Hindawi
Wireless Communications and Mobile Computing
Volume 2019, Article ID 1684906, 12 pages
https://doi.org/10.1155/2019/1684906

Research Article

A LoRaWAN Testbed Design for Supporting Critical Situations: Prototype and Evaluation

Jorge Navarro-Ortiz[1], **Juan J. Ramos-Munoz**[1], **Juan M. Lopez-Soler**[1], **Cristina Cervello-Pastor**[2] and **Marisa Catalan**[3]

[1]*Department of Signal Theory, Telematics and Communications, University of Granada (UGR), Granada, Spain*
[2]*Department of Network Engineering, Universitat Politècnica de Catalunya (UPC), Castelldefels, Spain*
[3]*Fundació i2CAT, Barcelona, Spain*

Correspondence should be addressed to Jorge Navarro-Ortiz; jorgenavarro@ugr.es

Received 29 November 2018; Accepted 7 February 2019; Published 21 February 2019

Academic Editor: Laurie Cuthbert

The Internet of Things is one of the hottest topics in communications today, with current revenues of $151B, around 7 billion connected devices, and an unprecedented growth expected for next years. A massive number of sensors and actuators are expected to emerge, requiring new wireless technologies that can extend their battery life and can cover large areas. LoRaWAN is one of the most outstanding technologies which fulfill these demands, attracting the attention of both academia and industry. In this paper, the design of a LoRaWAN testbed to support critical situations, such as emergency scenarios or natural disasters, is proposed. This self-healing LoRaWAN network architecture will provide resilience when part of the equipment in the core network may become faulty. This resilience is achieved by virtualizing and properly orchestrating the different network entities. Different options have been designed and implemented as real prototypes. Based on our performance evaluation, we claim that the usage of microservice orchestration with several replicas of the LoRaWAN network entities and a load balancer produces an almost seamless recovery which makes it a proper solution to recover after a system crash caused by any catastrophic event.

1. Introduction

The Internet of Things (IoT) is one of the hottest topics in communications today. Although previous forecasts may have overestimated the growth of connected IoT-devices, it is clear that the current and short-term market revenues are impressive: from $151B in 2018 up to $1,567B by 2025. Current IoT-devices vary from 6 to 9 billion devices (e.g., the current number of IoT devices in 2018 is 7 billion, according to IoT-analytics [1]) whereas forecasts estimate from 20 to 30 billion IoT devices by 2020 (e.g., Ericsson figure is 28 billion by 2021 [2]).

Many of the IoT services follow the category of massive Machine-Type Communications (mMTC), one of the three major 5G use cases (in addition to enhanced mobile broadband and ultrareliable and low latency MTC). Since mMTC communications assume a massive number of devices, in most of the cases battery-powered and in a high number of locations and environments, its main requirements are low-power communications and a wide range coverage.

Two main technologies which fulfill these requirements are being used for these applications: cellular evolution and Low Power Wide Area Networks (LPWAN).

Regarding cellular evolution, the Third Generation Partnership Project (3GPP) has tried to adapt the existing mobile standards for the requirements of IoT devices. In this way, cellular IoT standards utilize the existing mobile network infrastructures in an effort to integrate both worlds. Some of the main cellular technologies for IoT are Extended Coverage Global System for Mobile communications (EC-GSM), LTE Cat-0 (new low complexity Long Term Evolution device, defined in 3GPP Release 12), LTE-M, and narrow-band IoT (NB-IoT). NB-IoT addresses the specific requirements of mMTC but, unlike LTE Cat-0 and LTE-M devices, requires a specific frequency band different from those used for LTE or LTE-Advanced.

To the contrary of cellular technologies, LPWAN technologies have been born with IoT requirements from the very beginning. Previous attempts such as local or mesh networks can accommodate part of IoT services, such as low battery consumption and optimization for low data rates but are not intended for global coverage. Some of the most popular LPWAN technologies are LoRaWAN, SigFox, RPMA, and NWave. They offer long range (up to several tens of kilometers), very low power consumption (years of battery operation), and very low bandwidth (tens of kbps) and utilize license-exempt frequency bands. Another advantage of LPWANs is that they require a much lower investment compared to mobile networks, allowing new players to compete with current Mobile Network Operators (MNOs). For this reason, many MNOs (e.g., KPN, Orange, SK Telecom, Bouygues Telecom, Swisscom, and SoftBank) have started to deploy LoRaWAN (Long-Range Wide Area Network) to complement their current cellular networks deployments.

In this article, we propose the design of a LoRaWAN testbed for supporting critical situations, i.e., an IoT testbed that shall be able to automatically recover if part of its network infrastructure is destroyed. Since, in the case of LoRaWAN, the radio equipment is cheap and can be easily replaced, we will focus on the core network infrastructure.

This testbed will be integrated with the demonstrator from the 5G-City [3] project. 5G-City is a Spanish coordinated research project among five universities and research centres (Universitat Politècnica de Catalunya, Universidad de Granada, Fundació i2CAT, Universidad Carlos III, and Universidad del Pais Vasco), which aims at providing an adaptive management of 5G services to support critical events in cities. In that sense, 5G-City is focused on one of the most difficult situations for current and future communications systems: unexpected events that affect a relatively large amount of mobile users which are concentrated in a small area, such as traffic jams due to congestion or accidents, disasters, or any other emergency situations that may affect a large number of users. An overview of the 5G-City demonstrator is shown in Figure 1.

One of the objectives of the 5G-City research project is the design of a virtualized 5G network for massive IoT and broadband experience. Within the 5G context, different wireless technologies will coexist as technological alternatives for the interconnection between information producers and consumers.

In the case of IoT, one of the wireless technologies that will be included in the 5G-City demonstrator will be the LoRaWAN network prototype proposed in this paper. For that purpose, this prototype will be integrated with the 5G-City 4G/5G network demonstrator following our previous work in [4].

The literature that defines the state of the art regarding the evaluation of LoRaWAN and LoRa testbeds in real deployments is very rich in quality and quantity. A number of papers have been devoted to measuring LPWAN performance metrics in both indoor and outdoor deployments, as well as in rural, urban, and suburban scenarios.

Almost invariably all of these works focus on coverage measurements. Particularly, different evaluations are reported, ranging from covered distance [5–8] to percentage of packets successfully delivered [9], in different scenarios and operation conditions, i.e. different payload sizes [10] and different impairments or spreading factors [11, 12], in high-density urban environments with many multifloor buildings [13].

In [14] a comparison of different LoRaWAN testbeds is provided in terms of several metrics (RSSI, SNR, and distances covered). Reference [15] evaluates the packet transmission time for different spreading factors. Additionally, [16] evaluates the average LoRaWAN throughput as a function of the spreading factor for different payload sizes.

However, none of the reported papers evaluates the LoRaWAN resilience dimension under a critical situation, i.e., the elapsed time for recovery and packet losses impact after a system crash. Note that these quantitative evaluations will definitively help to determine the LoRaWAN suitability for IoT deployments under critical circumstances.

