electronics

MDPI

Article

Approach for Designing Real-Time IoT Systems

Stanisław Deniziak *, Mirosław Płaza * and Łukasz Arcab

Faculty of Electrical Engineering, Automatic Control and Computer Science, Kielce University of Technology, Al. Tysiaclecia P.P. 7, 25-314 Kielce, Poland

* Correspondence: s.deniziak@tu.kielce.pl (S.D.); m.plaza@tu.kielce.pl (M.P.); Tel.: +48-41-342-4167 (M.P.)

Abstract: Along with the rapid development of Internet of Things (IoT) technology over the past few years, opportunities for its implementation in service areas that require real-time requirements have begun to be recognized. In this regard, one of the most important criteria is to maintain Quality of Service (QoS) parameters at an appropriate and sufficiently high level. The QoS level should ensure the delivery of data packets in the shortest time possible while preventing critical parameters relevant to real-time transmission from being exceeded. This article proposes a new methodology for designing real-time IoT systems. The premise of the proposed approach is to adapt selected solutions used in other types of systems working with real-time requirements. Some analogy to embedded systems with a distributed architecture has been noted and used in this regard. The main differences from the concept of built-in systems can primarily be seen in the communication layer. The methodology proposed in this article is based on the authors' proposed model of real-time system functional specification and its mapping to the IoT architecture. In addition, the developed methodology makes extensive use of selected IoT architecture elements described in this article, as well as selected task scheduling methods and communication protocols. The proposed methodology for designing RTIoT systems is based on dedicated transmission serialization methods and dedicated routing protocols. These methods ensure that the time constraints for the assumed bandwidth of IoT links are met by appropriately prioritizing transmissions and determining communication routes. The presented approach can be used to design a broad class of RTIoT systems.

Keywords: real time; IoT; tasks scheduling; communication protocols

check for updates

Citation: Deniziak, S.; Płaza, M.; Arcab, Ł. Approach for Designing Real-Time IoT Systems. *Electronics* **2022**, *11*, 4120. https://doi.org/10.3390/electronics11244120

Academic Editors: Juan M. Corchado, Byung-Gyu Kim, Carlos A. Iglesias, In Lee, Fuji Ren and Rashid Mehmood

Received: 8 November 2022
Accepted: 7 December 2022
Published: 10 December 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

For a number of years, there has been a continuous and rapid development of products and services in the area of Internet of Things (IoT) technology [1–4]. Research in this arena is focused on introducing new solutions in the form of dedicated, smart, specialized and autonomous systems [5–8]. The main reason for the widespread use of IoT solutions is usually the desire to improve people's lifestyle and comfort, the optimal use of natural resources, and the low energy consumption of these solutions [9–12]. IoT solutions in this area often provide previously unattainable levels of efficiency while minimizing the contribution of the human factor. This trend is being forced by the ever-increasing demands of profit optimization in all industries and economies, as well as the quality and safety of life demands of specific groups of consumers and even entire global communities [13].

The most up-to-date literature describes numerous examples of implementations of classical IoT solutions for various purposes. As is well known, the so-called "smart" systems, e.g., smart homes, smart factories, smart cities, and smart grids, have gained great popularity [14,15]. In addition, a number of systems have been developed for the medical [16], automotive [17], agricultural [18], property protection [19], logistics [20], and sports and recreation [21] industries, as well as many other areas related to the immediate needs of the public. Smart systems for managing amusement parks [7,8,22]; automated customer service solutions based on speech recognition techniques and voice assistants [23]; facial recognition and motion detection systems used for security and access control [24];

smart systems for managing [25,26], controlling [27], and optimizing production processes [28–30]; and real-time object location systems [12,31,32] are being implemented. The aforementioned solutions are just a few examples of the known applications of IoT technology. The possibilities of implementing traditional solutions are very wide and mainly depend on the needs of the recipients and the ingenuity of their designers, thus influencing the development of a certain area of the service market.

At the same time, it should be noted that services and products where real-time transmission requirements play a key role are still major constraints in the implementation of IoT solutions. The need to design real-time applications for IoT (RTIoT) solutions was recognized more than 10 years ago [33]. The first technologies and standards in this area also began to emerge at that time. Examples include: ITCC (Intel Time Coordinated Computing) [34,35], TSN (Time Sensitive Network) [36], M2M (Machine to Machine Communication) [37], SaaS (Sensing as a Service) [38], and a multi-layered reference model dedicated to IoE (Internet of Everything) technologies [3]. However, most work to date in the field of RTIoT systems has only been concerned with trying to develop selected elements of the IoT architecture [39] or applications and technologies [40,41] optimized for the highest possible QoS (Quality of Service). These mostly deal with issues related to real-time transmission and protocols, server applications, or real-time databases. As previously mentioned, there was a need for real-time transmission early in the development of the internet, mainly for the purpose of transmitting multimedia data. The RTP/RSTP [42] or XMPP [43] protocols in use at the time did not allow for the parameters associated with time constraints to be met. It was not until the development of the IEEE 802.1 standard in the form of the TSN solution that some conditions were created for the development of data transmission methods that considered time parameters. However, this standard only defines basic mechanisms for time synchronization, transmission scheduling or resource reservation. It does not specify specific communication protocols or algorithms for scheduling and routing data transmission [44]. In terms of server applications, there is ongoing work on cloud computing. This work focuses on developing real-time cloud services, scheduling tasks according to real-time requirements, and developing web servers that handle requests with time constraints [45–47]. Real-time databases, on the other hand, define real time as the fastest execution of queries to a database. Examples of such databases are VoltDB [48] and Firebase [49].

Although it is possible to find numerous publications presenting examples of IoT applications running in real time, these solutions are not fully compatible with the concept of a real-time system. Typically, these solutions are highly optimized in terms of data processing and data transmission performance so the time requirements under typical operating conditions are met. However, such solutions do not provide the required QoS level when operating conditions change (e.g., with higher network load). Moreover, providing high performance is often associated with significant costs of computing and communication resources. Additionally, the methodologies of designing RTIoT systems that take cost and QoS optimization into account are unknown. Constraints related to real-time requirements are currently one of the main challenges faced by IoT system research, and they were also the main motivation for the authors of this study to carry out the presented research.

The purpose of this work was to identify the problems and needs that exist in the design of RTIoT systems, to identify research directions to solve these problems, and to propose a design methodology that considers the requirements of RTIoT systems. The contribution to the body of knowledge of this article is:

- The proposal of a methodology for designing real-time IoT systems considering the application of edge computing, fog computing, SDN (Software-Defined Networking), and NFV (Network Function Virtualization) concepts.
- An overview of the components, methods and technologies of distributed real-time systems, with particular emphasis on scheduling methods and communication protocols in the context of their applications in RTIoT systems.
- The proposal of a general RTIoT system architecture covering a broad class of applications.

The starting point for the RTIoT system design methodology proposed in this article is an analysis of approaches used in other classes of real-time systems. In particular, one can see a certain analogy with embedded systems, or rather, IoT systems can be seen as a certain generalization or development of the concept of embedded systems with a distributed architecture. The differences are particularly evident in the communication layer. This article presents a proposed methodology based on analogous approaches to the synthesis of embedded systems.

The rest of this article is organized as follows. Section 2 describes the functional specification model and synthesis process for real-time systems, and Section 3 considers selected components of IoT systems in relation to real-time requirements. Section 4 is devoted to an analysis of task scheduling methods and communication protocols possible to apply in RTIoT systems, and Section 5 describes the new methodology for designing RTIoT systems proposed by the authors. The summary and conclusions are provided in Section 6.

2. Functional Specification Model and Synthesis of Real-Time Systems

The design methodology for complex IT systems usually consists of two stages:

- Developing a high-level model: The system model allows for an analysis to validate and optimize the proposed solution. This is particularly important in the case of distributed systems, where the verification of the communication and synchronization mechanisms used between processes is of significance.
- Mapping the model to the target architecture: It is possible to map to a standard architecture (e.g., a multi-core processor) or to synthesize a specialized architecture optimized for a given application.

2.1. IoT Application System Model

The first stage of design is to create a system model of the application. The system model represents the application function at the highest abstraction level, where tasks (processes) and interactions between tasks are distinguished. At this level, what a task does is not as important as when it is activated, what the synchronization between tasks is, and how the tasks communicate with each other. Typically, system models are graph models. Existing methods use various system models. We can distinguish, for example:

- Task Graph (TG): one of the simplest and most popular methods of representing functions at the system level. It is a directed acyclic graph in which nodes represent tasks and edges represent the order in which tasks are performed (usually representing transmissions). The task is activated when all preceding ones are completed. In this way, sequential dependencies between tasks are presented. Transmission volumes are represented by edge weights. A sample task graph is shown in Figure 1. The graph describes a six-task system. Tasks on the same path in the graph are sequentially executed, while tasks from parallel paths can be executed in parallel. Extended versions of task graphs can also be found in the literature, e.g., conditional task graphs and multimodal task graphs [50,51]. Such models allow for the specification of special cases, such as the conditional or alternative performance of certain tasks.
- SDF (Synchronous Data Flow) or SSDF (Statically Schedulable Data Flow): models representing data flow (often used in modeling telecommunications applications). Like the TG, SDF is a directed graph. Unlike the TG, cycles can occur in SDF. The synchronization mechanism is described by determining the number of tags generated by the execution of a given task (for each output edge). The number of tags taken from each input is also specified for each task. The task is activated if the required number of tags is available on all inputs.
- STATECHARTS: a model based on a FSM (Finite State Machine), models based on a description in the form of various forms of automation enable the specification of control flow. Unlike a classic FSM, state charts enable parallel descriptions and task specifications. Tasks are related to transitions between states.

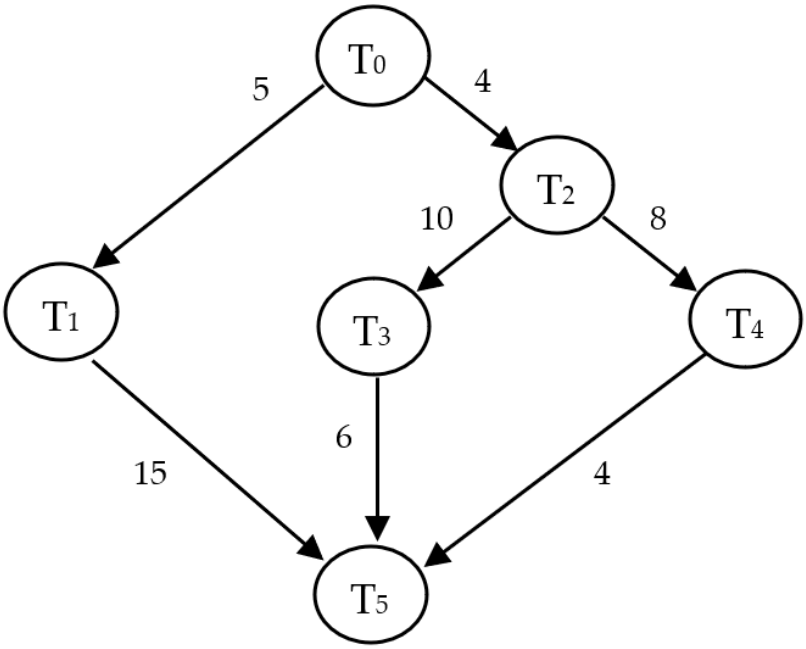

Figure 1. Sample task graph.

