Therefore, modeling and simulation are crucial factors that support IoT system development. However, an integrated approach synergistically providing both aspects is lacking [24]. This challenge is still an open issue that needs future attention from the research community.

There are infrastructural orchestration challenges which include determining which functions to distribute to the corresponding layer of IoT system architecture [25–34]. Key challenges are related to heterogeneity (various hardware specifications, communication protocols, and architectures), resource-constraint devices, limited power supply, various workloads, various IoT application demands, etc. Therefore, many authors addressed offloading challenge and provided offloading techniques while incorporating various resource allocation strategies in different scenarios. Some solutions include a rewriting-based approach to design Cloud-Fog self-adaption and orchestration behaviors to manage infrastructure reconfiguration [32,35] including multi-objective optimization based resource allocation and utilization [36]. Other authors propose a smart layer between IoT devices and computing nodes to incorporate an intelligent task-scheduling technique and computing resource allocation algorithm [37–41]. For this purpose, various flexible middleware has been proposed [42–44] as well as mechanisms of data caching [45,46]. Also, there is a hybrid cloud solution based on the integration of resources between private and public clouds to enable horizontally scale on-premises infrastructure [47]. Therefore, there are many algorithms for task scheduling in the IoT-Cloud environments [6,40,48–56]. For example, in the study [34], authors used the Particle Swarm Optimization algorithm (PSO) to consider multiple parameters including response time, network bandwidth, energy consumption, and latency. In the study [57], a semi-dynamic real-time task scheduling algorithm is proposed for bag-of-tasks applications in the cloud–fog environment. In the study [58] authors proposed an enhanced firefly algorithm adapted for tackling workflow scheduling challenges in a cloud-edge environment. The survey [59] focuses on UAV-enabled MEC solutions based on artificial intelligence approaches with their respective advantages and limitations. In the study [60] authors propose a distributed algorithm that jointly optimizes offloading decisions and resource allocation. This is a deep reinforcement learning algorithm with centralized training and distributed execution proposed to optimize the real-time transmission power. Shakarami et al. provided a comprehensive summary of the computation offloading approaches in mobile edge computing based on Machine learning [61]. Study [62] provides a comprehensive survey on the use of ML in MEC systems by pointing out which challenges can be solved using ML approaches. The study [63] reviews the existing literature on task scheduling for IoT systems and identifies several challenges, such as the heterogeneity of devices, the need for energy-efficient scheduling, and the need for real-time scheduling. Papers [64] and [65] overview computation offloading algorithms and discuss the challenges and potential future research directions. In addition to the mentioned works, there is a large number of research that emphasize the importance of finding suitable methods for optimizing the distribution of resources in the IoT systems.

Most of the described solutions are based on the implementation of a specific infrastructure that includes servers, base stations, and network devices. Also, most of these approaches may not be able to identify an efficient strategy within an acceptable time frame. Other limitations include high computation complexity, omitting some parameters in the problem formulation, unrealistic assumptions, and not being evaluated in real-case scenarios [66]. Nevertheless, the solution proposed in this paper is adaptable for implementation on any infrastructure. It is independent of the type of IoT application. This solution is based on a simplified algorithm with low computational complexity but considers both computational and communication KPIs. These factors make it practical for application in various real-case IoT scenarios.

3. IoT system modeling

IoT system modeling refers to creating a representation of a system. This representation helps to understand the components, interactions, and behaviors of the IoT system. The conceptual model defines the high-level structure that describes how the IoT system operates over time. The behavioral model may include details such as the data flow between components. The modeling helps to identify and resolve potential performance issues and leads to more efficient, effective, and reliable IoT systems. Modeling an IoT system provides a framework for testing, validating, and optimizing the system's performance.

IoT processes such as identifying, sensing, networking, and computation enable connectivity between physical and virtual objects. The IoT technology stack provides seamless connectivity anytime and anywhere by anyone and anything to facilitate the development of intelligent services and applications with self-configuring capabilities. For this paper, IoT technologies are systematized according to functional blocks composed of four main groups (domains) as shown in

Table 1. These domains include various technologies with specific functionalities and capabilities. Fig. 1 depicts the main dataflow steps in a generic IoT system.

Some challenges in developing IoT services are their complexity and heterogeneity in all layers of the system architecture. Several systems (generic), hardware/network, software, and process architectures have been proposed to support the development of IoT applications. These architectures are required to support full interoperability, multisystem integration, cross-domain interactions, and simple and scalable management functionalities. However, there is a lack of formal models for IoT system descriptions that can help select the corresponding architecture and technologies according to specific application demands (use cases). IoT devices are typically resource-constrained with identifying, sensing, networking, computation, and other capabilities. Therefore, most IoT applications rely on computing systems to offload certain functions. In recent years, several computing paradigms have been proposed, including cloud computing, fog computing, MCC, MEC, and cloudlets. These terms are often used interchangeably, with the same or similar meanings. The distinction between these computing paradigms lies in the "distance" from the IoT objects and the type of infrastructure. This paper does not include a profound analysis of the diverse meanings, but considers that fog computing is broader than the notion of the other related terms mentioned above, and it can be used as a general term that abstracts all similar concepts. The infrastructure of these concepts includes small datacenters placed close to the edge of the network, which provide some computation and communication capabilities. These computing systems can be considered as part of the IoT system architecture. Therefore, a generic IoT system architecture (designed from a system point of view) can be presented as a three-tier structure (physical tiers) that includes different interaction models between IoT devices and computing systems for offloading resources and tasks (Fig. 2). In this model, each IoT device is connected to one or more computing systems, whereas all layers of the system architecture, as well as components in each layer, can be interconnected depending on the integration model.

Layer 1 (IoT devices or the perception layer) encompasses things or IoT devices (terminal nodes – TNs) such as mobile phones, smart vehicles, and other objects with embedded sensors that are responsible for

Table 1
IoT enabling technologies and functional blocks.

Domain	Enabling technologies, samples and use cases		Functionalities
Application Domain	*IoT applications Architectures*	Smart home, Smart cities, Smart traffic, Smart healthcare, Industrial IoT, Smart agriculture, Smart farm, Smart meters, Environment monitoring, etc. Software architectures, SOA, RESTful, etc.	*Visualization, Application development environment, System and devices monitoring, control and management, Data and services management…*
Middleware Domain	*Software and APIs*	OS (Contiki, FreeRTOS, LiteOS, Android, Riot OS, uClinux, Mbed…), APIs (JML, WebGL, RAML…), Embedded and custom apps built using «thing» data	*Data storage, Data aggregation and processing, big data analysis, Decision support (Expert and DSS systems), Machine Learning…*
	Fog and Cloud platforms	OpenIoT, Amazon, Google Cloud, Libellium, IBM Watson, FIWARE, Arkessa, OnePlatform, SensorCloud, SmartThings, ThinkWorks, Oracle IoT, Plotly, Nimbits, ThinkSpeak, Xively, etc.	
	Data processing mechanisms	Data mining, Big Query, Cloud Datalab, Apache Hadoop, Kafka, Storm, RapidMQ, Scribe, SPARQL, SciDB, Semantic technologies (JSON, W3C, OWL, RDF, EXI, WSDL…), etc.	
	Data storage	Storage infrastructure (public, private, hybrid), DB (MongoDB, Cassandra, Hadoop, HBase, CouchDB, Redis…), Storage architecture, etc.	
Networking Domain	*Communication protocols*	Application (CoAP, MQTT, AMQP, XMPP, DDS, WebSocket…), Transport (TCP, UDP), Network (IPv4, IPv6), Routing (RPL), Service Discovery (mDNS, DNS-SD, SSDP, SLP…), etc.	*Seamless connectivity (anytime, anywhere by anyone and anything), Data transfer (device-to-device, device-to-application, application-to-device) …*
	Network interface	IEEE 802.11, 802.15.4, IEEE 802.15.1, IEEE 802.16, 3GPP, IEEE 802.15.6, WSN, Z-Wave, IEEE 802.3, RFID, NFC, UWB, IrDA, PLC, CAN, etc.	
	Adoption mechanisms	Adoption layer (6LoWPAN, 6TiSCH, 6Lo, IEEE 1095.1…), Connectivity interfaces (RJ45, ODB2, RS-232, RS-485, PLC, USB, Modus, SPI…), Gateways (Advantech, ADLINK, BRC…)	
Devices Domain	*Hardware platforms*	Arduino, Raspery Pi, Intel Galileo Gen, Intel Edison, Beaglebone Black, Broadcom, Netduino, Intel Edison, Flutter, Marvell, Tessel 2, Particle.io, Smart Things, etc.	*Identification, sensing, actuating, data processing and computation, power supply…*
	Embedded objects	Embedded sensors, Actuators, RFID/NFC tags, Identification (EPC, uCode, QR…), touch screen displays, firmware, onboard software, etc.	
	Mechanical & Electrical parts	Mechanical and electrical parts (e.g., batteries), Processing units (e.g., microcontrollers, microprocessors), Digital Signal Processors, Peripheral Controller Chips, etc.	

sensing events (e.g. temperature, humidity, pressure, etc.) and transmitting data to other TNs (for acting or to work collaboratively) or transmitting data to some offloading system for further processing. In addition to these capabilities, TNs can be aware of their geospatial location through GPS or other positioning techniques as well as perform some basic computing processes. They can be mobile or fixed at one location while using different communication technologies to send and receive data.

Layer 2 (fog layer) encompasses infrastructures such as local servers, computers, set-top boxes, routers, gateways, access points, etc. These nodes are typically located at the edge of the network or near TNs (source of data); however, this is not an exclusive rule. This infrastructure has computing and storage capabilities in addition to routing data to the upper layer or back to the TNs. Therefore, they can be used for data aggregation, data filtering and preprocessing, real-time processing, temporary data storage, and other services. All TNs that are in the range of some fog nodes (FN) can be grouped into logical groups, which we denote as a virtual cluster (VC). Each TN can join (if authorized) or leave the VC according to its geospatial location (availability of the FN).

Layer 3 (cloud layer) constitutes data centers with huge storage and computing capabilities. This infrastructure enables large-scale complex pattern-data analysis and long-term data storage. Cloud computing is inefficient in some use cases because of the heavy traffic loads on networks. The centralized approach is not always sufficient, and cloud computing cannot handle all requirements, particularly when running applications with low latency and geo-distributed demands. In such cases, local processing or the use of fog infrastructure is often the only valid solutions. IoT devices can be connected to the cloud directly (without an FN as a sublayer) or through FNs that are connected to the cloud data centers by using the Internet.

According to this approach, composite entities of the IoT system (IoT_{sys}) can be defined as a 4-tuple:

$$IoT_{sys} = \langle A,\ D,\ N,\ C \rangle \tag{1}$$

where A is the set of applications, D is the set of IoT devices (terminal nodes), N is the set of networks (communication technologies), and C is the set of computing systems for offloading tasks.

3.1. Application domain

IoT application is a program designed to perform some specific functions and manages different services for the user. IoT_{sys} includes one or more IoT applications:

$$A = A_{pp_1}, A_{pp_2}, \ldots, A_{pp_n} \tag{2}$$

Each IoT application ($i = 1, 2, \ldots n$) denoted as A_{pp_i} is defined as a 5-tuple:

$$A_{pp_i} = \langle A_{id},\ A_{type},\ A_{sp}, A[S],\ A_{st},\ A_{req} \rangle \tag{3}$$

A_{id} is an integer representing the application ID and each A_{pp_i} has a unique ID. This ID uniquely identifies the application running on the device or computing system. For A_{id} can be used digit number or some other notion (e.g. *ba.fsk.myapp*).

A_{type} denotes a type of application, such as client, web, mobile, and hybrid applications. Client applications are also known as desktop applications. They are intended to be installed on a desktop (user terminal) for user interaction. These applications are developed specifically for one platform and are typically standalone applications. Web applications can be used in a browser and do not require installation on client

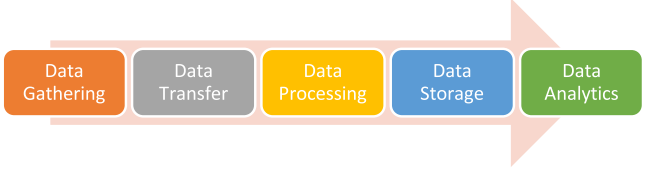

Fig. 1. Data flow of a generic IoT system. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

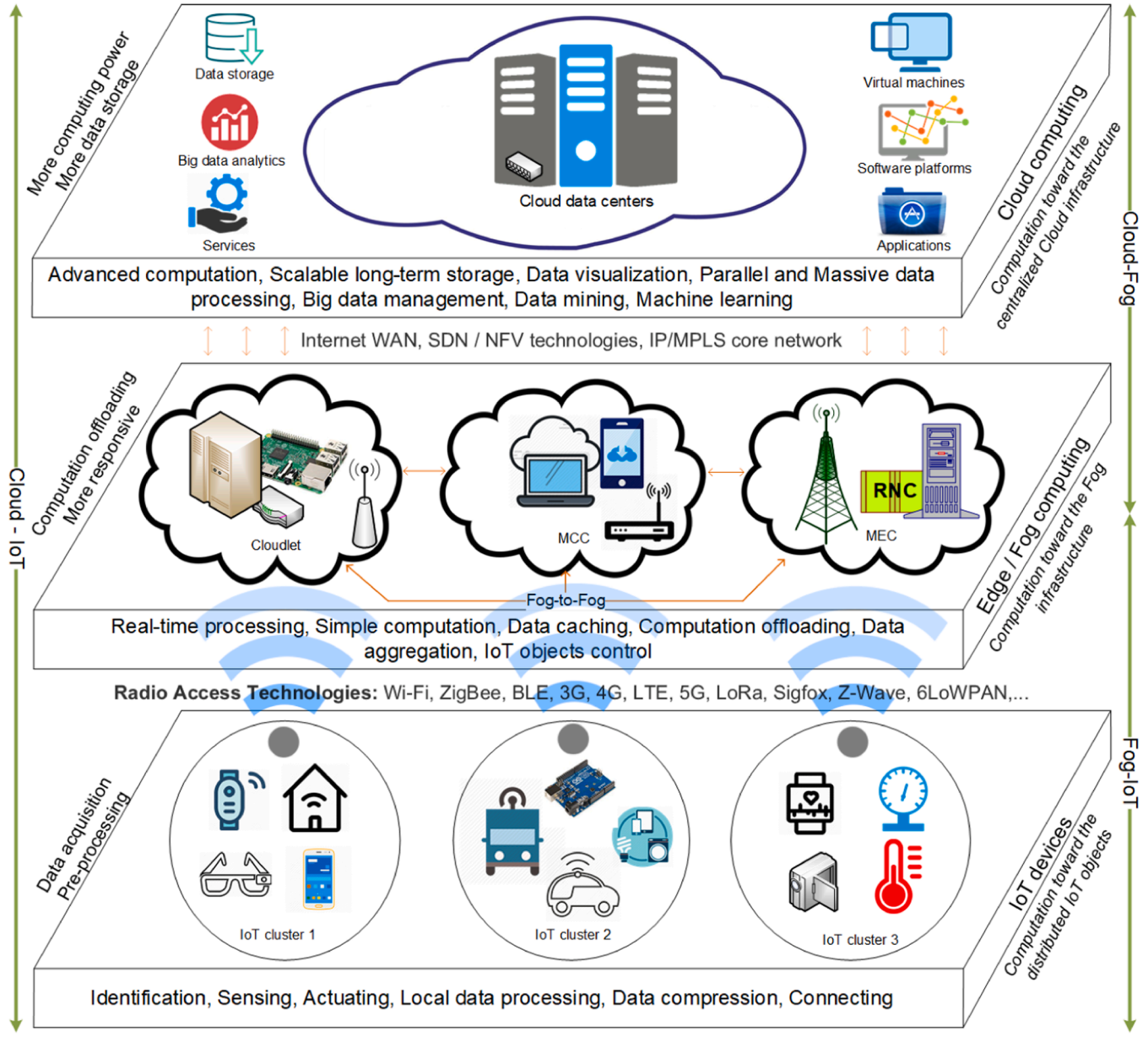

Fig. 2. Integration models of IoT-Fog-Cloud systems. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

devices. These applications typically support different browsers running on a range of OS (Operating Systems) and platforms while allowing clients to access them instantly from a local or remote system without installation. Mobile applications have been developed for handheld devices that can be used anywhere and at any time. These applications run locally on an IoT device, so they can be fast and responsive, but device resources may prove to be a constraint. Other specific types of applications combine these basic types. These applications can be considered hybrid applications.

A_{sp} indicates the infrastructure required to run A_{pp_i}. This includes IoT device denoted as D_{ev}, software components S_{oft}, network infrastructure N_{et}, and offloading systems O_{sys_i}. An IoT device can be an object that collects or generates data. These objects are typically equipped with sensors and have computing and communication capabilities. S_{oft} includes components such as OS, APIs (Application Programming Interface), and programming language or SDKs (Software Development Kits). Network infrastructure is used for data transfer and includes different technologies and protocols. Today, most IoT applications rely on the support of offloading systems, which are mostly based on cloud technologies, but not exclusively.

$$A_{sp} = \langle D_{ev}, S_{oft}, N_{et}, O_{sys_i} \rangle \tag{4}$$

$A[S]$ indicates the type of event or set of services $S = (s_1, s_2, \ldots, s_n)$ required by the IoT application, where n is the total number of distinct services. There are many service types such as sensing, actuating, networking, computation, storage, etc. These services produce several program instructions that must be executed to run the IoT application. Therefore, a service indicates an action (task) and each service is defined by a type (S_T) and workload time (S_w).

$$S_i = \langle S_t, S_w \rangle \tag{5}$$

The application is in the active state if any required service is being performed. A_{st} defines whether the application is inactive or idle state and can be represented as a Boolean:

$$A_{st} = \begin{cases} 0, & idle \\ 1, & active \end{cases} \tag{6}$$

A_{req} represents an IoT application profile that dictates the minimum performance requirements for an application to operate properly. Various performance indicators can be used to create IoT application profiles according to the resource demands. For example, some of the most important performance-based requirements (indicators) are service execution time, storage capacity, hardware utilization (e.g., CPU and RAM), energy efficiency, and network performance such as network delay, throughput, jitter, and loss. There are other important indicators depending on various use cases (e.g., mobility, reliability, security, and resource usage fairness). Therefore, it is necessary to determine the key performance requirements for IoT applications, which can be formulated as

$$A_{req} = \langle req_1,\ req_1, \ldots, req_n \rangle \tag{7}$$

where req_i $(i = 1, \ldots, n)$ is i_{th} performance requirement or key performance indicator (KPI), and n is the number of KPIs used for creating IoT application profile. Some KPIs for IoT applications are described in the next section.

