On Extending ETSI MEC to Support LoRa for Efficient IoT Application Deployment at the Edge

Adlen Ksentini and Pantelis A. Frangoudis

Abstract

The Internet of Things (IoT) undergoes a rapid transformation this last decade, thanks to the appearance of low-power wide area network technologies, such as LoRa/LoRaWAN, SigFox, and narrowband IoT, which allow reducing the deployment cost of sensors and other IoT devices. Many emerging services such as smart city, Industry 4.0, and autonomous driving are based on IoT devices and applications to collect and analyze data and control end devices (i.e., actuators). Among these services, several IoT applications, such as data analytics, need to be deployed at the edge to either reduce the latency to access data or treat the high amount of generated data locally. However, in the context of LoRa/LoRaWAN, most of the current IoT service deployments run the applications at a central cloud to ease the integration with existing software as a service (SaaS) platforms, without exploiting the benefits of edge computing. In this article, we propose a new framework that leverages the ETSI multi-access edge computing (MEC) model to deploy LoRa-based IoT applications at the edge. In particular, the proposed model takes advantage of the ETSI MEC features, such as dynamic deployment of an IoT application at the edge and application life cycle management. In addition, the proposed framework allows running an IoT application as a 5G network slice at the edge.

Introduction

Edge computing is enabling a new generation of services that operate close to end users, aimed at reducing end-to-end latency. Edge computing allows the deployment of two types of services:

- Services that require low-latency access to user plane traffic
- Context-aware services that adapt the delivered service according to the users' environment

Two main technological solutions have emerged to make edge computing a reality: fog computing and multi-access edge computing (MEC). The former is an architecture that adds computing capabilities to edge routers and end-user devices, while the latter is an operator-oriented architecture. Indeed, MEC adds computing capability in the vicinity of radio base stations, and proposes an orchestration as well as a management framework to handle the life cycle management (LCM) of edge applications. Fog computing is under development, where the OpenFog Alliance[1] has recently issued a reference architecture for fog computing,[2] while the European Telecommunications Standards Institute (ETSI) is providing specifications to MEC via the MEC Industry Specification Group (ISG).[3] The latter has published several documents aimed at describing envisioned use cases, a reference MEC architecture, a MEC application model (descriptor), MEC services, a MEC orchestration and management framework, and so on. Besides enabling applications at the edge, MEC provides services, accessible via high-level application programming interfaces (APIs), which give information on the mobile user and cellular base station context, such as the radio channel quality of users and user locations. This allows building context-aware applications.

At the same time, deploying Internet of Things (IoT) applications at the edge, such as data analytics, Industry 4.0, and time-critical IoT services, represents one of the main use cases being showcased by ETSI MEC, but is also highly relevant in the fog computing space [1, 2]. Besides enabling low-latency access, placing IoT services at the edge enables reducing the amount of IoT traffic to carry to the remote server located at the central cloud, which could significantly strain the core network infrastructure given the massive amount of traffic that can be generated by IoT devices.

Regarding the connectivity of the IoT devices to the network, many technologies are in competition; on one hand, narrowband technologies such as narrowband IoT (NB-IoT), LTE-Mobile (LTE-M), and SigFox; and on the other hand, low-energy and long-range technologies such as LoRa/LoRaWAN.[4] The latter is gaining momentum due to its low complexity and reduced cost to deploy. Indeed, LoRa/LoRaWAN has many interesting features:

- It uses free industrial, scientific, and medical (ISM) bands.
- It relies on an ALOHA-based medium access control (MAC) protocol, which reduces the complexity of the devices compared to other contention-based mechanisms, such as carrier sense/collision avoidance multiple access (CSMA/CAMA).
- It uses a novel physical layer (based on spread spectrum) that allows increasing range and reducing energy consumption, hence reducing the deployment cost.

Adlen Ksentini is with EURECOM; Pantelis A. Frangoudis is with TU Wien.

[1] The OpenFog consortium merged with the Industrial Internet Consortium on January 31, 2019.

[2] https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf

[3] https://www.etsi.org/technologies/multi-access-edge-computing

[4] LoRa refers to the patented physical layer, while LoRaWAN represents the higher layers of the system and is specified by the LoRa Alliance (https://www.lora-alliance.org/). Throughout this article, we use the terms LoRa and LoRaWAN interchangeably to designate the same technology.

Digital Object Identifier:
10.1109/MCOMSTD.001.1900051

2471-2825/20/$25.00 © 2020 IEEE

Regarding the connectivity of the IoT devices to the network, many technologies are in competition: on one hand, narrowband-based technologies such as NB-IoT, LTE-M, and SigFox; and on the other hand, low-energy and long-range technologies such as LoRa/LoRaWAN. The latter is gaining momentum due to its low complexity and reduced cost to deploy. Indeed, LoRa/LoRaWAN has many interesting features.