There are also a number of other methods that often target specific classes of applications. The system model should not impose any constraints on the target system architecture if this is not directly implied by the design assumptions. The architecture should be the result of system optimization. Therefore, the first step in the methodology should be to propose a method for specifying the application at the system level. This model will be the starting point for further design (scheduling, mapping into architecture, and optimization). Therefore, it should contain all the information so that the above tasks can be performed. Each method has limitations (e.g., the task graph offers no way of describing the control functionality). In the case of RTIoT, the model should allow for the specification of both computational and transmission tasks (communication tasks). The model should also target so-called reactive systems, i.e., systems in which particular sequences of actions/tasks are activated as a result of reactions to specific events.

2.2. Mapping Functional Specifications to Real-Time IoT Architecture

In a typical approach to mapping a system specification into a distributed information system architecture, three main steps are performed [52]:

- Resource allocation: At this stage, the target hardware architecture of the system is determined. Resources can be processors, specialized hardware modules, and communication channels (buses, communication processors, etc.). At the same time as the allocation, the connections between system components are determined.
- Assignment of tasks to resources: This step involves assigning individual tasks to resources. Tasks are assigned to computing modules, and transmissions are assigned to communication channels. Task allocation is closely related to resource allocation. For specialized resources, only tasks corresponding to the function performed by the resource can be assigned. Transmissions must only be assigned to communication channels between resources with assigned communicating tasks. When two tasks are assigned to the same resource, transmissions between them are ignored.
- Task prioritization: Task scheduling is necessary to determine the order in which tasks are performed when more than one task is assigned to a resource. Scheduling must consider the sequential relationships between tasks. During scheduling, speed optimization is performed. In the case of real-time systems, the main goal is to arrange the execution of tasks

and transmissions in such a way that all timing constraints are met (hard-constrained systems) or that timing constraint overruns are minimized (QoS maximization).

IoT systems are characterized by a specific architecture, and for this reason, it is not possible to directly apply existing methods of synthesizing real-time systems to IoT applications. The main distinguishing features of these systems are:

- A distributed architecture based on internet infrastructure.
- Communication through internet links, thus not ensuring that time constraints are met.

From the above-mentioned properties of IoT systems comes the need to develop a specialized method for mapping specifications to the target architecture, as well as the need to develop solutions to ensure the implementation of transmission that meets real-time requirements.

3. Selected Elements of IoT System Architecture

There are many proposals for IoT system architectures. In order for a given architecture to enable the implementation of a real-time system, it must meet certain requirements in terms of the predictability of operation and the possibility of optimization in terms of response time. This section presents an overview of the various elements of the IoT architecture, with a focus on how they can be used in real-time systems.

3.1. Edge and Fog Computing

Today, the amount of data that are generated by IoT systems and then transmitted over the network is increasing at a very high rate. Therefore, it is important to use the throughput of IoT systems as efficiently as possible. The concepts of edge computing and fog computing [53] can be used to optimize this problem. One of the main features of edge computing is the clear separation of the physical devices that make up the edge infrastructure from the core network devices [54]. Edge devices can perform data preprocessing functions at the network edges, which significantly improves the ability to maintain the required transmission time parameters in the context of real-time systems. If the tasks performed by edge devices are subject to time constraints, then these devices are implemented as real-time embedded systems enabling real-time data processing [55,56]. Therefore, it is becoming increasingly desirable to use edge computing for smart data processing and performing specific optimization tasks [57,58]. The partial offloading of centralized cloud systems is also possible through fog computing mechanisms [59]. Solutions of this class can perform services by computation at the level of a distributed, decentralized network infrastructure (e.g., using local IoT networks/systems) [60]. Fog computing and edge computing also provide some opportunity to apply function virtualization techniques on a massive scale [61–65], which consequently reduces the demand for resources and computing power of a data center. Both of the solutions under consideration significantly reduce the amount of data that must be transmitted and processed by the core network/server infrastructure, thus relieving the burden on traditional cloud computing and data center systems. If the target resources are less loaded, then it is easier to meet the assumptions that guarantee throughput and latency at levels appropriate for real-time solutions [66]. The primary purpose of edge infrastructure and the use of distributed computing is to solve a number of critical tasks affecting the latency, throughput, energy consumption, and scalability of IoT systems [67]. Moreover, these solutions allow for a high degree of autonomy in managing the optimal configuration of a system [68,69] and increasing the level of security [70]. All of the above aspects favorably affect the ability to implement IoT services with time requirements.

3.2. Programmable Networks and Virtualization Techniques

In the context of real-time IoT tasks, new developments in Software-Defined Networking (SDN) and mass-scale Network Function Virtualization (NFV) methods are important elements to consider. SDN networks are commonly associated with IoT technology because the architecture provides a more flexible and manageable environment. In addition, more

and more research (for example, on blockchain) is focused on implementing solutions that combine SDN and IoT technologies with NFV methods. Therefore, it is anticipated that NFV will be a very important tool to support the automation of network solutions, including in real-time transmission issues.

In SDN, management tasks are separated from data transmission tasks. In this approach, network devices solely act as relays that transmit sequentially routed packets from the source to the destination, with all policies in place for both communication and security [71]. The management space in SDN is a highly centralized area, while the data space operates on a distributed plane. Potential SDN applications related to real-time transmission are shown in [72], where a four-tier cloudlet architecture (based on SDN) in combination with a WBAN (Wireless Body Area Network) architecture was described. The paper described a cloudlet—a mobile, cloud-based data center system that is located at the edge of the network. This approach was shown to enable efficient traffic processing, optimal resource utilization, and reduced latency, which is of great importance for real-time systems. The approach proposes that all network controllers be located in a cloudlet, resulting in better optimization in terms of QoS mechanisms, as well as a complete view of the network. On the other hand, the authors of [73] proposed the concept of a three-layer SDN application based on a WBAN scheme called an SDWBAN (Software-Defined Wireless Body Area Network). In this solution, network traffic prioritization is performed. Higher-priority packets are analyzed and sent through the network first, which is made possible by optimal resource allocation (e.g., throughput), path allocation and the use of cloud computing. In this case, transmission latency is reduced by formulating a heuristic allocation model for minimizing the service time of individual services. Security issues for IoT systems are also an important issue. In this area, solutions using SDN are known, such as artificial intelligence mechanisms that automatically analyze network traffic while taking real-time transmission requirements into account [74].

In today's vast IT world, technology is beginning to play an increasingly important role in the move away from physical infrastructure building and solutions based on dedicated physical network devices to virtualization technology. An example of virtualization technology in relation to the IoT area is the NFV solution. The implementation of virtualization technology in IoT solutions provides flexibility for applications regardless of the platform on which this solution is based or built while maintaining the full management of logical network functions and the modeling of the full technology stack. An additional advantage of implementing NFV technology in IoT solutions is the ability to fully manage and configure individual virtualized network elements at all levels (from physical machines to virtualized machines to sublime network services) [75]. From the point of view of the IoT and real-time requirements, the use of NFV virtualization technology has begun to play a key role due to the requirements of real-time transmission, i.e., transmission with the shortest possible packet delivery and processing time while maintaining an appropriate QoS and meeting the deadline value requirements of individual packets [76]. The biggest advantage of virtualization technology is its versatility in application to various IoT architectures, although due to its definition and similar mechanisms, NFV technology perfectly complements IoT solutions based on the SDN model architecture.

3.3. Real-Time Database Systems

To ensure the best possible QoS performance in real-time transmission, the solutions described in the previous sections are crucial from the points of view of hardware technologies and transmission and data processing concepts. At the same time, as the data volume generated by IoT systems continues to grow, there is also a steady increase in the need for solutions to store that data while supporting mechanisms that consider real-time requirements [77]. One of the directions is the concept "fast data", which involves developing database technology to ensure the fastest possible execution of queries. Examples of such databases are Druid [78] and Volt Active Data [79].

Some of the most well-known approaches for storing large datasets are *SDDS* (Scalable Distributed Data Structure) structures, which enable the virtualization of distributed memory by aggregating resources at network nodes [80]. Considering real-time requirements and the need for QoS parameters, an extension of this concept in this direction are SDDS LH/RP (Scalable Distributed Data Structure Linear Hashing/Range Partitioning) approaches dedicated to real-time cloud applications [81]. An SDDS server schedules queries using real-time queuing methods that consider importance attributes, which enables the achievement of a higher level of execution of queries in a certain time in the case of SDDS LH/RP approaches than in the case of FIFO (First In, First Out) queuing approaches [82]. With such an implementation, QoS metrics in real-time IoT systems can be significantly improved [83]. The steady increase in demand for big data processing environments has led to the development of more dedicated data storage models, such as NoSQL [84] and NewSQL [85]. These IoT system solutions are beginning to gain value from the point of view of time requirements due to the need to address the processing of large volumes of data while maintaining QoS parameters.

In terms of real-time database systems, issues remain to be addressed in terms of ensuring that hard real-time policies are met for queries that update data that are already recorded/stored and for bucket splitting operations [86,87].

4. Task/Transmission Scheduling and Communication Protocols

The real-time system design methodology proposed in this paper assumes the appropriate scheduling of calculations and transmissions. In classic system solutions, scheduling can be implemented through FIFO queuing or other static or dynamic scheduling methods [88]. Static scheduling methods are mainly concerned with the design of embedded systems, which are most often used to control various devices when there are time constraints. These systems feature a dedicated architecture and perform predetermined functions. Therefore, it is possible to estimate the execution time of each function and to perform a ranking, ensuring that all time constraints are met. IoT systems can be viewed as a generalization of embedded systems with a distributed architecture. The main difference is the communication between system components. In the case of embedded systems, the components are directly connected and the transmission time is predetermined. In the case of IoT systems, transmission time also depends on other factors, e.g., on the traffic and throughput of links and on the route determined by routing processes. Thus, it is not possible to directly apply known static task scheduling methods to IoT systems, though these methods can be considered from the point of view of ensuring optimal cooperation with relevant communication protocols. Therefore, this section examines the scheduling methods and selected communication protocols used in data transmission tasks that assume the need to meet real-time assumptions.