3.2. IoT device

IoT devices encompass objects equipped with physical and virtual entities including hardware (electrical and mechanical components with embedded sensors, processors, connectivity antennas, etc.) and embedded software (operating system, onboard application, etc.). These devices are usually resource-constrained because of their hardware specifications as mentioned before, and this influence on overall IoT_{sys} performances. IoT_{sys} includes one or more IoT devices:

$$D = \langle D_{ev_1}, D_{ev_2}, \ldots, D_{ev_n} \rangle \tag{8}$$

Each IoT device $(j = 1,\ 2,\ \ldots,\ n)$ denoted as D_{ev_j}, can be defined by nine-tuple:

$$D_{ev_j} = \langle D_{id},\ D_{ip}, D_{type},\ D[S],\ D(A), D_{sp},\ D_l,\ D_{st}, V_i \rangle \tag{9}$$

Each IoT device has a unique identifier D_{id} represented as an integer and an unique IP address D_{ip}. D_{type} defines the type of device such as a microcontroller, smartphone, tablet, or another object with capabilities such as sensing, actuation, data storage and processing, connecting with other objects, etc. $D[S]$ indicates the type of event or service being performed by the IoT device. Multiple applications can run within the same device and $D[A]$ is a set of applications running on D_{ev}. Depending on the device performance specification (D_{sp}), it can be determined which IoT application may run on a specific device, considering that each application requires some computing, memory, communication, and other capabilities. Therefore, D_{sp} includes the hardware performance specification and can be formulated as:

$$D_{sp} = \langle D_{cpu}, D_{ram}, D_{disk}, D_{ni},\ D_{en}, D_{os} \rangle \tag{10}$$

D_{cpu} denotes CPU characteristics (e.g. CPU speed, bus specification, cache memory, etc.). The RAM memory specifications (D_{ram}) include memory size, memory access time, cache characteristics, etc. Storage specifications such as disk capacity, write and read speed, disk access time, etc. are denoted as D_{disk}. Tuple D_{Ni} describes the network interfaces (e.g. LTE, Wi-Fi, ZigBee, Bluetooth, BLE, etc.) embedded in D_{ev}. Tuple D_{en} describes the power supply and consumption details including battery characteristics, voltage, number of units, energy consumption rate when the device is active or in idle mode, etc. D_{os} denotes the operation system running on D_{ev_j}.

D_l denotes geo-spatial location of D_{ev_j} and can be defined as a four-tuple:

$$D_l = \langle x,\ y,\ z,\ t \rangle \tag{11}$$

where, x, y, and z represent the longitude, latitude, and altitude, respectively, and t denotes the time point at which D_{ev} has these coordinates.

D_{ev_j} status is active if one or more services are being performed, otherwise the device is in the idle state. D_{st} defines whether the device is in an active or idle state and can be represented as a Boolean:

$$D_{st} = \begin{cases} 0, & idle \\ 1, & active \begin{cases} computation \\ sending\ data \\ receiving\ data \end{cases} \end{cases} \tag{12}$$

All IoT devices can be grouped into logical groups (virtual entities or virtual clusters) denoted by $V_i = (V_1, V_2, \ldots, V_k)$. Belonging to a particular cluster at a certain moment t depends on D_{ev_j} location and connectivity capabilities. For example, V_i can be formed using devices connected to the same access point (AP). Each IoT device can be associated with one or more V_i, which is responsible for providing some computing and communication services. Devices in the same cluster can exchange data, and in some cases can be used for computation offloading. Each virtual cluster (i) is defined by three-tuple:

$$V_i = \langle V_{id},\ V(D), V_R \rangle \tag{13}$$

V_{id} is the unique identifier of the cluster. $V(D)$ stores the IDs of all devices that are authorized to join this cluster. This array should have a dynamic length because of changes in the number of devices present in V_i. The tuple V_R denotes the region (geographic terrain) covered by some V_i and it depends on the deployed network. Regions of different clusters can overlap.

The specifications of IoT devices, as well as other characteristics, affect the overall IoT_{sys} performance including program execution time, energy consumption, storage capacity, etc. More details about the KPIs related to IoT devices are provided in the following section.

3.3. Network infrastructure

An IoT network includes hardware and software infrastructure that enables connectivity between different objects, including IoT devices and offloading systems. IoT_{sys} can deploy multiple communication technologies for connectivity purpose.

$$N = \langle N_{et_1}, N_{et_2}, \ldots, N_{et_3} \rangle \tag{14}$$

Each network $(k = 1,\ 2,\ \ldots,\ n)$ can be defined as 4-tuple:

$$N_{et_k} = \langle N_{type}, N_{cov},\ N_{mod}, N_{arc},\ N_{sp}, N_{st} \rangle \tag{15}$$

N_{type} denotes the type of network, e.g. Wi-Fi, ZigBee, BLE, LTE, etc. There are number of wireless communication technologies adapted for IoT such as various WSNs (Wireless Sensor Networks), LR-WPAN (Low-Rate Wireless Personal Area Networks), and LP-WAN (Low Power Wide Area Networks) [67]. N_{cov} indicates network coverage, which represents the maximum distance at which IoT objects can transmit data to the destination host. According to the coverage range, the networks can be grouped as follows: BAN (Body Area Network), PAN (Personal Area Network), LAN (Local Area Network), MAN (Metropolitan Area Network), and WAN (Wide Area Network).

$$N_{cov} = (BAN,\ PAN,\ LAN,\ MAN,\ WAN) \tag{16}$$

N_{mod} indicate the type of interaction between IoT sub-systems. Motivated by the models presented in IETF RFC 7452 [68], we outline the following communication (interaction) models:

$$N_{mod} = \{(DxD),\ (DxF),\ (FxF),\ (DxC), (FXC)\} \tag{17}$$

DxD is the communication between IoT devices, *DxF* is the communication between IoT devices and fog nodes, *FxF* is the communication

between different fog nodes, *DxC* is the communication between IoT device and cloud, and *FxC* is the communication between fog and cloud. Thus, it is possible to include other interaction models based on different subsystems.

N_{arc} denotes the IoT protocol architecture (protocol stack) used for data exchange between the IoT subsystems. Some protocols used in traditional Internet services are not the corresponding option for IoT because of the constraints of low-power and lossy networks (LLNs). Consequently, protocols have been created to connect things (devices). For example, there are new application layer protocols such as CoAP (Constrained Application Protocol), MQTT (Message Queuing Telemetry Transport), MQTT-SN (MQTT for Sensor Networks), AMQP (Advanced Message Queuing Protocol), XMPP (Extensible Messaging and Presence Protocol), and DDS (Data Distribution Service).

N_{sp} is a set of network specifications such as supported interfaces, bandwidth, scalability, mobility support, and energy consumption. Each IoT application has specific requirements, and the corresponding communication technology can be selected according to various demands. For example, various applications require different bandwidths for the uplink (↑) and downlink (↓).

$$b = \begin{cases} b\uparrow, \ uplink \\ b\downarrow, \ downlink \end{cases} \tag{18}$$

Each network can be in active state when it is used for sending or receiving data or idle state when there is no data transfer. N_{st} defines the states that can be represented as a Boolean:

$$N_{st} = \begin{cases} 0, \ idle \\ 1, \ active \begin{cases} transmitting \\ receiving \end{cases} \end{cases} \tag{19}$$

Many network parameters affect the entire IoT_{sys} performance. These parameters can be used to create IoT application profiles and evaluate the system. Network-based KPIs include the network throughput, network delay, jitter, and packet loss.

3.4. Offloading systems

Deploying some offloading systems helps overcome some limitations of IoT devices, such as processor and memory resource constraints. A summary of the computational architecture models is presented in Table 2. These systems have emerged to support the requirements of IoT applications and meet their needs with minimum help from the Internet infrastructure [69–71]. These systems are usually the part of IoT_{sys} architecture layer known as IoT middleware. They are designed as intermediaries between the IoT devices and applications. IoT_{sys} can deploy multiple computing systems for offloading processes.

$$O = \langle O_{sys_1}, O_{sys_2}, \ldots, O_{sys_n} \rangle \tag{20}$$

Each of these computing systems is defined as a seven-tuple:

$$O_{sys_n} = \langle O_{id}, \ O_{type}, O_l, \ O_{sp}, O(D), \ O(C), \ O(A), O[S], O_{st} \rangle \tag{21}$$

O_{id} is an integer representing the unique identification of the IoT system. O_{type} represents the infrastructure type (e.g. server, gateway, router, access point, etc.). The main distinction between different offloading systems is the type of infrastructure, as described in the previous section. O_l denotes the geospatial location of the computational system. A similar concept can be used for IoT devices. This characteristic influences several performance indicators such as network latency. O_{sp} denotes system specifications that include parameters such as CPU, RAM, storage, programmability, network interface, supported APIs, operation systems, security mechanisms, etc. *O(D)* consists a database of device's IDs eligible to connect to offloading system O_{sys_n}. IoT devices and virtual clusters can have one-to-one or one-to-many mapping with offloading instances. $O(C)$ stores the *IDs* of the interconnected offloading subsystems. $O(A)$ denotes whether some application instances run in the computing subsystem. $O[S]$ indicates the type of event or service performed by the offloading system. O_{st} indicates whether the offloading system is active or idle.

$$O_{st} = \begin{cases} 0, \ idle \\ 1, \ active \begin{cases} computation \\ sending \ data \\ receiving \ data \end{cases} \end{cases} \tag{22}$$

Various computing systems can be used for computation offloading, which assume data transfer from IoT devices to O_{sys}.

The process of computation offloading affects IoT_{sys} parameters such as service latency, energy consumption, network performances, etc. This issue highlights the challenge of resource optimization by allocating functions over IoT system components.

4. Key performance indicators

Several metrics for quantitative analysis can be used to create IoT application profiles. Most previous studies have been based on an analysis of individual metrics observed in specific use cases. Approaches that integrate more than one metric to allow for a more comprehensive

Table 2
Summary of computing models.

Computing model	Features	Computing infrastructure	Access Network
Cloud computing	Distributes some resources, processes, and services to large data centers (DCs) which may be located anywhere in the world.	Large data centers (DCs) that relies on a shared pool of computing resources (e.g. servers, storage, applications, and services).	WAN (Wide Area Networks) - Internet
Fog computing (FC)	Distributes resources, processes, and Cloud-based services anywhere along the continuum from Things to Cloud.	Any device with computing, storage and networking capabilities (e.g. local servers, hardware platforms, routers, access points, cellular base stations, set-top boxes, etc.).	Wireless and wired network technologies: Wi-Fi, BLE, ZigBee, 3 G, LTE, PLC, Ethernet, etc.
Mobile cloud computing (MCC)	Offloading data processing and storage from the mobile device to powerful and centralized computing platforms.	Centralized computing platforms (data centers) located at infrastructure outside of the mobile devices.	Radio Access Technologies (e.g. Wi-Fi, Bluetooth, 3 G, 4 G, 5 G)
Mobile edge computing (MEC)	Provides cloud-computing capabilities by using computers or clusters of computers (i.e. cloudlets) that are topologically closer than traditional cloud centers.	Nodes or small-scale data centers co-located within radio network base stations or at the Radio Network Controllers (RNCs).	Radio Access Technologies (e.g. Wi-Fi, 3 G, 4 G, 5 G, WiMAX, etc.)
Cloudlet (Cl)	Fixed infrastructures that provide virtual machine capabilities for provisioning computing and networking resources	Fixed infrastructures (e.g. cluster of computers) installed between the IoT devices and centralized cloud.	Wi-Fi (primary), WiMAX, 3 G, 4 G, 5 G, PLC, etc.
Edge computing (EC)	Resources, processes, and services allocation to the IoT devices ("things") side. It is extension of the MEC concept to address both mobile and fixed devices.	IoT devices (fixed or mobile), computers or clusters of computers.	Wireless and wired network technologies: Wi-Fi, BLE, ZigBee, 3 G, LTE, PLC, Ethernet, etc.

evaluation of IoT systems are lacking. In addition, some models that include both computational and communication aspects are mostly oversimplified, and many important factors are neglected. For example, the analytical models proposed by Chen [72] and Fan et al. [73] include several computational and communication aspects, but they take high-level abstraction of some important KPIs, such as network propagation time. The evaluation method proposed in this study includes multiple metrics to enable resource optimization and find eligible deployments of IoT architectures, interaction models, technologies, and protocols. In addition, this model can be used for offloading decisions and performance optimization and to create a classification scheme for IoT applications according to their performance requirements.

IoT system performance level $IoT_{sys}(p)$ can be observed through the fulfillment of specific performance requirements $(p_{i_{req}})$where p_i is i_{th} performance indicator:

$$IoT_{sys}(p) = \left\{p_{i_{req}}, p_i\right\} \tag{23}$$

Specific IoT application demands exist in terms of computing and communication capabilities, mobility, reliability, privacy, and security. Therefore, a large number of performance indicators can be used as utility metrics for IoT_{sys} evaluation. Aslanpour et al. [74] summarized the performance metrics for evaluating IoT and computing systems as well as metrics that are commonly measurable for all layers. These metrics can be grouped into four domains, as shown in Fig. 3.

Regardless of the technology stack used to build IoT applications, it ultimately runs on infrastructure, which greatly affects application performance. To evaluate IoT_{sys}, it is necessary to capture all important performance parameters and interpret them in a meaningful manner. However, it is practically impossible to consider all possible parameters. Therefore, realistic assumptions were made. These assumptions involve limiting the number of parameters based on which a sufficient performance analysis can be performed. Therefore, it is challenging to identify the key performance indicators (KPIs), which are the most important performance metrics.

These performance indicators can be used for IoT_{sys} evaluation (e.g. QoS assessments) and creating IoT application profiles. From an IoT application point of view, two general and measurable groups of KPIs for performance efficiency are QoS (Quality of Service) and resource utilization. Each group contained several indicators. This paper considers the service latency $(T_{exe.})$ and energy consumption $(E_{dev.})$ as the most important KPIs.

$$p_i = \langle T_{exe}, E_{dev} \rangle \tag{24}$$

These performances depend on the specifications (capabilities) of all IoT subsystems, their interactions, required services, and system workload. Therefore, IoT system specification $(IoT_{sys_{sp}})$ can be presented as follows:

$$IoT_{sys_{sp}} = \langle A_{sp}, D_{sp}, C_{sp}, N_{sp} \rangle \tag{25}$$

Depending on the nature of the application, there is always a requirement for some computation and communication services for different types of data (e.g. text, video, audio). The computation includes data processing, which refers to transforming the input data into meaningful outputs. Communication services refer to the data transfer between IoT devices and offloading systems. These two types of services can be used to evaluate KPIs (T_{exe}, E_{dev}) and overall IoT_{sys}. Therefore, the system workload is the result of communication and computation services that depend on the number of devices (k), input data rate (D_{rate}) measured in bps (bits per second), and workload time (t_{w_s}) measured in seconds.

$$S_i = \langle S_{t_i}, S_{w_i} \rangle \tag{26}$$

$$S_{t_i} = \begin{cases} Computation\ (data\ processing) \\ Communication\ (data\ transfer) \end{cases} \tag{27}$$

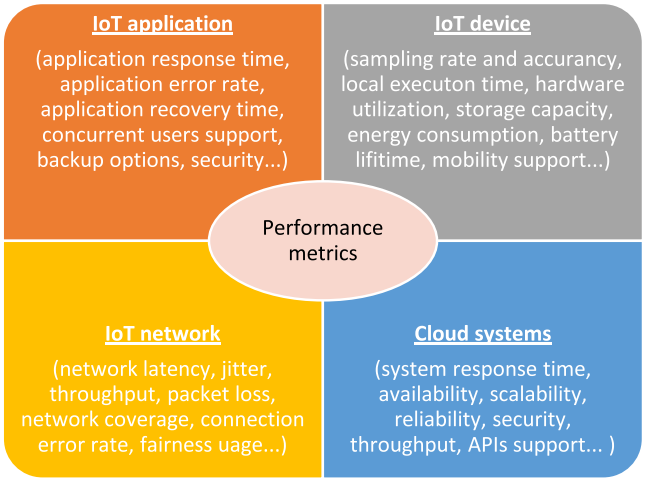

Fig. 3. Performance metrics for IoT system evaluation. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$$S_{w_i} = \langle k, D_{rate}, t_{w_s} \rangle \tag{28}$$

For a small-time interval, approximately the current system load can be obtained. Shannon's formula can be used to estimate D_{rate} such as reported in previous studies [75–78]. These forms depend on the parameters of the equations. The following formula can be used in this study:

$$D_{rate} = B_{ij} log_2 \left(1 + \frac{|h_{ij}|^2 P_{tx}^i}{P_{Nj}}\right) \tag{29}$$

$B_{ij}$is bandwidth, P_{tx}^i is the transmission power of the device, h_{ij} is the channel gain between the source and destination nodes, and P_{Nj} is the noise power on the receiver side. Shannon's formula is applicable to scenarios with a source and receiver (or other similar scenarios). However, in an IoT system, there is no direct topology, and the previous assumption is for a directly connected topology, that is, a direct connection between a source and receiver. Therefore, the reduced abstract topology of a system can be used and mathematically represented using a graph. This approach enables multiple topologies inside, but the reduced abstract topology will still be represented mathematically. Therefore, the system load is the amount of data generated by all devices in the system (D). If there are N devices in the system where D_{rate_k} is the data rate generated by k devices for some timet_{w_k} , then the total system load can be calculated using the following expression

$$D = \sum_{k=1}^{N} D_{rate_k} \cdot t_{w_k} \tag{30}$$

4.1. Service latency

Service latency or service execution time $(T_{exe.})$ is the amount of time that the IoT system takes to execute all computing and communication requests (the total application execution time). This represents the overall time consumed from the moment of a service request to the moment all tasks are successfully processed. Therefore, this is the time interval between the application request and the acquisition of results. According to the three-tier model presented in the previous section, service time can be divided into the following intervals:

- data transfer time from IoT device to fog infrastructure,
- data transfer time from IoT device to the cloud,
- data transfer time from fog to the cloud,
- notification transfer time from the cloud to the fog,

- notification transfer time from the cloud to the IoT device,
- notification transfer time from the fog to the IoT device,
- computation time in the IoT device,
- computation time in the fog layer,
- computation time in the cloud.