To ensure reliable communication, LoRaWAN gateways are deployed (covering a large area) to forward any received packet from the LoRaWAN devices to a LoRaWAN network server, which will be in charge of deleting duplicate packets and forwarding packets to an application server hosted in the cloud. Existing LoRaWAN deployments are driven by either commercial operators, which own the LoRaWAN infrastructure (gateways and LoRaWAN servers) and offer cloud services to recover data from the devices provided by the consumers, or initiatives like The Things Network (TTN),[5] where consumers provide and connect their own LoRaWAN gateways and devices, and use the TTN servers to process data in the cloud. Although the LoRaWAN network of devices and gateways is distributed, all the data are carried from devices to the application servers in the cloud in both configurations. While these deployments are suitable for time-tolerant IoT services, they cannot satisfy time-sensitive or context-oriented IoT services that need to be deployed at the edge. Furthermore, in 5G, it is expected that the IoT provider (or vertical) deploys its LoRaWAN-based service as a network slice [3, 4] at the edge, taking advantage of the dynamicity and flexibility provided by network slicing. Examples include an IoT-based metering report service that may require a temporary deployment, or a smart city service where the city may use the network operator's local infrastructure (edge cloud and LoRaWAN gateways and servers) to reduce setup and operational costs compared to a full deployment, as is typically the case today. Besides, the city can manage the generated data locally, without the need to carry them to remote cloud servers, hence reducing security threats and complying with citizen privacy requirements.

In this article, we devise a novel framework that enables executing IoT services in a MEC-ready environment, taking advantage of features provided by the ETSI MEC model, including dynamic deployment of MEC applications, orchestration and management of MEC IoT applications, standard interfaces to expose and consume native and third-party MEC services, and so forth. Our main objective is to extend the current MEC model to include LoRaWAN networks. Indeed, the current version of ETSI MEC is built on top of 3GPP mobile networks, such as LTE and 5G, ignoring other types of wireless networks. The contributions of this work are the following.

- Extension of the southbound API of MEC to communicate with LoRaWAN gateways
- Specification of a northbound API that allows IoT applications to interact with a LoRaWAN (devices, gateways, and network server)
- Integration of a LoRaWAN server as a MEC service
- Use of the MEC Orchestrator to dynamically deploy IoT applications relying on LoRaWAN, in the form of a network slice, at the edge
- Ensuring the isolation of the network slices at the edge
- Development of a proof of concept of the MEC LoRa framework

Our results can be of interest to the ETSI MEC standarization community, particularly regarding its ongoing (Phase 2) activities related with non-3rd Generation Partnership Project (3GPP) access and the MEC IoT API (ETSI GS MEC 033). This article is structured as follows. We provide an overview of the LoRaWAN architecture and deployment options, with a focus on the role of edge computing, as well as an overview of ETSI MEC. Our key contributions around integrating LoRaWAN with ETSI MEC are presented in detail before we conclude the article.

Related Work

LoRaWAN and Edge Deployment

The key elements in a LoRaWAN architecture are the gateways and the network server, whereas devices are provided by a LoRaWAN device provider. An application server is the external service subscribed/deployed by the application/device provider to recover application data sent from the devices and format the data by using a high-level API. Usually, devices and the back-end applications belong to the IoT service provider (i.e., vertical). Several commercial application servers using central clouds exist in the market, such as Azure IoT[6] and IBM Watson.[7] In LoRaWAN, the devices implement a simple MAC protocol (ALOHA) to access the channel. Each time a device has data to send, it uses the available channel and broadcasts the message. The LoRa device is not associated with a specific gateway. Rather, each gateway receiving the signal (packet) forwards it to its associated network server, which will delete the duplicate messages and send one message to the application server located in the cloud, using HTTP (REST) or publish/subscribe protocols such as Message Queuing Telemetry Transport (MQTT). The network server is also in charge of sending messages from the IoT applications to the devices by selecting the appropriate gateway, for example, to update the devices' firmware.

So far, two types of deployments are observed:

- Operator-based deployment, where a network operator deploys a LoRaWAN network (mainly gateways and a network server), while the IoT application provider deploys its devices and uses the operator API to connect to an end application server hosted in the cloud
- Community-based solutions, such as TTN, which provide a network server, while the application provider deploys both gateways and IoT devices, and connects the gateways to the network server via specific APIs.

In addition, the IoT service provider needs to connect its application to the TTN network server to obtain the data from the devices. Accordingly, deploying LoRa-based IoT services at the edge, close to the devices and network server, is not well supported and studied in the literature, even though such deployments are needed. To the best of our knowledge, only a few works have considered the deployment of a LoRaWAN service at the edge. In the work of Truong [5], the edge is used only for computing, while in this work we rely on and extend the well-defined ETSI MEC features to dynamically deploy an IoT network slice at the edge, and ease the deployment of IoT analytics and low-latency services close to the end devices using LoRaWAN. On the other hand, Sanchez-Iborra *et al.* [6] propose to deploy LoRa

[5] https://www.thethingsnetwork.org/

[6] https://azure.microsoft.com

[7] https://www.ibm.com/cloud/watson-iot-platform/

RAN management, IPv6 header compression, storage offloading, and context-aware services for device control at the mobile edge in a converged MEC-LoRaWAN environment. Although they cast their work in a MEC setting, convergence is not discussed in the context of the ETSI MEC reference architecture and the use of its standard interfaces (a major focus of our work).

