Service Oriented Architecture for Interconnecting LoRa Devices with the Cloud

Konstantinos Tsakos, Euripides G.M. Petrakis

Abstract The contribution of this work is two-fold: First, we show how Low Power Wide Area Networks (LPWANs) can be interconnected with the cloud; second, leveraging on PaaS functionality, we develop LoRaWare, a Service Oriented Architecture (SOA) and system that allows developers to enhance the capabilities of LoRa enabled applications using advanced cloud services such as, selective publication and subscription to data, IoT connectivity using MQTT protocol, persistent storage etc. We experimented with LoRa and LoRaWAN, the latest successful representative of LPWAN protocols. We applied a typical experimental setup with LoRa environmental sensors transmitting measurements over long distances to gateways. LoRa gateways receive LoRa packets from sensors in range and re-transmit them to the cloud using an IP protocol (typically UDP). In this work, we opt for MQTT, a more elaborate lightweight publish-subscribe IP protocol offering advanced security, better routing control and visibility of the communication (i.e. easier handling and control of data packets). To support connection of LoRa IoT networks with the cloud, we developed the Network Server, a cloud service that encompasses the necessary functionality for porting LoRa packets to applications (i.w. encrypts/decrypts, de-duplicates, authenticates LoRa packets and converts LoRa payloads to JSON). We developed our solution in Fiware, the cloud infrastructure of the European Union. The reason for our selection is that, currently, Fiware supports interconnection with LoRa only via the network of another provider (i.e. The Things Network). The Network Server is the only solution for connecting LoRa networks directly to Fiware. We run an exhaustive set of experiments in order to study system response time and scalability as well as, the practical range efficiency of LoRaWAN protocol.

1 Introduction

The idea of Internet of Things (IoT) combined with cloud computing, opens new horizons in the field of real time data collection and analysis [2]. Cloud computing emerges as the key platform for IoT data storage, processing and analytics due to its simplicity, scalability and affordability (i.e. no up-front

Konstantinos Tsakos
School of Electrical and Computer Engineering, Technical University of Crete (TUC), Chania, Crete, Greece, e-mail: ktsakos@isc.tuc.gr

Euripides G.M. Petrakis
School of Electrical and Computer Engineering, Technical University of Crete (TUC), Chania, Crete, Greece, e-mail: petrakis@intelligence.tuc.gr

investment, low operation costs). IoT manufacturers have developed products, which typically use short range protocols such as Bluetooth and later Bluetooth Low Energy (BLE) that connect IoT devices to a gateway (e.g. a smartphone or tablet) and then to the cloud via a cellular network and the internet. However, the commercial success of this solution is questionable as the number of IoT devices (e.g. sensors) that can be connected to a gateway using BLE is limited and the energy toll on gateways can be huge, imposing that the gateway must be connected to a sustainable power source and be close to the IoT devices. Bluetooth and BLE, the same as WiFi and ZigBee, are not suited for long-range transmission, while cellular networks are costly, consume a lot of power, and are expensive.

Low Power Wide Area Network (LPWAN) technologies fill the gap between mobile (3G, 4G, LTE) and short-range wireless (e.g. Bluetooth, WiFi and ZigBee) networks [6]. LPWAN technology falls short in terms of QoS compared to cellular standards and this means that an operator cannot use it to provide the kind of Service Level Agreements (SLAs) that are critical for customers in certain application domains (e.g. remote health monitoring) [1]. However, LPWAN specifications fit well the requirements of many IoT applications (e.g. smart cities, home automation, industrial automation, environmental monitoring) who need to transmit small quantities of data periodically over a long range, while maintaining long battery life (e.g. an environmental sensor).

The present work is the technological bridge between the two important technologies referred to above namely, LorRaWAN and Cloud computing. The focus of this work is on interconnecting LoRa gateways with the cloud. We develop a Network Server whose purpose is to receive LoRa packets from gateways and port LoRa payloads to other cloud services. The Network Server applies a sequence of operations (i.e. decrypts, de-duplicates, authenticates LoRa packets) and, finally transforms LoRa packets to JSON format. For outgoing packets the same solution is applied in reverse order: Packets are encrypted and transmitted to target gateways.