Different cases of IoT system implementation may involve different time intervals, depending on the implemented system architecture. In general, service latency is distributed among IoT system components, including IoT devices, networks, and computing systems. Regardless of the different interaction models, the service time consists of the computation latency (T_{cp}) and communication latency or delay (T_{cm}) and can be calculated as the sum of these two time-related metrics:

$$T_{exe} = T_{cm} + T_{cp} \tag{31}$$

The time frames can be generated from the interaction model presented in Fig. 2. These frames can be used to show a subset of the time performances that are of interest for a given IoT application. The execution time must be reduced when scheduling tasks to the fog or cloud. The service time must be less than the requirements of an IoT application (T_{req}). Therefore, the objective function (2) aims to minimize service latency by reduction of computation or/and communication time.

$$Objective : min\ (T_{exe}) = T_{cm} + T_{cp} \leq T_{req} \tag{32}$$

Communication latency (T_{cm}) or network latency, is the time interval between the generation of a packet at a source node and its reception at the target node. In an IoT environment, network latency can be defined using two different approaches, depending on the specific use case. One-Way Delay (OWD) is the time from sending the first packet from the source to the network until the last packet arrives at the destination. The term IP Transfer Delay (IPTD) is used for this type of latency in IP-based networks. In addition, Round-Trip Time (RTT) can be used to measure network latency. It is the response time, which represents the time interval from the moment the first packet is sent from the source to the receiver until the source receives a response. In the three-tier architecture, RTT represents the time it takes for the data to travel from the IoT device to the corresponding computing system (fog and/or cloud) and the time required to send data back (response) to the source. Therefore, the communication latency can be divided into different intervals, including the data transfer time from the IoT device to the fog layer, data transfer time from the fog layer to the cloud, data transfer time from the Cloud to Fog layer or IoT device, and the data transfer time from the fog layer to the IoT device.

Both IPTD and RTT consist RAN (Radio Access Network) latency and WAN (Wide Area Network) latency. RAN latency represents the time it takes to transfer data over RAN such as Wi-Fi, Bluetooth, ZigBee, mobile networks (3 G, 4 G, 5 G), etc. which are used for connecting IoT device with the fog layer. Also, these technologies can be used for interconnection between IoT devices or interconnection between multiple fog infrastructures. There are different approaches to estimate the RAN latency such as the mathematical expressions proposed in [79–82]. WAN latency is the time interval of transferring data between fog and cloud infrastructure. It depends on the cloud server's location; for example, if the cloud servers are located in the same region as the RAN, then this latency is less than if the cloud server's position is more remote from the RAN. In general, there are four key causes of both types of network latency including packet processing delay (d_{proc}), queuing and buffering delay occurred at network nodes (d_{queue}), transmission delay (d_{tr}), and channel propagation delay ($d_{prop.}$).

The transmission and propagation delays are fixed depending on the type of link. Processing delay depends on CPU speed, CPU load, and complexity of table lookup while queuing delay depends on load (packet arrival intensity, packet size) and model queuing delay. By combining these components, the network latency for the path of N network sections can be calculated using the following equation:

$$T_{cm} = \sum_{i=1}^{N} \left(d_{i_{proc}} + d_{i_{queue}} + d_{i_{trans}} + d_{i_{prop}} \right) \tag{33}$$

Processing delay is the amount of time required to process the packet header (e.g., update the TTL), check for errors, determine the packet's next destination, and queue it for delivery. This latency occurs at the following nodes: source before sending, intermediate router, and destination before delivery to the application. The nodal processing latency is usually negligible compared to other components in the latency equation because of the very fast processing speed in new systems ($d_{i_{proc}} \rightarrow 0$).

The data transmission delay is also known as the "store-and-forward" delay. This is the amount of time required to push all the data into the transmission media (channel). Therefore, in IP networks, packet transmission delay is the time it takes for node i to copy the packet bits into the buffer, as well as to serialize packets over the communication link. Thus, the average transmission delay of a packet is proportional to its size (P_S) in bits and inversely proportional to the link speed (R_L) measured in bits per second (bps).

$$d_{trans} = \frac{P_S}{R_L} \tag{34}$$

The queuing delay is the amount of time the packets wait in the queue until they can be processed. This is the time the packets spend in routing queues, while the link is busy sending other packets. Queuing is required when the arrival rate of packets is faster than the transmission rate. As a queue begins to fill up owing to traffic arriving faster than it can be processed, the amount of latency a packet experiences going through the queue increases. This delay depends on the arrival and departure rates of packets, arrival patterns, buffer size, and queue depth, measured in bits. Depending on the QoS configuration, some packets can be prioritized over others. For example, multiple queues can be used, and different packets can be assigned to different queues. The maximum queuing latency is proportional to the queue depth, which depends on the load factor (intensity or rate of traffic arriving at a queue, transmission rate of the departure link, and load time). The network node (e.g., the router) has a finite amount of buffer memory to hold the queue. Therefore, a router that receives packets at a high rate can experience a full queue. In this case, the router has no option other than simply discarding excess packets. In this case, Queueing Theory can be used to calculate this type of delay. Examples of typical delay contributions by network routers, including queuing and processing delays, are provided in the ITU-T Rec. G.114, Y.1541, and Y.1542.

Propagation delay is the amount of time required for data to travel from source i to receiver j and depends on the length of the physical link (l_{ij}) and propagation speed of the media (c). This value can be calculated using the following equation:

$$d_{pr} = \frac{l_{ij}}{c} \tag{35}$$

When using a radio broadcast, a signal propagates at close to the speed of light. The speed of light (speed of electromagnetic wave) in a vacuum is 3.0 × 108 m/second. In a cable, c is 2.3 × 108 m/second while in a fiber 2.0 × 108 m/second. Propagation latency is negligible when the distance between the source and destination is small such as in the case of nodes in the same LAN (Local Area Network). In ITU-T Rec. G.114 [83] there are some descriptions for one-way propagation time according to the different transmission medium. For the route length (l_{ij}) calculation can be used ITU-T Rec. G.826, ITU-T Y.1541, ITU-T Y.1542, etc. For example, according to ITU-T Y.1541:

$$l_{ij} = \begin{cases} l_{ij\ (AR)} x 1,25\ for\ l_{ij\ (AR)} > 1200\ km \\ l_{ij\ (AR)} x 1,50\ for\ l_{ij\ (AR)} \leq 1200\ km \end{cases} \tag{36}$$

Computation latency (T_{cp}) corresponds to the amount of time required to perform computing tasks for a specific request. The

computational latency did not include any network latency. It depends on the computing power of the deployed infrastructure, such as the CPU performance and memory access time. Owing to the computing power limitations of IoT devices, some high-computing tasks may decrease the level of QoS performance, such as service time. Therefore, it is necessary to offload more advanced computational tasks from IoT devices to fog or cloud computing systems. Therefore, this latency includes the time period of executing operations at IoT device (local execution time) denoted as t_L and time period of executing tasks at other computing systems (remote execution time) denoted as t_{offi}. It can be deployed in several different computing instances for offloading tasks such as those described in Section 3. Therefore, if k computing systems are deployed at different layers of the system architecture (for offloading tasks), then T_{cp} can be calculated by using the following equation:

$$T_{cp} = t_L + \sum_{i=1}^{k} t_{ofi} \tag{37}$$

According to the three-tier model, T_{cp} consists local processing time (t_L), remote execution time including processing time at fog layer (t_F), and processing time at cloud computing system (t_C). It can deploy multiple fog nodes and servers in the cloud to offload tasks. For simplification purposes, a useful assumption is that all offloading tasks at different fog nodes are performed simultaneously because the time required to transfer data between fog layers is negligible. The same realistic assumption is made when distributing tasks between cloud servers. Therefore, the computation time for multiple fog layers is the maximum time required to execute tasks at a fog node (t_{Fi}). Hence, the computational latency can be calculated using the following equation:

$$T_{cp} = t_L + t_F + t_C \tag{38}$$

$$t_F = \arg\max_{i=1\ldots k} \{t_{Fi}\} \tag{39}$$

$$t_C = \arg\max_{j=1\ldots n} \{t_{Cj}\} \tag{40}$$

The computation time of each subsystem $t_{p_i} = \{t_L, t_{Fi}, t_{Cj}\}$ depends on the computing power of the system and the number of operations (tasks) that need to be executed. Generally, the computing power depends on the speed and architecture of the central processing unit (CPU), random access memory (RAM), internal hard drive (HDD or SSD) speed, graphics processing unit (GPU), etc.. Therefore, the computation time of each subsystem (t_{p_i}) consists of several components including CPU time, memory and disk time, GPU, etc.

$$t_{p_i} = t_{cpu_i} + t_{I/O_i} \tag{41}$$

Considering that computation latency is tightly coupled with the hardware specifications, nature of the application, and number of application instructions, our model takes on some level of abstraction. As the computation service is the process of transforming the input data into a meaningful form (information) in the CPU, CPU execution time is a key metric for computation latency. The CPU is the "brain" that handles all the computations needed to process instructions before sending output to the user. The CPU execution time consists of the user CPU time and the system CPU time (t_{OS}) and it doesn't count I/O time or time spent on running other programs. The following function describes CPU time for device i:

$$t_{cpu_i} = \frac{I_{CC_i}}{f_{cpu_i}} + t_{OS} = I_{CC_i} \cdot t_{cc_i} + t_{OS} \tag{42}$$

System CPU time is usually very small and it can be neglected ($t_{OS} \to 0$). Therefore, our focus is on user CPU time which represents the time spent executing the lines of code that are in the IoT program. It depends on the computing capability of device $i(f_{cpu_i})$ and the number of CPU clock cycles needed for a program(I_{CC_i}). f_{cpu_i} is measured as CPU clock rate(Hz), I_{CC_i} is the number of CPU clock cycles executed in system i, and t_{cc_i} is clock cycle time. I_{CC_i} depends on instruction type such as input data size, programming language, the complexity of the program algorithm, etc. However, not all types of program instructions take the same time to execute. Therefore, CPI (clock cycles per instruction) can be used as the average number of clock cycles each instruction takes. If I_{app_j} is the number of instructions for a program, then I_{CC_i} can be calculated by using the following equation:

$$I_{CC_i} = \sum_{j=1}^{k} I_{app_j} \cdot CPI_j \tag{43}$$

CPI varies by instruction type, programming language, compiler, and hardware implementation of the computing device. Based on hardware implementation, CPI can be grouped into different classes of instructions such as Class A, Class B, and Class C, that require one, two, and three CPI, respectively. The number of instructions for a program can be measured by using profilers/simulators without knowing all of the implementation details. Some authors present task profiles to estimate these parameters [84,85]. Some previous studies [84,86,87] can be used to derive a relationship between I_{CC_i} and input data size (D) as follows:

$$I_{CC_i} = D \cdot X \tag{44}$$

X is the processing intensity or processor workload that represents the number of processor cycles required to process a single bit of data. This random variable depends on the application parameters such as the complexity of the programming algorithm, the type of data being processed, the operating system, other active processes, etc. Some previous research presented X as a random variable with an empirical distribution [87–90]. As shown in [86,87], the number of CPU cycles per bit can be modeled by a Gamma distribution. Therefore, computation time (t_{p_i}) can be calculated by using the following equation:

$$t_{p_i} = \frac{\beta_i \cdot D \cdot X}{f_{cpu_i}} \tag{45}$$

β is the percentage of data processed in the system i. However, this model for CPU execution time should be expanded taking into account some other parameters such as spreading tasks across the multiple CPUs, or dual-socket processors. Also, the very interesting issue is the optimization the usage of multiple cores by writing program code to take advantage of this architecture.

Beside CPU execution time, the main areas of concern are related to CPU speed, RAM and storage characteristics. CPU process data that comes from some input component (e.g. network, the hard drive). The most of data go in RAM first as it is the main holding area of the computing system. The CPU then stores pieces of data it will need to access, often in a cache, and maintains certain special instructions in the register. The time a program or device takes to locate information and make it available to the CPU for processing depends on RAM memory (main memory). RAM is used by the computing system to store data that it is actively using. RAM delay is time period it takes to transmit data between a computer's RAM and its CPU. The access time of memory should be fast enough to keep up with the CPU. Otherwise, the CPU will waste a certain number of clock cycles, which makes it slower and processing time will be longer. Since RAM is not necessarily fast compared to the computer's processor, RAM delay can occur, causing a delay between the time a computer's hardware recognizes the need for a RAM access (initiates a request for data) and the time the data (or instruction) is available to the CPU. If the CPU requests data that is not stored in the cache, then it will have to wait for the RAM to retrieve the data. Generally, there are two main types of RAM memory with different performances. DRAM (Dynamic Random-Access Memory) chips for personal computers have access times of 50 to 150 nanoseconds while Static RAM (SRAM) has better performances with access times as low as 10 nanoseconds. RAM is much faster than hard drive storage, so the goal is to keep as much data in it as possible. However, it is shared by other

programs and the operating system itself that is grabbing the available RAM. This can result in the problem of "running out of RAM" which implies slower performance.

Disk drives (storage system) can be a crucial issue. The access time for disk drives depends on the type of disk (e.g. HDD or SSD). HDD access time includes the time it takes for the read/write head to locate a sector on the hard drive disk (called the seek time). This is an average time since it depends on how far away the head is from the desired data. Therefore, the disk time must be included if some program requires to read data from the local disk to execute tasks. For example, face recognition application first detects faces in an image (captured by the camera) and then attempts to identify the face from a pre-populated database which can be stored at a local device. In this case, its required to take into account and time required to read data from the disk. This time period depends on data size required to read from disk and hardware specifications such as rotational speed, read seek time, number of heads, sectors per track, bytes per sector, etc. This information is usually provided by the disk drive manufactures. This latency can be critical if there is a need to read big data from disk. The emergence of solid-state drives (SSD) has dramatically accelerated storage access. Therefore, high-speed SSDs can be used as a storage or it can be dedicated to caching process as a separate cache space. The main characteristics of SSD which influence access time is that information is stored in microchips and there are no moving parts.

In computer architecture, it is used the memory hierarchy which separates memory and storage into a hierarchy based on response time. The number of levels in this hierarchy and the performance at each level has increased over time. Time to read data from the memory or disk can be reduced by using cache because caching enables to increase computation speed. Therefore, two options must be distinguished including the case without caching option on computation device and option when caching process is enabled. Cache is an area of temporary volatile storage either on the controller, reserved section of main memory or an independent high-speed storage device that has a faster access time than the main memory and storage. Generally, two types of caching are commonly used in personal computers: memory caching and disk caching. Some memory caches are built into the architecture of microprocessors such internal caches known as Level 1 cache (L1). External cache memory, also known as Level 2 (L2) and Level 3 (L3) caches sit between the CPU and the DRAM. A memory cache is a portion of memory made of SRAM instead of the slower DRAM used for main memory. Memory caching is effective because most programs access the same data or instructions over some time. By using SRAM as a memory cache, the computer avoids accessing the slower DRAM. Disk caching works under the same principle as memory caching, but instead of using high-speed SRAM, a disk cache use other locations such as a separate SSD. These caching processes can dramatically improve computing performance. For example, data requested for the computing process might already be in the cache from a previous operation, thus eliminating the need to read data from the disk drive or memory access.

4.2. Energy consumption

Increasing demand for data transferring and processing cases development of more and more sophisticated and power-hungry technologies. Appropriate mechanisms are needed to increase energy efficiency. Therefore, energy consumption is another important KPI that includes several metrics such as an energy efficiency (e.g. energy to deliver a single bit of data), power usage effectiveness (e.g. power consumed by IoT devices for computation processes), etc. Energy consumption (E) is the total amount of work performed by a system over a time period (T) while power (P) is the rate at which the work is performed by the system [91].

According to the three-tier model there are different sources of energy consumption for each layer. In the fog and cloud, the energy consumption is caused by four main sources including computational processes, communication processes, cooling systems, and idle energy. However, in most of the cases, fog and cloud infrastructure have access to a permanent power supply. On the other hand, IoT devices often operate wirelessly on limited battery supply. Therefore, IoT devices may have stringent constraints on the power supply. Due to the small size of IoT devices (in most of the cases) and mobility requirements in some cases, property devices rely on batteries for power supply. Some computation processes may drain the available battery energy and because of that, the energy efficiency is one of the major research fields in the IoT industry. Consequently, the battery life can be considered as IoT system lifetime which depends on the battery capacity (the remaining energy) and the total energy consumption. This is the reason to put IoT device power consumption in the focus of this paper.

From the point of view of a single IoT device requesting to run an application, the main sources of energy consumption are related to computation (E_{cp}), and communication processes (E_{cm}). There are some other sources of energy usage (E_{other}) such as sensing process, GPU (Graphics processing unit), display, etc. When tasks are offloaded from IoT device, it is in the idle state which implies lower energy consumption (E_{idle}). Therefore, the energy consumed by IoT devices for executing IoT application is a function of overall hardware resource usage by all service processes:

$$E_{dev} = F\{E_{cp}, E_{cm}, E_{idle}, E_{other}\} \tag{46}$$

A model of energy usage for IoT device can be built by considering the proposed three-tier model including computing and communication processes. It consists of energy spent on local computation ($E_{loc.}$), energy spent for offloading processes ($E_{off.}$), and energy while in the idle state (E_{idle}).

$$E_{dev} = E_{loc.} + E_{off.} \tag{47}$$

The energy required for local computation depends on the power consumption required for local computation tasks (P_L) and local computation time (t_L).

$$E_{loc.} = P_L \cdot t_L \tag{48}$$

P_L depends on IoT device and type of computation tasks. There are some researches that provide P_L values for different processes and devices such as [75,79,84,92,93] while t_L can be calculated by using the model described in this paper. Some authors proposed other models such as [76,94] where computational energy is calculated as a product of I_{CC_i} and CPU energy consumption required by each CPU cycle.

$$E_{loc.} = k \cdot I_{CC_i} \cdot f_{cpu_i}^2 = k \cdot D \cdot X \cdot f_{cpu_i}^2 \tag{49}$$

k is constant related to the hardware specifications, D is input data size in bits, and X is computation intensity/workload (required CPU cycles per bit). In [91] authors described different energy consumption models according to energy used by some system components such as CPU, memory, and I/O devices.