ETSI MEC

Since its creation in 2013, the ETSI MEC ISG has been working on the development of standardization activities around MEC. The first released document covers the reference architecture [7]. A high-level representation of this architecture is shown in Fig. 1.

It introduces three main entities:

- The MEC host, which is the key element. It provides the virtualization environment to run MEC applications, while it interacts with the mobile network entities via the MEC platform (MEP) to provide MEC services and data offload to MEC applications. Generally, a MEC host corresponds to the edge computing platform running a virtualized infrastructure manager (VIM) to handle the instantiation of MEC applications as virtual machines (VMs) or containers.
- The MEC platform (MEP), which acts as an interface between the mobile network and the MEC applications. It provides the Mp1 interface for MEC applications to expose and consume MEC services, and interacts with the mobile network over the Mp2 reference point. The latter is used to obtain statistics from the RAN on user equipments (UEs) and evolved NodeBs (eNBs), e.g., in order to provide the radio network information service (RNIS) [8] and the location service [9], and to appropriately steer user-plane traffic to MEC applications.
- MEC applications that run on top of a virtualized platform.

Another concept introduced by the MEC ETSI group is the *MEC service*, which is either a service provided natively by the MEC platform, such as the RNIS and traffic control, or a service provided by a MEC application (e.g., video transcoding). A MEC service provided by a third-party application should be registered with the MEP and made available via the Mp1 interface. The service may be discovered by other MEC applications once registered by the MEP. This implies that in order to take advantage of MEC features, third-party applications need to be designed with awareness of MEC platform functions and APIs [10], including the ability to discover the MEP (e.g., via DNS) and access its service endpoints.

Regarding the management plane, the ETSI MEC group introduced the Mobile Edge Orchestrator (MEO), which is in charge of the life cycle of MEC applications (instantiation, orchestration, and management), and acts as the interface between the MEC host and the operations support system (OSS)/business support system (BSS) [11]. Several interfaces (noted as Mmx) have been specified for the management plane of MEC. The Mm1 interface is used to communicate with the OSS/BSS, allowing the latter to request the deployment

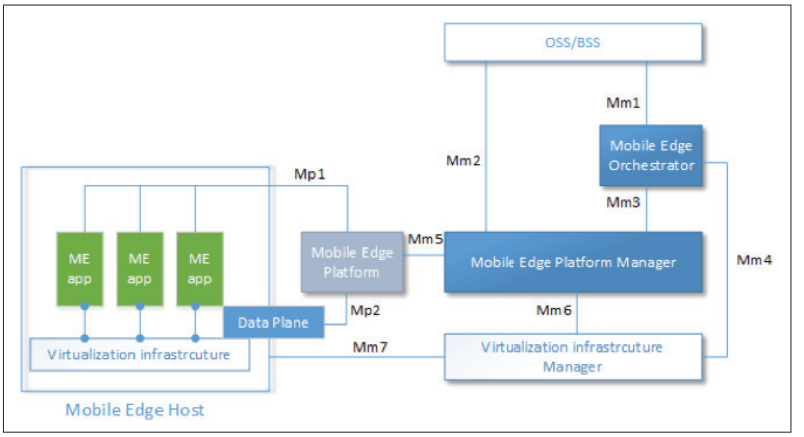

FIGURE 1. High-level representation of the MEC architecture (based on [7]).

of a MEC application (e.g., an IoT application), which is already available in the application catalogue or is first onboarded by the OSS/BSS. The MEO uses Mm4 to upload the image of the application to the edge VIM, which is in charge of instantiating the application at the MEC host. The MEP Manager (MEPM) element is in charge of the life cycle management of the deployed MEC applications, and the configuration of the MEC platform via the Mm5 reference point, including MEC application authorization, the type of the traffic that needs to be offloaded to a MEC application, DNS management, and so on. The MEPM uses the Mm3 interface to configure the MEC application and communicate with the VIM, located at the MEC host, to obtain information on the virtual resources used by a MEC application. This information is used by the MEO to check the MEC application resource status and, if deemed appropriate, to decide if more resources are needed for the MEC applications. This information is also exposed to the OSS/BSS through the Mm2 interface.

