FUDGE: a frugal edge node for advanced IoT solutions in contexts with limited resources

Kiyoshy Nakamura
Universitat Politécnica de Valéncia
Valencia, Spain
minapin@posgrado.upv.es

Pietro Manzoni
Universitat Politécnica de Valéncia
Valencia, Spain
pmanzoni@disca.upv.es

Marco Zennaro
ICTP
Trieste, Italy
mzennaro@ictp.it

Juan-Carlos Cano
Universitat Politécnica de Valéncia
Valencia, Spain
jucano@disca.upv.es

Carlos T. Calafate
Universitat Politécnica de Valéncia
Valencia, Spain
calafate@disca.upv.es

José M. Cecilia
Universitat Politécnica de Valéncia
Valencia, Spain
jmcecilia@disca.upv.es

ABSTRACT

The growing connection among IoT and AI poses many challenges that call for novel approaches and rethinking of the entire architecture, communication, and processing to meet the requirements in latency, reliability, and use of resources. Edge computing is a promising approach in this sense. Moreover, it can become beneficial to bring advanced, i.e., AI-based, IoT solutions in areas where connectivity is scarce and where in general resources are limited.

In this paper we describe an edge/fog generic architecture to allow the adoption of edge solutions in IoT deployments in poorly connected and resource limited scenarios. To this end we integrated, using microservices, an MQTT based system that can collect ingress data, handle their persistency, and coordinate data integration with the cloud using a specific service called aggregator. The edge stations have a dedicated channel with the aggregator that is based on LoRa to enable long-range transmissions with low power consumption.

Some details of the implementation aspects are described along with some preliminary results. The initial tests of the architecture indicated that is flexible and robust enough to provide a good platform for the deployment of advanced IoT services in contexts with limited resources.

CCS CONCEPTS

• **Networks** → *Peer-to-peer protocols*; **Network architectures**; • **Information systems** → Collaborative and social computing systems and tools.

KEYWORDS

LoRa, IoT, Edge computing

advanced IoT solutions in contexts with limited resources . In *FRUGALTHINGS '20: ACM Workshop on Experiences with the Design and Implementation of Frugal Smart Objects, September 21, 2020, London, UK*. ACM, New York, NY, USA, 6 pages. https://doi.org/XXX

1 INTRODUCTION

The proliferation of smart IoT in association with artificial intelligence (AI) is giving rise to the development of new applications that not only require additional compute resources but add new constraints of privacy and low latency. This trend has motivated the recent adoption in IoT of the edge/fog computing paradigm, which introduces a middle tier of resources between the cloud and the IoT devices [1, 2].

While cloud refers to computing powered by large, distributed groups of servers, the edge refers to compute on the edge of the network, closer to or at the data source itself. Fog computing is inclusive of computing anywhere along the continuum, from cloud to the edge, into close proximity to the *things* [3].

For example, Machine Learning (ML) inference on the edge shows an increasingly attractive potential. By performing inference on-device, and near-sensor, ultra-low-power machine learning hardware like TinyML enables greater responsiveness and privacy while avoiding the energy cost associated with wireless communication, which at this scale is far higher than that of compute [4]. A recent trend is to deploy well trained artificial intelligence (AI) model on edge servers, whose capacities are somewhat bigger than those of IoT devices. However, since the training of AI demands huge computation and memory resources, the training process is preferred to be done in the cloud.

This link among IoT, cloud, edge, and AI, poses many challenges that call for novel approaches and rethinking of the entire architecture, communication, and processing to meet the requirements in latency, reliability, and use of resources [5]. Edge computing can also become beneficial to bring advanced IoT solutions in areas where connectivity is scarce and where in general resources are limited, like for example in rural areas areas and in developing countries.

In this paper, we detail the design of a generic frugal edge/fog architecture to provide computing services in poorly connected and resources limited scenarios. We first outline the concept of the "aggregator". The aggregator coordinates the data flow between the services on the edge nodes with those in the cloud. It uses a

polling technique so to: (1) obtain a higher flexibility in the use of the channel, (2) allow the use of sleep mode at the edge node, and (3) handle disconnection periods.