Leveraging on advanced FiWare services we developed LoRaWare, an innovative IoT platform that supports a complete sequence of services for IoT data in the cloud that includes, selective publication and subscription of users and services to data, processing of events based on rules, persistent storage, data analytics functions and application specific services. The platform is secure by design allowing access to data and services based on authorization and user roles. To show proof of concept and support the LoRa claims of range efficiency, we developed an application with LoRa sensors transmitting temperature and humidity measurements to LoRa gateways and from there to the Network Server in the cloud.

As of July 2018, Fiware announced availability of a LoRaWAN IoT Agent developed by The Things Network [1](TTN) and Loriot.io[2]. The same as our

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

[2] https://www.loriot.io

Network Server, the new IoT Agent's ambition is to provide an interface with LoRa networks. However, it requires that application owners register their devices, gateways and applications to the network of a third party (i.e. the TTN network[3]); from where, LoRa payloads are transmitted to Fiware using an IP protocol. For applications owners and developers who need to install and maintain a LoRa IoT network on their own and connect to Fiware directly (i.e. not via TTN network), our Network Server is the only solution.

We run an exhaustive set of experiments using real and simulated (but realistic) data in order to study the system response time and system scalability. We report average end-to-end processing times (i.e. from the time IoT data are received by the network server to the time they are stored in the database) and also, average time spent on each service in the processing sequence in the cloud. Our experimental results demonstrate that our system is capable of performing in real - time (or close to real time) for up to thousands of requests. In a different experiment, we study the practical range of LoRa transmission in a complex terrain with two gateways placed 5.7Km apart from each other. The experimental results reveal that the error rate for packages captured by any of the two gateways increases drastically with the distance from the gateways.

The rest of this paper is structured as follows: Sec. 2 describes IoT networks with LoRaWAN support. Sec. 3 presents the architecture of the cloud back-end that includes the Network Server. Implementation issues are discussed in Sec. 4. Sec. 5 presents our experimental results followed by conclusions in Sec. 6

2 Sensing Environment

Fig. 1 highlights LoRaWare ecosystem consisting of (a) the sensing platform, (b) the back-end (cloud) implementing (among other services) a Network Server and, (c) the applications which run in the cloud and allow consumers (end-users) to receive information from selected devices (sensors).

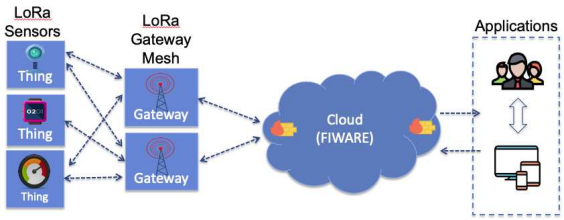

Fig. 1 LoRaWare ecosystem

Sensors are not associated with a specific gateway. Instead, data transmitted by a sensor can be received by multiple gateways (i.e. all gateways in range). Each gateway will forward the received packets to the Network

[3] https://github.com/Fiware/iot.IoTagent-LoraWAN/blob/master/README.md

Server. The intelligence and complexity of the LoRa sensing network is pushed to the network server which manages the active connections.

2.1 LoRa Gateway

The LoRa nodes (i.e. sensors) transmit LoRa modulated packets which are captured by one or more gateways (up-link transmission). A gateway receives LoRa packets from sensors in range and re-transmits them to the cloud (i.e. to the Network server) using an IP protocol (e.g. UDP). This operation is carried-out using the packet forwarder[4], an open source software by Semtech which runs on every gateway. The packet forwarder applies UDP for the communication between the LoRa Gateways and the Network Server. Conversely, LoRa gateways receive down-link messages from the Network server using an IP protocol from where they are forwarded to LoRa devices using LoRaWan. Fig. 2 illustrates this process.