4.1. FIFO

Depending on the application, FIFO queuing mechanisms can be implemented as a hardware sliding register or using different memory structures. Such solutions often use network devices such as switches or routers. These devices queue data packets as they travel from source to destination. Typically, one FIFO structure is used per connection [89]. Some network devices also have multiple FIFO structures/implementations for the simultaneous and independent queuing of different types of data. Most software implementations of FIFO queuing do not meet security conditions and require an additional security mechanism to verify the chain of data structures [90]. Achieving good performance in terms of throughput and scalability while maintaining a high level of security in IoT systems has always been a major challenge. The packet queuing process strategy represented by the FIFO algorithm contributed to the later development of algorithms such as SAGA-PBFT (Security-Aware Genetic Algorithm-based Practical Byzantine Fault Tolerance), which was developed on the basis of the PBFT (Practical Byzantine Fault Tolerance) algorithm. The

SAGA-PBFT algorithm is used in solutions such as the block chain, where security aspects and transmission time requirements are extremely important [91].

4.2. Static and Dynamic Task Scheduling Methods in Real-Time Systems

Static task scheduling algorithms assume that before the target scheduling process, a set of input tasks is processed according to one of the specified sorting rules. These rules define how the input task array is sorted. With this approach, it is possible to determine which packages, criteria, or parameters in the IoT architecture will have the highest priority at the outset. Static task scheduling algorithms include LS (List Scheduling) mechanisms. In this group, we can distinguish the following sorting rules: LPT (Longest Processing Time), SPT (Shortest Processing Time), and RPT (Reverse Processing Time). In this case, the drawback of using list scheduling in real-time solutions is the additional overhead of executing the processes implemented in the sorting rules before the data packet is forwarded to the next network layers. This adversely affects packet delivery times, so meeting the goals of RTIoT solutions is difficult to implement [92]. Another popular static task scheduling algorithm is the Rate Monotonic Scheduling (RMS) method. This algorithm assigns priorities to individual tasks before they begin execution that do not change during the algorithm's run time. Priorities are assigned to each task by taking the frequency of their appearance into account (the task with the shortest period will have the highest priority). RMS is typically used to schedule single-processor tasks [93], which is a major limitation of it for the design of extended RTIoT solutions. On the other hand, its expropriation feature (the highest priority task is performed) is decent in terms of maintaining the best possible QoS performance and meeting real-time conditions.

For dynamic scheduling in the context of the research presented in this article, it is worth considering the EDF (Earliest Deadline First) and LLF (Least Laxity First) algorithms. Real-time systems distinguish between two types of input data. The first type comprises data generated over certain fixed periods of time (e.g., traffic resulting from the functionality of the communication protocols used), and the second type comprises data generated as a result of actual events (e.g., data from various types of sensors). In IoT systems, which deal with data generated as a result of interactions taking place, the EDF algorithm [46] may find application in scheduling processes. In this solution, the value referred to as the deadline for each task is taken as the priority value, so the lower it is, the closer the deadline for a given task is. An additional advantage of the EDF algorithm over others is its good utilization of CPU resources and fairly fast response times, which, from the point of view of designing real-time IoT systems, can have key roles in ensuring adequate QoS parameters and other real-time requirements. Another example of a dynamic scheduling algorithm that can find application in a real-time IoT system is the LLF algorithm. It uses the deadline value, which, along with the time left to complete a given task, is used to calculate the laxity parameter used to prioritize the execution of individual tasks. In this case, the lower the value of laxity, the higher priority the task will be and the faster it should be completed [94]. A certain disadvantage of this algorithm is when there are many tasks with the same or similar laxity value simultaneously, the demand for computing power increases [95]. This represents a kind of limitation from the point of view of applications for RTIoT systems, since one of the priorities of IoT systems is to optimize their energy intensity [96]. On the other hand, however, the advantages of this algorithm include resistance to overloading, to which systems using the EDF algorithm are prone, which translates into the possibility of using it in the design and implementation of real-time systems [83]. Considering the above, it can be concluded that both EDF and LLF can be used for certain tasks in the design and implementation of RTIoT solutions.

4.3. Communication Protocols

In real-time IoT systems, in addition to the described task scheduling techniques, the use of appropriate types of transmission media and the implementation of proper communication protocols, with particular emphasis on routing protocols, also play important

roles. Some well-known routing protocols to consider when designing RTIoT solutions are: RSTP (Real Time Streaming Protocol), WebRTC (Web Real Time Communication), XMPP (Extensible Messaging and Presence Protocol), MQTT (Message Queue Telemetry Transport), CoAP (Constrained Application Protocol), WebSocket, and 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks).

The RSTP is an application layer protocol for the control, transmission and delivery of data while maintaining real-time transmission criteria. The most common implementations of this protocol are systems used to transmit voice and video in real time. Data are continuously delivered, even when RTSP requests are not received by the server. A new area in which this protocol is being applied is the technology area of the teaching of language, which includes listening, speaking, reading and writing [97]. Another protocol implemented in real-time IoT systems is WebRTC. Its most common applications are solutions for transmitting video images or streaming data in the broadest sense while achieving the lowest possible levels of latency and data transmission loss, as exemplified by applications for voice transmission and videoconferencing. Research [98] conducted in the area of the WebRTC protocol indicates that it yields better response times and connection stabilization times in web applications compared with other streaming protocols such as the RTSP [99]. In addition, from the point of view of QoS parameters and energy consumption, the implementation of the WebRTC protocol allows one to achieve better values, which translates into meeting the real-time transmission requirements of solutions implemented in IoT systems. The XMPP protocol, on the other hand, is based on XML [100]. The protocol is commonly available and can be freely modified, and many XMPP servers are available [101]. This translates into its main use being implementations in systems and applications for exchanging text-based content such as popular instant messaging applications. The simplest and lightest protocol for real-time data transmission is the MQTT protocol, which is based on the publish/subscribe pattern. It is designed for transmission for devices that do not require a high throughput. By limiting the transmission speed, the protocol provides a greater reliability. The protocol is perfect for IoT machine-to-machine connections, mobile devices, and where throughput and energy savings are required. By using this protocol, one can easily transfer data during a conversation from one device to another and continue the conversation [102]. An example of a protocol that uses the UDP protocol for communication and is also applicable to RTIoT systems is the CoAP protocol. It has been optimized for constrained devices and networks (used in wireless sensor networks) [103]. The CoAP protocol extends the paradigm of ubiquitous web services in mobile applications to IoT and M2M solutions, which can then be developed using shared and reusable RESTful web services. In doing so, this protocol considers IoT constraints and requirements or QoS requirements [104]. A two-way data exchange channel over a single TCP connection is provided by the WebSocket protocol. The protocol is designed to operate on ports 80 and 443, similarly to HTTP, allowing for interactions between a web browser (or other client application) and a web server at a lower load than alternatives while making it much easier to transfer real-time data to and from the server, which is an asset when considering its use in IoT systems [105]. Thanks to the previously mentioned interactions between the web browser and the server, the protocol has found applications in the development of online games, particularly [106]. With the development of technology, the need for IPv4 address allocation has gradually increased, which has contributed to basically exhausting the entire pool of this addressing. In response to the introduction of the new IPv6 addressing standard and the need to handle packets containing such addresses, the 6LowPAN routing protocol, which is characterized by a low power consumption, was created. The shortcomings of this protocol include the large size of the header relative to the data themselves; from the point of view of application considerations, the implementation of this protocol in RTIoT solutions can contribute to an increase in the level of latency or packet loss with a direct impact on the deterioration of QoS parameters. Today, further research is underway to address the challenges of the 6LowPAN protocol, which include fragmentation, header compression and security [107].

5. RTIoT System Design Methodology

The implementation of the functional model of the real-time system proposed in Section 2 requires the implementation of selected elements included in the IoT solution architecture. Typical IoT systems primarily consist of sensors providing data to the system, end devices (usually also equipped with sensors), a server application, and a client application [5]. The sensors can be directly connected to the server via internet links or can be an integral part of the end device. Indirectly, there may also be edge devices whose job is to pre-process data. Communication is usually carried out via low-energy technologies such as BLE (Bluetooth Low Energy) [108], ZigBee [109], and LoRaWAN (Long Range Wide Area Network) [110]. End devices using sensors communicate with an IoT system via internet links. They can also include actuators and controllers, so an IoT system can remotely control such devices. Server applications are usually installed in cloud computing or on a dedicated server, while the client applications used to operate the system can come in the form of mobile, desktop or web applications. All of the aforementioned components communicate using the relevant protocols.

In the case of real-time systems, there is a time limit on obtaining a response, which means that once a request is sent, a response is expected to be obtained within an assumed maximum time (deadline). Exceeding this time can be treated as an error (systems with hard constraints) or as a reduction in service quality (systems with soft constraints) [111]. Requests can be sent by either end devices or server or client applications. In order to reduce the occurrence of the aforementioned errors or to maintain the required level of quality of service in IoT systems, one can use the solutions described earlier, such as edge computing, fog computing, SDN, NFV, real-time databases, scheduling methods, and relevant communication protocols. These elements are described in Sections 3 and 4 of this article, and based on them, an illustrative architecture for a real-time IoT system can be proposed, as shown in Figure 2. In the architecture, the following computing layers that can perform various tasks of the IoT system can be distinguished:

- Sensor and actuator layer (SL): the layer consisting the interface between an IoT system and its environment that enables the collection of data from the environment and the control of the elements of the environment.
- Edge layer (EL) (optional): an intermediate layer that enables the distributed processing of data without the need to send them to a central system server.
- Cloud layer (CL): the layer that contains the system's servers and databases and that usually controls the operation of the entire system.
- User layer (UL): the layer that uses user applications (mobile, web or desktop). These applications allow a user to interact with the rest of the IoT system.