MEC has been envisioned to host a plethora of IoT services, and it is typically assumed to interact only with 3GPP-based mobile networks. In this environment, Husain *et al.* [12] propose a design for integrating IoT service layer components in network slices, with the possibility of deploying such components at the mobile edge when the service requirements (e.g., in terms of latency) mandate it. As mentioned earlier, though, many IoT deployments are using other types of networks, such as LoRaWAN and Sigfox. The requirements and necessary extensions for ETSI MEC to support emerging IoT services have not yet been well studied. We are only aware of the work of Zanzi *et al.* [13], who introduce a MEC IoT platform as a new element in the ETSI MEC architecture. This builds on the concept of virtualized IoT gateways, where upper-layer gateway functionality is executed within the MEC IoT platform. However, issues pertinent to the integration of different access technologies, including southbound interfaces (Mp2), and technical details on how MEC functional blocks and interfaces are involved in IoT service deployment are not discussed. In the following section, we introduce our new framework

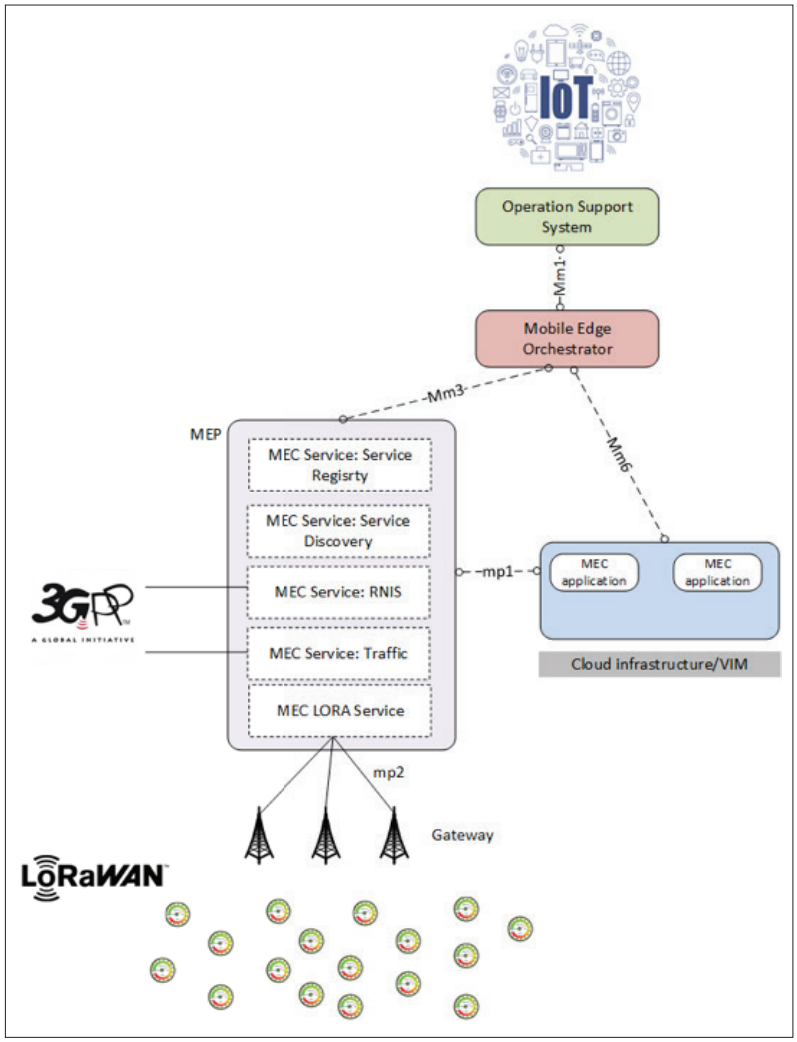

FIGURE 2. LoRaWAN integrated in the ETSI MEC architecture.

that enables the deployment of IoT services in MEC, using a LoRaWAN network.

MEC LoRa Service

The current version of ETSI MEC does not support interaction with LoRaWAN networks, even though there is a need to deploy IoT applications at the edge in a dynamic way, and building on the features provided by the ETSI MEC framework, such as LCM and low-latency access to the user data plane. To enable MEC supporting LoRaWAN-based IoT services at the edge, we introduce a new MEC service, namely the *MEC LoRa service*. Like other MEC services, such as the RNIS, traffic control, and DNS, MEC LoRa is included in the MEP's service catalogue and announced to the MEC applications via the MEC discovery service. The MEC LoRa service will be provided by the network operator within its MEP. In addition, the MEC LoRa service can cover the following two scenarios:

- The gateway, IoT devices, and IoT MEC applications are provided and managed by the vertical (or IoT service owner, e.g., a city). In this case, the MEC LoRa service should expose, via the Mp1 interface, the necessary APIs to register the gateways to the LoRaWAN network server and the IoT devices as well as the IoT applications.
- The IoT devices as well as the IoT applications are provided and managed by the vertical. In this case, the vertical uses the LoRaWAN gateways deployed by the network operator. Regarding the Mp1 interface, no changes from the precedent case are envisioned, except for the API related to the registration of the LoRaWAN gateways, which needs to be associated with the LoRa network server.