Energy for offloading tasks consist energy required for sending data for offloading process and receiving results if it is required (E_{comm}), and the energy used while being in the idle state (E_{id}) while waiting results:

$$E_{off.} = E_{cm.} + E_{idle} \tag{50}$$

E_{id} is energy used while IoT device is in the idle state which means the device is waiting for the result of the remote computation. Hence, it depends on power consumption while being in idle state and time spent for remote computation (t_{off}). Value for P_{id} can be found in some previous researches such as [75,95].

$$E_{idle} = P_{idle} \cdot t_{off} \tag{51}$$

$$t_{off} = T_{exe} - (t_L + T_{cm}) \tag{52}$$

In general, energy usage for the communication process consists the energy needed to transmit data to offloading infrastructure (E_{tx}), and

energy during reception of data (E_{rx}).

$$E_{cm} = E_{tx} + E_{rx} \tag{53}$$

E_{tx} differs from E_{rx} which depends on deployed communication technology (e.g. Wi-Fi, LTE). Different RATs require different power for sending and receiving data. These values depend on power consumption for transmitting (P_{tx}) and receiving data (P_{rx}) and time spent on sending data (t_{tx}) and receiving data (t_{rx}).

$$E_{tx} = P_{tx} \cdot t_{tx} \tag{54}$$

$$E_{rx} = P_{rx} \cdot t_{rx} \tag{55}$$

Huang et al. [95] proposed a power model for data transfer including different power levels for uplink and downlink:

$$P_{tx} = p_u \tau_u + \beta \tag{56}$$

$$P_{rx} = p_d \tau_d + \beta \tag{57}$$

Uplink throughput is τ_u, downlink throughput is τ_d, while p_u and p_d are power required for data transfer in uplink and downlink, respectively while β is idle power. These values depend on communication technology [79,96,97], protocols [98], and deployed device [99]. For simultaneous uplink and downlink transfers, the power level can be calculated by using the following equation:

$$P_{trx} = p_u \tau_u + p_d \tau_d + \beta \tag{58}$$

Dividing the Eqs. (56) and (57) with the bandwidth, results the equation for the energy efficiency of the network in terms of the energy required to transmit a certain amount of data (energy per bit). In this way, the energy required for sending $E(D)_{tx}$ and receiving $E(D)_{rx}$ data can be estimated.

$$E(D)_{tx} = p_u + \beta \tau_u^{-1} \tag{59}$$

$$E(D)_{rx} = p_d + \beta \tau_d^{-1} \tag{60}$$

D_{tx} and D_{rx} are the amounts of data (in bits) sent and received by the IoT device. Finally, by considering previous expressions, the equation for calculating energy consumption is:

$$E_{dev} = P_L \cdot t_L + P_{tx} \cdot t_{tx} + P_{rx} \cdot t_{rx} + P_{id} \cdot t_{off} \tag{61}$$

$$E_{dev} = P_L \cdot t_L + (p_u \tau_u + \beta) \cdot t_{tx} + (p_d \tau_d + \beta) \cdot t_{rx} + P_{id} \cdot t_{off} \tag{62}$$

$$E_{dev} = P_L \cdot t_L + \left(p_u + \beta \tau_u^{-1}\right) \cdot D_{tx} + \left(p_d + \beta \tau_d^{-1}\right) \cdot D_{rx} + P_{id} \cdot t_{off} \tag{63}$$

One of the main challenges is the power supply of an IoT device. In some applications, IoT devices use batteries where it is not possible to provide a battery replacement. The initial value of the energy by some device is $E_{dev}(i)$. Each IoT device consumes this energy due to the processing data, sending and receiving data, and while it is in idle state. The available energy $E_{dev}(t)$ is reduced over some time t. Therefore, the residual energy $E_{dev}(r)$ or the system lifetime can be estimated by using the following equation:

$$E_{dev}(r) = E_{dev}(i) - E_{dev}(t) \tag{64}$$

The battery lifetime or the remaining battery lifetime $T(sys)$ depends on the capacity of the battery or the remaining energy and power required to perform all the services. Energy is consumed depending on the power required for local data processing (P_{cp}), data transmission (P_{cm}), and other processes (P_{other}) other processes such as sensing, detection, etc.

$$T(sys) = \frac{E_{dev}(r)}{P_{cp} + P_{cm} + P_{other}} \tag{65}$$

One of the challenges is related to the power supply of the IoT devices. In some cases, IoT devices use batteries, and it is not possible to replace batteries owing to dynamic environments. Battery constraints are often used as an indicator of IoT device lifetime and can be used as one of the most important QoS metrics[100,101]. Therefore, the objective function (26) aims to minimize energy consumption to maximize the overall system lifetime.

$$Objective : min\ (E_{dev}) \tag{66}$$

5. Performance evaluation model

Most IoT applications rely on computing systems to meet QoS requirements. However, there are some bottlenecks for each model of integration of IoT and other systems, which pose the problem of selecting the corresponding technologies and the model of integration. The approach of IoT system modeling provides an overview of possible technologies and infrastructure for deployment in the developing process. It is possible to use different metrics for IoT_{sys} evaluation such as described in the previous Section. However, there is a need for a quantitative evaluation method that includes multiple metrics to find an eligible deployment for any layer of the IoT system architecture. In this study, an analytical-based evaluation model was proposed. This model can be used as an efficient tool for selecting the corresponding architecture and technologies according to IoT application requirements. It can be used as a framework for a decision module for IoT_{sys} that has the responsibility to decide whether to offload some tasks to other systems (e.g. network or cloud infrastructure) and how to do it. Therefore, it is necessary to estimate KPIs to make offloading decisions according to the IoT application profile.

There is a correlation between the application performance and the features of the IoT infrastructure. However, different KPIs (for example QoS metrics) cannot be compared against each other as such because of different values range and important levels (weighted factors). To overcome this issue, a method that provides unitless values in the normalized range is required. In this context, an application utility function $IoT_{sys}(p)$ can be used to evaluate different IoT_{sys} models using KPIs.

$$IoT_{sys}(p) = \sum_{i=1}^{n} w_i \cdot f(p_i) \tag{67}$$

n is the number of KPIs (P_i) used for IoT_{sys} performance evaluation. $f(p_i)$ corresponds to the specific utility functions used to evaluate each KPIs which are scaled within the same range [0,1]. w_i are weight coefficients for $f(p_i)$. Specific utility functions are multiplied by weight coefficients to determine which KPIs are considered more important. The application utility function $IoT_{sys}(p)$ represents the degree of performance of the IoT system. Therefore, our problem formulation aims to provide an offloading decision for IoT applications to maximize QoS performance and resource utilization. $IoT_{sys}(P)$ is a user-centric QoS and resource utilization function that means the performance level is considered for the particular application. Monitoring of the level of utility function, which is an indicator of IoT_{sys} performance level, can be performed by applying statistical tests and analysis. A higher value indicates a positive effect on IoT_{sys} performance. Therefore, the problem formulation is equivalent to:

$$\max_{p_i} IoT_{sys}(p) \tag{68}$$

According to previous considerations, the application and KPIs utility functions have the following properties:

- $IoT_{sys}(p)$ is an increasing function of IoT_{sys}.
- $f(p_i)$ is an increasing function of P_i.
- $IoT_{sys}(p)$ and $f(p_i)$ are scaled with the range [0,1].

The challenge is to determine KPIs and their weighted coefficients as well as to model $f(P_i)$ for each KPI. Service latency and energy con-

sumption are the main KPIs for system-related characteristics, which indicate the performance level of an IoT system. Therefore, the overall performance level can be calculated as the sum of the utility function values for each of these KPIs:

$$IoT_{sys}(p) = w_{T_{exe}} \cdot f(T_{exe}) + w_{E_{dev}} \cdot f(E_{dev}) \tag{69}$$

$f(T_{exe.})$ and $f(E_{con.})$ are utility functions for execution time and energy consumption, respectively, whereas $w_{T_{exe.}}$ and $w_{E_{con.}}$ are their weighted coefficients. Weighted coefficients (w_i) represent the relative importance of the KPIs for some IoT application profiles.

$$\sum_{i=1}^{n} w_i = 1 \tag{70}$$

w_i depends on the IoT application requirements. For example, an analytic hierarchy process (AHP) can be used to obtain their values, as described in [102]. AHP is a mathematical method for analyzing and organizing complex decisions using ratio-scale measurements.

Determining $f(P_i)$ for each KPI is a nontrivial problem that can be addressed by modeling specific utility functions using sigmoid curves. This approach is well known, and sigmoid functions are often used for QoS evaluation and for solving resource allocation problems [103,104]. There are two different cases, $f(p_i) = \{f(p_i\uparrow); f(p_i\downarrow)\}$, depending on the KPIs meanings. For example, for parameters such as network throughput, a higher value represents better performance. Therefore, the utility function must increase the sigmoid function $f(p_i\uparrow)$. Some other parameters such as service latency and energy consumption need to be minimized for increasing performance level (the related utility functions need to be decreasing sigmoid functions): $f(p_i\downarrow)$.

$$f(p_i\uparrow) = L\left(\frac{A}{1 + e^{-\alpha_k \frac{(p_i - r_i)}{Z_i}}} - U\right) \tag{71}$$

$$f(p_i\downarrow) = 1 - L\left(\frac{A}{1 + e^{-\alpha_k \frac{(p_i - r_i)}{z_i}}} - U\right) \tag{72}$$

$$L = \frac{1 + e^{\frac{\alpha_k \cdot r_i}{z_i}}}{e^{\frac{\alpha_k \cdot r_i}{z_i}}}, \cdot U = \frac{1}{1 + e^{\frac{\alpha_k \cdot r_i}{z_i}}} \tag{73}$$

where A is the maximum value of $f(p_i)$. Therefore, for real values of the function in the range [0, 1], $A = 1$. p_i indicates the measured performance level for some KPI (p_i), and Z_i is a tunable parameter for adjusting the function to unitless values. The steepness of the curve depends on α_k $(k = 1,\ 2,\ 3)$ and R_i $(i = 1,\ 2,\ 3)$ which represent the sensitivity to performance degradation and sigmoid midpoint (center points of the function curves), respectively. R_i can be represented as a reference value for the KPIs, and a higher value of R_i represents a more demanding application. With a higher level of α_k, the IoT application is relatively firm in terms of the performance degree. Here, L and U are coefficients used to verify that $f(0) = 0$ and $f(\infty) = 1$, which indicates that an infinite resource assignment leads to a maximum performance level.

In addition to determining KPIs, weighted coefficients, and the model for calculating utility function, it is necessary to determine IoT application reference values (r_i). These references can be requirements or objective values for $(p_{i_{req}})$. This can be obtained from recommendations (e.g., for network-based QoS parameters). Another way of obtaining values for R_i is to use a maximum or mean value of p_i, depending on the IoT application type. Using these equations, the performance level for each KPI as well as for the overall IoT_{sys} can be calculated.

The proposed method provides the flexibility to include measurable parameters, even if the reference values are unknown. In addition, it does not depend on the choice of KPI because of the unitless parameters and value range independence. This method provides the possibility of adjusting the utility function based on the sensitivity of the individual parameters. Therefore, it has several advantages over other methods such as [71] and [101,105].

6. Simulation results and model validation

A realistic scenario for an IoT system can be used to evaluate the proposed model. Various tools, such as NS-3 and EdgeCloudSim network simulators can be used to verify the results. The use of comprehensive network simulators is helpful for simultaneously considering many different interacting elements of an IoT system [15,106–109]. Therefore, a comparison of the results obtained by numerical analysis with the network simulator NS-3 and some experimental measurements provides the validity of the proposed model and the possibilities of applying this model in realistic IoT scenarios.

In this paper, an IoT-based warehouse monitoring system [110] was used to evaluate the proposed model. Some unexpected events, such as temperature fluctuations, humidity changes, and other unwanted events, may destroy the value of goods and products stored in premises, such as warehouses. Under such circumstances, continuous monitoring and condition control are required. IoT enables the development of cost-effective solutions. Fig. 4 shows the structural model of an IoT system based on a three-tier architecture.

The described IoT system for monitoring premises may include several applications, such as monitoring the conditions where goods are stored (e.g. warehouses), detecting anomalies of measured sizes (e.g. temperature fluctuations), triggering certain actions with an actuator, controlling access to premises, etc. The specifications of technologies and infrastructure can cover a large number of components, depending on the IoT system structure. The key components of the system are listed in Table 3.

Sensors were used to measure temperature, humidity, and illumination. They are connected to a microcontroller that performs filtering, aggregation, pre-processing, and data forwarding to the IoT gateway. For example, the NodeMCU (ESP8266EX MCU) is a cost-effective microcontroller that acts as an aggregator of multiple raw datasets collected from sensors. It was equipped with a Wi-Fi module (IEEE 802.11 b/g/n) to send data to the network gateway.

The gateway is implemented by using the Raspberry Pi 3 Model B + with a 1.4 GHz processor clock speed (Broadcom BCM2837B0, Cortex-A53 64-bit). It enables the predefinition of decisions by implementing a program code to decide whether to forward data to a cloud server, local server, or terminal. Therefore, Raspberry Pi is a fog layer in the proposed system architecture. For advanced data processing and storage, a cloud server with a CPU @ 3,60 Ghz (8CPUs) and 16 GB RAM was used. A key role of this infrastructure is to analyze and store data in a database (for example, MongoDB) and host a web application. MongoDB), and to host a web application. This implementation requires a permanent internet connection. However, a local server can be used as an alternative solution to make the system cost-effective, secure, and reliable. This alternative solution eliminates the need for a permanent internet connection. For experimental measurements and analytical evaluations, a local server with a CPU @ 2.10 Ghz (4CPUs) processor and 4 GB of RAM was used. Mobile applications (for example Android or iOS) allow stakeholders (e.g. product owners) to have an insight into the warehouse conditions in real time. In this case, the terminal device can be considered as another fog layer.

6.1. Scenarios for IoT system analysis

To achieve the desired level of QoS and optimize the use of available resources, an appropriate decision-making mechanism must be implemented. This mechanism sends instructions to the individual system components. There are several possible implementations of the proposed IoT system (Fig. 5). Different systems can be used for data processing, such as IoT devices (Dev), Raspberry Pi (F1), local servers (F2), and cloud servers (C). Therefore, these implementations include

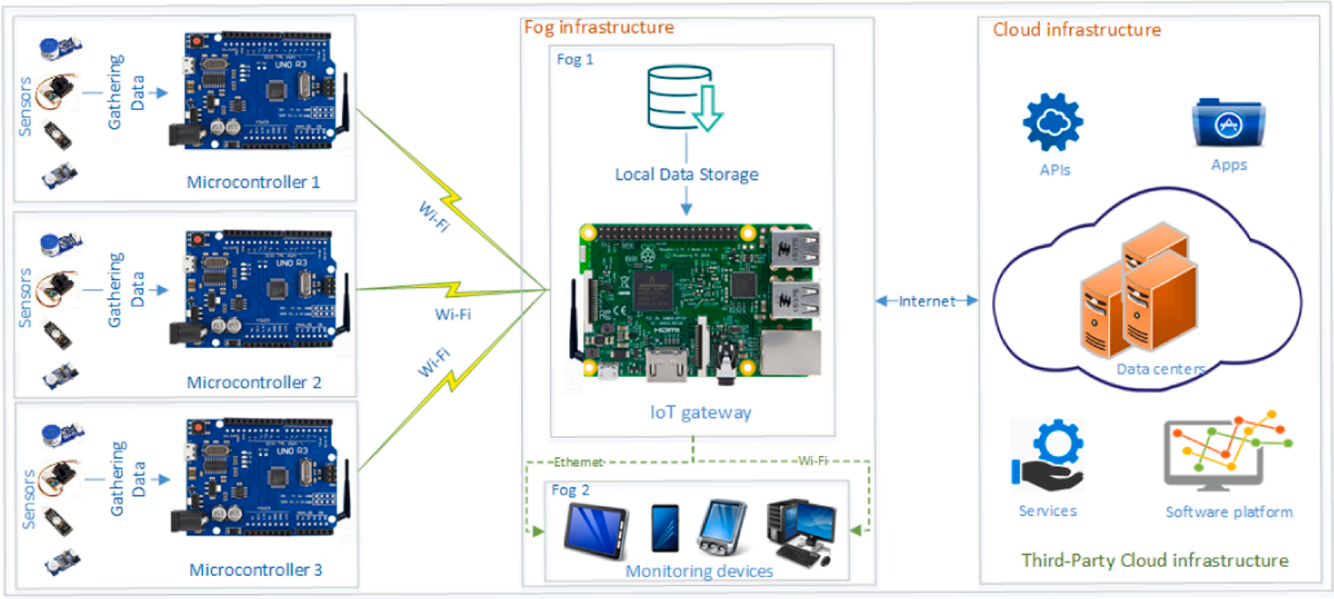

Fig. 4. The structure of the IoT system. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

different integration models, and the following alternatives can be distinguished:

- The *Dev (no offloading)* scenario implies that data processing is performed on an IoT device (NodeMCU), and the results are transmitted over the network gateway to the destination (monitoring center and/or actuator). In this case, a time interval for data processing is set on the IoT device (e.g., the microcontroller processes the collected data every second), or the microcontroller processes the data after collecting a certain amount of data from the sensor (e.g., 320 KB).
- The *F1-D (LAN)* scenario involves processing data in Fog Layer 1 (Raspberry Pi) and then transmitting the results to the destination via a LAN (Ethernet).
- The *F1-D (Wi-Fi)* scenario involves processing data in Fog Layer 1 (Raspberry Pi) and then transmitting the results to the destination using Wi-Fi.
- The *F2 (LAN)* scenario involves processing data in Fog Layer 2 (local server or terminal device). When data processing and visualization are performed by the terminal device, then F2 is the final destination. If the results need to be forwarded to another device (e.g., an actuator), the resultant transmission time can often be neglected if the destination device is in the same location (e.g., F2 and the destination device are in the same LAN). This assumption is based on the fact that the results generally involve a much smaller amount of data than the original and unstructured data, and are transmitted over small distances. It is assumed that data from the IoT device to the gateway are sent over Wi-Fi and then transmitted over the LAN to the desired destination.
- The *F2 (Wi-Fi)* scenario implies the same processes as *F2 (LAN)*, only it uses other communication technologies to transmit the results. The results from the gateway were forwarded via Wi-Fi to the final destination (F2).
- The *C-D (LAN)* scenario implies that data processing is performed on the cloud server, and then the results are returned via the Internet to the gateway and further through the LAN to the destination.
- The *C-D (Wi-Fi)* scenario implies that the data processing is performed on the cloud server, and then the results are returned via the Internet to the gateway and further through the Wi-Fi to the destination.

The key issue is to choose the best option for task allocation to achieve the desired level of QoS as well as to optimize resource utilization.

Fig. 6 shows the algorithm of the described system. If the IoT

Table 3
Technology and infrastructure specification.