The main objective of this paper is the proposal of a self-healing LoRaWAN network architecture in order to provide resilience under critical situations such as earthquakes, fires, or hurricanes. Under such conditions, part of the network equipment may become faulty. By virtualizing the different entities in the core network, i.e., converting them into VNFs (Virtual Network Functions), we will be able to reduce costs, increase flexibility, and provide resilience. We have implemented different options for the virtualization of the LoRaWAN core network entities which will be compared in terms of time for recovery, packet losses, and resource usage.

For this objective, the rest of the article is organized as follows. Section 2 provides a LoRaWAN technology overview, including main transmission characteristics and architecture. Section 3 describes LoRaWAN implementation issues. Particularly, both NFV (Network Function Virtualization) orchestration options using microservices and virtual machines are explained. Section 4 explains our testbed. It includes the hardware setup and the software platform details. Section 5 identifies the evaluated four use cases, and it includes the obtained results. Finally, Section 6 sums up the paper and provides the main conclusions.

2. LoRaWAN Overview

LoRaWAN [17] is a standardized Low Power Wide Area Network (LPWAN) which uses LoRa [18] or FSK modulations. LoRa is a proprietary modulation owned by the French company Semtech. This modulation is based on CSS (Chirp Spread Spectrum) and it features the same low-power characteristics of FSK but increases the coverage range. The bandwidth of a LoRa signal can be 125, 250, or 500 kHz, and different spreading factors (SF) can be used to achieve a trade-off between data rate and coverage.

The spreading factor is defined as $SF = \log_2(R_C/R_S)$, where R_C is the chip rate and R_S is the symbol rate. Since the chip rate is constant for a fixed bandwidth ($R_C = BW$ chips/sec, where BW is the bandwidth), a higher SF implies a lower data rate but increases the transmission range due to a higher robustness. Codes from different SFs are orthogonal,

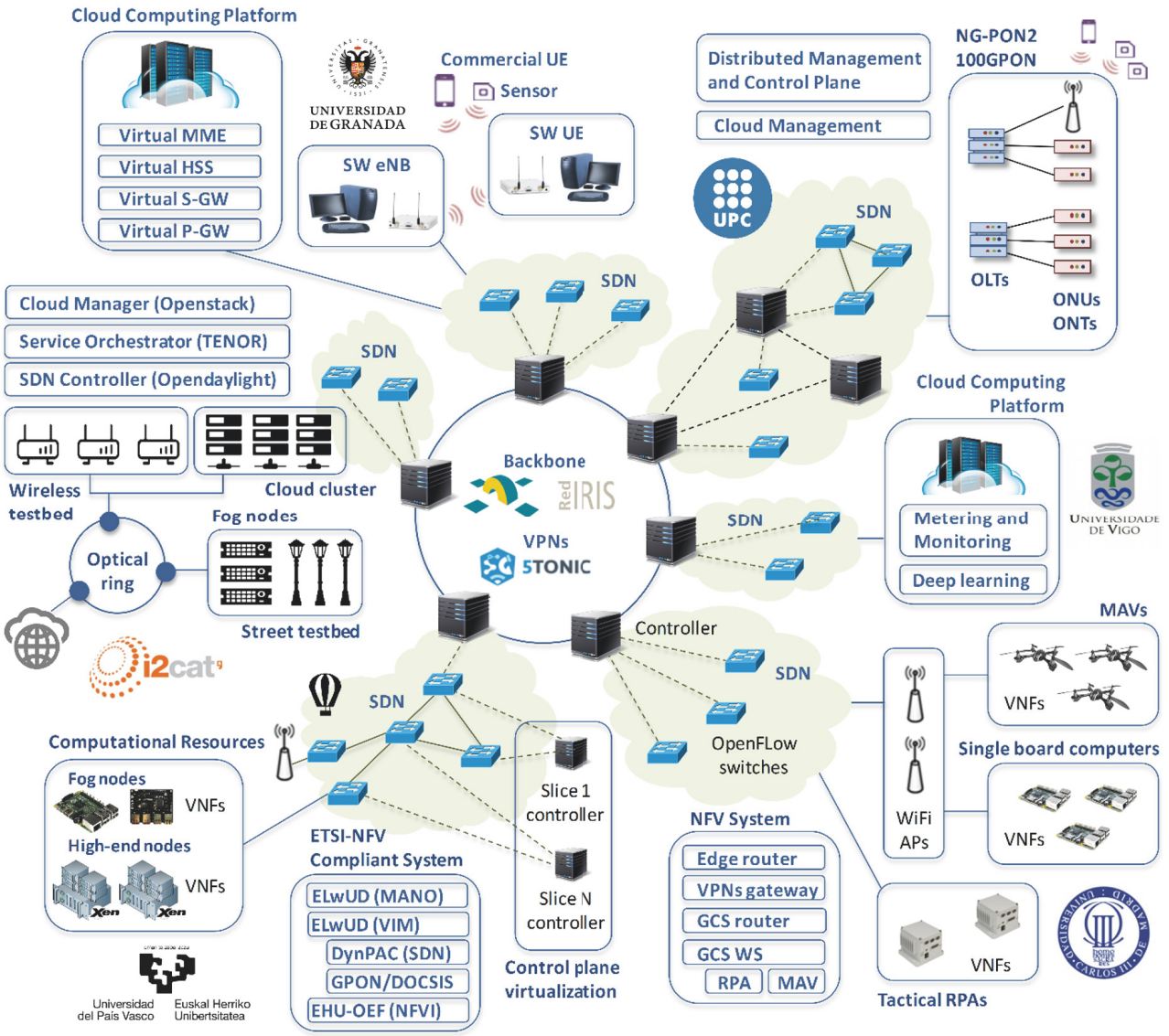

Figure 1: Testbed of the 5G-City research project [3].

so multiple frames can be simultaneously transmitted on the same channel as long as they utilize different SFs.

LoRaWAN is an open standard managed by the LoRa Alliance. LoRaWAN defines the Medium Access Control (MAC) layer on top of the LoRa physical layer. It also defines the system architecture.

The MAC layer utilizes a duty cycle to reduce the probability of collisions in a simple and hardware-efficient manner. Depending on regional regulations [19], this duty cycle can be, e.g., 1%, meaning that a LoRaWAN node can only transmit 1% of the time, thus affecting its maximum data rate. This limitation makes the SF selection have a high impact on the transmission rate since it is determining the Time on Air (ToA). ToA can be computed as

$$ToA = T_{preamble} + T_{payload} = T_{preamble} + T_S \times n_{payload}, \quad (1)$$

where $T_{preamble}$ and $T_{payload}$, respectively, are the preamble and payload transmission time, whereas $T_S = 2^{SF}/BW$ is the symbol period.

In the case of European regulations, the duty cycle is 1% and the combination between SFs and bandwidths produces the different data rates (DR) included in Table 1. This table also includes the ToA and the minimum time between consecutive frames (i.e., to fulfill the duty cycle limitation) assuming an application payload of 12 bytes.