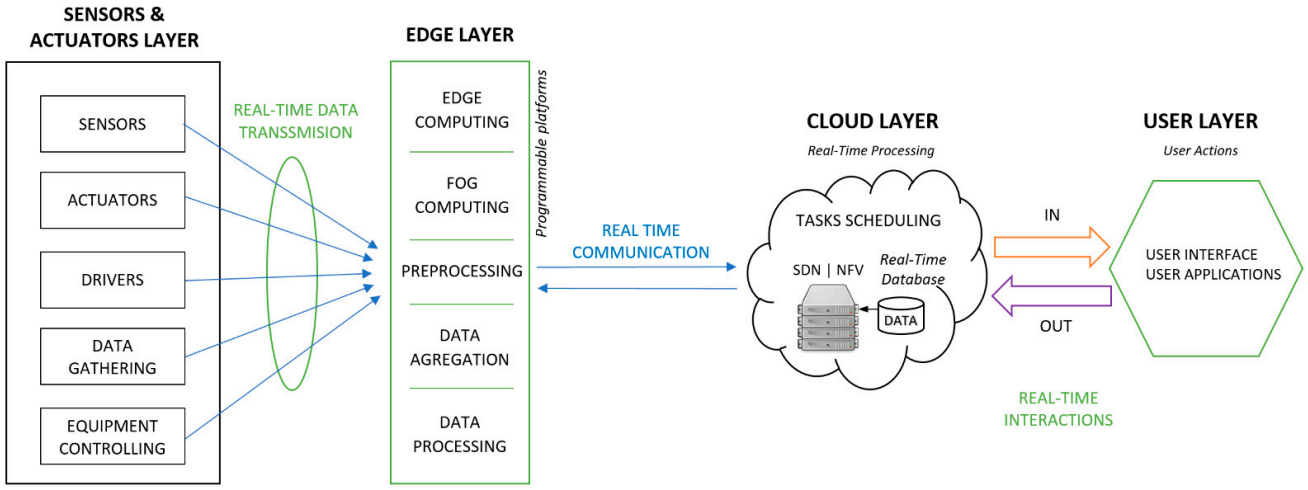

Figure 2. General architecture of a real-time IoT system.

There are communication links between different layers that enable data transfer.

5.1. System Specifications

The starting point for designing an RTIoT system is a system specification in the form of an attributed set of task graphs. Each task graph describes one function of the system. The following attributes are defined for each graph:

- Maximum frequency of graph activation: This attribute specifies the maximum frequency of appearance of input events that cause the execution of functions represented by a given TG.
- Maximum number of TG instances: This attribute specifies the maximum number of simultaneous instances of the task graph. This corresponds to the maximum number of simultaneous events that cause the activation of the functions described by the TG.
- A set of time constraints: The time constraints are associated with the selected paths in the task graph and define the maximum time in which all tasks must be completed from the activation of the task that starts the path to the completion of the task that ends the path.
- For each task, an attribute is specified that assigns the task to a specific layer of the architecture. This attribute is defined by the designer.

Example task graphs are shown in Figures 3–6. These graphs describe nine basic functions of a smart city system for managing parking spaces. Figure 3 shows a user-activated task graph for the following system functionalities: searching for the parking space closest to the user's current location, the function of finding the user's car in the parking lot based on their license plate number, the function of reserving any free parking space based on the entered search criteria, the function of charging a parking fee for the used parking lot, and the function of retrieving information on weather conditions.

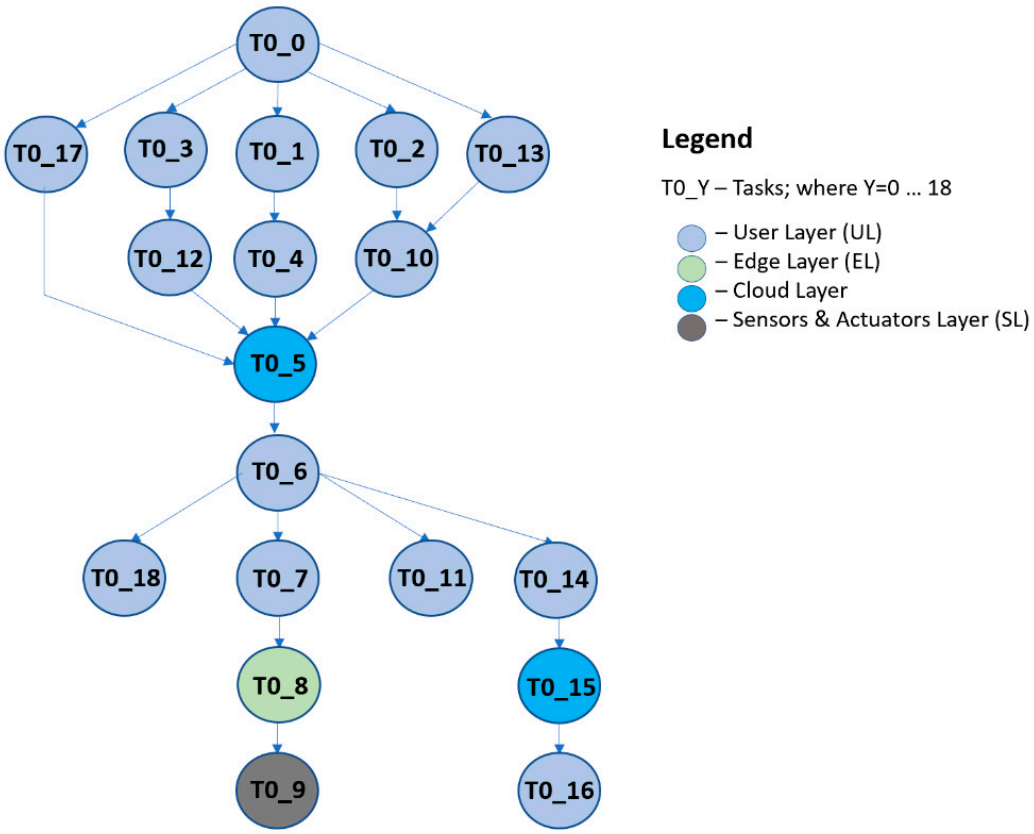

Figure 3. Task graph of the parking space management system.

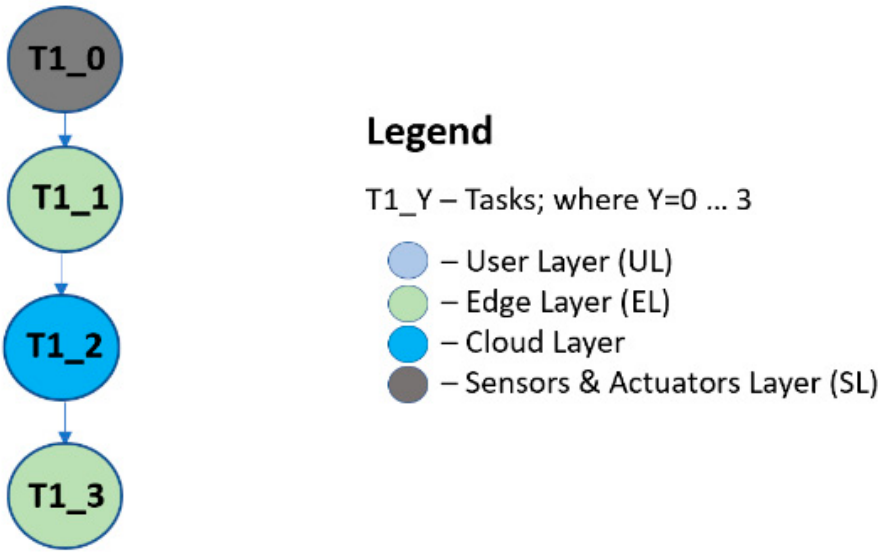

Figure 4. Task graph for the problem of monitoring/managing parking spaces.

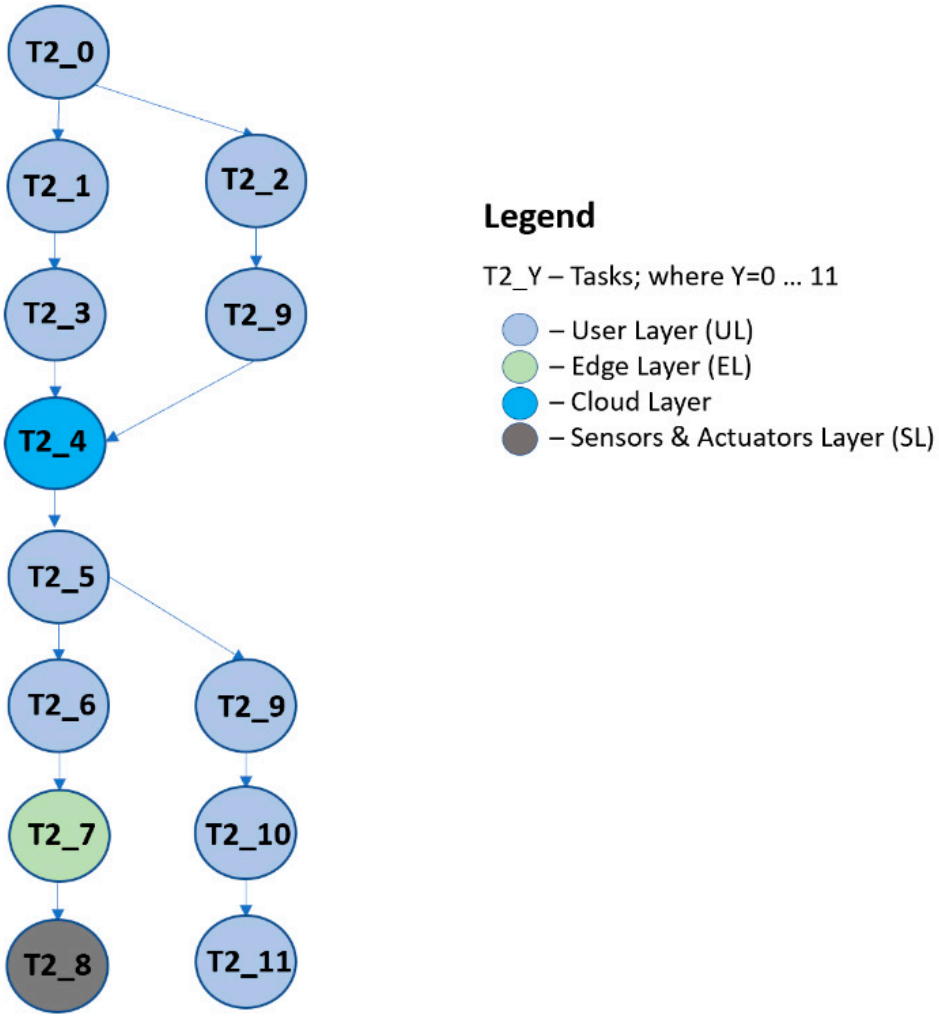

Figure 5. Task graph of basic system functions triggered from the application.