It is worth recalling that the LoRaWAN network server provides three key functions: authentication and authorization of devices, management and optimization of the network, and interfacing with the IoT application. It receives messages from the IoT devices via the LoRaWAN gateways, manages device authentication, routes packets to the application, and manages gateways and devices. Furthermore, the LoRaWAN network server dynamically selects the best gateway for device data routing (in the case of packets from the application to the devices), as well as deduplicating packets and optimizing radio resource use. To do so, in this framework, we leverage the Mp1 interface by providing the necessary APIs that allow:

- Registering the IoT application and devices involved in the communication
- Using the callback mechanism to redirect the data obtained from devices to the application
- Sending messages to the devices

Figure 2 shows the envisioned MEC system featuring the MEC LoRa network service. In the proposed framework, the MEP has a new service (i.e., the MEC LoRa service), which is connected to the LoRaWAN gateways. As indicated, the proposed framework supports two deployment scenarios:

- The gateways are owned by the network operator.
- The gateways are owned by the vertical and are connected to the MEC LoRa service.

For both scenarios, we assume that the gateways cover a specific region or area. The Mp2 interface, which is needed to connect the MEP to the gateways, is based on IP (no other specific protocol is need). Usually, LoRa gateways use a packet forwarder layer that has one interface connected to the LoRa network (radio interface) and one interface to the IP network, and hence to the LoRa network server.

It is worth mentioning that in MEC, several MEPs are deployed to cover a specific geographic location. The number of MEPs to deploy depends on the type of region covered (e.g., rural or urban). For the former, we can assume that a MEP covers a wide region, while in the latter, a MEP covers a small one.

In the following, we describe the different components involved in building an IoT slice using a LoRa network.

LCM of an IoT MEC Application

As defined in ETSI MEC, a MEC application's LCM is handled by the MEO. If a vertical wishes to deploy an IoT slice at the MEC, the first step is to onboard the MEC application image (i.e., VM or container image) at the MEO catalogue. The onboarding process consists of providing metada-

ta on the MEC application and the location of the application image. These metadata are described in a specific format, which is known as the Application Descriptor (AppD) [11]. It includes information on the location of the virtual image, security information, and other fields related to the requirements of the MEC application, such as its maximum tolerated latency, traffic steering rules, and required MEC services. In the case of an IoT MEC application, the AppD should include the following fields:

- The geographic location where the MEC IoT application needs to be deployed (i.e., the location/region where the IoT devices are deployed)
- The required MEC service, which corresponds to the MEC LoRa service
- The maximum tolerated latency by the IoT application

Since the MEC application image is onboarded, the MEO creates an identifier for the MEC application, which is communicated to the vertical and used by the latter to instantiate the MEC application. Following the request of the vertical to instantiate the MEC application, the MEO uses the AppD, and more specifically the three fields described earlier, to select the appropriate MEP that satisfies the combined requirements, and requests the deployment of the MEC application to the VIM (at the selected MEC host). Once the MEC application is up, the next step consists of allowing the latter to discover the MEP resources over the Mp1 reference point, and therefore the API related to the MEC LoRa service. It should be noted that since the MEC application is running in a sliced environment, the MEP should allow the service discovery only to the LoRa service; other MEC services will be hidden to the MEC IoT application.

MEC Service Mp1 Interface

The Mp1 interface will be used for two purposes:

- Discover the services available at the MEP (in this case, MEC LoRa service API endpoints)
- Interact with the MEC LoRa service

The service discovery is already a function of the MEP, but it needs to be adapted to support network slicing at the MEP. Indeed, to ensure isolation and security as needed in a sliced environment, the deployed applications should have access only to their authorized MEC services. For this purpose, we propose that the MEO, when instantiating the MEC application, indicates to the MEP the authorized MEC services to which the MEC application should have access. Therefore, the MEP should identify the MEC application when the latter requests the MEC service catalogue available at the MEP. Regarding the MEC LoRa service, the Mp1 API should provide the necessary functions to interact with and configure it. The Mp1 API is divided in three parts:

- Application Service, which allows registering and removing the IoT applications. The registration process allows the creation of an application end-device inique identifier (APPEUI).
- Device Service, which allows registering the devices by indicating the device identifier (DEVEUI), the application where the device belongs (APPEUI), device latitude and longitude, device activation parameters (over-the-air activation — OTAA, or activating a device by personalization — ABP, the security keys in case of ABP, etc.). To recall, OTAA and ABP are the two modes used to activate IoT devices in LoRaWAN.
- Gateway Service, which allows registering gateways with the network server. Mp1 should also allow registering a gateway with the MEC LoRa service by indicating the IP address of the gateway, its EUI, and other specific parameters (e.g., the power and name). This service should also allow getting statistics on the gateway state.