The architecture of a LoRaWAN network is based on a star topology, as shown in Figure 2. This figure shows the different entities in a common LoRaWAN deployment. It includes the Radio Access Network (RAN) and the Core Network (CN). The RAN is composed of nodes and gateways, which act as base stations forwarding the frames received from the radio interface to the core network entities. The CN is composed

Table 1: Parameters for the different LoRaWAN DRs.

DR	SF	BW (kHz)	data rate (bps)	ToA (ms)	Time between frames (s)
0	SF12	125	250	1482.8	148.3
1	SF11	125	440	823.3	82.3
2	SF10	125	980	411.6	41.2
3	SF9	125	1760	205.8	20.6
4	SF8	125	3125	113.2	11.3
5	SF7	125	5470	61.7	6.2
6	SF7	250	11000	30.8	3.1

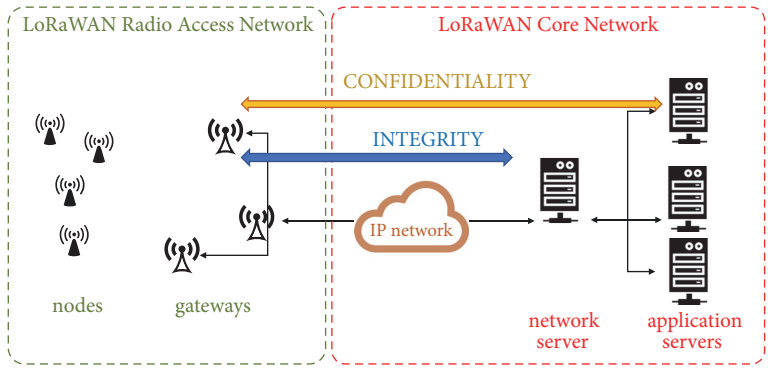

Figure 2: LoRaWAN network architecture.

of an IP network and two types of servers: network and application servers.

All the frames forwarded from the gateways to the network server are integrity protected (thanks to a Message Integrity Code (MIC) generated with a network session key, `NwkSKey`), whereas privacy is kept up to the application server (thanks to the encryption of the payload with an application session key, `AppSKey`). The network server sends packets to the appropriate application server, which handles the customer application and processes the customer data. Since this architecture achieves end-to-end security, the IP network infrastructure can be from a different provider.

In order to exchange the required session keys (`NwkSKey` and `AppSKey`), the LoRaWAN standard defines two activation methods when the node is attached to the network: Activation by Personalization (ABP) and Over-the-Air Activation (OTAA).

In the first case, the developer shall include this information in both the nodes (i.e., stored in their firmware) and the servers. Thus, no signalling is needed. In the second case, the node shall send a `JoinRequest` frame with a device identifier (`DevEUI`), an application identifier (`AppEUI`), and a random challenge (`DevNonce`). Upon receiving this frame, the gateway shall send a `JoinResponse` frame with the device address (`DevAddr`), a network identifier (`NetID`), and another random challenge (`AppNonce`). With these data and a preshared key (`AppKey`), both the node and the servers are able to derive the same `NwkSKey` and `AppSkey`, which are used for the subsequent transmissions. Both activation types are summarized in Figure 3.

LoRaWAN allows nodes to have bidirectional communications with gateways although asymmetric, since uplink transmissions (from nodes to gateways) are strongly favored. Three types of devices are defined (classes A, B, and C) with different capabilities. Class A is the most energy efficient and must be supported by all nodes. Class A nodes use pure ALOHA for uplink access, and they can only receive a downlink frame after a successful uplink transmission. This class is intended for battery-operated sensors. Class B nodes utilize beacons sent from the gateway to determine whether they have to receive downlink frames or not, using scheduled receive windows at a predictable time without the need of successful uplink transmissions. This class is intended for battery-operated actuators. Finally, class C nodes are always listening to the radio interface except when they are transmitting. Due to its power consumption, class C is intended for main powered actuators. As commented, class A is mandatory for all LoRaWAN nodes, and the three classes may coexist in the same network.

3. Implementation of a Virtualized LoRaWAN Network Architecture

This section presents the two options which have been implemented for the virtualization and the automatic orchestration of a LoRaWAN network. As commented, the first proposal is based on microservices using the Kubernetes platform. The second implementation is based on virtual machines, using OpenStack along with its modules for the automatic deployment of the LoRaWAN services.

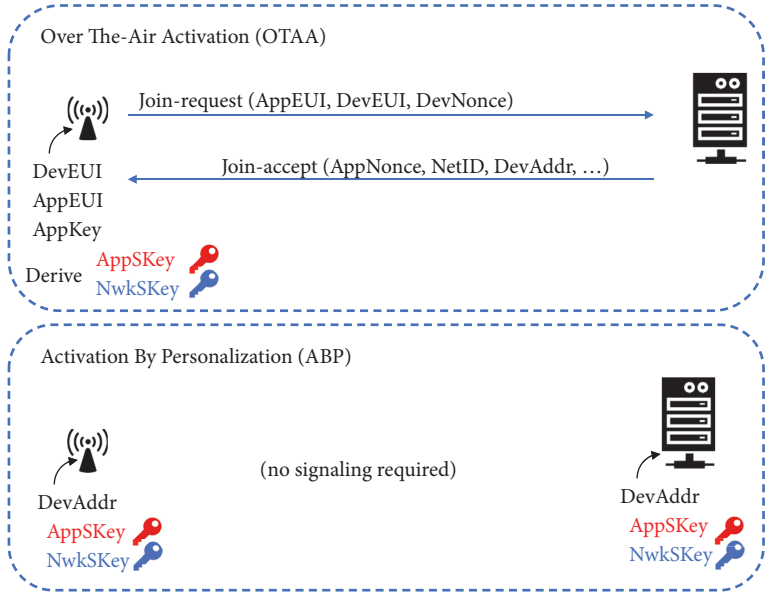

Figure 3: LoRaWAN activation types.

3.1. LoRaWAN NFVs Orchestration Using Microservices. Due to the architecture of a LoRaWAN network and the lightweight functionalities of the different entities, they can be deployed as microservices. With the success of containerization technologies, such as Docker [20], microservices can be realized as containers that result to be extremely fast to start up and can be easily deployed.

In order to implement a LoRaWAN network which supports autorecovery in the case of an emergency situation such as an earthquake, fire, hurricane, or any other situation that may destroy part of the core network infrastructures, the usage of a microservice orchestration platform may suit these requirements due to its efficiency in terms of CPU, memory, and storage consumption compared to virtual machines [21–23].

In particular, we propose to utilize the Kubernetes platform [24] (also named K8S) for the container orchestration. Figure 4 presents the proposed architecture. The details about the hardware and software that implement the 5G-City Kubernetes cluster are described in Section 4.