Edge nodes in this architecture are called "FUDGE" (FrUgal eDGE node). A FUDGE is based on microservices, an architectural approach where software is composed of small independent services that communicate over well-defined APIs. This architecture make applications easier to scale and faster to develop. In a FUDGE various microservices coexist and interchange data with other microservices using an API based on the MQTT pub/sub system. Data, indicated as *content*, enters locally into the FUDGE through so called "content proxys". Three main components can be found in a FUDGE: an MQTT broker, a persistency manager, and a content forwarder. The MQTT broker handles the flow of the content inside the node. The persistency manager element takes care of storing the content that is labeled as persistent in a time-series database. This allows to maintain the temporal evolution of the system while retaining all the required data to handle asynchronous operation or possible disconnections. Moreover, data analytic tools can be used to extract metrics that make sense out of the collected data, and monitoring apps through customizable dashboards. Finally, the content forwarder is in charge of talking with the aggregator using a protocols called "LoRaCTP" (LoRa Content Transfer Protocol). LoRaCTP is a reliable transport protocol designed on top of LoRa, that allows to transfer blocks of bytes ("content") adapting to the quality of the channel. LoRa and LoRaWAN are one of the most popular and successful technologies in the LPWANs space that offer long ranges with low power and low complexity although with low data rates [6, 7].

The paper is organized as follows. Section 2 gives an overview of the current activities in the area, Section 3 describes the overall fog architecture proposed, and in Section 4 the details of the FUDGE design are presented. Section 5 describe some details of the prototype that was designed to test the architecture. Finally, Section 6 describe some conclusions and future works.

2 RELATED WORKS

Other works in the literature addressed the combination of LoRaWAN and MQTT to provide IoT services, for example see [8–14]. In [15] a LoRa-MQTT gateway device is described for supporting the sensor-to-cloud data transmission in smart aquaculture IoT applications. In this work the authors focus on the integration of the collection of data from sensor devices and their transmission to a cloud based data storage server. In [16] the authors describe the design of an Internet of Things based platform having as main objective the real-time management of energy consumption in water resource recovery facilities and their integration in a future demand side management environment. A very interesting solution is presented in [17] where a low-cost remote monitoring system for dangerous areas based on drones is described that again takes advantage of LoRa and MQTT as the basic technologies. Also, in [18] an open-source earthquake and weather monitoring system is presented based on a Long Range (LoRa)-based star topology with a fully energy-autonomous sensor node.

IoT generates massive amounts of information, and AI helps to make sense of all this data, turning it into predictive findings and prescriptive recommendations [19]. Various works appeared recently that take advantage to this combination. For example in [20] the authors use a class of advanced machine learning techniques, namely deep learning (DL), to facilitate the analytics and learning in the IoT domain. Also, in [21] Akbar et al propose a proactive architecture which exploits historical data using machine learning (ML) for prediction in conjunction with Complex Event Processing (CEP). Finally, an interesting work by Verma et al [5] reviews the state-of-the-art of the analytics network methodologies, which are suitable for real-time IoT analytics.

Finally, fog computing, as an architecture, supports a growing variety of applications, including those in the Internet of Things (IoT), fifth-generation (5G) wireless systems, and embedded artificial intelligence (AI). In [22], Chiang and Zhang summarize the opportunities and challenges of fog, focusing primarily in the networking context of IoT. In [23], the authors state that over time, because of the increasing number of IoT devices, managing them by a fog node has become more complicated. The problem they address in their study is the transmission rate of various IoT devices to a fog node in order to prevent delays in emergency cases. They formulate the decision making problem of a fog node by using a reinforcement learning approach in a smart city as an example of a smart environment and then develop a *Qlearning* algorithm to achieve efficient decisions for IoT transmission rates to the fog node.

Our work aims to offer an edge/fog generic architecture to allow the adoption of edge solutions in IoT deployment in poorly connected and resources limited scenarios. To this end we integrated, using microservices, an MQTT based systems that can collect ingress data, handle their persistency, and coordinate data integration with the cloud using a specific service called aggregator.

3 THE OVERALL ARCHITECTURE

Figure 1 shows the overall architecture of the system we propose. Its basic structure is based on a central node called "aggregator" and on various edge stations. The edge stations have a dedicated channel with the aggregator that is based on LoRa to enable long-range transmissions with low power consumption. On top of LoRa we designed a reliable transport protocol called "LoRaCTP" (LoRa Content Transfer Protocol)[1] that allows to reliably transfer blocks of bytes ("content") bidirectionally and adapting to the quality of the channel; the details of LoRaCTP are outside the scope of in this paper.