Domain	Enabling technologies and infrastructure	
Application Domain	*Application*	Monitoring and control of warehouse conditions
	OS and API	OS (Android for mobile devices, Apache web server, Windows server); RTOS (Raspbian for Raspberry Pi platform); API (WebGL JavaScript API, HTTP Server API, API for Raspberry Pi, Google APIs for android)
	SDK and IDE	Web app: HTML, CSS, JavaScript, PHP; Mobile app: Android SDK with API libraries and debugging tools; Raspberry Pi: Arduino IDE, Python; Microcontroller programming: Arduino IDE
	Architecture	RESTful (IP address + port number, e.g., 8125)
Middleware Domain	*Cloud and fog platforms*	Cloud platform: private cloud infrastructure; Fog 1: Raspberry Pi 3 Model *B+*; Fog 2: Local Windows Server
	Data manipulation	NoSQL database (MongoDB) File format: CSV (*comma-separated values*)
Networking Domain	*Protocol stack*	Application (CoAP, HTTPS), Transport (UDP), Network (IPv4)
		Network interface (IEEE 802.11n)
	Network interface	WLAN: ESP8266 Wi-Fi module; LAN: Ethernet 1000Base-T, Cable Cat 6, 1 Gbps; WAN: Internet
Devices Domain	*Microcontroller and equipment*	NodeMCU (ESP8266EX MCU), Tensilica L106 32-bit RISC processor 160 MHz, voltage transformers, batteries, resistors, etc.
	Sensors and actuators	Actuators for automatic regulation of measurement parameters Sensors: temperature, humidity, light, etc.

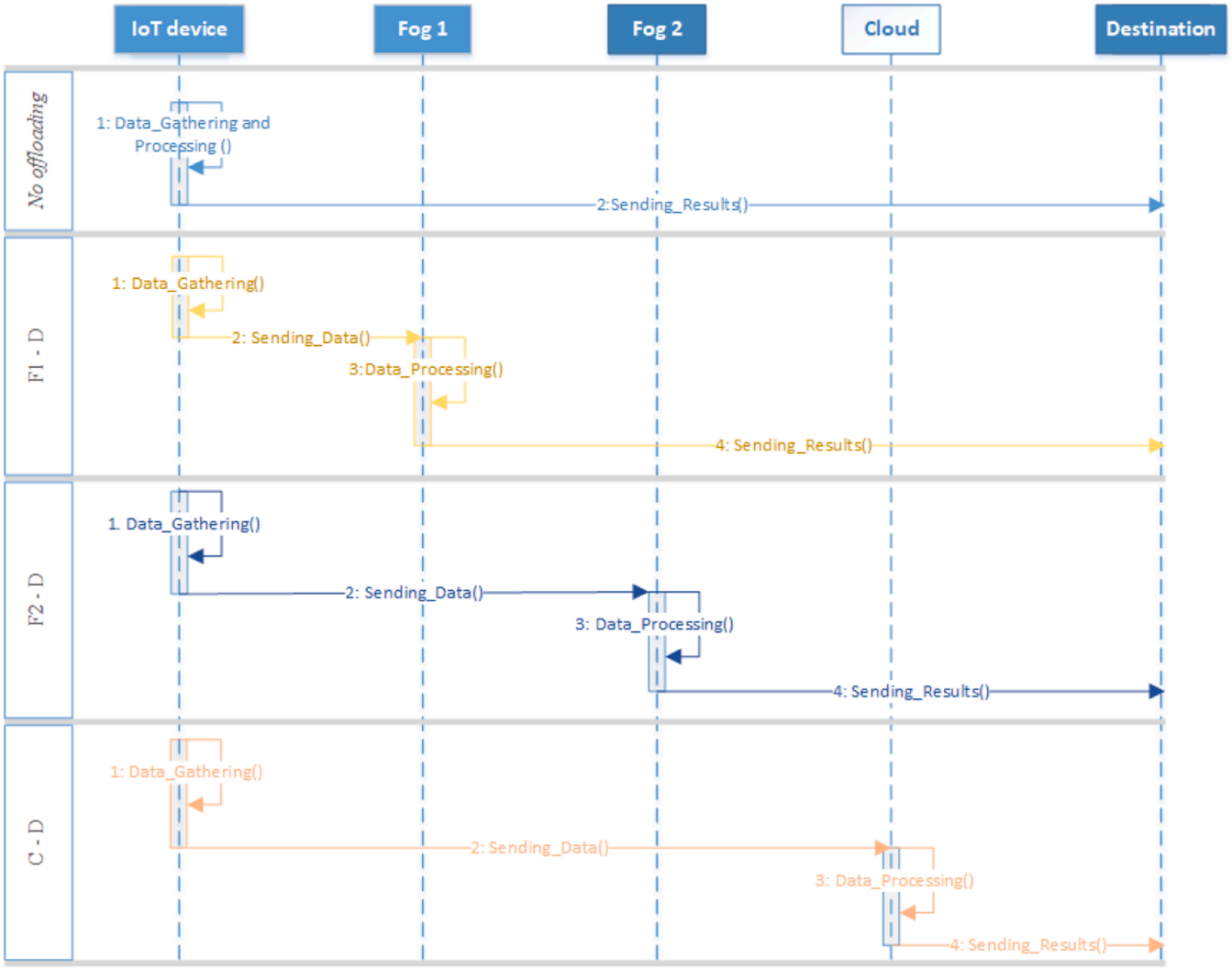

Fig. 5. Tested IoT system scenarios (Data flow of a tested IoT environment). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

monitoring and control system requires real-time operation, then the service execution time is set as one of the key performance indicators. Energy consumption is another key indicator of IoT system performance. In terms of power, the most critical component is the IoT device because of battery usage. Therefore, the power consumption of IoT devices is considered to be one of the KPIs.

$$p_i = \langle T_{exe}, E_{dev} \rangle \tag{74}$$

There may be different numbers of IoT devices (k) in the system that generate data. If the devices have the same capabilities and configuration, they generate the same amount of data over time (D). Therefore, the system load can be observed as a function of the number of IoT devices and the amount of data generated from each device.

$$S_{sys(w)} = \langle k, D \rangle \tag{75}$$

6.2. The application execution time

Application execution time represents the period required to perform all the necessary operations required by the application ($T_{exe.}$). This time consists of two components: the data transfer time (network delay) and the data processing time. Furthermore, these times can be divided into different intervals depending on the system components. Data transfer time ($T_{cm.}$) is the time required for data to be transferred to a destination. This includes the time of data transfer to the processing system and the time of transmission of the results to the destination. Data processing time ($T_{cp.}$) is the time required for a particular system to process data. Data processing can be performed by IoT device or the offloading systems. The time segments depend on the integration model, which means that each model includes different components of time.

6.2.1. Data transfer time

The NS-3 network simulator was used to verify the proposed analytical model. Scenarios with different numbers of nodes (IoT devices) with the same settings were tested to generate the same amount of traffic. This implies that the system load depends on the number of devices. Wi-Fi was used to send data from the IoT device to the gateway. A Wi-Fi or LAN (Ethernet) network is used to transfer the results (processed data) from the processing system to the destination. The application of the NS-3 Network Simulator implies additional simulation settings, as presented in A maximum deviation of 4.6 % was obtained for the data transfer scenario to the fog layers, whereas for the other scenarios, the deviation was less than 2 %. The biggest difference is the radio link transmission because the NS-3 simulator includes some other elements, such as the Yans channel features. Nevertheless, the small differences in the results indicate that the application of the proposed analytical model for estimating network delay is acceptable in most systems.

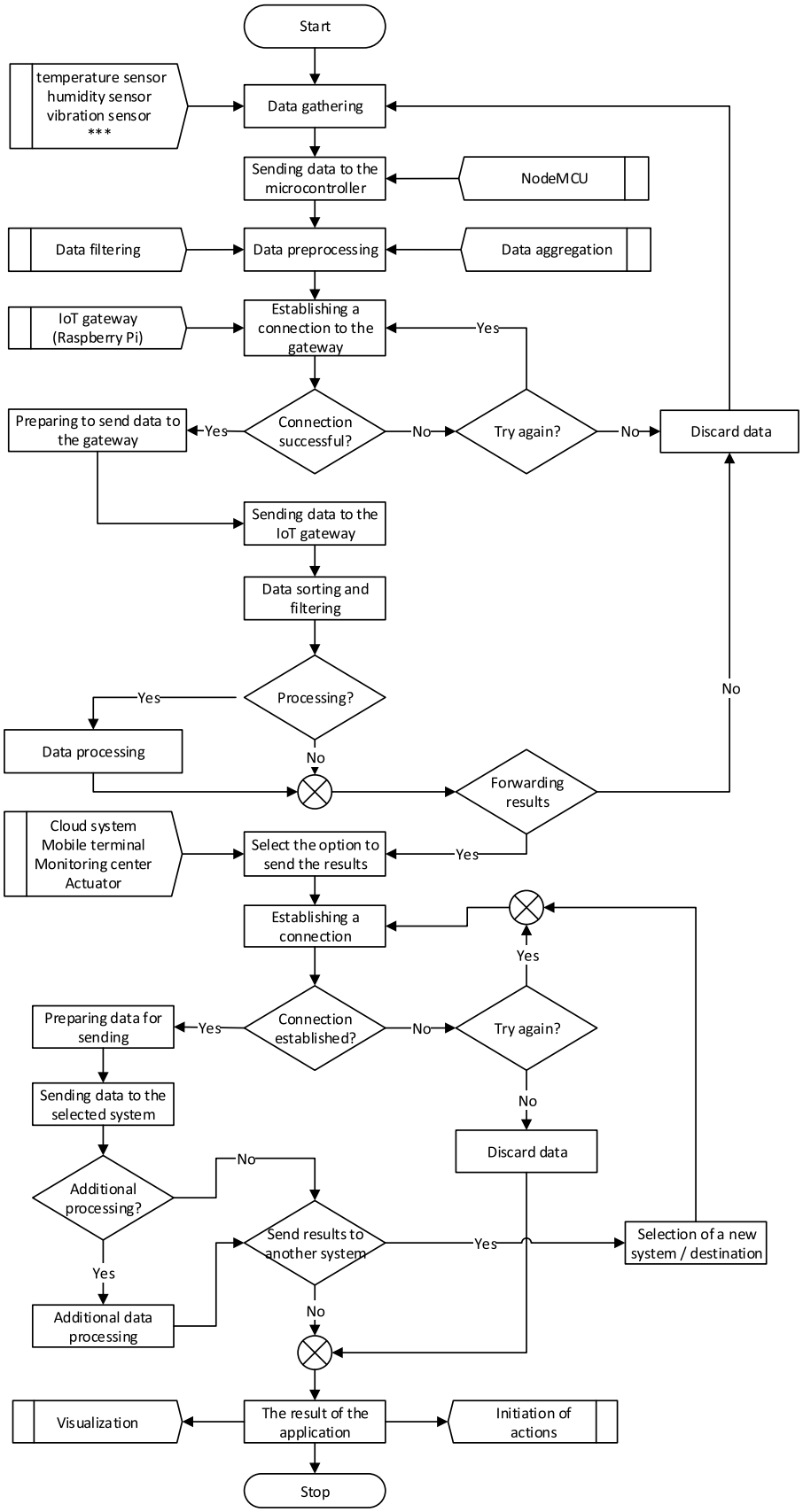

Fig. 6. The IoT system algorithm.

Table 4. The results for different scenarios are shown in Fig. 7. These results vary depending on the number of nodes, distance of the data source from the destination, and communication technology applied. However, as these are the result values expressed in milliseconds, these differences are not significant for most applications.

A maximum deviation of 4.6 % was obtained for the data transfer scenario to the fog layers, whereas for the other scenarios, the deviation was less than 2 %. The biggest difference is the radio link transmission because the NS-3 simulator includes some other elements, such as the Yans channel features. Nevertheless, the small differences in the results indicate that the application of the proposed analytical model for estimating network delay is acceptable in most systems.

A comparison of the results allows the selection of the best service distribution solution depending on the network delay as one of the possible metrics. The data transfer times for the different layers of the system architecture varied considerably. The increase in network latency is particularly pronounced when data are transferred to the cloud system. Therefore, different integration models significantly affect the network QoS performance.

6.2.2. Data processing time

The application of the proposed analytical model for estimating the data processing time imposes the problem of determining the random variable X. For this purpose, a custom programming code was developed. Therefore, the estimation of X was based on the experimental measurements of the data processing time. It is not practical to perform measurements for each device individually or for different workloads. Measurements were made on several devices with different hardware and software characteristics to estimate the mean value of parameter X. To create a more realistic scenario, different files of different sizes, including 10, 40, 160, 320, 640, 1280, and 2560 (KB) were used. It is considered that the data comes from a different number of nodes including 1, 4, 8, 16, 32, 64. Experimental measurements were performed on a Raspberry Pi (Fog 1), local server (Fog 2), and cloud server. In addition, other services were launched on the local server to test how much the value of X depends on other services. X was estimated for each scenario, including 1500 (IoT gateway), 700 (local server), and 430 (cloud server). Systems with better performance have better efficiency (fewer CPU cycles are required to process the same amount of data). Therefore, the value of X for the other systems can be assumed (for example, X for NodeMCU is 2000). There were significant differences in the data processing times for the different systems (Fig. 8). A similar process can be used for other types of applications that require different services and data processing in other formats. With an increase in the number of nodes (IoT devices), which implies an increase in computing requirements, the difference in processing times on different systems increases.

Data processing on the device itself is constant (it does not depend on the number of devices), because each device has the same characteristics and settings. In the case of fewer IoT devices (up to 10), it is useful to use an offloading system. As the number of devices increases (to more than 10), it is not efficient to use Raspberry Pi (Fog Layer 1) for data processing. Using a local server (fog 2 layer) was better when there were fewer than 35 devices. However, it is better to use a cloud system for a larger number of devices. The cloud system achieved the best results for the maximum number of devices (64). The graph shows an exponentially increasing trend as the system load increases (i.e., the number of data-generating devices). Thus, if more devices are included in the system, the assumption is that a similar situation will occur in the Cloud as with Fog 1 and Fog 2. Therefore, at some point of increasing system load, using IoT devices for data processing could be a better solution (if possible). However, the Cloud system has the option of load-balancing. This is the process of distributing workloads across multiple computing resources to achieve the best performance.

All IoT devices have the same characteristics and settings; therefore, the data-processing times of these devices are identical. However, the deployment of offloading infrastructure implies that the processing times on Fog 1, Fog 2, and the cloud infrastructure differ significantly in cases of change in the number of traffic-generating devices (system load). A comparison of the results obtained experimentally and using the proposed model is shown in Fig. 10. The results show that increasing the number of nodes reduces the percentage difference between the proposed model and the experimental measurement. For example, for Fog 1, the infrastructure differences in the results of the proposed analytical model and experimental measurements were reduced from 8 % for one node to 3 % for 64 nodes. This situation is similar to the cloud system, where the differences are reduced from 7 % for one node to 0.4 % for 64 knots. The differences in Fog 2 infrastructure were slightly larger, as other programs were active. Thus, it was confirmed that the system load significantly affects the processing time estimate, but also that the value of variable X depends on the hardware characteristics of the system, the amount and type of data, and the system load on other processes and services.

Appropriate network time must also be added to the processing time. The total execution time of a service depends on both components (computation and communication times). For example, data transfer to the cloud requires significantly more time than data transfer to the fog infrastructure. However, using a cloud system for data processing significantly reduces the computation time. This imposes the problem of determining the optimal offloading solution for different scenarios (applications).

6.2.3. Service latency

Service execution time or service latency ($T_{exe.}$) consists of the computation latency (T_{cp}) and communication latency or delay (T_{cm}) and can be calculated as the sum of these two time-related metrics. Therefore, the real-time performance is obtained by summing the appropriate communication (data transfer) and computation (data processing) times. Owing to the large difference in the transmission and data processing times for each scenario, the summary times are expressed in seconds. The service execution times for all scenarios can vary significantly depending on the number of nodes and the deployed offloading system (Fig. 9).

When it comes to fewer nodes (~10), it is useful to distribute the data processing service to some offloading systems. For a higher system load, using a local server (Fog 2) or cloud system for offloading IoT devices shows significantly better results than using Raspberry Pi. However, as the number of devices increased, the load on the local server also increased.

For a higher number of IoT devices (~ 30), the service time increases significantly compared with the time it takes for each device to perform data processing. In such cases, the allocation of data-processing services to the cloud proved to be the best model. The trend of the results shows that with increasing system load, there may be a case where the best alternative is to use IoT data-processing devices themselves (if they have processing capabilities). However, in such cases, there may be other challenges such as higher energy consumption.

Table 4
Network simulation parameters.

Parameter	Value
Number of nodes (IoT devices)	1–64
Area size	50 × 30 m
Mobility model	None
The size of the data to be transferred	320 KB
Result file size (after processing)	1 KB
MAC/PHY standard	IEEE 802.11n, Yans channel
WAN latency	30 ms (constant propagation model)
Traffic intensity from one device	1 Mbps - CBR (*Constant Bit Rate*)
Transport protocol	UDP
Packet size + header	1024 + 28 bytes
LAN capacity and technology	IEEE 802.3ab (1000Base-T), 1 Gbps

Fig. 7. Network latency: Analytical (proposed) model vs. Simulation model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 5. shows the contribution of network delay to the total service latency for each observed scenario. Certain patterns can be observed, depending on the integration model. Without the offloading system (data are processed on the devices), the computation time is critical. In other cases, variations occur depending on the system load (i.e., the number of devices). Without offloading IoT devices, the processing time is much longer than the network time. In this case, it is crucial to improve the performance of the device to reduce the overall service time. When using Raspberry Pi for data processing, the processing time was dominant. However, for smaller numbers of devices, this difference is much less pronounced. When a local or cloud server is deployed for data processing, the network delay is more dominant in a scenario with a smaller number of devices. However, as the number of devices increases, the impact of the data processing time increases. The results show that it is very important to determine which time component is more significant to identify system components where performance improvements are possible.

Fig. 11. shows a comparison of the results obtained experimentally and using the proposed model. Differences in results are 3–5 (%) for fog 1, 4–7 (%) for fog 2, and 0.4–2 (%) for cloud. When using the local server (fog 2) this difference is somewhat more pronounced because this server was overloaded with other services during the measurement to determine the dependence of variable X on the system load on other applications and services. The differences are significantly smaller in systems with better computing performance (cloud).