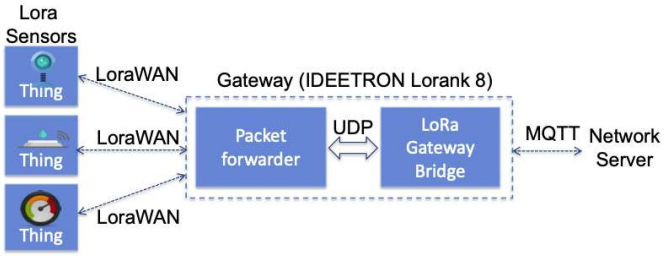

Fig. 2 LoRa gateway

Other protocols higher in the internet stack can be also applied for the communication between a gateway and the network server (e.g CoAP, MQTT). In this work, we opt for MQTT[5], an elaborate lightweight publish-subscribe IP protocol (i.e. a LoRa payload can be sent to many MQTT clients simultaneously each serving a different purpose or application). Besides, secure one-to-many communication (i.e. UDP is more susceptible to spoofing and denial of service attacks), MQTT offers better routing control and visibility of the communication (i.e. easier handling and monitoring of the packets sent by the gateways). Relying on TCP, MQTT applies TLS (Transport Layer Security) for data encryption and secure communication. Overall, MQTT should be preferred over UDP unless speed of processing is a key factor and error correction or flow control is not necessary.

We developed the "LoRa gateway bridge" service that receives LoRa packets from the packet forwarder and converts UDP to MQTT/TCP prior to transmission to Network Server. For the publish-subscribe communication between the network server and the LoRa gateway bridge we relied

[4] https://github.com/kersing/packet_forwarder
[5] http://mqtt.org

on Eclipse Mosquitto message broker[6] supporting protocol versions 3.1 and 3.1.1. The gateway then allows one-to-many communication subscribing to different topics (e.g. areas of interest where data are published in order to serve different applications or users). At the same time, the Network Server can serve many clients (subscribing to many topics simultaneously). When a client subscribes to a topic, it is entitled to receive data published on this topic.

3 Back-end Services

LoRaWare implements in the cloud the services illustrated in Fig. 3. Important advantages of LoRaWare design are (a) adoption of the NGSI[7] (i.e the Fiware data exchange model based on JSON) for subscribing LoRa devices to the cloud and (b) the use of the Publish-Subscribe service of Fiware for making IoT data available to users and for users to subscribe to this information, and get notified on changes of this information or when new information becomes available. The entry point to the cloud is the Network Server.

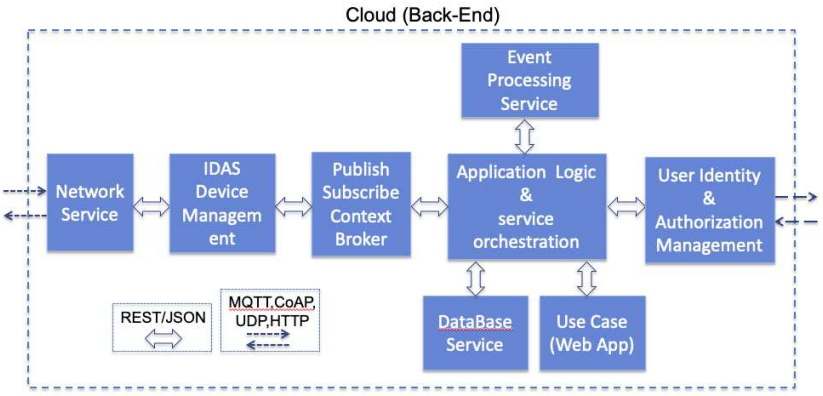

Fig. 3 Cloud platform

Network Server: The Network Server manages active sessions (i.e. devices that have joined the network), serves new LoRa nodes when they join the network, decodes and decrypts the physical LoRa payload, deduplicates the received packets (which may be captured by multiple gateways), authenticates the packets to make sure that they are not due to attacks (e.g. denial of service attacks). Also, it manages the state of each node using mac-commands (e.g. to change data rate, channels, etc.). Finally, it maps LoRa context (i.e. sensor names, types) and content information (i.e. measurements) to NGSI. Fig. 4 illustrates this process as a sequence of steps which are explained below.