The LoRaWAN network and application servers are based on the LoRaWAN Server Project [25], an open-source project that provides the components for building LoRaWAN networks. Figure 5 summarizes the interaction between the different dockers and the exposed services to allow external connectivity. The docker images that implement the LoRaWAN network and application servers, along with the required services, have been modified and stored in a personal repository to support the communication with Kubernetes.

3.2. LoRaWAN NFVs Orchestration Using Virtual Machines. In the case of the implementation using virtual machines, due to its popularity, large community, high availability of modules, and being open-source, we have opted for using OpenStack. Our OpenStack testbed is based on the Rocky release and has been installed using the DevStack scripts.

Apart from the primary OpenStack services (Keystone for the identity service, Glance for the image service, Nova for the provision of compute instances or virtual machines, Neutron for network connectivity, and Horizon for the dashboard user interface), Heat and Ceilometer have been installed for the orchestration and the telemetry service. The chosen hypervisor is KVM (Kernel-based Virtual Machine) using QEMU Copy-on-write (qcow2) as the virtual machine image format.

For comparison purposes, the virtual machine images are based on CentOS 7 cloud images, similarly to the Kubernetes deployment. For the same reason, the LoRaWAN network and application servers have also been installed using the LoRaWAN Server Project [25]. Our prototype using OpenStack is depicted in Figure 6.

For our OpenStack testbed, we have developed a module which automatically starts the provision of resources for a new instance, which is then launched. This procedure is triggered once that the original instance is destroyed due to, e.g., a catastrophic event. The module utilizes the API provided by the heat orchestrator and the metrics from ceilometer.

4. Testbed Proposal

This section describes both the hardware and software used for the design of our LoRaWAN network prototype.

4.1. Hardware Setup. The radio access network of our LoRaWAN network prototype is composed of 5 gateways and 12 nodes. The gateways are LiteGateways from iMST [26], which utilize one Raspberry Pi connected to an iMST ic880A LoRaWAN concentrator and 868MHz antenna (see Figure 7).

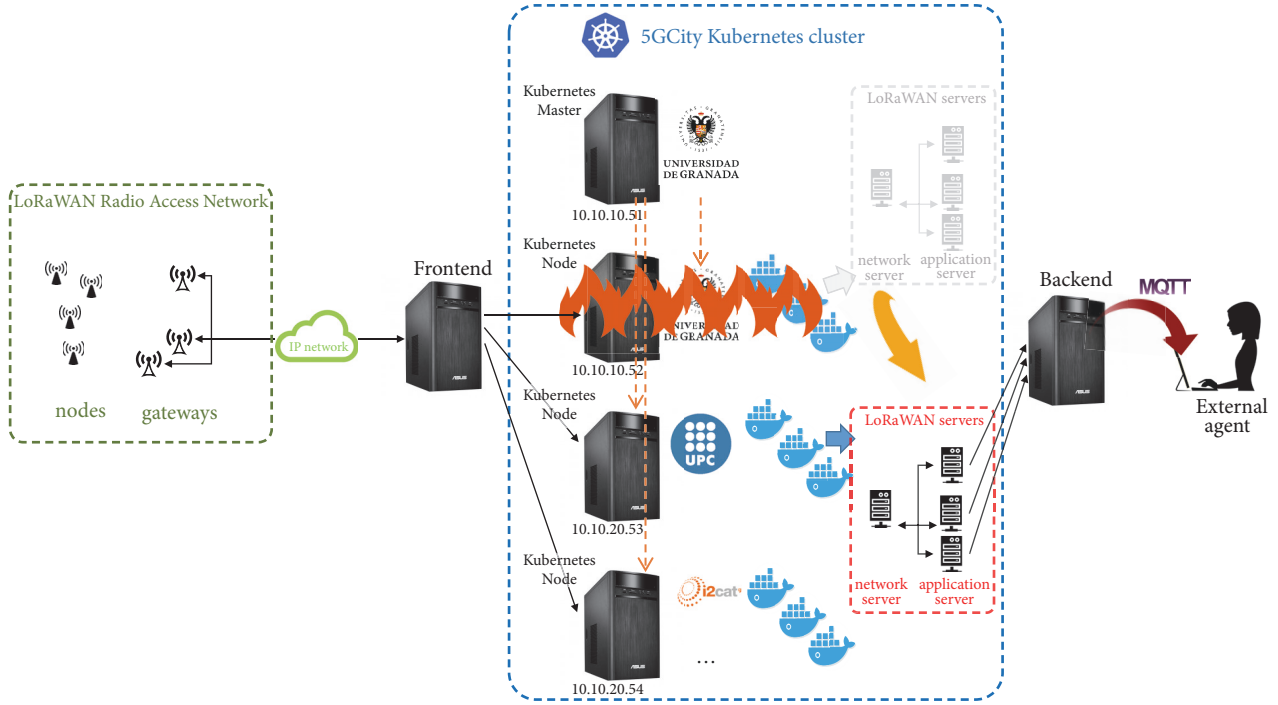

Figure 4: Proposed network architecture based on microservices.

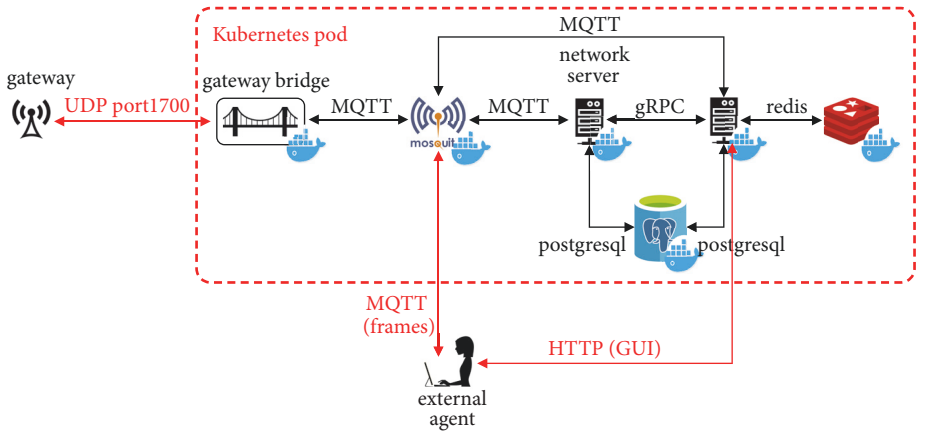

Figure 5: Kubernetes pod (group of colocated containers that are tightly coupled and need to share resources) for LoRaWAN deployment.

The nodes are TTGO-LoRa32 devices (see Figure 8) which are based on the ESP32 microcontroller with a Semtech's SX1276 LoRa transceiver.