6.3. Energy consumption analysis

IoT devices consume the available energy for two key processes: communication (data transfer) and computation (data processing). The devices are configured to send and process data and not to receive feedback or instructions. Therefore, the model for estimating power consumption takes the following form:

$$E_{dev} = P_L \cdot t_L + \left(p_u + \beta \tau_u^{-1}\right) \cdot D_{tx} + P_{id} \cdot t_{off} \tag{76}$$

In our case, an application requires $P_L = 1\ W$ W for processing data, and while it is in idle state it uses $P_{idle} = 0.02\ W$ [75,92,111]. Other parameters were set to $p_u = 0.28317$ W/Mbps, and $\beta = 0.13286$ W [95]. The assumption is that the system has sufficient capacity to support the given load. Accordingly, the losses are minimal and can be neglected. The data transmission bandwidth is set to $\tau_u = 1\ Mbps$. All IoT devices have the same features and settings. Devices can be in one of three possible states including data processing, data sending, and idle. IoT devices consume different amounts of energy depending on their activity. An energy-per-bit model can be used to estimate the energy consumption for data transmission:

$$E(D)_{tx} = p_u + \beta \tau_u^{-1} = 0.41603 \text{ J/Mb} \tag{77}$$

All devices use the same amount of energy to process data locally and send them to the offloading system. It is because devices are configured to collect and send the same amount of data at the same speed and via

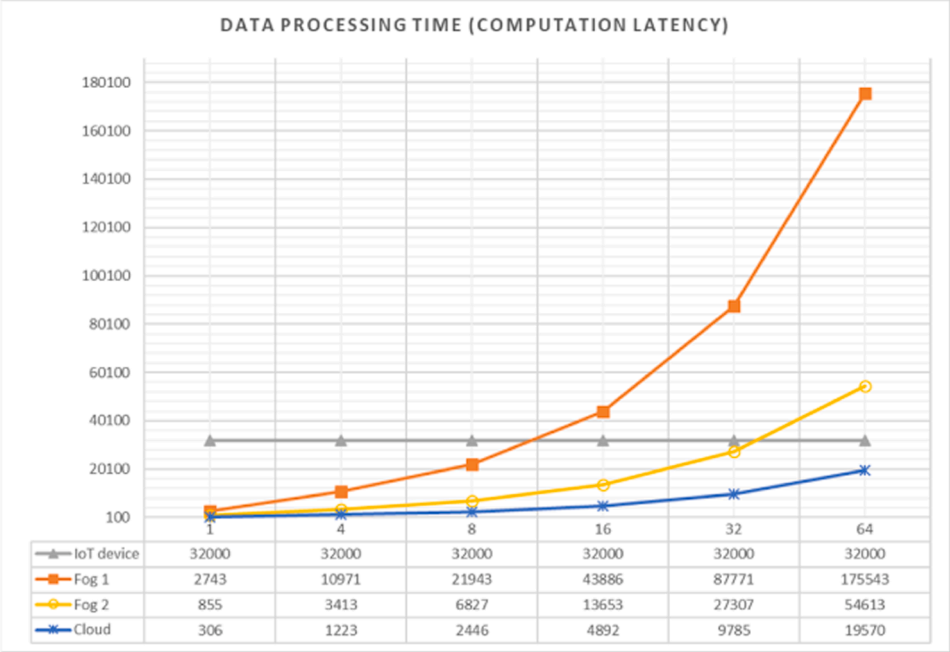

Fig. 8. Computation latency (data processing time) for different computing systems. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

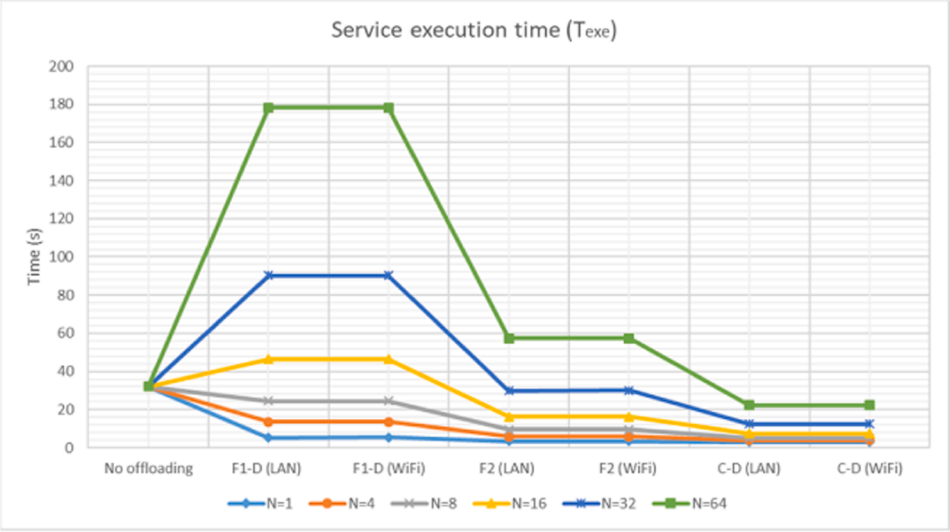

Fig. 9. Service latency (total service execution time). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the same communication technology. For these settings, the power consumption of each device is the same. Smaller differences in energy consumption are when deploying some offloading system.

IoT devices consume idle power when during offloading but with lower performance capabilities because the devices stay longer in idle state. However, since P_{idle} is a small value, then significant differences can only be expected for systems with higher loads. The results (Fig. 12) showed that $P_{tx} = 0.41$ W is used to send data over Wi-Fi. It is useful to compare with results obtained by using the method verified in another research [95]. The comparison results are with slight differences (about 1 %). The results show that offloading IoT devices through the allocation of data processing services to other computer systems is of great benefit when it comes to power consumption. Using any offloading system (fog1, fog2, cloud) reduces power consumption. However, the question is how to optimize the system according to multiple KPI such as energy consumption and service execution time.

6.4. Selecting the best model for IoT system

To select the optimal IoT system model, the following matrix can be used:

$$IoT_{sys}(C) = \left\{ \overbrace{\begin{matrix} S_{w_1} = \langle 1, 320 \rangle \\ \vdots \\ S_{w_k} = \langle 64, 320 \rangle \end{matrix}}^{System\ loads} \overbrace{\begin{pmatrix} IoT_{sys}(p_{11}) & \cdots & IoT_{sys}(p_{1n}) \\ \vdots & \ddots & \vdots \\ IoT_{sys}(p_{1k}) & \cdots & IoT_{sys}(p_{kn}) \end{pmatrix}}^{IM_1 \ \cdots \ IM_n} \right\}_{IoT_{sys}(p) = \langle p_{i_{req}}, p_i \rangle, \quad P_i = \langle T_{exe}, E_{dev} \rangle} \tag{78}$$

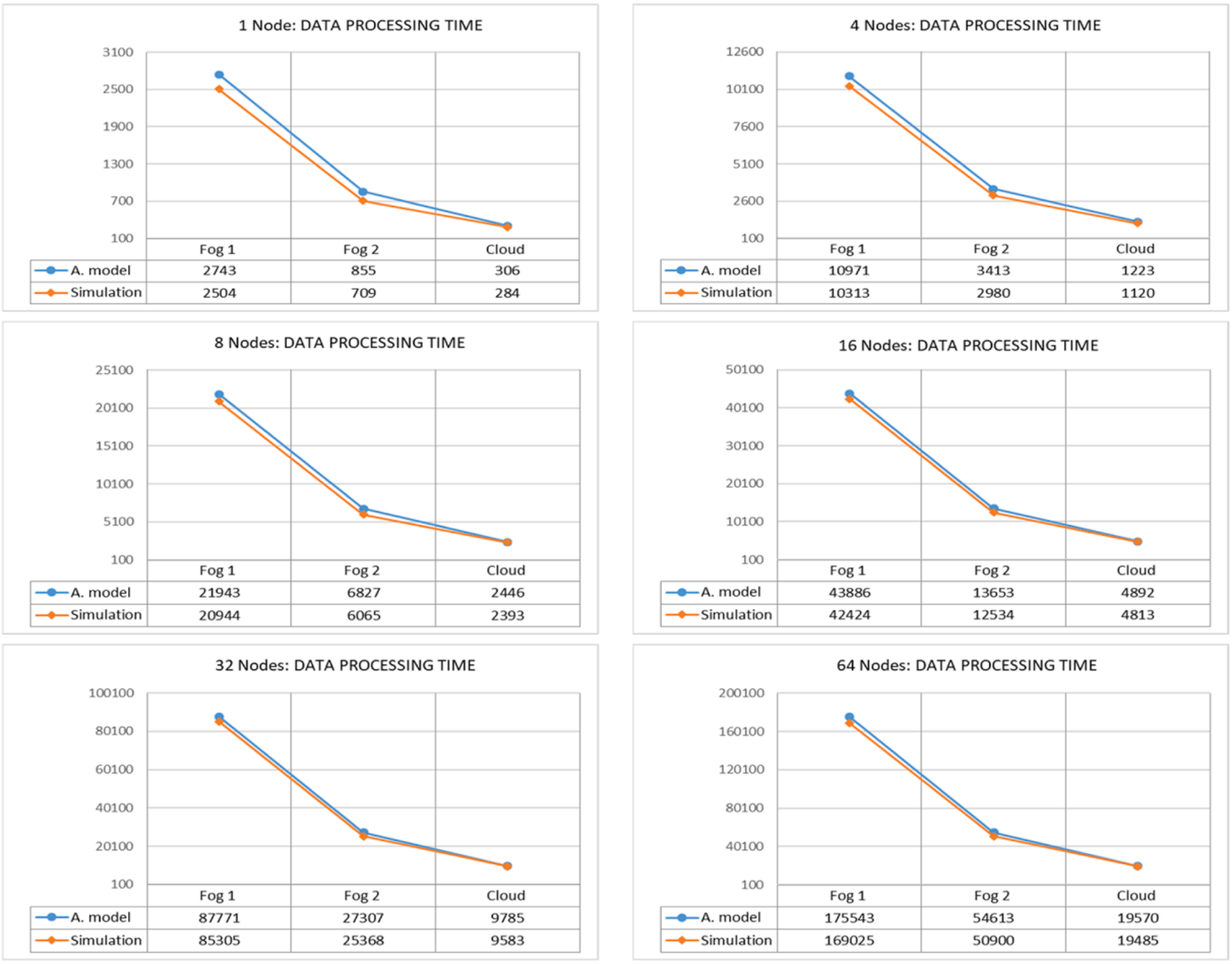

Fig. 10. Computation latency: Analytical (proposed) model vs. Simulation model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 5
Contribution of network delay to the total service execution time.

IoT devices	No offloading	F1-D (LAN)	F1-D (Wi-Fi)	F2 (LAN)	F2 (Wi-Fi)	C-D (LAN)	C-D (Wi-Fi)
1	0.03%	48.95%	49.03%	75.48%	75.49%	89.79%	89.82%
4	0.03%	19.02%	19.07%	43.13%	43.19%	68.33%	68.39%
8	0.03%	10.48%	10.51%	27.51%	27.61%	51.82%	51.89%
16	0.03%	5.52%	5.54%	16.05%	16.17%	34.95%	35.02%
32	0.03%	2.84%	2.84%	8.85%	9.00%	21.19%	21.24%
64	0.03%	1.44%	1.44%	4.78%	4.94%	11.88%	11.91%

System load $S_{w_k} = \langle k, D \rangle$depends on the number of IoT devices in the system $k = \langle 1; 4; 8; 16; 32; 64 \rangle$. Each device generates the same amount of data ($D = 320\ kB$). Data processing is performed by different systems including IoT devices without offloading (Dev), Raspberry Pi (F1), local server (F2), or Cloud system (C). Wi-Fi or Ethernet has been used to exchange data between these systems in the local network. An internet connection is required for sending data to the Cloud system. According to these considerations, possible alternatives include seven integration models: $IM_n = \langle \text{Dev},\ \text{F1} - \text{D (LAN)},\ \text{F1} - \text{D (Wi} - \text{Fi)},\ \text{F2 (LAN)},\ \text{F2 (Wi} - \text{Fi)},\ \text{C} - \text{D (LAN)},\ \text{C} - \text{D (Wi} - \text{Fi)} \rangle$.

The KPIs for performance analysis are service execution time and power consumption $p_i = \langle T_{exe}, E_{dev} \rangle$. These KPIs were estimated using the proposed analytical models. The proposed method of the utility function was used to calculate the value $IoT_{sys}(p_{kn})$ for each scenario. These scenarios include a combination of different system loads (k) and integration models (IM_n). The $IoT_{sys}(p_{kn})$ values are components of the specified matrix. The following parameters were used to apply custom sigmoid functions: $\frac{\alpha_k}{Z_i} = 0.1$ (for both KPIs), $R_{T_{exe}} = 30\ s$ (average service execution time), $R_{E_{dev}} = 5.5\ J$ (average energy consumption). The application of the proposed utility function (UF) gives the same results as the analysis of individual KPIs (Fig. 13). Therefore, this function is acceptable for the analysis of individual KPIs.

The application execution time indicates that it is not always useful to offload IoT devices. For a smaller number of devices ($\sim$ 10), it is

Fig. 11. Service latency: Analytical (proposed) model vs. Simulation model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

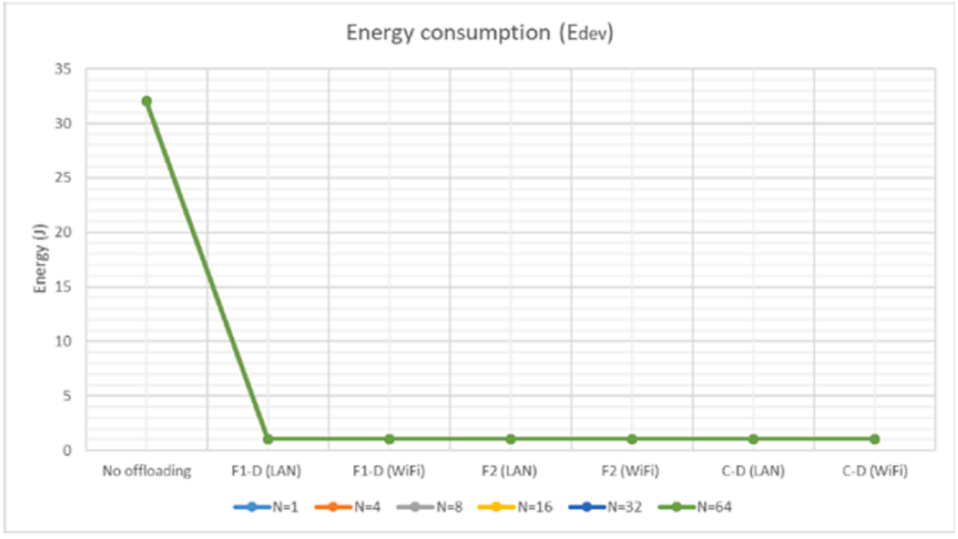

Fig. 12. Energy consumption by IoT devices. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

useful to distribute the data processing services to other systems. For 16–32 devices, it is better to offload the devices using a local server or cloud system. However, in this scenario, it is not useful to use the Raspberry Pi (fog 1) to offload the IoT device. As the system load increase, the Raspberry Pi shows lower performances due to limited processing capabilities compared to a local or cloud server.

For 64 devices, it is useful to distribute data processing service to a system with significantly better performances (e.g. Cloud). However, the trend of the results shows that for a large increase in system load, offloading the IoT devices would not be useful. Energy consumption shows that it is useful to offload IoT devices through the allocation of data processing services to any other computing system. This result shows that IoT devices use more energy to process data and send results than just sending data to other computing systems.

The challenge is to choose the best system model while considering both KPIs. Also, one of the challenges is the selection of a weighted value for each of these KPIs. Assuming that both of these indicators have the same importance, i.e. $w_{T_{exe}} = w_{E_{dev}} = 0.5$, applying the utility function shows the results shown in Fig. 14(a). In each scenario, it is useful to distribute the data processing service to one of the offloading systems (fog 1, fog 2, cloud). For a smaller number of IoT devices (less system load), the results are similar for each scenario. As the system load grows, it is more useful to allocate data processing services to better-performing systems (local server or cloud). For the largest system load in the observed scenarios, the best solution is to integrate IoT and cloud systems. However, the analysis of the result trends suggests that there may be problems if sufficient resources are not available on cloud servers. Such problems are solved by scaling mechanisms. Also, it is possible to use the processing capabilities of the IoT devices themselves or to perform partial offloading. This issue remains open for future research.

To verify the proposed model, it is useful to compare the results with the CF (*Cost Function*) method which is a previously validated approach in the scientific literature [83]. CF method shows the results shown in Fig. 14(b). The results are almost identical for both methods, which confirms the validity of the application of the proposed method based on custom sigmoid functions. However, the application of the CF method implies certain assumptions. For example, service execution time and energy consumption benchmarks can be assumed to be consistent with the required values for performing an application on an IoT device. Though, this approach cannot be applied in our scenarios since these values could be less than the obtained results. In such a case, the results are not normalized to the range [0, 1], which makes it impossible to summarize the results for different KPIs. Accordingly, the maximum obtained values for these KPIs (the worst cases for all scenarios) were used. Therefore, the application of the proposed method allows significantly better flexibility in the inclusion of additional KPIs, independence in the choice of benchmarks, and adjustment of the sensitivity of the system to individual KPIs. This can provide better representations of results and applications in different scenarios.

KPIs can have different weighted values, which significantly affects the results. Some results for different weight factors for time and energy are shown in Fig. 15. Therefore, the proposed approach is practical for achieving the desired level of system performances and optimizing the use of available resources. The open issue is to analyze the behavior of IoT systems with higher workloads, application of different technologies, partial offloading processes, etc. The proposed approach can be used for this research and applying the proposed methods enable the corresponding modeling of IoT systems which improve overall system performance.

7. Conclusion

IoT applications bring new challenges owing to the increased number of connected devices, increasing traffic demands, integration of more advanced and sophisticated technologies, big data demands, security and privacy risks, etc. In this paper, various approaches for distributing resources, processes, and services among IoT system architecture layers are presented to overcome some of these issues. Most IoT solutions rely on cross-domain integration with offloading systems such as Cloud computing, MCC, MEC, fog computing, cloudlet, hybrid cloud platforms, etc. These solutions break through future developments with the challenge of finding eligible deployments of different architectures, integration models, technologies, protocols, etc.

This paper provides insights into IoT-enabling technologies by presenting functional domains and a model for the description of IoT systems. Key performance indicators (KPIs) were identified and discussed for performance evaluation. From the viewpoint of IoT applications, the most important QoS and resource utilization parameters include service latency and energy consumption. An analytical model was used to evaluate the KPIs. Finally, this paper presents a mathematical method based on multiple metrics for overall IoT system evaluation. This method can be used as an effective tool for selecting the corresponding architecture, technologies, and protocols according to the IoT application requirements. It can easily include any metric according to specific needs. This approach enables the deployment of adequate components for each layer of IoT system architecture. The simulation results proved the importance of choosing an appropriate model for integrating the IoT and some offloading systems.