[6] https://mosquitto.org

[7] https://orioncontextbroker.docs.apiary.io/#introduction/preface/status

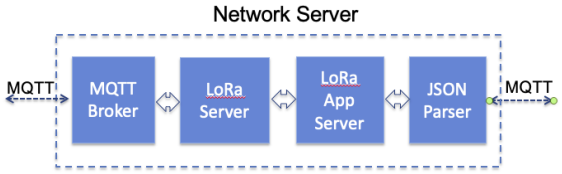

Fig. 4 Network server

MQTT Broker: It subscribes to payload published on LoRa gateway bridge. In analogy with the MQTT broker on gateway, it is implemented using Eclipse Mosquitto. The MQTT broker subscribes to one or more gateways and conversely, a payload which is published on a gateway can be sent to subscribed MQTT Broker clients on the Network Server.

LoRa Server: It is responsible for managing active node sessions (i.e. nodes which have joined the network). For the active sessions, it (a) de-duplicates the received data, (b) authenticates this data, (c) decrypts/encrypts LoRa packets using the network key, (d) forwards (encrypted) data to the LoRa application server, (e) asks the application server for feedback. Finally it is responsible for managing the state of a LoRa node through so called mac-commands (e.g. to change the data-rate, channels, etc.).

LoRa Application Server: It is compatible with the LoRa Server component (above) and is responsible for (a) the device "inventory" (i.e. a database for devices, gateways and device data), (b) handling of join-requests and (c) encryption and decryption of application payloads using the application key. It is responsible for handling users, nodes, applications and access rights (i.e. via user interface or API). An application is a collection of devices with the same purpose or of the same type and can be associated with one more gateways. It responds to requests issued by LoRa server for which nodes are allowed to join the network and if so, which settings to use. When a new node joins the network, permission is granted by LoRa Application Server. It offers node management per application, per organization and gateway management per organization. It also offers functionality for user and sensor management (e.g for assigning users or sensors, to organizations or applications). For authentication and authorization, users can be created in LoRa Application Server. A user itself can be an administrator or a regular user. Administrators are authorized to perform any action (e.g. add new network servers). LoRa Application Server is entitled to connect to one or multiple LoRa Server network-server instances. Finally, LoRa Application Server is responsible for handling users, nodes and applications.

LoRa Server and LoRa Application Server are parts of the open-source LoRaWAN network server project[8] which provides many components for building LoRa networks. Both services can be accessed using a RESTful or gRPC API and implement a storage for information pertinent to these services (e.g. nodes, users, subscriptions, access rights etc.).

[8] https://www.loraserver.io

JSON Parser: It parses the MQTT payload, filters-out metadata of LoRa payloads, keeps sensor values and subscribes this information to the MQTT IoT agent of IDAS service (beloq). The agent accepts a different payload format from the one that LoRa Application Server publishes. It is implemented in PyThon using Paho Client[9] client class. This class provides functionality to make publishing one off messages to an MQTT server very straightforward. Our script runs as a daemon and its role is twofold: (a) it listens to the topic where the LoRa Application Server publishes data and, (b) it publishes the new JSON payload on a topic where the IoT Agent of IDAS subscribes to receive data.

IDAS Backend Device Management service[10]: This service is provided by Fiware. It collects data from the Network Server and translates them to NGSI format. Devices running a different protocol are served by a separate IoT Agent (i.e. IDAS Backend Device Management is a composition of IoT Agents). An IoT Agent listens to an IoT protocol (MQTT in our case) and transforms the LoRa payload to NGSI. Developers should know in advance which protocol they will be using to connect their devices to the cloud and select the right IoT Agent to use. IoT Agents publish IoT values to the Publish-Subscribe Context Broker; users or other services can subscribe to this information and get notified on value changes or, every time a new value becomes available (e.g. for triggering actions when new values become available). Fig. 5 illustrates this process.