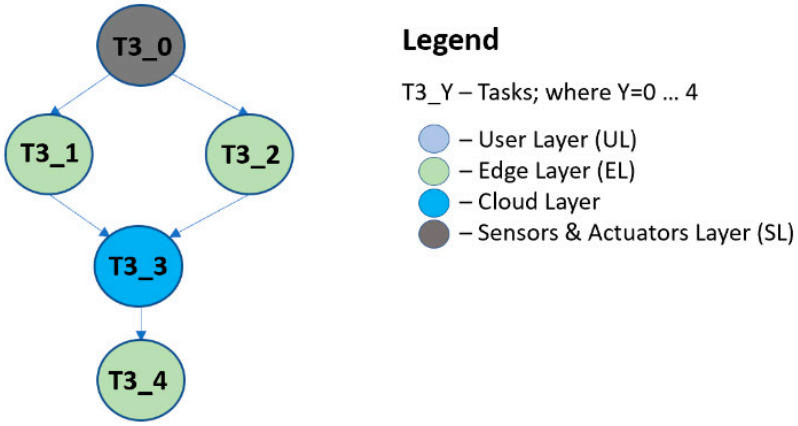

Figure 6. Task graph for the problem of collecting/updating data on weather conditions.

For all system functionality, the starting point, or trigger (T0_0), is the end-user activity in a dedicated application made by selecting the appropriate function at the UL layer level. Depending on the selected function, a further sequence of events follows. When the user selects the parking space search function (T0_1), the user's current location is retrieved using the GPS module (T0_4). If the car search function (T0_2) is selected, the user must enter the license plate number of the searched vehicle (T0_10) into the application. On the other hand, when the function of reserving a free parking space is selected from the user's application (T0_03), the end user enters data/criteria for the search function, such as city and street (T0_12). The user also has the ability to download information on weather conditions (T0_17). In the next step (T0_5), the central server (a dedicated cloud application running in the CL layer) receives queries from the UL layer and starts the processes necessary to perform the called functions. For example, databases can be searched to find free parking spaces while considering information from the GPS tracker. A search for license plate numbers can be run, along with the retrieval of the vehicle location data assigned to them (parking/parking space number). Taking the user's criteria into account, free parking spaces can be searched for, or the fee due can be calculated based on the time the parking space is occupied and the parking rate for the given parking space. The (T0_6) task runs functions in the user application responsible for receiving and processing data coming from the CL layer. In the next step, the user, based on the generated data, selects a parking space by activating the "Reserve" function (T0_7); in turn, for the process of searching for a car based on the entered license plate number (T0_10), the end result is the presentation of the results of the query processing in the end-user application (T0_11) or the start of the payment process using a debit/credit card or bank transfer (T0_14). It is also possible to visualize data on weather conditions (T0_18). Once the task is completed (T0_7), the server-level procedure for processing a parking space reservation request (T0_8) is started, and the state of the sensor is changed; this is realized, for example, by changing the color of the marking of a given space from green to red (T0_9). The task (T0_14) is followed by the payment process through data exchange with the bank/payment card architecture (T0_15). At the end of this path, the result is received in the form of information on the payment in progress (T0_16).

Figure 4 shows a task graph for the functionality of monitoring and updating the status of individual parking spaces.

The starting point/trigger for system function aimed at monitoring or managing parking spaces is to register a change in sensor state/status (T1_0) at the sensor (sensor) layer level. Then, via the network infrastructure, such information is transmitted to the servers that make up the parking system (T1_1). In the next step (T1_2), the central server (a dedicated cloud application running at the CL layer) receives queries from the EL layer and runs the processes necessary to execute the called functions—in this case, a function to update the database with the latest parking sensor states/statuses. In order

for the process to be considered successful, the end result at the edge layer is to run a procedure that receives a confirmation of the completion of cloud processing for the database update (T1_3).

Figure 5 presents a task graph for the following functionalities: the function of reserving an available parking space for given time criteria (T2_1) and the function of plotting a route leading to a vehicle in the parking lot (T2_2). As for the task graph in Figure 3, the trigger for these functionalities is also the end-user activity of selecting the appropriate function from the user application (T2_0). Depending on the user's selection of a specific function, they may be prompted in the next step of the process to provide the data necessary to perform the search function (T2_3). If the function of routing to a vehicle is implemented, the results of calling the cloud processing presentation function may be displayed in the user's application (T2_9). In the next step (T2_4) the central server (a dedicated cloud application running in the CL layer) receives queries from the UL layer and starts the processes necessary to perform the called functions. For example, a database can be searched for free parking spaces given the criteria set by the user or a process can be run to search the database to determine the location of a vehicle with a specific license plate number. Then, the function responsible for receiving and processing data coming from the CL layer (T2_5) is run in the user's application. In the next step, based on the generated data, the user selects a parking space by running the reservation function (T2_6). Alternatively, from the user's application level, a function for the presentation of the results of cloud query processing (T2_9) is run. After the task (T2_6) is completed, the procedure for processing a parking reservation request (T2_7) is started from the EL layer. The next step could be a task to start the navigation function in the user's application interface (T2_10). Finally, a change in the state of the sensor is made; this can be visualized, for example, by changing the color of the marking of the given parking space from green to red (T2_8). For another process, it could be running a function designed to retrieve current position data from a GPS module.

The last graph shows the tasks for the functionality of reading and processing information on the value of the outdoor temperature, as well as reading and processing information on the state of atmospheric pressure.

Similar to the task graph shown in Figure 4, the tasks described for the problem of collecting/updating data on weather conditions also have sensor readings (T3_0) as the starting point/trigger for system operation. Then, through the network infrastructure, the temperature (T3_1) or the atmospheric pressure (T3_2) information is transmitted to the servers that make up the CL layer level of the system. In the next step (T3_3), the central server receives queries from the EL layer and starts the processes necessary to perform the called functions. Processes can be considered successfully completed when the end result at the EL layer is the launch of a procedure that receives confirmation of the completion of cloud processing for database updates (T3_4).

5.2. Mapping Specifications to RTIoT System Architecture

The assignment of tasks to the RTIoT architecture layer is performed by the designer at the system specification stage. The next step is to map the specifications to the different layers of the RTIoT architecture. Figure 7 shows the results of the specification mapping introduced in Section 5.1.

For simplicity, only one instance of each graph was assumed. This corresponds to a system with one user and single sensors (one parking lot and one parking space). In a real system, one instance is created for each user and one instance is created for each sensor that can activate the TG. For this purpose, it is necessary to assume the maximum load on the system, i.e., the maximum number of simultaneous users and the maximum number of parking lots and stands served by the system. For clarity, the figure also does not indicate transmission volumes and time constraints. Based on the mapping, it is possible to obtain information about the transmissions between the layers and the sequence dependencies between these transmissions. For example, there may be 10 transmissions between the

UL and CL layers: T0_4→T0_5, T0_10→T0_5, T0_12→T0_5, T0_17→T0_5, T0_5→T0_6, T0_14→T0_15, T0_15→T0_16, T2_3→T2_4, T2_9→T2_4, and T2_4→T2_5. It can also be noted that there are sequence dependencies between some transmissions, which means that these transmissions will not compete with each other for the communication link. This is important information from the point of view of transmission scheduling.

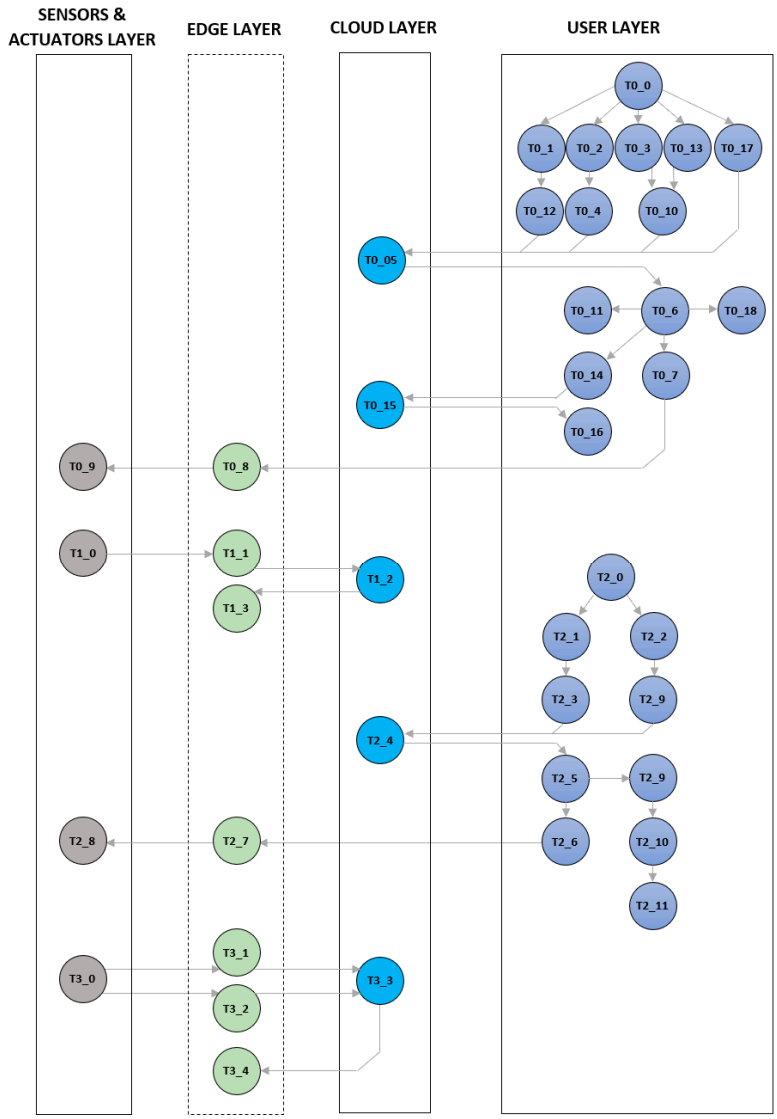

Figure 7. Mapping the described processes to RTIoT system architecture.

Assuming the maximum frequency of activation of individual tasks and the size of individual transmissions, it is possible to estimate the throughput requirements of communication links. These requirements can be reduced by appropriately scheduling transmissions so that time-constrained transmissions are executed in an order that ensures the maximum QoS.

5.3. RTIoT System Optimization

In the RTIoT system, each function, represented by a task graph, can be simultaneously activated by different sources (users and sensors). This can be represented by multiple instances of a given graph. When designing an RTIoT system, one specifies the requirements for the system by identifying the following constraints:

- Maximum number of instances of a given graph: This parameter determines the maximum load on the system in terms of the number of simultaneously activated tasks.