The proposed method of IoT system performance evaluation requires fewer system resources than the methods based on complex algorithms presented in other studies. This makes the described approach practical for applications in real IoT scenarios. Furthermore, the proposed method is independent of the type of IoT application. Therefore, the proposed evaluation method can be used in various scenarios, for example, when

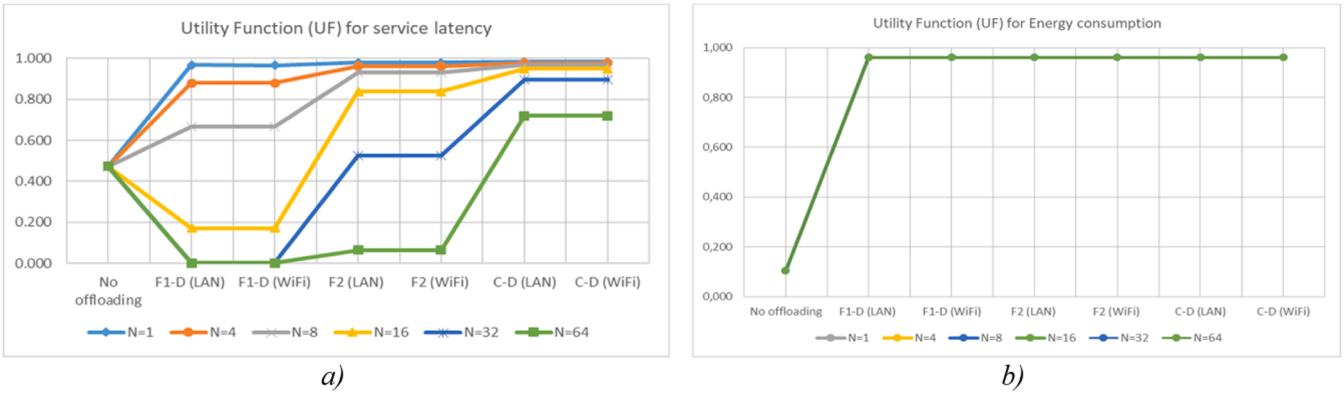

Fig. 13. Utility function for: (a) service latency; (b) energy consumption. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

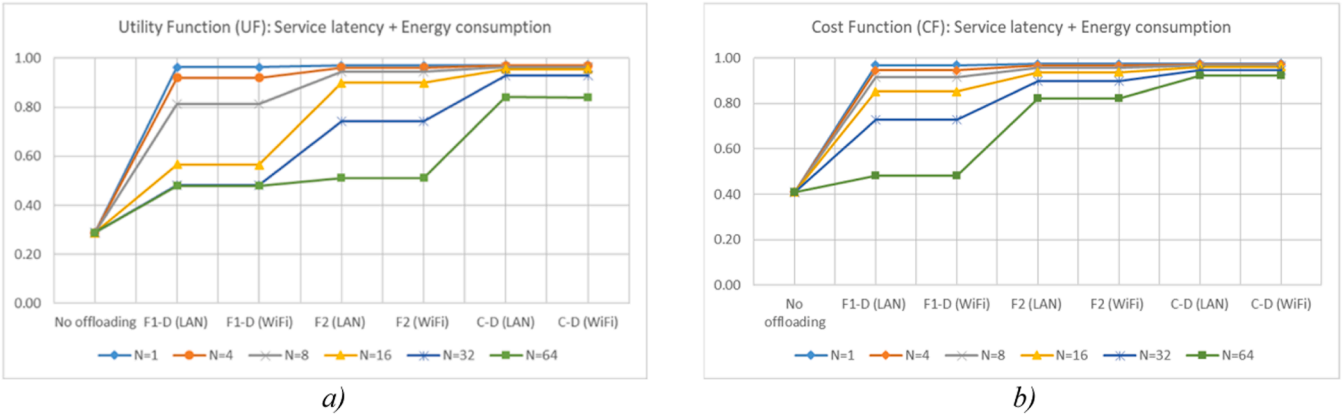

Fig. 14. Comparison of results: (a) UF; (b) CF method ($\mathbf{w}_{\mathbf{T}_{exe}} = \mathbf{w}_{\mathbf{E}_{dev}} = 0.5$). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

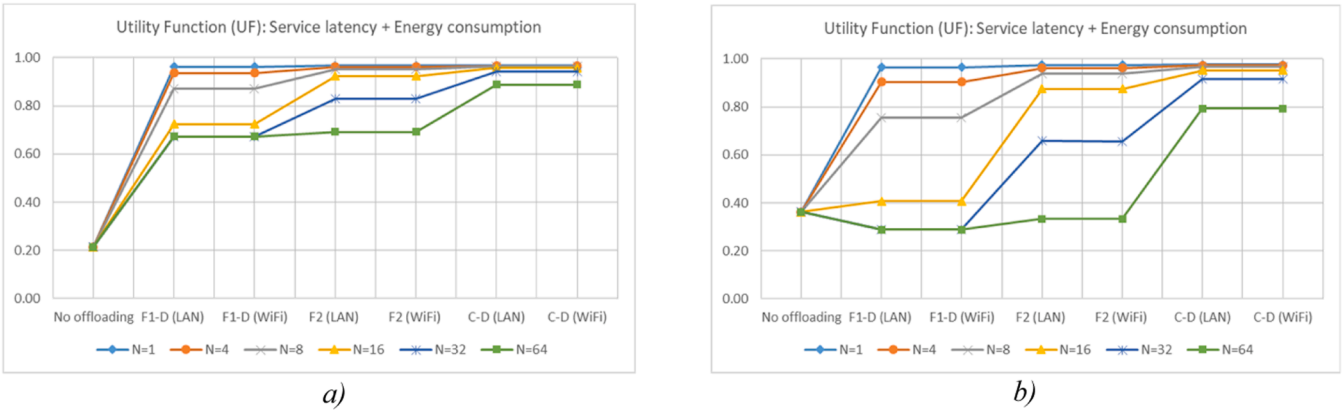

Fig. 15. Results of the UF method for different weight factors (**a**) $\mathbf{w}_{\mathbf{T}_{exe}} = 0.3,\ \mathbf{w}_{\mathbf{E}_{dev}} = 0.7$; (**b**) $\mathbf{w}_{\mathbf{T}_{exe}} = 0.7,\ \mathbf{w}_{\mathbf{E}_{dev}} = 0.3$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

IoT devices are located in a fixed place, but also in cases of mobility. However, the proposed approach has some limitations related to the limited number of observed KPIs. It is necessary to consider additional KPIs for different IoT applications to improve the obtained results. The question of accuracy and precision of the estimation arises, which requires additional research and improvement of the KPI estimation methods. Additionally, it is necessary to examine the effectiveness of the proposed approach in realistic scenarios. Furthermore, future research should test how the proposed approach deals with group mobility and handovers. Based on the above statements, there are many open issues for future research.

Future research may use the proposed method in different scenarios with various system loads, the employment of heterogeneous IoT components and technologies, and the impact of mobility on offloading policies and system performance. It is necessary to make additional efforts to improve the analytical model because the accuracy of the results depends on the estimated values of KPIs. Thus, the evaluation of service latency and energy consumption (KPIs) requires more specific measurements. Future work should also include efforts to design an autonomous service placement framework and self-adaptation according to the proposed three-tier architecture of the fog ecosystem. Enabling self-adaptation involves designing a system that can autonomously monitor, analyze, and respond to changes in its environment or operational conditions. The prerequisite for this research was to investigate the importance of each QoS metric for different IoT applications. It is necessary to derive a taxonomy for classifying QoS metrics for IoT system evaluation. This issue implies the need to investigate QoS metrics for various IoT applications and how they affect the performance of the entire IoT system. Therefore, this paper is valuable to the research community as it can be used as a starting point for analyzing various solutions for IoT systems and for performance optimizations. In addition, the proposed approach and evaluation method are effective tools for creating a comprehensive QoS classification scheme for IoT applications according to their performance profiles.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Supplementary materials and data used in this research are available upon request.

References

[1] D.H. Abdulazeez, S.K. Askar, Offloading mechanisms based on reinforcement learning and deep learning algorithms in the fog computing environment, IEEE Access 11 (2023) 12554–12585, https://doi.org/10.1109/ACCESS.2023.3241881.

[2] A. Yousefpour, et al., All one needs to know about fog computing and related edge computing paradigms: a complete survey, J. Syst. Archit. 98 (Sep. 2019) 289–330, https://doi.org/10.1016/j.sysarc.2019.02.009.

[3] H. Elazhary, Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: disambiguation and research directions, J. Netw. Comput. Appl. 128 (2019) 105–140, https://doi.org/10.1016/j.jnca.2018.10.021.
[4] Y. Ai, M. Peng, K. Zhang, Edge computing technologies for Internet of Things: a primer, Digit. Commun. Netw. 4 (2) (2018) 77–86, https://doi.org/10.1016/j.dcan.2017.07.001.
[5] R.S. Alonso, I. Sittón-Candanedo, Ó. García, J. Prieto, S. Rodríguez-González, An intelligent Edge-IoT platform for monitoring livestock and crops in a dairy farming scenario, Ad Hoc Netw. 98 (Mar. 2020), 102047, https://doi.org/10.1016/j.adhoc.2019.102047, 1–23.
[6] Q.D. La, M.V. Ngo, T.Q. Dinh, T.Q.S. Quek, H. Shin, Enabling intelligence in fog computing to achieve energy and latency reduction, Digit. Commun. Netw. 5 (1) (2019) 3–9, https://doi.org/10.1016/j.dcan.2018.10.008.
[7] J. Schmitt, J. Bönig, T. Borggräfe, G. Beitinger, J. Deuse, Predictive model-based quality inspection using machine learning and edge cloud computing, Adv. Eng. Inform. 45 (2020), 101101, https://doi.org/10.1016/j.aei.2020.101101, 1–10.
[8] Z. Zhao, P. Lin, L. Shen, M. Zhang, G.Q. Huang, IoT edge computing-enabled collaborative tracking system for manufacturing resources in industrial park, Adv. Eng. Inform. 43 (C) (2020), 101044, https://doi.org/10.1016/j.aei.2020.101044, 1–12.
[9] S. Zahoor, R.N. Mir, Resource management in pervasive Internet of Things: a survey, J. King Saud Univ. - Comput. Inf. Sci. 33 (8) (Oct. 2021) 921–935, https://doi.org/10.1016/j.jksuci.2018.08.014.
[10] E. Badidi, A. Ragmani, An architecture for QoS-aware fog service provisioning, Proc. Comput. Sci. 170 (2020) 411–418, https://doi.org/10.1016/j.procs.2020.03.083.
[11] S. Ebneyousef, A. Shirmarz, A taxonomy of load balancing algorithms and approaches in fog computing: a survey, Cluster Comput. (Feb. 2023), https://doi.org/10.1007/s10586-023-03982-3.
[12] L. Bulej, et al., Managing latency in edge–cloud environment, J. Syst. Softw. 172 (Feb. 2021) 11087, https://doi.org/10.1016/j.jss.2020.110872, 1–15.
[13] P. Hu, S. Dhelim, H. Ning, T. Qiu, Survey on fog computing: architecture, key technologies, applications and open issues, J. Netw. Comput. Appl. 98 (2017) 27–42, https://doi.org/10.1016/j.jnca.2017.09.002.
[14] W. Shi, J. Cao, Q. Zhang, Edge computing: vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646.
[15] J. Mass, S.N. Srirama, C. Chang, STEP-ONE: simulated testbed for Edge-Fog processes based on the opportunistic network environment simulator, J. Syst. Softw. 166 (2020), https://doi.org/10.1016/j.jss.2020.110587.
[16] I. Farris, A. Orsino, L. Militano, A. Iera, G. Araniti, Federated IoT services leveraging 5G technologies at the edge, Ad Hoc Netw. 68 (2018) 58–69, https://doi.org/10.1016/j.adhoc.2017.09.002.
[17] G. Kecskemeti, G. Casale, D.N. Jha, J. Lyon, R. Ranjan, Modelling and simulation challenges in internet of things, IEEE Cloud Comput. 4 (1) (Jan. 2017) 62–69, https://doi.org/10.1109/MCC.2017.18.
[18] K. Bierzynski, A. Escobar, Cloud, Fog and Edge: cooperation for the future?, in: 2017 Second International Conference on og and Mobile Edge Computing (FMEC), 2017, pp. 62–67, https://doi.org/10.1109/FMEC.2017.7946409.
[19] P.P. Ray, A survey on Internet of Things architectures, J. King Saud Univ. - Comput. Inf. Sci. 30 (3) (Jul. 2018) 291–319, https://doi.org/10.1016/j.jksuci.2016.10.003.
[20] B. Memić, A.H. Džubur, E. Avdagić-Golub, Green IoT: sustainability environment and technologies, Sci. Eng. Technol. 2 (1) (2022) 24–29, https://doi.org/10.54327/set2022/v2.i1.25.
[21] S. Sarkar, S. Misra, Theoretical modelling of fog computing: a green computing paradigm to support IoT applications, IET Netw. 5 (2) (Mar. 2016) 23–29, https://doi.org/10.1049/iet-net.2015.0034.
[22] N. Kumari, A. Yadav, P.K. Jana, Task offloading in fog computing: a survey of algorithms and optimization techniques, Comput. Netw. 214 (2022) 1–24, https://doi.org/10.1016/j.comnet.2022.109137.
[23] W. Li, et al., System modelling and performance evaluation of a three-tier Cloud of Things, Future Gener. Comput. Syst. 70 (May 2017) 104–125, https://doi.org/10.1016/j.future.2016.06.019.
[24] G. Fortino, R. Gravina, W. Russo, C. Savaglio, Modeling and simulating Internet-of-Things systems: a hybrid agent-oriented approach, Comput. Sci. Eng. 19 (5) (2017) 68–76, https://doi.org/10.1109/MCSE.2017.3421541.
[25] A. Ikram, A. Anjum, R. Hill, N. Antonopoulos, L. Liu, S. Sotiriadis, Approaching the Internet of Things (IoT): a modelling, analysis and abstraction framework, Concurr. Comput. Pract. Exp. 27 (8) (2015) 1966–1984, https://doi.org/10.1002/cpe.3131.
[26] A. Čolaković, M. Hadžialić, Internet of Things (IoT): a review of enabling technologies, challenges, and open research issues, Comput. Netw. 144 (Oct. 2018) 17–39, https://doi.org/10.1016/j.comnet.2018.07.017.
[27] A. Munir, P. Kansakar, S.U. Khan, IFCIoT: integrated Fog Cloud IoT: a novel architectural paradigm for the future Internet of Things, IEEE Consum. Electron. Mag. 6 (3) (Jul. 2017) 74–82, https://doi.org/10.1109/MCE.2017.2684981.
[28] K.-H.N. Bui, J.J. Jung, Computational negotiation-based edge analytics for smart objects, Inf. Sci. (Ny). 480 (2019) 222–236, https://doi.org/10.1016/j.ins.2018.12.046.
[29] A. Brogi, S. Forti, QoS-aware deployment of IoT applications through the Fog, IEEE Internet Things J. 4 (5) (Oct. 2017) 1185–1192, https://doi.org/10.1109/JIOT.2017.2701408.
[30] M. Chiang, T. Zhang, Fog and IoT: an overview of research opportunities, IEEE Internet Things J. 3 (6) (Dec. 2016) 854–864, https://doi.org/10.1109/JIOT.2016.2584538.
[31] X. Deng, J. Li, E. Liu, H. Zhang, Task allocation algorithm and optimization model on edge collaboration, J. Syst. Archit. 110 (2020) 10177, https://doi.org/10.1016/j.sysarc.2020.101778.
[32] K. Khebbeb, N. Hameurlain, F. Belala, A Maude-based rewriting approach to model and verify Cloud/Fog self-adaptation and orchestration, J. Syst. Archit. 110 (2020) 10182, https://doi.org/10.1016/j.sysarc.2020.101821.
[33] C.T. Joseph, K. Chandrasekaran, IntMA: dynamic Interaction-aware resource allocation for containerized microservices in cloud environments, J. Syst. Archit. 111 (2020) 10178, https://doi.org/10.1016/j.sysarc.2020.101785.
[34] S.S. Gill, P. Garraghan, R. Buyya, ROUTER: fog enabled cloud based intelligent resource management approach for smart home IoT devices, J. Syst. Softw. 154 (2019) 125–138, https://doi.org/10.1016/j.jss.2019.04.058.
[35] R.A. Haidri, C.P. Katti, P.C. Saxena, Cost effective deadline aware scheduling strategy for workflow applications on virtual machines in cloud computing, J. King Saud Univ. - Comput. Inf. Sci. 32 (6) (2020) 666–683, https://doi.org/10.1016/j.jksuci.2017.10.009.
[36] B. Shrimali, H. Patel, Multi-objective optimization oriented policy for performance and energy efficient resource allocation in Cloud environment, J. King Saud Univ. - Comput. Inf. Sci. 32 (7) (2020) 860–869, https://doi.org/10.1016/j.jksuci.2017.12.001.
[37] F. Murtaza, A. Akhunzada, S. ul Islam, J. Boudjadar, R. Buyya, QoS-aware service provisioning in fog computing, J. Netw. Comput. Appl. 165 (2020) 10267, https://doi.org/10.1016/j.jnca.2020.102674, 1–14.
[38] T.C.S. Xavier, et al., Collaborative resource allocation for Cloud of Things systems, J. Netw. Comput. Appl. 159 (2020) 10259, https://doi.org/10.1016/j.jnca.2020.102592, 1–18.
[39] C. Li, J. Tang, Y. Luo, Dynamic multi-user computation offloading for wireless powered mobile edge computing, J. Netw. Comput. Appl. 131 (2019) 1–15, https://doi.org/10.1016/j.jnca.2019.01.020.
[40] T. Wang, J. Zhou, A. Liu, M.Z.A. Bhuiyan, G. Wang, W. Jia, Fog-based computing and storage offloading for data synchronization in IoT, IEEE Internet Things J. 6 (3) (2019) 4272–4282, https://doi.org/10.1109/JIOT.2018.2875915.
[41] S. Tuli, R. Mahmud, S. Tuli, R. Buyya, FogBus: a blockchain-based lightweight framework for Edge and Fog computing, J. Syst. Softw. 154 (2019) 22–36, https://doi.org/10.1016/j.jss.2019.04.050.
[42] J. Zhang, M. Ma, W. He, P. Wang, On-demand deployment for IoT applications, J. Syst. Archit. 111 (2020) 10179, https://doi.org/10.1016/j.sysarc.2020.101794.
[43] A. Farahzadi, P. Shams, J. Rezazadeh, R. Farahbakhsh, Middleware technologies for cloud of things: a survey, Digit. Commun. Networks 4 (3) (2018) 176–188, https://doi.org/10.1016/j.dcan.2017.04.005.
[44] G. Merlino, R. Dautov, S. Distefano, D. Bruneo, Enabling workload engineering in edge, fog, and cloud computing through openstack-based middleware, ACM Trans. Internet Technol. 19 (2) (Apr. 2019), https://doi.org/10.1145/3309705.
[45] S. Safavat, N.N. Sapavath, D.B. Rawat, Recent advances in mobile edge computing and content caching, Digit. Commun. Netw. 6 (2) (2020) 189–194, https://doi.org/10.1016/j.dcan.2019.08.004.
[46] H. Wei, H. Luo, Y. Sun, M.S. Obaidat, Cache-aware computation offloading in IoT systems, IEEE Syst. J. 14 (1) (2020) 61–72, https://doi.org/10.1109/JSYST.2019.2903293.
[47] Y. Mansouri, V. Prokhorenko, M.A. Babar, An automated implementation of hybrid cloud for performance evaluation of distributed databases, J. Netw. Comput. Appl. 167 (2020) 10274, https://doi.org/10.1016/j.jnca.2020.102740, 1–14.
[48] L.F. Bittencourt, A. Goldman, E.R.M. Madeira, N.L.S. Da Fonseca, R. Sakellariou, Scheduling in distributed systems: a cloud computing perspective, Comput. Sci. Rev. 30 (2018) 31–54, https://doi.org/10.1016/j.cosrev.2018.08.002.
[49] I. Ahmad, M.G. AlFailakawi, A. AlMutawa, L. Alsalman, Container scheduling techniques: a Survey and assessment, J. King Saud Univ. - Comput. Inf. Sci. 34 (7) (2022) 3934–3947, https://doi.org/10.1016/j.jksuci.2021.03.002.
[50] S.K. Panda, S.S. Nanda, S.K. Bhoi, A pair-based task scheduling algorithm for cloud computing environment, J. King Saud Univ. - Comput. Inf. Sci. 34 (1) (2022) 1434–1445, https://doi.org/10.1016/j.jksuci.2018.10.001.
[51] B. Keshanchi, A. Souri, N.J. Navimipour, An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: formal verification, simulation, and statistical testing, J. Syst. Softw. 124 (2017) 1–21, https://doi.org/10.1016/j.jss.2016.07.006.
[52] Y. Hao, Y. Miao, L. Hu, M.S. Hossain, G. Muhammad, S.U. Amin, Smart-Edge-CoCaCo: AI-enabled smart edge with joint computation, caching, and communication in heterogeneous IoT, IEEE Netw. 33 (2) (2019) 58–64, https://doi.org/10.1109/MNET.2019.1800235.
[53] M. Aazam, S. Zeadally, E.F. Flushing, Task offloading in edge computing for machine learning-based smart healthcare, Comput. Netw. 191 (2021) 10801, https://doi.org/10.1016/j.comnet.2021.108019, 1–11.
[54] Q. Kang, H. He, H. Song, Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm, J. Syst. Softw. 84 (6) (2011) 985–992, https://doi.org/10.1016/j.jss.2011.01.051.
[55] N. Moganarangan, R.G. Babukarthik, S. Bhuvaneswari, M.S. Saleem Basha, P. Dhavachelvan, A novel algorithm for reducing energy-consumption in cloud computing environment: web service computing approach, J. King Saud Univ. - Comput. Inf. Sci. 28 (1) (2016) 55–67, https://doi.org/10.1016/j.jksuci.2014.04.007.
[56] K.N. Vhatkar, G.P. Bhole, Optimal container resource allocation in cloud architecture: a new hybrid model, J. King Saud Univ. - Comput. Inf. Sci. 34 (5) (2022) 1906–1918, https://doi.org/10.1016/j.jksuci.2019.10.009.