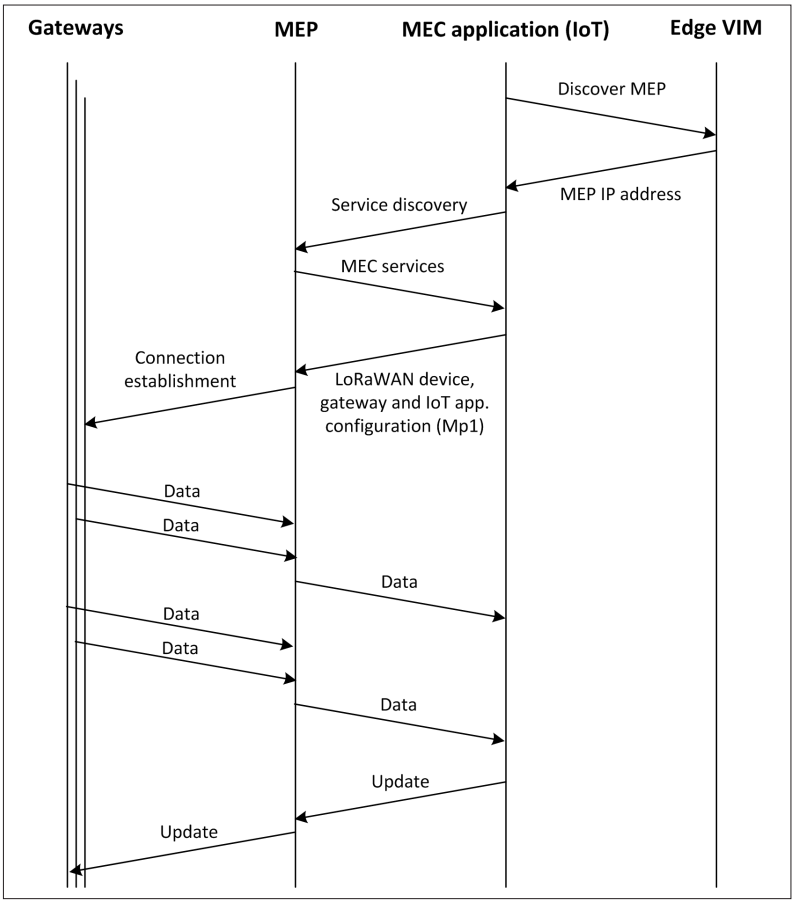

Figure 3. MEC LoRa service discovery.

Figure 3 illustrates how a deployed MEC application discovers the Mp1 details, including the MEC LoRa service API. Once the MEP application is on, it starts by discovering the serving MEP. This information is obtained from the local MEC host (or edge VIM) via a DNS lookup. Indeed, the MEC host is assumed to run a DNS server, which includes the information on the local MEP. After the discovery process, the application requests the MEC service catalogue available at the MEP. The latter replies with only the MEC services that the MEC application has the right to access; in this case, the MEC LoRa service API. The MEC application uses the Mp1 API to register the IoT application, devices and, if needed, the gateways (depending on the scenario). The MEC application indicates via a callback (i.e., an IP address-port pair and a function endpoint) where to receive the data sent by the IoT devices. The MEC LoRa service uses a publish/subscribe mechanism

Once the MEC application is up, the next step consists in allowing the latter to discover the MEP resources over the Mp1 reference point, and therefore the API related to the MEC LoRa service. It should be noted that since the MEC application is running in a sliced environment, the MEP should allow the service discovery only to the LoRa service; other MEC services will be hidden to the MEC IoT application.