Our core network prototype is composed of two servers with an Intel Core i7-7820X CPU (8 cores operating at 3.6 GHz) and 32 GB of RAM located in University of Granada (UGR). These two servers act as the master node of the Kubernetes cluster and the first worker node (minion-1). In addition, we have other two servers located in Barcelona at Universitat Politècnica de Catalunya (UPC) (based on a six-core Intel i7-5820K operating at 3.3 GHz) and Fundació i2CAT (based on a Intel Xeon E312xx (Sandy Bridge) with 32 cores operating at 2.5 GHz), which acts as the second and third worker nodes (minion-2 and minion-3), respectively.

The IP network is a direct Ethernet connection between the gateways and the master server located at University of Granada, which also implements the frontend using one NGINX ingress controller.

4.2. Software Configuration. The nodes are programmed using the Arduino framework, based on the IBM's LMIC library [27]. Before transmitting a LoRaWAN frame, the nodes connect to a server (named *experiment manager*) to ask whether it shall transmit or not, thus allowing us to control the network load. In addition, the transmission parameters are also commanded from the server. These parameters include the spreading factor and the time between frames, which is composed of a constant term and a random term.

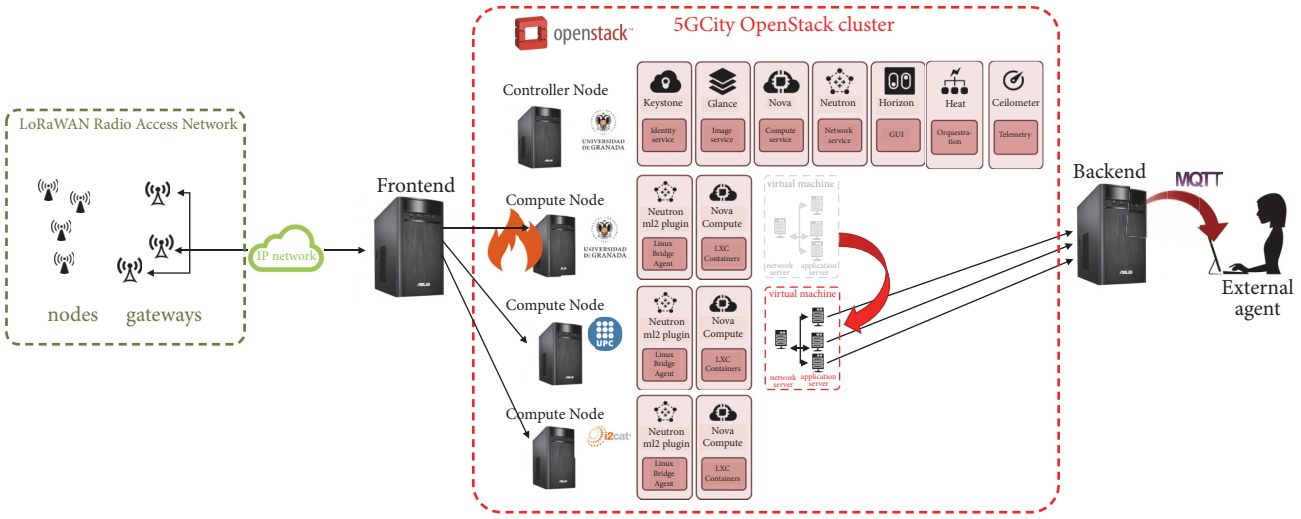

FIGURE 6: Network architecture based on OpenStack and virtual machines.

FIGURE 7: LoRaWAN gateways used in the proposed testbed.

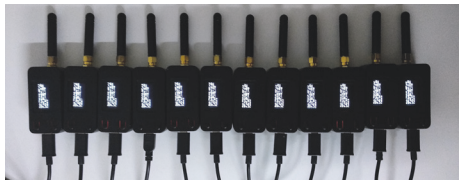

FIGURE 8: LoRaWAN nodes used in the proposed testbed.

FIGURE 9: Detail of a LoRaWAN node.

Figure 9 includes an example of the "TTGO #1" node, which is connected to our server through the Wi-Fi network "5gcity-testbed" with the IP address shown. The last transmission parameters were the usage of spreading factor 7 (SF7) and a time between frames with a fixed part (f) of 10 seconds plus a random part (r) between 0 and 10 seconds. It shall be noted that the ESP32 contains the hardware to generate true random numbers whenever an RF subsystem is running (i.e., Bluetooth or Wi-Fi is enabled) [28].

In order to avoid that the connection between the nodes and the experiment manager may impact the results of the experiments, e.g., due to additional delays, all the nodes connect to the Wi-Fi network only when they are switched on; i.e., there is no connection establishments during the experiments. The mean response time, between the request from the node and the response from the server, has been measured around 85 ms, which is much lower than the time between LoRaWAN frames (which has a minimum value of 6.2 seconds according to Table 1 for SF7 and 125 kHz and has never been lower than 10 seconds in the performed experiments). Besides, the server collects stats from the nodes since it knows when they are going to transmit a frame and with which parameters.

As previously stated, the gateways are based on the Raspberry Pi platform with the iC880A concentrator. The software is based on the reference gateway implementation from TTN-ZH (Zurich community of The Things Network) [29], which has been configured to use our own LoRaWAN network servers. The gateways are also connected to the experiment manager, which collects the logs related to their LoRaWAN activity.

Our Kubernetes cluster is based on Kubernetes version 1.5.2. For portability and reproducibility purposes, both master and worker nodes have been virtualized and executed using VirtualBox version 5.2.22. The host operating system is Ubuntu Server 16.04.05 (64 bits), and the guest operating system is CentOS 7.5.1804 (64 bits) running with 1 core and 2 GB of RAM. We utilize Vagrant version 2.1.5 in order to automatize the virtual machines deployment, and Ansible version 2.6.4 to automatize the installation and configuration of the required packages.

In order to simplify the requirements for connecting the worker and master nodes, a VPN was created using OpenVPN version 2.4.6 using client certificate authentication. In