Content sent is encrypted using the certificate of the aggregator. The certificate must be manually pre-installed in the edge nodes. This approach guarantees both privacy in the communication and that only the authorized aggregator can handle the incoming data. We understand that this is a minimum level of security but we consider that given the context is enough to define a sufficiently reliable system.

The aggregator coordinates the data flow with the edge nodes using a polling approach. Polling was chosen due to the difficulty in LoRa to properly detect and handle collisions, and because of the necessity, given the low bandwidth offered by LoRa to guarantee transmission reliability to take place at maximum speed. Moreover,

[1]The code of LoRaCTP can be found here: https://github.com/pmanzoni/loractp

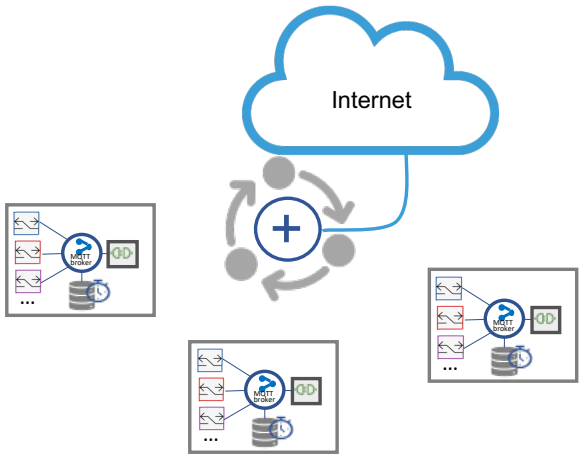


Figure 1: Overall architecture of the proposed system.

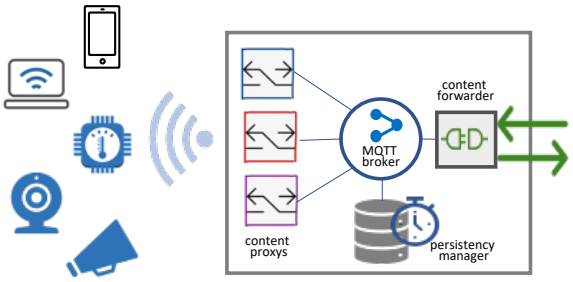


Figure 2: The FUDGE (frugal edge-node) structure.

polling does not require for more complex hardware/software to manage links establishment with the edge devices. overall, this approach allows: (1) a greater flexibility in the use of the channel, (2) the use of sleep mode at the edge node, and (3) ease the handling of disconnection periods.

During polling, once a edge node is selected, a push/pull sequence in followed where first the edge node *pushes* the data to the aggregator and a second phase where the edge nodes *pulls* the data stored in the aggregator. The *content forwarder* is in charge of this task; see Section 4 for details. Basically, in the push phase the content forwarder sends all the content tagged as `Global` that are kept stored by the *consistency manager*. In the pull phase the aggregator sends the content that it kept stored since the previous polling phase. The aggregator returns content whose topics were registered from any service in the edge node. The idea behind this is to give the possibility to local processes to receive both (1) replies to requests sent locally and (2) data from other services in the cloud. An example of the latter can be input data coming from a LoRaWAN network server, like *The Things Networks*[2] but the general idea is to collect data from other MQTT enabled cloud services, like *firebase*[3] or *flespi*[4]. This last part still has to be completely designed.

Finally, as a future idea we'll evaluate the possibility to extend the star topology that we depicted in Figure 1 to a mesh like to distribute the load among various aggregators and to reduce latency.

4 THE FRUGAL EDGE-NODE

In this Section the structure and organization of an edge node is detailed. The edge nodes are called "FUDGE" (FrUgal eDGE node) and their structure is shown in Figure 2. A FUDGE is based on microservices. Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs. This architecture make applications easier to scale and faster to develop. In a FUDGE various microservices can coexist which interchange data with other microservices using an API based on the MQTT pub/sub system.

Three basic elements constitute a FUDGE: an MQTT broker, a persistency manager, and a content forwarder. Beside these three main elements, there are as many "content proxys" as the various *content sources* this FUDGE is handling; more details on the content proxys can be found in Section 5.1.

The MQTT broker is the core of the FUDGE and handles the flow of the content inside the node. The persistency manager element takes care of storing the content that is labeled as persistent in a time-series database. This allows to maintain the temporal evolution of the system while retaining all the required data to handle asynchronous operation or possible disconnections. Moreover, data analytic tools can be used to visualize and extract metrics from the collected data, or implement custom monitoring dashboards. Finally, the content forwarder is in charge of talking with the aggregator using LoRaCTP to interchange all the required content, as described in Section 3.