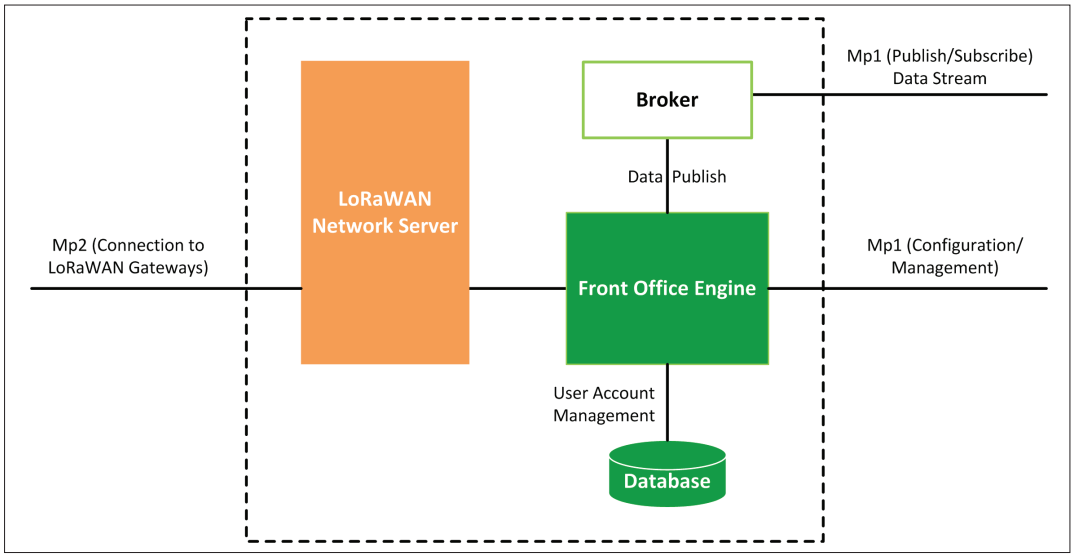

FIGURE 4. The MEC LoRa service components.

```
'applications':{
  'application1':{
    'description application': 'New_App',
    'application name': 'New_App',
    'devices':{
      'device1':{
        'authentication method': 'otaa',
        'description device': 'brand new device',
        'device eui': '6102040405060710',
        'device name': 'device_otaa',
        'configuration':{
          'application key': '53acaeaa3438c7ff6726876dae220032',
          'device eui': '6102040405060710',
          'network key': 'a65fc1832975b4c4fd9e162fb4debc49'
        }
      },
      'device2':{
        'authentication method': 'abp',
        'description device': 'brand new device 2',
        'device eui': '7102030405060710',
        'device name': 'device_abp',
        'configuration':{
          'app session key': '3c31b4615edd69eb6a5143e624cdcf1b',
          'device address': '0062342f',
          'device eui': '7102030405060710',
          'network key': 'a65fc1832975b4c4fd9e162fb4debc51'
        }
      }
    }
  },
```

FIGURE 5. Excerpt of an Mp1 message.

to redirect the received messages from the IoT devices to the appropriate MEC application. For certain reasons (e.g., firmware update), the IoT application may need to send data to the device. This should also be supported over the Mp1 interface.

Implementation and Preliminary Results

We have implemented the MEC LoRa service by extending our OAI-MEP implementation,[8] which supports most of the ETSI-specified services, such as RNIS, traffic redirection, service discovery, and service registration. The MEC LoRa service is based on the open source implementation of the LoRa server.[9] The MEC LoRa service is illustrated in Fig. 4. It is composed of the LoRa network server, a front office engine, a broker to push data toward the registered applications, and a database. The LoRa network server is in charge of receiving/transmitting data from/to LoRa devices via the LoRa gateways. The front office is written in Python and uses the Flask library.[10] It is the interface between the IoT MEC application (via Mp1) and the LoRa network server. It translates the Mp1 messages (e.g., device configuration) to a LoRa server configuration using the API provided by the latter. The front office engine is also in charge of user account management and authentication to ensure slice isolation. Finally, the front office engine uses a broker to push data received from the LoRa server to the registered MEC IoT application, allowing asynchronous communication. The broker pushes the data to the MEC IoT application using the IP and port specified in the callback information obtained via Mp1.

We deploy LoRa devices and LoRa gateways on top of Rasperry Pi 3. To deploy and instantiate a MEC application (IoT analytics), we use the MEO of OAI-MEP that allows us to onboard and instantiate MEC applications on top of lxc containers,[11] representing an edge VIM infrastructure. To interact with the MEO, we use a web proxy as described in [7]. The vertical selects the AppD describing an IoT analytics application, onboards the IoT MEC application, and instantiates it on top of our lxd-based VIM. Once deployed, the instance's credentials are available to the vertical, which allows the latter to have access to the deployed application via SSH. After the deployment, the MEC application can discover the MEP service and the Mp1 API endpoints to access the MEC LoRa service. In the case of the LoRa server implementation, two types of interfaces are provided, a REST API and a GUI, which allow configuration of the application, devices, and gateway. We extend the API by providing a callback registration mechanism, which allows the application to specify the IP and port number to which to send the data pushed by the broker. The vertical then starts the IoT analytics application, and begins receiving the device messages.

Figure 5 shows an excerpt of an Mp1 message used to register devices as well as the application to the MEC LoRa service when the vertical is using the operator's LoRa gateways. It uses a JSON message format.

[8] A Python-based implementation of the ETSI MEP based on OAI (http://openairinterface.org) for 4G connectivity.

[9] As of this writing, LoRa Server has been renamed to ChirpStack. https://www.chirpstack.io/

[10] https://www.palletsprojects.com/p/flask/

[11] https://linuxcontainers.org/

App. onboarding	116.4s
App. instantiation	58.5s
MEP discovery	25 ms
Service discovery	25 ms
Total	173.05s

Table 1. Time needed to deploy a LoRa IoT network slice.

Each device is described using its EUI, as well as other security parameters, which allows activating the devices (e.g., ABP or OTAA). In the case of ABP, more information is needed, such as the devices' addresses.