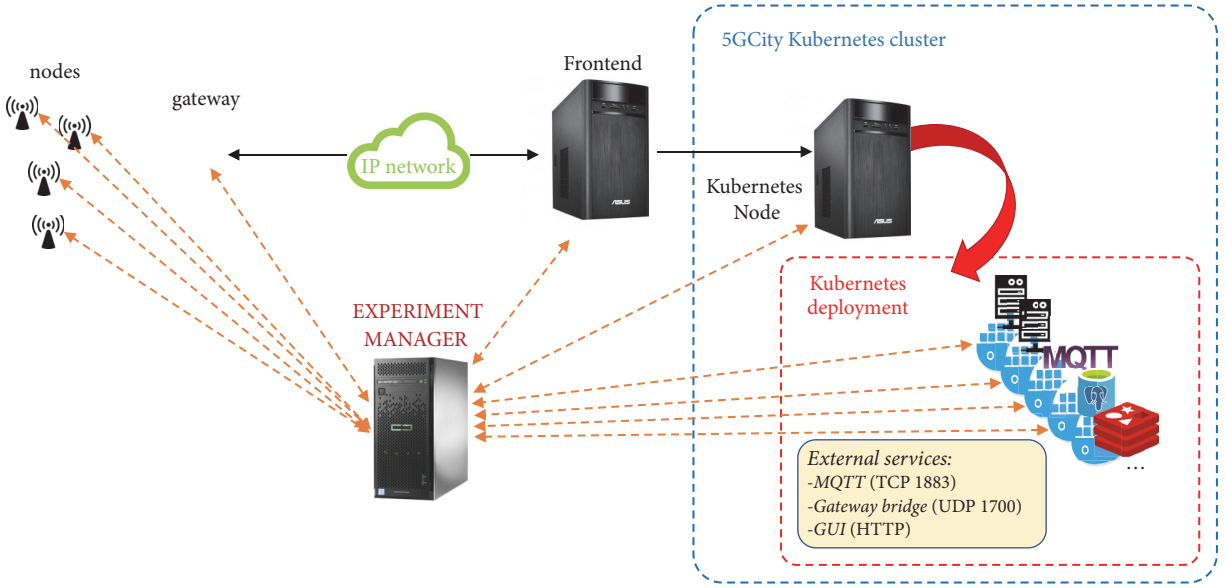

Figure 10: Experimentation testbed.

this way, only the master node is required to have a public IP address. In addition, it eliminates the possibility of issues due to firewall rules. In our VPN, the master node acts as the OpenVPN server whereas the worker nodes act as OpenVPN clients. The workers also collect tcpdump traces on the TCP/UDP ports that are used by services related to the LoRaWAN deployment, which are later sent to the experiment manager.

The dockers that implement the LoRaWAN deployment are also connected to the experiment manager, which starts/stops the services depending on the experiment and collects the required logs and stats.

The experiment manager connects to the different entities using SSH connections, which enable us to execute commands (e.g., to start or to stop a particular service), to upload files (e.g., a configuration file), or to download files (e.g., logs, tcpdump traces, or stats). In the case of the LoRaWAN nodes, they connect to the experiment manager after transmitting one LoRaWAN frame to request the transmission parameters for the next frame. If the connection manager commands the node not to transmit, it will ask again after 10 seconds. Figure 10 shows a simplified view of the testbed for experimentation.

The NGINX ingress controller has been configured with the default values for load balancing the different external services, i.e., the UDP port 1700 (which is used by the lorawan-gateway-bridge container), the TCP port 1883 (which is used by the MQTT broker), and the TCP port 443 (which is used for the HTTPS-based GUI). The main parameters are *max_fails*=3 and *fail_timeout*=30s, meaning that the backup server will be used after the main server has failed to respond to at least 3 packets in a period of 30 seconds.

Based on this experiment manager, we have developed a framework based on scripting to generate different scenarios for both the radio access network and the core network and to automatically collect statistics, which will be used in the next section for the experimental evaluation.

5. Use Cases and Experimental Results

Considering the two options that we have followed for the virtualization of the LoRaWAN network entities, i.e., using microservices (Kubernetes) and virtual machines (OpenStack), the following use cases have been tested:

(i) *UC1*: Kubernetes deployment with one replica and default parameters: the chosen version of Kubernetes considers a deployment (i.e. a set of pods and services available externally) to be unavailable after five minutes. Thus, a new replica will be deployed after this time.

(ii) *UC2*: Kubernetes deployment with one replica and an eviction pod timeout of 30 seconds. Instead of waiting the default 300 seconds, this use case attempts to react faster to possible unavailability of worker nodes due to a catastrophic situation. We did not select a lower timeout value (e.g. 3 seconds) in order to avoid new replica deployments due to temporary network fluctuations.

(iii) *UC3*: Kubernetes deployment with two replicas: in this case, the replica at UGR is chosen initially, and the replica available at UPC/i2CAT will be used for backup.

(iv) *UC4*: OpenStack deployment with one replica and default parameters: similar to the first use case but using the OpenStack platform.

The reason of selecting these four use cases is twofold. First, testing and comparing different configurations using

microservices have been proved to be suitable for the deployment of LoRaWAN servers. For that purpose, we want to compare the usage of Kubernetes with default values (UC1, with a timeout of 300 seconds), with two replicas in order to achieve a solution (almost) without service interruption (UC3) and an intermediate situation (UC2). Second reason is to compare both Kubernetes (with containers, UC1) and OpenStack (with virtual machines, UC4) with their default configurations.

Since we want to simulate a high-loaded IoT scenario, nodes will transmit 12-byte frames using SF7 and 125 kHz, leading to a minimum time between frames of 6.2 seconds due to the duty cycle (see Table 1). Since we want transmissions to be uncorrelated, the time between frames are composed of a fixed part, f=10 seconds, and a random part, r, which follows a random uniform distribution between 0 and 10 seconds. This also avoids the problem of collisions due to the simultaneous powering on of the nodes.

With these values, the average time between frames is 15 seconds, i.e., leading to a load of $12 nodes/(15 sec/frame/node) = 0.8$ frames/sec, which is similar to approximately 1000 nodes transmitting one frame every 20 minutes. These frames will be received by only one gateway, meaning that it will be high-loaded since other works (e.g., [30]) suggest that the maximum load that a LoRaWAN gateway can support without frame losses is around 0.1 frames/sec. This is the maximum load that can be generated with the real equipment in our testbed. It is left for future work to include a load testing tool which would allow us to evaluate the performance of our testbed under stress conditions (e.g., in [31], the authors emulate the load generated from 14,000 nodes).

In the proposed use cases, the main performance indicators are the recovery time, i.e., the time that elapses from the failure in one worker node until another worker node executes the pod with the LoRaWAN deployment, and the lost frames during that recovery. It shall be noted that the lost frames will depend on the gateway load, and the results shown in this section are given for the aforementioned load of 0.8 frames/sec. Additionally, we also want to show the different requirements, in terms of CPU and memory, between the usage of a Kubernetes cloud or OpenStack.

Figures 11 and 12 depict the main performance indicators related to the recovery of the LoRaWAN core network after an equipment failure due to, e.g., a catastrophic situation. All the use cases (UC1 to UC4) are included. These *box-and-whisker* charts include a box bounded by the first and third quartile and lines that extend to the minimum and maximum, respectively. The median is also included as the line that divides the box.

As shown, UC1 takes between 5 and 6 minutes (with an average of 321 seconds and an standard deviation of 13.6) to recover due to the default value of the pod eviction timeout (300 seconds). In the case of UC2, we have reduced this timeout to 30 seconds, leading to a recovery time of around one minute (average of 64 seconds with an standard deviation of 14.7). To conclude with the Kubernetes-based use cases, UC3 has an almost negligible recovery time. This is because two replicas are already executed, and the second

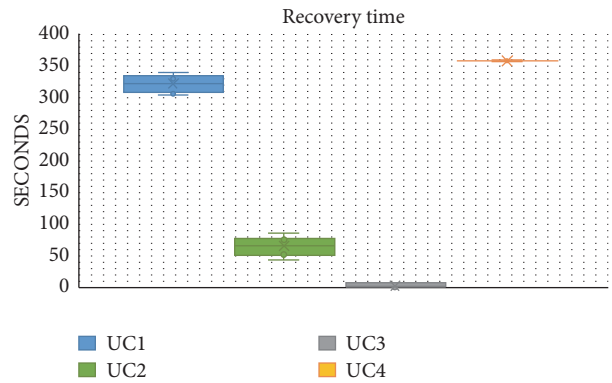

Figure 11: Recovery time after server failure.