- Maximum processing time (deadline): This parameter specifies the maximum time that can elapse from the start of the t_{start} task to the completion of the t_{stop} task. For a given system, there can be multiple time constraints that define different paths in task graphs. For soft-real-time systems, a soft constraint is defined as d^{s}_{max} and a hard constraint is defined as d^{h}_{max}.

The goal of optimizing RTIoT systems is to achieve the highest possible QoS. The QoS for a single constraint can be defined as:

$$QoS_i = \begin{cases} 0 & \text{if } t_i > d^{h}_{max} \\ 1 & \text{if } t_i < d^{s}_{max} \\ 1 - \frac{t_i - d^{s}_{max}}{d^{h}_{max} - d^{s}_{max}} & \text{in other cases} \end{cases} \quad (1)$$

where t_i is the current execution time of tasks covered by the i-th constraint.

Then, the total QoS for the system can be determined as the average value of all QoS_i:

$$QoS = \sum_{i=0}^{n} \frac{QoS_i}{n} \quad (2)$$

where n is the number of all constraints in all instances of task graphs.

While intending to obtain the highest possible QoS value, the optimization of the system consists in prioritizing the execution of tasks and transmissions in such a way as to minimize any exceedance of time constrains. For this purpose, the scheduling methods described in Section 4 can be used. However, in order to be able to perform tasks and transmissions according to the optimized prioritization, it is necessary to consider the following properties of the RTIoT architecture:

- Transmissions are carried out over shared internet links: To ensure predictable transmission times, it is necessary to develop routing methods that consider the required order and priority of transmissions.
- Individual tasks and transmissions are assigned to resources distributed over the internet. Therefore, in order to execute them in the required order, it is necessary to use time synchronization mechanisms. This can be accomplished by developing appropriate communication protocols.

Assuming that the aforementioned solutions are available, the presented methodology can enable the automatic synthesis of QoS-optimized RTIoT systems. The quality of a given solution is evaluated by calculating the QoS value defined by Equation (2). This value is calculated for the worst-case scenario, i.e., assuming the maximum system load and maximum computation and transmission times.

6. Conclusions

This article presents a methodology for designing RTIoT systems that includes the following elements:

- Four-layer generic RTIoT system architecture model.
- Functional specification method in the form of a set of task graphs with assignment of tasks to IoT architecture layers.
- A method for mapping functional specifications into an RTIoT system architecture.
- Requirements for communication protocols and routing methods used in RTIoT systems.

For the proposed RTIoT architecture model, a method is presented to ensure that the functional specification is mapped into an optimized IoT architecture. The starting point is the specification of the functions of a given system in the form of task graphs. For each task, their assignment to the different layers of the IoT architecture and time constraints are specified. The specification is then mapped into the target architecture by globally prioritizing tasks and transmissions. The scheduling method optimizes the schedule to achieve the highest possible QoS. The presented methodology is based on

analogous methodologies used in the synthesis of embedded systems. However, due to the nature of the IoT architecture, it was necessary to consider specific features of the IoT architecture such as the lack of the centralized control of all elements of the system, the sharing of communication and computing infrastructure with other systems, and the dynamic number of tasks in the system. This paper also provides an overview of existing methods and technologies relevant to RTIoT systems. The purpose of this review was to present current solutions that can be applied to RTIoT systems, as well as to indicate the missing elements that require development in order to create a complete design environment for RTIoT systems. In this way, this paper can also be a valuable resource for designers and others interested in RTIoT.

The proposed methodology was illustrated with an example of designing a practical RTIoT system that performs the functions of a smart system for handling a city's parking lots. The example demonstrated the practical usefulness of the proposed methodology in the design and optimization of RTIoT systems. To the best of our knowledge, this is the first paper to present a complete methodology for the design of a broad class of RTIoT systems while considering the QoS optimization of such systems.

Further work will involve the development of routing methods and communication protocols to enable the implementation of systems designed according to the methodology presented in this work. The result will be a complete RTIoT system design and implementation environment that ensures the creation of systems with a high level of QoS.

Author Contributions: S.D.: conceptualization, methodology, investigation, formal analysis, validation, visualization, writing—original draft, writing—review and editing, supervision, and funding acquisition; M.P.: conceptualization, methodology, investigation, formal analysis, validation, visualization, writing—original draft, writing—review and editing, and supervision; Ł.A.: conceptualization, methodology, investigation, formal analysis, validation, visualization, writing—original draft, and writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Qadri, Y.A.; Nauman, A.; Zikria, Y.B.; Vasilakos, A.V.; Kim, S.W. The Future of Healthcare Internet of Things: A Survey of Emerging Technologies. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 1121–1167. [CrossRef]
2. Sadhukhan, P. An IoT-based E-Parking System for Smart Cities. In Proceedings of the 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, India, 13–16 September 2017.
3. Płaza, M.; Belka, R.; Szcześniak, Z. Towards a different world–On the potential of the internet of everything. *IAPGOS* **2019**, *2*, 8–11. [CrossRef]
4. Shuja, J.; Humayun, M.A.; Alasmary, W.; Sinky, H.; Alanazi, E.; Khan, M.K. Resource efficient geo-textual hierarchical clustering framework for social IoT applications. *IEEE Sens. J.* **2021**, *21*, 25114–25122. [CrossRef]
5. Khanna, A. IoT based Smart Parking System. In Proceedings of the 2016 International Conference on Internet of Things and Applications (IOTA), Maharashtra Institute of Technology, Pune, India, 22–24 January 2016.
6. Ahmad, I.; Pothuganti, K. Design & implementation of real time autonomouscar by using image processing & IoT. In Proceedings of the 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT 2020), Tirunelveli, India, 20–22 August 2020.
7. Belka, R.; Deniziak, S.; Płaza, M.; Hejduk, M.; Pięta, P.; Płaza, M.; Czekaj, P.; Wołowiec, P.; Ludwinek, K. Integrated visitor support system for tourism industry based on IoT technologies. In Proceedings of the SPIE 2018, Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments, Wilga, Poland, 26 May–4 June 2018; p. 108081J. [CrossRef]
8. Pięta, P.; Deniziak, S.; Belka, R.; Płaza, M.; Płaza, M. Multi-domain model for simulating smart IoT-based theme parks. In Proceedings of the SPIE 2018, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments, Wilga, Poland, 26 May–4 June 2018; p. 108082T. [CrossRef]
9. Ivankova, G.V.; Mochalina, E.P.; Goncharova, N.L. Internet of Things (IoT) in logistics. *IOP Conf. Ser. Mater. Sci. Eng.* **2020**, *940*, 1–7. [CrossRef]
10. Song, Y.; Yu, F.R.; Li Zhou, F.; Yang, X.; He, Z. Applications of the Internet of Things (IoT) in Smart Logistics: A Comprehensive Survey. *IEEE Internet Things J.* **2021**, *8*, 4250–4274. [CrossRef]

11. Muangprathub, J.; Boonnam, N.; Kajornkasirat, S.; Lekbangpong, N.; Wanichsombat, A.; Nillaor, P. IoT and agriculture data analysis for smart farm. *Comput. Electron. Agric.* **2019**, *156*, 467–474. [CrossRef]
12. Płaza, M.; Belka, R.; Płaza, M.; Deniziak, S.; Pięta, P.; Doszczeczko, S. Analysis of feasibility and capabilities of RTLS systems in tourism industry. In Proceedings of the SPIE 2018, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments, Wilga, Poland, 26 May–4 June 2018; p. 108080C. [CrossRef]
13. Kodali, R.K.; Rajanarayanan, S.C.; Koganti, A.; Boppana, L. IoT based security system. In Proceedings of the TENCON 2019–2019 IEEE Region 10 Conference (TENCON), Kochi, India, 17–20 October 2019.
14. Kang, B.; Park, S.; Lee, T.; Park, S. IoT-based monitoring system using tri-level context making model for smart home services. In Proceedings of the 2015 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 9–12 January 2015.
15. Chen, X.-Y.; Jin, Z.-G. Research on Key Technology and Applications for Internet of Things. *Phys. Procedia* **2012**, *33*, 561–566. [CrossRef]
16. Balandina, E.; Balandin, S.; Koucheryavy, Y.; Mouromtsev, D. IoT Use Cases in Healthcare and Tourism. In Proceedings of the 2015 IEEE 17th Conference on Business Informatics, Lisbon, Portugal, 13–16 July 2015.
17. Philip, B.V.; Alpcan, T.; Jin, J.; Palaniswami, M. Distributed Real-Time IoT for Autonomous Vehicles. *IEEE Trans. Ind. Inform.* **2019**, *15*, 1131–1140. [CrossRef]
18. Saraf, S.B.; Gawali, D.H. IoT Based Smart Irrigation Monitoring And Controlling System. In Proceedings of the 2017 2nd IEEE International Conference On Recent Trends in Electronics Information & Communication Technology (RTEICT), Bangalore, India, 19–20 May 2017.
19. Pawar, S.; Kithani, V.; Ahuja, S.; Sahu, S. Smart Home Security using IoT and Face Recognition. In Proceedings of the 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 16–18 August 2018.
20. Lu, D.; Teng, Q. Application of Cloud Computing and IoT in Logistics. *J. Softw. Eng. Appl.* **2012**, *5*, 204–207. [CrossRef]
21. Ebling, M.R.; Watson, J. IoT: From Sports to Fashion and Everything In-Between. *IEEE Pervasive Comput.* **2016**, *15*, 2–4. [CrossRef]
22. Tianxiang, Z. A Mobile Architecture to Real-time Device Safety Monitoring of Amusement Park Ride Based on the Internet of Things. *Contemp. Logist.* **2011**, *5*, 42–46.
23. Isyanto, H.; Arifin, A.S.; Suryanegara, M. Design and Implementation of IoT-Based Smart Home Voice Commands for disabled people using Google Assistant. In Proceedings of the 2020 International Conference on Smart Technology and Applications (ICoSTA), Surabaya, Indonesia, 20 February 2020.
24. Patel, S.; Kumar, P.; Garg, S.; Kumar, R. Face Recognition based smart attendance system using IoT. *Int. J. Comput. Sci. Eng.* **2018**, *6*, 871–877. [CrossRef]
25. Avatefipour, O.; Sadry, F. Traffic Management System Using IoT Technology-A Comparative Review. In Proceedings of the 2018 IEEE International Conference on Electro/Information Technology (EIT), Rochester, MI, USA, 3–5 May 2018.
26. Płaza, M.; Deniziak, S.; Płaza, M.; Belka, R.; Pięta, P. Analysis of parallel computational models for clustering. In Proceedings of the SPIE 2018, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments, Wilga, Poland, 3–10 June 2018; p. 108081O. [CrossRef]
27. Pradityo, F.; Surantha, N. Indoor Air Quality Monitoring and Controlling System Based on IoT and Fuzzy Logic. In Proceedings of the 2019 7th International Conference on Information and Communication Technology (ICoICT), Kuala Lumpur, Malaysia, 24–26 July 2019.
28. Kalsoom, T.; Ahmed, S.; Rafi-ul-Shan, P.M.; Azmat, M.; Akhtar, P.; Pervez, Z.; Imran, M.A.; Ur-Rehman, M. Impact of IoT on Manufacturing Industry 4.0: A New Triangular Systematic Review. *Sustainability* **2021**, *13*, 12506. [CrossRef]
29. Jayaraman, P.P.; Perera, C.; Georgakopoulos, D.; Dustdar, S.; Thakker, D.; Ranjan, R. Analytics-as-a-service in a multi-cloud environment through semantically-enabled hierarchical data processing. *Softw. Pract. Exp.* **2010**, *47*, 1139–1156. [CrossRef]
30. Mourtzis, D.; Vlachou, E.; Milas, N. Industrial Big Data as a result of IoT adoption in Manufacturing. *Procedia CIRP* **2016**, *55*, 290–295. [CrossRef]
31. Ramnath, S.; Javali, A.; Narang, B.; Mishra, P.; Routray, S.K. IoT based localization and tracking. In Proceedings of the 2017 International Conference on IoT and Application (ICIOT), Nagapattinam, India, 19–20 May 2017.
32. Khelifi, F.; Bradai, A.; Benslimane, A.; Rawat, P.; Atri, M. A Survey of Localization Systems in Internet of Things. *Mob. Netw. Appl.* **2019**, *24*, 761–785. [CrossRef]
33. Bąk, S.; Czarnecki, R.; Deniziak, S. Synthesis of real-time cloud applications for Internet of Things. *Turk. J. Electr. Eng. Comput. Sci.* **2015**, *23*, 913–929. [CrossRef]
34. Intel. Intel Time Coordinated Computing Tools (Intel TCC Tools). 2022. Available online: https://www.intel.com/content/www/us/en/developer/tools/time-coordinated-computing-tools/overview.html (accessed on 10 September 2022).
35. Intel. Real-Time at the Edge: Overview. 2022. Available online: https://www.intel.com/content/www/us/en/design/technologies-and-topics/iot/real-time.html (accessed on 10 September 2022).
36. Lee, J.; Park, S. Time-Sensitive Network (TSN) Experiment in Sensor-Based Integrated Environment for Autonomous Driving. *Sensors* **2019**, *19*, 1111. [CrossRef]
37. Rawat, P.; Singh, K.D.; Bonnin, J.M. Cognitive Radio for M2M and Internet of Things: A survey. *Comput. Commun.* **2016**, *94*, 1–29. [CrossRef]