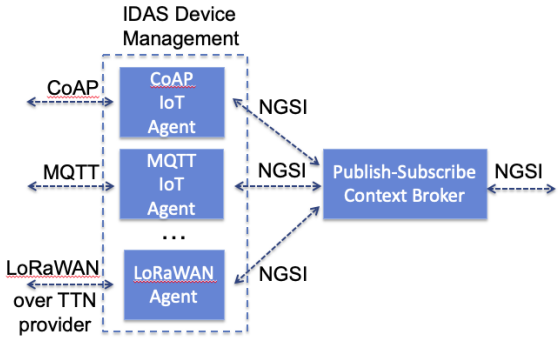

Fig. 5 Connecting IoT devices to FIWARE

Publication and Subscription service: This service acts as a mediator for the data sent to the back-end and the end-users or applications. It is implemented using Publish Subscribe Context Broker[11] service of FIWARE. Services or users can subscribe to data produced in the front-end. Sensors are listed as "public entities" and users or other services are subscribers to these

[9] https://www.eclipse.org/paho/

[10] https://catalogue-server.fiware.org/enablers/backend-device-management-idas

[11] https://catalogue-server.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker

entities. Each time a new sensor is registered to the system or a new measurement becomes available, this component is updated. When a new sensor is used, a new entity is created in the broker. When a new measurement becomes available by a sensor, a notification is sent to all entities subscribed to this information.

Event management service: It is a rule engine that gets input from the publication and subscription service. The main purpose of this component is to analyse data that become available to the publish and subscribe service (above). When a sensor creates a new measurement, its corresponding entities in the publication and subscription services are updated as well. The event processing service subscribes to this information and gets a notification about the change. This triggers the execution of rules that govern the application: If the value is within the limits of these rules no action is taken; otherwise (e.g. the temperature exceeds its threshold), the service notifies the subscribed user to take action. In turn, rule violations may triggers events (e.g. an alert is generated like sending an SMS to subscribed users). Complex Event Processing (CEP)[12] of FIWARE is a reference implementation of this service.

User Identity and Authorization Management service: Implements user management mechanisms and access control based on user roles (e.g. administrator, user) and access policies using FIWARE Keyrock Identify management (IDM)[13] service of FIWARE. It provides a Single Sign On (SSO) service for secure access to services, data and networks and, applies (a) identification services for users, (b) management of their profiles, (c) authorization services based on OAuth2.0[14] (i.e. applications accessing protected resources are checked whether they are authorized to do so using their OAuth2.0 credentials).

Storage service: It is a shared database where data for user actions, sensors (history data), messages, rules, events (e.g. rule violations) are permanently stored. It is implemented as a non-SQL database using MongoDB.

Application Logic: Its purpose is to orchestrate, control and execute services running in the cloud. Threshold violations (reported by the Event Management service below) are typically managed by this component. When a request is received (from a user or service), it is dispatched to the appropriate service. For example, services regarding user accounts and access rights are dispatched (from application logic) to user management service. It is tightly related with the connections and the publication and subscription services which form the communication channel with the gateway. Finally, application logic performs basic security controls (e.g. checking if a session between the mobile application and the cloud is active).

---

[12] http://fiware-iot-stack.readthedocs.io/en/latest/cep/index.html

[13] https://catalogue-server.fiware.org/enablers/identity-management-keyrock

[14] https://oauth.net/2/

Additional services can also be imported to the cloud. For example, a Mashup Editor (e.g. NodeRed[15]) to support composing new applications with minimal effort, Business Support Services (BSS) dealing with subscription, system monitoring, billing and accounting services required for commercial exploitation of the platform), data analytics services etc.

4 LoRaWare Implementation

To show proof of concept, we developed an example application (use case) where humidity and temperature measurements produced by LoRa nodes are monitored in real - time on the cloud. The sensor measurements are updated asynchronously (using AJAX calls). Humidity and temperature charts as a function of time are displayed on a Web interface. The application subscribes to the Publication and Subscription service (above) which is responsible for updating the database service and the User Interface when new sensor measurements become available.

LoRaWare Sensing Environment: It is implemented based on Ideetron products (gateways and LoRa nodes). The Lorank8[16] is a LoRa gateway with professional specifications. According to specifications, it enables connection with many thousands end nodes and communication up to many Kms if mounted on a high point with free sight. Each LoRa node is implemented on Ideetron Nexus boards[17]. Every Node has one Nexus Board with a microcontroller and a PCB antenna for the transmission of RF packets. It connects to a Nexus Demoboard which has temperature, humidity and motion sensors mounted on it.