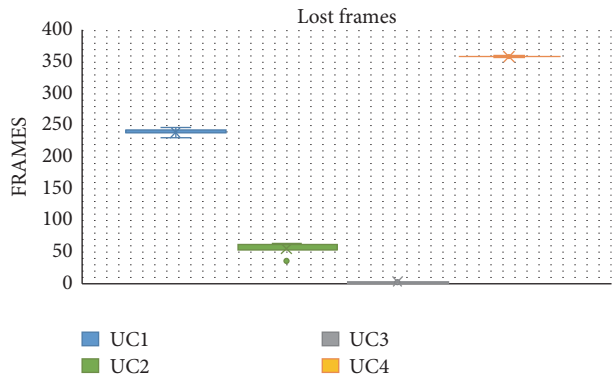

Figure 12: Frames lost during recovery.

one takes over when the first one fails. Since we utilized the default values for the NGINX frontend, only three packets are required to change from one replica to another. The rate of these packets depends on the transmissions from the LoRaWAN nodes and some periodic packets, being the recovery time of 4 seconds with a standard deviation of 2.4. As it was expected, the number of lost frames is approximately proportional to the recovery time.

In the OpenStack use case (UC4), the developed module waits 5 minutes (like the default timeout value for Kubernetes) before provisioning and launching the new instance, leading to a total recovery time of around six minutes (with an average of 358.1 seconds and a standard deviation of 0.72). As in the previous use cases, the number of lost frames is almost proportional to the recovery time.

Next, we compare the usage of resources of both options. For Kubernetes, we employed cAdvisor [32], a tool that provides the resource usage and performance characteristics of running containers. The resources used by the different containers that compose the Kubernetes deployment for LoRaWAN under a load of 0.8 packets/second are summarized in Table 2.

The results from Table 2 show that the CPU usage is almost negligible (lower than 0.01%) and the total memory usage of the LoRaWAN deployment is 39 MiB.

Table 2: Usage of resources for the containers of the LoRaWAN deployment.

Container	CPU %	MEM %	MEM (MiB)
lora-app-server	0.00	0.60	11.3
loraserver	0.00	0.40	7.6
lora-gateway-bridge	0.00	0.20	4.0
mosquitto	0.00	0.00	1.5
postgres	0.00	0.70	13.0
redis	0.00	0.00	1.6

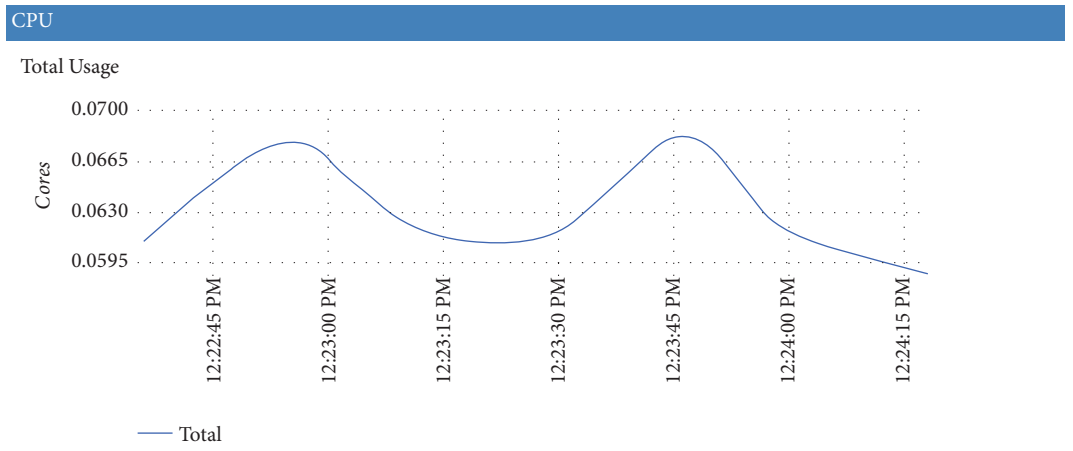

Figure 13: Total CPU usage including all Kubernetes processes.

Figure 13 presents the total CPU usage for one of the worker nodes of the Kubernetes cluster. As shown, Kubernetes (and docker in general) utilizes very few resources in terms of CPU, between 6% and 7% of one core. The main consumers are processes related to the management of the Kubernetes cloud (kubelet with 2.8%, dockerd-current with 0.5% and kube-proxy with 0.4%).

In terms of memory, around 1.6 GB are used by the worker node. The processes that reserve more memory are also related to Kubernetes (java with 392 MB, kubelet with 75.9 MB, dockerd-current with 51.5 MB, kube-proxy with 39.9 MB, and flanneld with 26.8 MB).

In the case of OpenStack, the total memory employed by the worker node is 8.45 GiB when no instances are deployed and 288 MiB more when one virtual machine with the LoRaWAN is executed. In terms of CPU, worker node consumes 1.28 CPU cores. This means that, in our given scenario, OpenStack requires more than 5 times the memory needed by Kubernetes and more than 18 times in terms of CPU consumption.

6. Conclusions

In this paper, we propose the usage of a microservices platform such as Kubernetes for the deployment of a LoRaWAN network infrastructure. Based on its orchestration capabilities, the proposed framework is able to support catastrophic situations and to rapidly recover from equipment failure in the core network. To evaluate the performance of this solution, a prototype testbed of a complete LoRaWAN network has been implemented. By using an experiment manager, we have been able to automatize the node traffic generation and the automatic recollection of stats, as well as the presence of failures. We have evaluated our implementation in terms of time to recover, lost frames, and resource usage. After the conducted evaluation, we claim that the usage of several replicas of the LoRaWAN core network entities and a load balancer, which automatically changes between servers in a fast and efficient way, produces an almost seamless recovery, what makes it a proper solution to recover after a system crash caused by any catastrophic event.

For future work, based on our previous analysis [33, 34], we plan to mathematically model the implemented VNFs in order to estimate the performance for a given configuration and to derive when to scale out by requesting more resources.

Data Availability

The data has been generated from live tests in our LoRaWAN testbed. Logs or any other information is available upon request to the authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is partially supported by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund (Projects TEC2016-76795-C6 and EQC2018-004988-P).

References

[1] K. L. Lueth, "State of the IoT 2018: Number of IoT devices now at 7B –market accelerating, 2018," https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/.

[2] A. Nordrum, "Popular Internet of Things forecast of 50 billion devices by 2020 is outdated," *IEEE Spectrum's General Technology Blog*, 2016, http://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated.