Finally, we evaluate the total time needed to deploy an IoT application in the form of a network slice at the edge when using the developed platform. This time duration is assumed from the moment the vertical requests the deployment of an application until the time when the application is registered with the MEP; it includes the onboarding time, instantiation time, MEP discovery time, and service discovery time. We chose to run our experiments on a low-end compute environment: Our MEC host (VIM platform) is an AMD FX-7500 Radeon R7, with 4 CPU cores at 2.1 GHz (maximum CPU frequency) and 8 GB RAM, running Linux kernel 4.4.0-97.

Table 1 summarizes the obtained results. We clearly observe that the time needed to onboard the application image represents 68 percent of the total time. However, the onboarding process is needed only once (i.e., the first time the application package is made available), which means that deploying another instance of the same application will consider only the instantation time as well as MEP discovery time, which is around 59 s.

Conclusion

In this article, we introduce a novel framework that extends the ETSI MEC model to support the deployment of LoRa-based IoT applications at the edge. The proposed framework allows an IoT vertical to deploy an IoT network slice at the edge, taking advantage of a low-latency and context-aware environment. We leverage the Mp1 ETSI MEC interface to abstract the low-level configuration of end devices and ease the registration of LoRa-based applications. Our preliminary results show that the deployment of an IoT slice at the edge requires less than a minute.

The proposed framework allows an IoT vertical to deploy an IoT network slice at the edge, taking advantage of a low-latency and context-aware environment. We leverage the Mp1 ETSI MEC interface to abstract the low-level configuration of end devices and ease the registration of LoRa-based applications. Our preliminary results show that the deployment of an IoT slice at the edge requires less than a minute.

Acknowledgment

This work was partially supported by the European Union's Horizon 2020 Research and Innovation Program under the 5G!Drones project (Grant No. 857031).

References

[1] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," *IEEE Internet of Things J.*, vol. 3, no. 6, 2016, pp. 854–64.
[2] J. An *et al.*, "EiF: Toward an Elastic IoT Fog Framework for AI Services," *IEEE Commun. Mag.*, vol. 57, no. 5, May 2019, pp. 28–33.
[3] I. Afolabi *et al.*, "Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions," *IEEE Commun. Surveys & Tutorials*, vol. 20, no. 2, 2018, pp. 2429–53.
[4] A. Ksentini, and P. Frangoudis, "Toward Slicing-Enabled Multi-Access Edge Computing in 5G," *IEEE Network*, vol. 34, no. 2, Mar./Apr. 2020.
[5] H.-L. Truong, "Enabling Edge Analytics of IoT Data: The Case of LoRaWAN," *Proc. 2018 Global Internet of Things Summit*, 2018.
[6] R. Sanchez-Iborra, J. Sanchez-Gomez, and A. F. Skarmeta, "Evolving IoT Networks by the Confluence of MEC and LP-WAN Paradigms," *Future Generation Comp. Sys.*, vol. 88, 2018, pp. 199–208.
[7] Multi-access Edge Computing (MEC); Framework and Reference Architecture, ETSI Group Spec. MEC 003, V2.1.1, Jan. 2019.
[8] Mobile Edge Computing (MEC); Radio Network Information API, ETSI Group Spec. MEC 012, V1.1.1, July 2017.
[9] Mobile Edge Computing (MEC); Location API, ETSI Group Spec. MEC 013, V1.1.1, July 2017.
[10] D. Sabella *et al.*, "Developing Software for Multi-access Edge Computing," ETSI, White Paper 20, Feb. 2019.
[11] Mobile Edge Computing (MEC); Mobile Edge Management; Part 2: Application Life Cycle, Rules and Requirements Management, ETSI Group Spec. MEC 010-2, V1.1.1, July 2017.
[12] S. S. Husain *et al.*, "Mobile Edge Computing with Network Resource Slicing for Internet-of-Things," *Proc. 4th IEEE World Forum on Internet of Things*, 2018.
[13] L. Zanzi *et al.*, "Evolving Multi-Access Edge Computing to Support Enhanced IoT Deployments," *IEEE Commun. Standards Mag.*, vol. 3, no. 2, Apr. 2019, pp. 26–34.

Biographies

Adlen Ksentini (adlen.ksentini@eurecom.fr) is an IEEE ComSoc Distinguished Lecturer. He obtained his Ph.D. degree in computer science from the University of Cergy-Pontoise in 2005. Since March 2016, he has been a professor in the Communication Systems Department at EURECOM. He has worked on several EU projects on 5G, network slicing, and IoT.

Pantelis A. Frangoudis (pantelis.frangoudis@tuwien.ac.at) is a university assistant at the Distributed Systems Group, TU Wien, Austria. He was a researcher at INRIA/IRISA, Rennes, France (2012–2017), joining as an ERCIM post-doctoral fellow, and in the Communication Systems Department, EURECOM, France (2017–2019). He has a Ph.D. (2012) in computer science from AUEB, Greece.