LoRaWare Back-End: It is implemented as a composition of autonomous RESTful services [7, 4] communicating with each other over HTTP (eventhough these can be deployed on the same VM). Individual services or groups of services are deployed on the same or different Virtual Machines (VMs) in the cloud. Network delays are expected due to the nature of this design. However, the experimental results (Sec. 5) demonstrate that the back-end is capable for responding in real time even under heavy workloads.

The back-end is implemented on a FIWARE cloud running in OpenStack. The services communicate asynchronously using Slim PHP (for handling REST requests), cURL for communicating data and requests between services and, finally, MongDB for the database. The user interface of the Web application was implemented using HTML and CSS/jQuery. The back-end is deployed on five VMs, all running locally (in the FIWARE node of TUC) with the exception of identification and authorization service that runs on a shared instance of Keyrock Identity Management installed on the FIWARE node of Spain (FIWARE is a federation of cloud infrastructures distributed across the EU). The four VMs that run locally are, the Network Server, the Database (MongoDB), one VM running the Publish-Subscribe service ser-

[15] https://nodered.org

[16] https://www.ideetron.nl/lora/?lang=en

[17] https://webshop.ideetron.nl/LoRa/

vices (IDAS Device Management and Publish-Subscribe Context Broker) and, one VM running application logic, event management and application logic services. Each VM has one processor (x86 64 processor architecture, 2,800MHz, first level cache size 32KB, second layer 4,096KB cache size), 2,048MB RAM, 20GB hard drive capacity, runs Ubuntu 14.04 and an Apache HTTP server (that handles PHP requests).

5 Performance Evaluation

We run an exhaustive set of experiments using real and simulated (but realistic) data in order to study the time performance of the cloud solution and the range efficiency of LoRa protocol using as a use case the Web application that monitors the humidity and temperature measurements from sensors.

Back-end System Evaluation: The purpose of this experiment is to evaluate (a) the response time and (b) the scalability of the back-end system (i.e. how system response times increase with the number of connected users). We measure overall system response time (i.e. from the time a measurement is received by the cloud to the time it is stored in the database), average response time of each service and also, average workload (i.e. CPU, memory usage) of each running VM.

The response time for the system to process 1 request is 170.5ms from which 16.8ms account for network delays (which is low since all VMs are running in the same cloud Infrastructure). Table. 1 reports the time spent on each service to process a single request.

Table 1 Response time for each service

Service	*MQTT Broker*	*LoRa Server*	*LoRa App. Server*	*JSON Parser*	*IoT Agent*	*Pub-Sub CB*	*Application*	*Storage*
Time (ms)	1	1	22	4	2	2	40	100

In order to study system scalability the system is stressed with many concurrent requests simulating the workload of a network with thousands of LoRa sensors. We used ApacheBench[18] to issue multiple simultaneous requests to each VM. In ApacheBench we are opted to define the total number of requests and how many of them will be executed simultaneously. In this experiment we report response time of three running VMs: The Network Server, the Publish - Subscribe services and the Use Case (i.e. application logic and event processing services). The performance of MongoDB database (fourth VM) has been studied elsewhere and is known to scale-up well [5]. The Identity Management service (fifth VM) is not expected to cause any performance bottlenecks as it is addressed only once by each user at login and, for authorizing access to resources.

[18] https://httpd.apache.org/docs/2.4/programs/ab.html

Table 2 reports average response times (per request) for various values of concurrency ranging from 1 through 300 (i.e. number of requests executed simultaneously). All measurements of time account also for the time spent for the communication between VMs or between services within the same VM. We notice a very low resource usage when concurrency = 1. This is expected, as only one request is executed at any time. Response times improve slightly with the simultaneous execution of 50 requests (i.e. the Apache HTTP server switches to multitasking). Processing capacities may increase or raise restrictions for space or bandwidth for concurrency > 100. The best values are obtained with concurrency values between 50 and 150, hence we conclude that the system optimal performance is for up to 150 concurrent users.