Since MQTT is at the core of the system, we define a standard format for *topics*. The structure used in the system is as follows:

`<device id>/<service id>/scope/persistency/...others ...`

where:

- `<device id>`: identifies the specific FUDGE node device
- `<service id>`: identifies the service that provides the contents. Content providers can be anything from a simple temperature sensors to cameras, messaging system, ...
- `scope`: indicates whether the content is to be used only by others services locally inside the FUDGE node ('L') or it has to be forwarded to the aggregator ('G')
- `persistency`: used to indicate whether the content has to be handled or not by the persistency manager. A ('P') will make the content become persistent, and a ('N') indicates that no action is required. A third value, ('X'), is used to perform searches in the persistency repository to retrieve data, and uses the `others` fields to indicate whether this is a *request* or a *response* for a search, or other details.
- `others`: more tags can be added if required by the specific service.

The content itself has to be structured using the JSON data-interchange format according to the format in Listing 1.

[2]https://www.thethingsnetwork.org/
[3]https://firebase.google.com/
[4]https://flespi.io/

```
payload = {
    "measurement": <measurement_id>,
    "tags": {
        "tag1": <tag1_value>,
        ...,
        "tagn": <tagn_value>
    },
    "fields": {
        "field1": <field1_value>,
        ...,
        "fieldn": <fieldn_value>
    }
}
```

Listing 1: Structure of the content based on JSON.

The <measurement_id> indicates the specific set of values. There can be various source generating data about the same set, e.g., various weather stations in an area close to the edge node providing the climatic values. The difference between the <tag_value> and the <field_value> is given by the underlying time-series database. Basically, a measurement that changes over time should be a field, and a metadata about the measurement should be a tag, for example the values of pressure and temperature are fields and the name of the weather station a tag.

In the following Section 5 more details are given on how a FUDGE works and about the content proxys.

5 PROTOTYPE IMPLEMENTATION

In this section we describe the details of a first prototype designed to test the feasibility of the FUDGE architecture. As the basic platform we used a Raspberry Pi 3 Model B+ that comes with a CPU Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz, has 1GB SDRAM, and IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE connectivity as main features.

The MQTT broker used was the Eclipse Mosquitto, an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on low power single board computers. The persistency manager used for data storage the support of InfluxDB[5], an open source time series database.

Mosquitto and InfluxDB are executed at start-up as Docker[6] containers. Then, the persistency manager starts by connecting to the InfluxDB server and by subscribing to the generic topic "rpired/#", (in this case the <device id> is rpired). On receiving a message from the broker the code executed was the one shown in Listing 2 (we deleted the low level details).

Basically, the topic of the incoming message is analyzed in search for the request to make its content persistent. If this is the case, a proper database record is created from the payload of the message, and eventually it's stored in the database.

Figure 3 shows the first results of the performance evaluation experiments we did. The figure shows the behavior of the delay when a content proxy generates a random content at a predefined

[5]https://www.influxdata.com/
[6]https://www.docker.com/

```
def on_message(mqttc, userdata, msg):
    top = msg.topic.split('/')
    if (top[3]=='P'): # Checking if data must be persistent
        jrecord = create_json_data(msg.topic, msg.payload)
        clientIX.write_points(jrecord, database=IXDB,
                              protocol='json')
    elif (top[3]=='X') and (top[4]=='request'):
        read_from_db_messapp(msg.topic, msg.payload)
    else:
        pass # nothing to do
```

Listing 2: Snippet of the code of the persistency manager to handle incoming messages.

frequency. In green is shown tha case of a 100 bytes content generated every 5 second, while in red is the case of a 100 KB content generated every 100msec. The time is measured from the moment the content is packed to create the message to be sent to the broker until the persistency manager completed the storing on the message in the DB. The first case can be considered as a very light source of load and was used as a reference; the mean value was 29 msec with a 75-percentile of 30 msec. The second case tried to stress a little the system but the performance stayed stable; the mean value was 84 msec with a 75-percentile of 63 msec.