[57] A.S. Abohamama, A. El-Ghamry, E. Hamouda, *Real-Time Task Scheduling Algorithm for IoT-Based Applications in the Cloud–Fog Environment*, vol. 30, no. 4, Springer, US, 2022, https://doi.org/10.1007/s10922-022-09664-6.

[58] N. Bacanin, M. Zivkovic, T. Bezdan, K. Venkatachalam, M. Abouhawwash, Modified firefly algorithm for workflow scheduling in cloud-edge environment, Neural Comput. Appl. 34 (11) (2022) 9043–9068, https://doi.org/10.1007/s00521-022-06925-y.

[59] S.M.A. Huda, S. Moh, Survey on computation offloading in UAV-Enabled mobile edge computing, J. Netw. Comput. Appl. 201 (2022) 10334, https://doi.org/10.1016/j.jnca.2022.103341, 1–26.

[60] S. Wang, X. Song, H. Xu, T. Song, G. Zhang, Y. Yang, Joint offloading decision and resource allocation in vehicular edge computing networks, Digit. Commun. Netw. (2023), https://doi.org/10.1016/j.dcan.2023.03.006.

[61] A. Shakarami, M. Ghobaei-Arani, A. Shahidinejad, A survey on the computation offloading approaches in mobile edge computing: a machine learning-based perspective, Comput. Netw. 182 (2020) 10749, https://doi.org/10.1016/j.comnet.2020.107496, 1–24.

[62] T.K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, N. Kato, Machine learning meets computation and communication control in evolving edge and cloud: challenges and future perspective, IEEE Commun. Surv. Tutor. 22 (1) (2020) 38–67, https://doi.org/10.1109/COMST.2019.2943405.

[63] T. Bu, et al., Task scheduling in the internet of things: challenges, solutions, and future trends, Cluster Comput. (2023), https://doi.org/10.1007/s10586-023-03991-2.

[64] S. Zhou, W. Jadoon, and I.A. Khan, "Computing offloading strategy in mobile edge computing environment: a comparison between adopted frameworks, challenges, and future directions," *Electronics (Basel)*, vol. 12, no. 11, pp. 1–30, May 2023, doi: 10.3390/electronics12112452.

[65] M. Maray, J. Shuja, Computation offloading in mobile cloud computing and mobile edge computing: survey, taxonomy, and open issues, Mob. Inf. Syst. 2022 (2022) 1–17, https://doi.org/10.1155/2022/1121822.

[66] K. Gasmi, S. Dilek, S. Tosun, S. Ozdemir, A survey on computation offloading and service placement in fog computing-based IoT, J. Supercomput. 78 (2) (2022) 1983–2014, https://doi.org/10.1007/s11227-021-03941-y.

[67] A. Čolaković, A.H. Džubur, B. Karahodža, Wireless communication technologies for the Internet of Things, Sci. Eng. Technol. 1 (1) (Apr. 2021) 1–14, https://doi.org/10.54327/set2021/v1.i1.3.

[68] H. Tschofenig, J. Arkko, D. Thaler, D. McPherson, Architectural considerations in smart object networking, IETF RFC 7452 (2015) 1–23 [Online]. Available: https://tools.ietf.org/html/rfc7452.

[69] P. Bellavista, J. Berrocal, A. Corradi, S.K. Das, L. Foschini, A. Zanni, A survey on fog computing for the Internet of Things, Pervasive Mob. Comput. 52 (2019) 71–99, https://doi.org/10.1016/j.pmcj.2018.12.007.

[70] J.C. Nobre, et al., Vehicular software-defined networking and fog computing: integration and design principles, Ad Hoc Netw. 82 (2019) 172–181, https://doi.org/10.1016/j.adhoc.2018.07.016.

[71] M. Gill, D. Singh, A comprehensive study of simulation frameworks and research directions in fog computing, Comput. Sci. Rev. 40 (2021), 100391, https://doi.org/10.1016/j.cosrev.2021.100391, 1–17.

[72] X. Chen, Decentralized computation offloading game for mobile cloud computing, IEEE Trans. Parallel Distrib. Syst. 26 (4) (Apr. 2015) 974–983.

[73] X. Fan, C.S. Ellis, A.R. Lebeck, The synergy between power-aware memory systems and processor voltage scaling, Int. Work. Power-Aware Comput. Syst. 3164 (2005) 164–179.

[74] M.S. Aslanpour, S.S. Gill, A.N. Toosi, Performance evaluation metrics for cloud, fog and edge computing: a review, taxonomy, benchmarks and standards for future research, Internet of Things 12 (2020), 100273, https://doi.org/10.1016/j.iot.2020.100273.

[75] D. Mazza, D. Tarchi, G.E. Corazza, A partial offloading technique for wireless mobile cloud computing in smart cities, in: 2014 European Conference on Networks and Communications (EuCNC), Jun. 2014, pp. 1–5, https://doi.org/10.1109/EuCNC.2014.6882623.

[76] L. Chen, X. Li, H. Ji, V.C.M. Leung, Computation offloading balance in small cell networks with mobile edge computing, Wirel. Netw. 25 (7) (Oct. 2019) 4133–4145, https://doi.org/10.1007/s11276-018-1735-y.

[77] J. Zhang, et al., Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks, IEEE Internet Things J. 5 (4) (Aug. 2018) 2633–2645, https://doi.org/10.1109/JIOT.2017.2786343.

[78] J. Wang, D. Li, Adaptive computing optimization in software-defined network-based industrial internet of things with fog computing, Sensors 18 (8) (Aug. 2018), 2509, https://doi.org/10.3390/s18082509.

[79] C. Ragona, F. Granelli, C. Fiandrino, D. Kliazovich, P. Bouvry, Energy-efficient computation offloading for wearable devices and smartphones in mobile cloud computing, in: 2015 IEEE Global Communications Conference (GLOBECOM), Dec. 2015, pp. 1–6, https://doi.org/10.1109/GLOCOM.2015.7417039.

[80] R. Rondón, M. Gidlund, K. Landernäs, Evaluating bluetooth low energy suitability for time-critical industrial iot applications, Int. J. Wirel. Inf. Netw. 24 (3) (Sep. 2017) 278–290, https://doi.org/10.1007/s10776-017-0357-0.

[81] R. Rondon, K. Landernas, M. Gidlund, An analytical model of the effective delay performance for Bluetooth low energy, in: 2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Sep. 2016, pp. 1–6, https://doi.org/10.1109/PIMRC.2016.7794553.

[82] E. Ghadimi, A. Khonsari, A. Diyanat, M. Farmani, N. Yazdani, An analytical model of delay in multi-hop wireless ad hoc networks, Wirel. Netw. 17 (7) (Oct. 2011) 1679–1697, https://doi.org/10.1007/s11276-011-0372-5.

[83] ITU-T, "Transmission systems and media, digital systems and networks." ITU-T, 2013. [Online]. Available: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-G.114-200305-I!!PDF-E&type=items.

[84] A.P. Miettinen, J.K. Nurminen, Energy efficiency of mobile clients in cloud computing, in: *2nd USENIX Workshop on Hot Topics in Cloud Computing*, 2010, pp. 1–7.

[85] S. Melendez, M.P. McGarry, Computation offloading decisions for reducing completion time, in: 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), Jan. 2017, pp. 160–164, https://doi.org/10.1109/CCNC.2017.7983099.

[86] W. Yuan, K. Nahrstedt, Energy-efficient CPU scheduling for multimedia applications, ACM Trans. Comput. Syst. 24 (3) (Aug. 2006) 292–331, https://doi.org/10.1145/1151690.1151693.

[87] J.R. Lorch, A.J. Smith, Improving dynamic voltage scaling algorithms with PACE, in: Proc. ACM SIGMETRICS international conference on Measurement and modeling of computer systems, 2001, pp. 50–61, https://doi.org/10.1145/378420.378429.

[88] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, D.O. Wu, Energy-optimal mobile cloud computing under stochastic wireless channel, IEEE Trans. Wirel. Commun. 12 (9) (2013) 4569–4581, https://doi.org/10.1109/TWC.2013.072513.121842.

[89] M. Yang, Y. Wen, J. Cai, C.H. Foh, Energy minimization via dynamic voltage scaling for real-time video encoding on mobile devices, in: 2012 IEEE International Conference on Communications (ICC), Jun. 2012, pp. 2026–2031, https://doi.org/10.1109/ICC.2012.6364132.

[90] G. Li, J. Wang, J. Wu, J. Song, Data processing delay optimization in mobile edge computing, Wirel. Commun. Mob. Comput. 2018 (2018) 1–9, https://doi.org/10.1155/2018/6897523.

[91] M. Dayarathna, Y. Wen, R. Fan, Data center energy consumption modeling: a survey, IEEE Commun. Surv. Tutor. 18 (1) (2016) 732–794, https://doi.org/10.1109/COMST.2015.2481183.

[92] R. LiKamWa, Z. Wang, A. Carroll, F.X. Lin, L. Zhong, Draining our glass, in: Proceedings of 5th Asia-Pacific Workshop on Systems, Jun. 2014, pp. 1–7, https://doi.org/10.1145/2637166.2637230.

[93] Y. Li, A.-C. Orgerie, I. Rodero, B.L. Amersho, M. Parashar, J.-M. Menaud, End-to-end energy models for Edge Cloud-based IoT platforms: application to data stream analysis in IoT, Future Gener. Comput. Syst. 87 (Oct. 2018) 667–678, https://doi.org/10.1016/j.future.2017.12.048.

[94] Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: the communication perspective, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2322–2358, https://doi.org/10.1109/COMST.2017.2745201.

[95] J. Huang, F. Qian, A. Gerber, Z.M. Mao, S. Sen, O. Spatscheck, A close examination of performance and power characteristics of 4 G LTE networks, in: Proceedings of the 10th international conference on Mobile systems, applications, and services, Jun. 2012, pp. 225–238, https://doi.org/10.1145/2307636.2307658.

[96] A. Carroll, G. Heiser, The systems hacker's guide to the galaxy energy usage in a modern smartphone, in: Proceedings of the 4th Asia-Pacific Workshop on Systems, Jul. 2013, pp. 1–7, https://doi.org/10.1145/2500727.2500734.

[97] G.P. Perrucci, F.H.P. Fitzek, J. Widmer, Survey on energy consumption entities on the smartphone platform, in: 2011 IEEE 73rd Vehicular Technology Conference (VTC Spring), May 2011, pp. 1–6, https://doi.org/10.1109/VETECS.2011.5956528.

[98] L. Sun, H. Deng, R.K. Sheshadri, W. Zheng, D. Koutsonikolas, Experimental evaluation of WiFi active power/energy consumption models for smartphones, IEEE Trans. Mob. Comput. 16 (1) (Jan. 2017) 115–129, https://doi.org/10.1109/TMC.2016.2538228.

[99] B. Dusza, C. Ide, L. Cheng, C. Wietfeld, An accurate measurement-based power consumption model for LTE uplink transmissions, in: 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Apr. 2013, pp. 49–50, https://doi.org/10.1109/INFCOMW.2013.6970731.

[100] N. Javaid, A. Ahmad, Y. Khan, Z.A. Khan, T.A. Alghamdi, A relay based routing protocol for wireless in-body sensor networks, Wirel. Pers. Commun. 80 (3) (Feb. 2015) 1063–1078, https://doi.org/10.1007/s11277-014-2071-x.

[101] Y. Liao, M. Leeson, M. Higgins, and C. Bai, "Analysis of In-to-Out wireless body area network systems: towards QoS-aware health internet of things applications," *Electronics (Basel)*, vol. 5, no. 4, pp. 1–26, Jul. 2016, doi: 10.3390/electronics5030038.

[102] Q. Zhang, D. Peng, Intelligent decision-making service framework based on QoS model in the internet of things, in: 2012 11th International Symposium on Distributed Computing and Applications to Business, Engineering & Science, Oct. 2012, pp. 103–107, https://doi.org/10.1109/DCABES.2012.26.

[103] A. Bozorgchenani, D. Tarchi, G.E. Corazza, Centralized and distributed architectures for energy and delay efficient fog network-based edge computing services, IEEE Trans. Green Commun. Netw. 3 (1) (Mar. 2019) 250–263, https://doi.org/10.1109/TGCN.2018.2885443.

[104] Y. Wang, A. Abdelhadi, T.C. Clancy, Optimal power allocation for LTE users with different modulations, in: 2016 Annual IEEE Systems Conference (SysCon), Apr. 2016, pp. 1–5, https://doi.org/10.1109/SYSCON.2016.7490537.

[105] A. Colakovic, H. Bajric, Assessing customer satisfaction based on QoS parameters, Int. J. Qual. Res. 11 (1) (2017) 221–240, https://doi.org/10.18421/IJQR11.01-14.

[106] R. Casadei, G. Fortino, D. Pianini, W. Russo, C. Savaglio, M. Viroli, A development approach for collective opportunistic Edge-of-Things services, Inf. Sci. (Ny). 498 (Sep. 2019) 154–169, https://doi.org/10.1016/j.ins.2019.05.058.

[107] G. D'Angelo, S. Ferretti, V. Ghini, Simulation of the Internet of Things, in: 2016 International Conference on High Performance Computing & Simulation (HPCS), Jul. 2016, pp. 1–8, https://doi.org/10.1109/HPCSim.2016.7568309.
[108] C. Sonmez, A. Ozgovde, C. Ersoy, EdgeCloudSim: an environment for performance evaluation of Edge Computing systems, in: 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), May 2017, pp. 39–44, https://doi.org/10.1109/FMEC.2017.7946405.
[109] X. Zeng, S.K. Garg, P. Strazdins, P.P. Jayaraman, D. Georgakopoulos, R. Ranjan, IOTSim: a simulator for analysing IoT applications, J. Syst. Archit. 72 (Jan. 2017) 93–107, https://doi.org/10.1016/j.sysarc.2016.06.008.
[110] A. Čolaković, S. Čaušević, A. Kosovac, E. Muharemović, A review of enabling technologies and solutions for IoT based smart warehouse monitoring system, New Technol., Dev.Appl. III 128 (2020) 630–637, https://doi.org/10.1007/978-3-030-46817-0_73.
[111] A. Bozorgchenani, D. Tarchi, G.E. Corazza, Centralized and distributed architectures for energy and delay efficient fog network based edge computing services, IEEE Trans. Green Commun. Netw. 3 (1) (2018) 1–14.