Table 2 Response time for each service

	VM1: Network Server			*VM2: IDAS & Pub/Sub CB*			*VM3: Application*		
Concurency	1	100	300	1	100	300	1	100	300
Time (ms)	22.7	21	22.1	29.9	9.4	5.7	14.5	6.3	11.9
CPU (%)	23–35	45–100	57–100	31–45	34–58	56–100	23	100	100
RAM(MB)	228–247	239–357	253–377	228–570	239–593	253–604	247	257	273

LoRaWare may potentially produce big amounts of data or receive many requests which can surpass the capacities that this experimental system setup is able to provide. An obvious solution would be to spawn additional VMs implementing the same service thus having more than on VM sharing the load.

LoRa Network Evaluation: Reliable communication of LoRa devices depends on number of transmission parameters [3]. The following parameter values are defined (in Arduino sketches): Spread factor =12, bandwidth =125kHz and transmission power = 17dbm. The code rate was set to 4/5 (the value was fixed by Ideetron). These values would maximize the effective range of LoRa (although for these values power consumption will be maximum as well).

We run the experiment in a complicated urban and sub-urban environmental terrain in the city of Chania, Crete. We placed two Lorank 8 gateways 5.7Km apart from each other. The first one is placed in an urban area and the second one in a sub-urban area of the city of Chania. Two mobile LoRa nodes transmitted 74 temperature and humidity measurements (37 each node). The longest effective range was 923m and 1,23Km for the gateway placed in the urban and sub-urban area respectively. Table 3 shows how error rate varies with distance. Despite the high error rate, its worth noting that more than 80% of transmitted packages were received successfully up to 350m and 700m from the gateway in the urban and sub-urban area respectively. These results are below our expectations. This could be due to

the noise interference and the non-uniform ground terrain with many obstacles that hinder transmission in the area of the experiment.

Table 3 Error Rate (ER) as a function of the distance from gateway

Distance (m)	100	300	500	800	1,000	1,500
ER Urban Area (%)	0	15	30	85	100	100
ER Sub-Urban Area (%)	0	0	20	35	65	90

6 Conclusions

We propose and implement the Network Server, a service that enables porting of LoRa networks to the cloud. As a result, the cloud supports unified management of LoRa payloads with minimum additional effort. In addition, we designed and implemented LoRaWare, an experimental infrastructure for studying the performance of LoRa networks. The response times reported for all cloud services are a good support to our claims of efficiency: LoRaWare is capable of responding in real-time (or close to real-time) even under heavy workloads. However, the effective range of LoRa is way lower than expected. Although these results are characteristic for a worst case environment (with many buildings, metal structures, rough terrain with many obstacles and gateways not optimally placed at high altitude), the results indicate that lots of experimentation is necessary for tweaking LoRa parameters in order to achieve closer to the optimal (i.e. theoretical) performance.

References

1. F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne. Understanding the limits of lorawan. *IEEE Comm. Magazine*, 55(9):34–40, 9 2017.
2. A. R. Biswas and R. Giaffreda. Iot and cloud convergence: Opportunities and challenges. In *IEEE World Forum on Internet of Things (WF-IoT)*, pages 375–376, Seoul, South Korea, March 2014.
3. M. Bor and U. Roedig. Lora transmission parameter selection. In *Intern. Conf. on Distributed Computing in Sensor Systems (DCOSS)*, pages 27–34, 5 2017.
4. K. Douzis, S. Sotiriadis, E.G.M. Petrakis, and C. Amza. Modular and generic iot management on the cloud. *Future Generation Computer Systems (FGCS)*, 78(1):369–378, January 2018.
5. E.G.M. Petrakis, S. Sotiriadis, T. Soultanopoulos, P. Tsiachri-Rentaa, R. Buyya, and N. Bessis. Internet of things as a service (itaas): Challenges and solutions for management of sensor data on the cloud and the fog. *Internet of Things*, 3-4:156–174, 10 2018.
6. U. Raza, P. Kulkarni, and M. Sooriyabandara. Low power wide area networks: An overview. *IEEE Communications Surveys and Tutorials*, 19(2):855–873, 11 2017.
7. S. Schreier. Modeling restful applications. In *ACM International Workshop on RESTful Design (WS-REST'11)*, pages 15–21, Hyderabad, India, 3 2011.