38. Perera, C.; Talagala, D.S.; Liu, C.H.; Estrella, J.C. Energy-Efficient Location and Activity-Aware On-Demand Mobile Distributed Sensing Platform for Sensing as a Service in IoT Clouds. *IEEE Trans. Comput. Soc. Syst.* **2015**, *2*, 171–181. [CrossRef]
39. Lasota, M.; Deniziak, S.; Chrobot, A. Scalable Distributed Datastore for Real-Time Cloud Computing. In *Federated Conference on Software Development and Object Technologies*; Springer: Cham, Switzerland, 2017; Volume 511, pp. 193–207.
40. Czarnecki, R.; Deniziak, S. Embedded Real-Time HTTP Server. *Int. J. Comput. Netw. Inf. Secur.* **2015**, *5*, 1–8. [CrossRef]
41. Zhao, J.-C.; Zhang, J.-F.; Feng, Y.; Guo, J.-X. The Study and Application of the IoT Technology in Agriculture. In Proceedings of the 2010 3rd International Conference on Computer Science and Information Technology, Chengdu, China, 9–11 July 2010.
42. Lee, J.; Kim, J.; Kim, S.; Lim, C.; Jung, J. Enhanced distributed streaming system based on RTP/RTSP in resurgent ability. In Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05), Jeju Island, Republic of Korea, 14–16 July 2005.
43. Kirsche, M.; Klauck, R. Unify to bridge gaps: Bringing XMPP into the Internet of Things. In Proceedings of the 2012 IEEE International Conference on Pervasive Computing and Communications Workshops, Lugano, Switzerland, 19–23 March 2012.
44. Krishnaa, G.G.; Krishnaa, G.; Bhalajia, N. Analysis of Routing Protocol for Low-Power and Lossy Networks in IoT Real Time. *Appl. Procedia Comput. Sci.* **2016**, *87*, 270–274. [CrossRef]
45. Biswas, A.R.; Giaffreda, R. IoT and Cloud Convergence: Opportunities and Challenges. In Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT), Seoul, Republic of Korea, 6–8 March 2014.
46. Ahmad, S.; Malik, S.; Ullah, I.; Fayaz, M.; Park, D.-H.; Kim, K.; Kim, D.H. An Adaptive Approach Based on Resource-Awareness Towards Power-Efficient Real-Time Periodic Task Modeling on Embedded IoT Devices. *Processes* **2018**, *6*, 90. [CrossRef]
47. Eugne, D.; Ngangue, N.; Andomaya, C. On Enhancing Technology Coexistence in theIoT Era: ZigBee and 802.11 Case. *IEEE Access* **2016**, *4*, 1835–1844.
48. Stonebraker, M.; Weisberg, A. The VoltDB Main Memory DBMS. IEEE Computer Society Technical Committee on Data Engineering. 2013. Available online: http://sites.computer.org/debull/A13june/VoltDB1.pdf (accessed on 10 September 2022).
49. Li, W.-J.; Yen, C.; Lin, Y.-S.; Tung, S.-C.; Huang, S.M. Just IoT Internet of Things based on the Firebase Real-time Database. In Proceedings of the 2018 IEEE International Conference on Smart Manufacturing, Industrial & Logistics Engineering (SMILE), Hsinchu, Taiwan, 8–9 February 2018.
50. Deniziak, S. Design Models in SYSTEMC Language. In *Technical Transactions*; 1-I; Krakow University of Technology: Krakow, Poland, 2007; pp. 17–33.
51. Wolf, W. *High-Performance Embedded Computing, Architectures, Applications, and Methodologies*; Elsevier: San Francisco, CA, USA, 2007.
52. Deniziak, S.; Tomaszewski, R. Co-synthesis of contention-free energy-efficient NOC-based real time embedded systems. *J. Syst. Archit.* **2019**, *98*, 92–101. [CrossRef]
53. De Donno, M.; Tange, K.; Dragoni, N. Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog. *IEEE Access* **2019**, *7*, 150936–150948. [CrossRef]
54. Alrowaily, M.; Lu, Z. Secure Edge Computing in IoT Systems: Review and Case Studies. In Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing (SEC), Seattle, WA, USA, 25–27 October 2018.
55. Wan, S.; Ding, S.; Chen, C. Edge computing enabled video segmentation for real-time traffic monitoring in internet of vehicles. *Pattern Recognit.* **2022**, *121*, 108146. [CrossRef]
56. Sun, Y.; Fei, T.; Li, X.; Warnecke, A.; Warsitz, E.; Pohl, N. Real-time radar-based gesture detection and recognition built in an edge-computing platform. *IEEE Sens. J.* **2020**, *20*, 10706–10716. [CrossRef]
57. Yu, W.; Liang, F.; He, X.; Hatcher, W.G.; Lu, C.; Lin, J.; Yang, X. A Survey on the Edge Computing for the Internet of Things. *IEEE Access* **2017**, *6*, 6900–6919. [CrossRef]
58. Hamdan, S.; Ayyash, M.; Almajali, S. Edge-Computing Architectures for Internet of Things Applications: A Survey. *Sensors* **2020**, *20*, 6441. [CrossRef]
59. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog Computing and its Role in the Internet of Things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, August 2012; pp. 13–16.
60. Oktian, Y.E.; Witanto, E.N.; Lee, S.-G. A Conceptual Architecture in Decentralizing Computing, Storage, and Networking Aspect of IoT Infrastructure. *IoT* **2021**, *2*, 205–221. [CrossRef]
61. Li, J.; Jin, J.; Yuan, D.; Zhang, H. Virtual Fog: A Virtualization Enabled Fog Computing Framework for Internet of Things. *IEEE Internet Things J.* **2018**, *5*, 121–131. [CrossRef]
62. Li, Y.; Xuan Phan, L.T.; Loo, B.T. Network Functions Virtualization with Soft Real-Time Guarantees. In Proceedings of the IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, USA, 10–14 April 2016.
63. Jawdhari, H.A.; Abdullah, A.A. The Application of Network Functions Virtualization on Different Networks, and its New Applications in Blockchain: A Survey. *Spec. Issue Comput. Technol. Inf. Manag.* **2021**, *18*, 1007–1044. [CrossRef]
64. Abdulqadir, H.R.; Zeebaree, S.R.M.; Shukur, H.M.; Sadeeq, M.A.M.; Salim, B.W.; Salih, A.A.; Kak, S.F. A study of moving from cloud computing to fog computing. *Qubahan Acad. J.* **2022**, *2*, 60–70. [CrossRef]
65. Cao, K.; Liu, Y.; Meng, G.; Sun, Q. An overview on edge computing research. *IEEE Access* **2020**, *8*, 85714–85728. [CrossRef]
66. Al-Shammari, B.K.J.; Al-Aboody, N.; Al-Raweshidy, H.S. IoT Traffic Management and Integration in the QoS Supported Network. *IEEE Internet Things J.* **2017**, *5*, 352–370. [CrossRef]