[3] C. Cervello-Pastor, "5G-City: Flexible management of 5G services oriented to support urban critical situations, project TEC2016-76795-C6 of the Spanish Ministry of Economy, Industry and Competitiveness, 2017-2019".

[4] J. Navarro-Ortiz, S. Sendra, P. Ameigeiras, and J. M. Lopez-Soler, "Integration of LoRaWAN and 4G/5G for the Industrial Internet of Things," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 60–67, 2018.

[5] X. Xiong, K. Zheng, R. Xu, W. Xiang, and P. Chatzimisios, "Low power wide area machine-to-machine networks: Key techniques and prototype," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 64–71, 2015.

[6] A. Cenedese, A. Zanella, L. Vangelista, and M. Zorzi, "Padova smart city: an urban internet of things experimentation," in *Proceedings of the 15th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM '14)*, pp. 1–6, Sydney, Australia, June 2014.

[7] T. Petrić, M. Goessens, L. Nuaymi, L. Toutain, and A. Pelov, "Measurements, performance and analysis of LoRa FABIAN, a real-world implementation of LPWAN," in *Proceedings of the 27th IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communications, PIMRC 2016*, pp. 1–7, Spain, September 2016.

[8] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, "Long-range communications in unlicensed bands: The rising stars in the IoT and smart city scenarios," *IEEE Wireless Communications Magazine*, vol. 23, no. 5, pp. 60–67, 2016.

[9] J. Petäjäjärvi, K. Mikhaylov, A. Roivainen, T. Hänninen, and M. Pettissalo, "On the coverage of LPWANs: Range evaluation and channel attenuation model for LoRa technology," in *Proceedings of the 14th International Conference on ITS Telecommunications, ITST 2015*, pp. 55–59, Denmark, December 2015.

[10] M. Aref and A. Sikora, "Free space range measurements with Semtech Lora™; technology," in *Proceedings of the 2014 2nd International Symposium on Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS-SWS)*, pp. 19–23, Odessa, Ukraine, September 2014.

[11] T. Wendt, F. Volk, and E. Mackensen, "A benchmark survey of long range (LoRaTM) spread-spectrum-communication at 2.45 GHz for safety applications," in *Proceedings of the 2015 16th IEEE Annual Wireless and Microwave Technology Conference, WAMICON 2015*, pp. 1–4, USA, April 2015.

[12] A. J. Wixted, P. Kinnaird, H. Larijani, A. Tait, A. Ahmadinia, and N. Strachan, "Evaluation of LoRa and LoRaWAN for wireless sensor networks," in *Proceedings of the 15th IEEE Sensors Conference, SENSORS 2016*, pp. 1–3, USA, November 2016.

[13] P. J. Radcliffe, K. G. Chavez, P. Beckett, J. Spangaro, and C. Jakob, "Usability of LoRaWAN technology in a central business district," in *Proceedings of the 2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, pp. 1–5, Sydney, Australia, June 2017.

[14] J. M. Marais, R. Malekian, and A. M. Abu-Mahfouz, "LoRa and LoRaWAN testbeds: A review," in *Proceedings of the IEEE AFRICON 2017*, pp. 1496–1501, South Africa, September 2017.

[15] N. Vatcharatiansakul, P. Tuwanut, and C. Pornavalai, "Experimental performance evaluation of LoRaWAN: A case study in Bangkok," in *Proceedings of the 2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, S. T. P. Chantamunee and S. Doung-in, Eds., pp. 1–4, Institute of Electrical and Electronics Engineers Inc., NakhonSiThammarat, Thailand, July 2017.

[16] A. Augustin, J. Yi, T. Clausen, and W. M. Townsley, "A Study of LoRa: long range & low power networks for the internet of things," *Sensors*, vol. 16, no. 9, article 1466, 2016.

[17] LoRaWAN™ 1.1 specification, "LoRa Alliance, Specification, Oct. 2017," https://lora-alliance.org/sites/default/files/2018-04/lorawantm_specification_-v1.1.pdf.

[18] LoRa™ modulation basics, "Semtech Corporation, Application Note AN1200.22, May 2015," https://www.semtech.com/uploads/documents/an1200.22.pdf.

[19] LoRaWAN™ 1.1 regional parameters, "LoRa Alliance, Specification, Jan. 2018," https://lora-alliance.org/sites/default/files/2018-04/lorawantm_regional_parameters_v1.1rb_-_final.pdf.

[20] Docker, "Docker, enterprise-grade container platform," https://www.docker.com.

[21] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, "A comparative study of containers and virtual machines in big data environment," in *Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 178–185, San Francisco, CA, USA, July 2018.

[22] M. Chae, H. Lee, and K. Lee, "A performance comparison of linux containers and virtual machines using Docker and KVM," *Cluster Computing*, 2017.

[23] A. M. Joy, "Performance comparison between Linux containers and virtual machines," in *Proceedings of the 2nd International Conference on Advances in Computer Engineering and Applications, ICACEA 2015*, pp. 342–346, India, March 2015.

[24] Google, "cadvisor (container advisor)," https://github.com/google/cadvisor.

[25] O. Brocaar, "LoRa server v2.3.0," https://www.loraserver.io/, 2018.

[26] IMST, "Lite gateway," https://shop.imst.de/wireless-modules/lora-products/36/lite-gateway-demonstration-platform-for-lora-technology.

[27] M. Kooijman, "IBM's LoRaMAC-in-C," https://github.com/matthijskooijman/arduino-lmic.

[28] "ESP32's random number generation," https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/system/system.html#_CPPv210esp_randomv.

[29] "TTN-ZH's iC880a-gateway," https://github.com/ttn-zh/ic880a-gateway/wiki.

[30] D. Bankov, E. Khorov, and A. Lyakhov, "On the limits of LoRaWAN Channel Access," in *Proceedings of the 2016 International Conference on Engineering and Telecommunication (EnT)*, pp. 10–14, 2016.

[31] Q. Zhou, K. Zheng, L. Hou, J. Xing, and R. Xu, "X-LoRa: An Open Source LPWA Network," http://arxiv.org/abs/1812.09012, 2019.

[32] Cloud Native Computing Foundation, "Kubernetes, production-grade container orchestration," https://kubernetes.io/.

[33] J. Prados-Garzon, P. Ameigeiras, J. J. Ramos-Munoz, P. Andres-Maldonado, and J. M. Lopez-Soler, "Analytical modeling for Virtualized Network Functions," in *Proceedings of the 2017 IEEE International Conference on Communications Workshops, ICC Workshops 2017*, pp. 979–985, France, May 2017.

[34] J. Prados-Garzon, J. J. Ramos-Munoz, P. Ameigeiras, P. Andres-Maldonado, and J. M. Lopez-Soler, "Modeling and Dimensioning of a virtualized MME for 5G mobile networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 5, pp. 4383–4395, 2017.