67. Cui, L.; Xu, C.; Yang, S.; Huang, J.Z.; Li, J.; Ming, X.W.Z.; Lu, N. Joint Optimization of Energy Consumption and Latency in Mobile Edge Computing for Internet of Things. *IEEE Internet Things J.* **2018**, *6*, 4791–4803. [CrossRef]
68. Atlam, H.F.; Walters, R.J.; Wills, G.B. Fog Computing and the Internet of Things: A Review. *Big Data Cogn. Comput.* **2018**, *2*, 10. [CrossRef]
69. Novo, O. Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT. *IEEE Internet Things J.* **2018**, *5*, 1184–1195. [CrossRef]
70. Ngabo, D.; Wang, D.; Iwendi, C.; Anajemb, J.A.; Ajao, L.A.; Biamba, C. Blockchain-Based Security Mechanism for the Medical Data at Fog Computing Architecture of Internet of Things. *Electronics* **2021**, *10*, 2110. [CrossRef]
71. Tayyaba, S.K.; Shah, M.A.; Khan, O.A.; Ahmed, A.W. Software Defined Network (SDN) Based Internet of Things (IoT): A Road Ahead. In Proceedings of the ICFNDS'17: Proceedings of the International Conference on Future Networks and Distributed Systems, Cambridge, UK, 19–20 July 2017; pp. 1–8.
72. Mazhar, N.; Salleh, R.; Zeeshan, M.; Hameed, M.M.; Khan, N. R-IDPS: Real time SDN based IDPS system for IoT security. In Proceedings of the 2021 IEEE 18th International Conference on Smart Communities: Improving Quality of Life Using ICT, IoT and AI (HONET), Karachi, Pakistan, 11–13 October 2021.
73. Hasan, K.; Wu, X.-W.; Biswas, K.; Ahmed, K. A Novel Framework for Software Defined Wireless Body Area Network. In Proceedings of the 2018 8th International Conference on Intelligent Systems, Modelling and Simulation (ISMS), Kuala Lumpur, Malaysia, 8–10 May 2018.
74. Yassein, M.B.; Aljawarneh, S.; Al-Rousan, M.; Mardini, W.; Al-Rashdan, W. Combined Software-Defined Network (SDN) and Internet of Things (IoT). In Proceedings of the 2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA), Ras Al Khaimah, United Arab Emirates, 21–23 November 2017.
75. Li, J.; Altman, E.; Touati, C. A General SDN-based IoT Framework with NVF Implementation. *ZTE Commun.* **2015**, *13*, 42–45.
76. Tang, W.; Zhang, R.; Feng, S. A Spatiotemporal Model for Hard-deadlineMulti-stream Traffic in Uplink IoT Networks. *IEEE Internet Things J.* **2021**, *9*, 601–615. [CrossRef]
77. Audsley, N.C.; Burns, A.; Richardson, M.F.; Wellings, A.J. Absolute and relative temporal constraints in hard real-time databases. In Proceedings of the Fourth Euromicro Workshop on Real-Time Systems, Athens, Greece, 3–5 June 1992.
78. Druid. Apache Druid Is a Real-time Database to Power Modern Analytics Applications. 2022. Available online: https://druid.apache.org/ (accessed on 10 September 2022).
79. Volt Active Data. Where We Sit in the Stack. 2022. Available online: https://www.voltactivedata.com/ (accessed on 10 September 2022).
80. Litwin, W.; Neimat, M.-A.; Schneider, D.A. LH*—A scalable, distributed data structure. *ACM Trans. Database Syst.* **1996**, *21*, 480–525. [CrossRef]
81. Lasota, M.; Deniziak, S.; Chrobot, A. An SDDS-Based Architecture for a Real-Time Data Store. *Int. J. Inf. Eng. Electron. Bus.* **2016**, *1*, 21–28. [CrossRef]
82. Hayatunnufus; Riasetiawan, M.; Ashari, A. Performance Analysis of FIFO and Round Robin Scheduling Process Algorithm in IoT Operating System for Collecting Landslide Data. In Proceedings of the 2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA), Medan, Indonesia, 16–17 July 2020.
83. Malik, S.; Ahmad, S.; Ullah, I.; Park, D.H.; Kim, D.H. An Adaptive Emergency First Intelligent Scheduling Algorithm for E cient Task Management and Scheduling in Hybrid of Hard Real-Time and Soft Real-Time Embedded IoT Systems. *Sustainability* **2019**, *11*, 2192. [CrossRef]
84. Pereira, D.A.; Ourique de Morais, W.; Pignaton de Freitas, E. NoSQL real-time database performance comparison. *Int. J. Parallel Emergent Distrib. Syst.* **2017**, *33*, 144–156. [CrossRef]
85. Kaur, K.; Sachdeva, M. Performance Evaluation of NewSQL Databases. In Proceedings of the 2017 International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 19–20 January 2017.
86. Singh, R.K.; Pandey, S.; Shanker, U. A Non-Database Operations Aware Priority Ceiling Protocol for Hard Real-Time Database Systems. In Proceedings of the 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kanpur, India, 6–8 July 2019.
87. Halang, W.A.; Stoyenko, A.D. *Real Time Computing*; Springer: Berlin/Heidelberg, Germany, 1994; pp. 261–282.
88. Abohamama, A.S.; El-Ghamry, A.; Hamouda, E. Real-Time Task Scheduling Algorithm for IoT-Based Applications in the Cloud–Fog Environment. *J. Netw. Syst. Manag.* **2022**, *30*, 54. [CrossRef]
89. Koukopoulos, D. The Impact of FIFO Compositions with Other Protocols on the Stability of Multimedia Networks Facing Dynamic Adversarial Attacks. In Proceedings of the MINES'13, 2013 Fifth International Conference on Multimedia Information Networking and Security, Beijing, China, 1 November 2013; pp. 575–578.
90. Kashyap, R.; Arora, K.; Sharma, M.; Aazam, A. Security-Aware GA based Practical Byzantine Fault Tolerance for Permissioned Blockchain. In Proceedings of the 2019 4th International Conference on Control, Robotics and Cybernetics (CRC), Tokyo, Japan, 27–30 September 2019.
91. Xu, G.; Liu, Y.; Xing, J.; Luo, T.; Gu, Y.; Liu, S.; Zheng, X.; Vasilakos, A.V. SG-PBFT: A Secure and Highly Efficient, Blockchain PBFT Consensus Algorithm for Internet of Vehicles, Computer Science. *J. Parallel Distrib. Comput.* **2022**, *164*, 1–11. [CrossRef]
92. Park, G.-L.; Shirazi, B.; Marquis, J.; Choo, H. Decisive path scheduling: A new list scheduling method. In Proceedings of the 1997 International Conference on Parallel Processing, Bloomington, IL, USA, 11–15 August 1997.

93. Zonios, C.; Tenentes, V. Energy Efficient Speech Command Recognition for Private Smart Home IoT Applications. In Proceedings of the 2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, Greece, 5–7 July 2021.
94. Teng, S.; Zhang, W.; Zhu, H.; Fu, X.; Su, J.; Cui, B. A Least-Laxity-First Scheduling Algorithm of Variable Time Slice for Periodic Tasks. *Int. J. Softw. Sci. Comput. Intell.* **2012**, *2*, 19.
95. Furst, J.; Chen, K.; Kim, H.-S.; Bonnet, P. Evaluating Bluetooth Low Energy for IoT. In Proceedings of the 2018 IEEE Workshop on Benchmarking Cyber-Physical Networks and Systems (CPSBench), Porto, Portugal, 10–13 April 2018.
96. Beshley, M.; Kryvinska, N.; Beshley, H.; Yaremko, O.; Pyrih, J. Virtual Router Design and Modeling for Future Networks with QoS Guarantees. *Electronics* **2021**, *10*, 1139. [CrossRef]
97. Schulzrinne, H.; Rao, A.; Lanphier, R. Real Time Streaming Protocol (RTSP). *RFC* **1998**, *2326*.
98. Rhinow, F.; Veloso, P.P.; Puyelo, C.; Barrett, S.; O Nuallain, E. P2P live video streaming in WebRTC. In Proceedings of the 2014 World Congress on Computer Applications and Information Systems (WCCAIS), Hammamet, Tunisia, 17–19 January 2014.
99. Jianbing, L.; Shuhui, C. The Design and Implementation of RTSP/RTP Multimedia Traffic Identification Algorithm. *J. Phys. Conf. Ser.* **2019**, *1168*, 1–8. [CrossRef]
100. Saint-Andre, P. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920, IETF. October 2004. Available online: https://www.rfc-editor.org/rfc/rfc6120 (accessed on 10 September 2022).
101. Saint-Andre, P. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence, RFC 3921, IETF. October 2004. Available online: http://www.kandroid.org/board/data/board/guestbook/file_in_body/1/xmpp.pdf (accessed on 10 September 2022).
102. MQTT. MQTT: The Standard for IoT Messaging. 2022. Available online: https://mqtt.org (accessed on 10 September 2022).
103. Bormann, C.; Castellani, A.P.; Shelby, Z. CoAP: An Application Protocol for Billions of Tiny Internet Nodes. *IEEE Internet Comput.* **2012**, *16*, 62–67. [CrossRef]
104. Porcius, M.; Fortuna, C.; Kandus, G.; Mohorcic, M. Integrating custom hardware into Sensor Web.SoftCOM 2010. In Proceedings of the 18th International Conference on Software, Telecommunications and Computer Networks, Split, Croatia, 23–25 September 2010.
105. Internet Engineering Task Force. The WebSocket Protocol. 2022. Available online: https://datatracker.ietf.org/doc/html/rfc6455 (accessed on 10 September 2022).
106. Muller, G.L. HTML5 WebSocket Protocol and Its Application to Distributed Computing. Available online: https://arxiv.org/abs/1409.3367 (accessed on 10 September 2022).
107. Ha, M.; Kim, D.; Kim, S.H.; Hong, S. Inter-MARIO: A Fast and Seamless Mobility Protocol to Support Inter-Pan Handover in 6LoWPAN. In Proceedings of the 2010 IEEE Global Telecommunications Conference GLOBECOM 2010, Miami, FL, USA, 6–10 December 2010; pp. 1–6.
108. Tosi, J.; Taffoni, F.; Santacatterina, M.; Sannino, R.; Formica, D. Performance Evaluation of Bluetooth Low Energy: A Systematic Review. *Sensors* **2017**, *17*, 2898. [CrossRef]
109. Varghese, S.G.; Kurian, C.P.; George, V.I.; John, A.; Nayak, V.; Upadhyay, A. Comparative study of zigBee topologies for IoT-based lighting automation. *IET Wirel. Sens. Syst.* **2019**, *4*, 201–207. [CrossRef]
110. Haxhibeqiri, J.; De Poorter, E.; Moerman, I.; Hoebeke, J. A Survey of LoRaWAN for IoT: From Technologyto Application. *Sensors* **2018**, *18*, 3995. [CrossRef] [PubMed]
111. Wu, C.-G.; Wang, L. A Deadline-Aware Estimation of Distribution Algorithm for Resource Scheduling in Fog Computing Systems. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 10–13 June 2019.