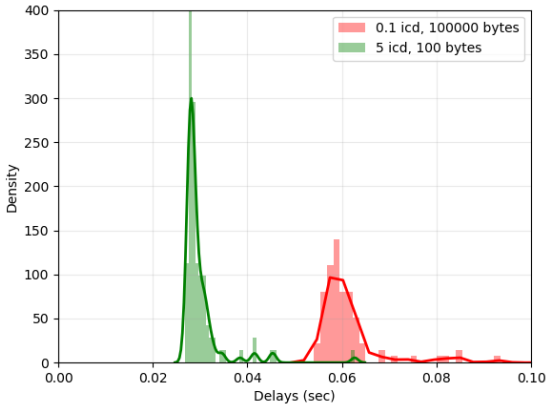


Figure 3: Results of the performance evaluation experiments. In green is shown that case of a 100 bytes content generated every 5 second, while in red is the case of a 100 KB content generated every 100msec.

In case it is a request of a search in the persistency repository ('X') a search in the database is started. Listing 3 shows, as an example, the case for the messaging service, execute calling function in line 8 in Listing 2. First, the request is transformed in a a response. Using the payload the proper SELECT to the database is prepared and issued to the DB server. Values are returned as a list of values, or if no value was found, the special label "NO DATA" is returned.

Finally, as indicate before, data analytic tools can be used to extract metrics that make sense out of the collected data, and monitoring apps through customizable dashboards. Figure 4 shows a

```
r = cIX.query("SELECT * FROM "+
                IXDB+".autogen."+pload["measurement"]+
                " where destination='"+
                pload["tags"]["destination"]+"'")
if len(r) == 0:
    mqttc.publish(topic, "NO DATA")
else:
    points = r.get_points()
    for point in points:
        payload = {
            "measurement": "messapp",
            "tags": {"sender": point['sender'],
                     "time": point['time']},
            "fields": {"message": point['message']}
        }
        jpaylaod = json.dumps(payload)
        mqttc.publish(topic, jpaylaod)
```

Listing 3: Snippet of a search for the messaging service.

```
{
    'data_format': 5,
    'humidity': 48.17,
    'temperature': 22.84,
    'pressure': 1014.65,
    'acceleration': 997.3645271414058,
        'acceleration_x': -44,
        'acceleration_y': 28,
        'acceleration_z': 996,
    'tx_power': 4, 'battery': 2907,
    'movement_counter': 234,
    'measurement_sequence_number': 14039,
    'mac': 'f4af92d97c3a'
}
```

Listing 4: Data provided by a Ruuvi device.

couple of examples of how Grafana[7], executed as a Docker container too, is used to monitoring data from the Ruuvi devices and from the device hosting the FUDGE itself.

5.1 Content proxys

In this development and testing phase we used, as content sources: (1) some Ruuvi[8] environmental sensors, (2) the internal system data from the Raspberry Pi, and (3) a messaging system. The code can be found in the following repository:

`https://github.com/pmanzoni/fudge`

Content proxys are the microservices in charge of getting the data from the various sensors, and publish it to the broker to be handled. The topic structure is indicated in Section 4. The code in Listing 5 is extracted from the Ruuvi's one and shows the basic structure of these services. Basically, data is obtained from the specific, physical source (line 4) and a JSON data structure is created (lines 7-12) including the specific measurement (e.g., "ruuvis"), the tags (e.g., the MAC address of the specific Ruuvi device), and as *fields* the values from the specific device, shown in Listing 4.

```
while True:
    for i in range(len(sensors)):
        # update state from the device
        state = sensors[i].update()

        devid = RTAGS[i]
        payload = {
            "measurement": "ruuvis",
            "tags": {"devid": devid
            },
            "fields": state
        }
        jpaylaod = json.dumps(payload)
        mqttc.publish("rpired/ruuvis/L/P",
                      payload=jpaylaod,
                      qos=0, retain=False)
```

Listing 5: Snippet of the Content proxys for the Ruuvi's device.

6 CONCLUSIONS

In this paper we detailed the design of a generic frugal edge/fog architecture to provide computing services in poorly connected and resources limited scenarios. We outlined the concept of the "aggregator", the process that coordinates the data flow between the services on the edge nodes with those in the cloud. We detailed the structure of the edge nodes, called "FUDGE" (FrUgal eDGE node). FUDGEs are based on microservices to make applications easier to scale and faster to develop. In a FUDGE interchange data using an API based on the MQTT pub/sub system. Data, indicated as *content*, enters locally into the FUDGE through so called "content proxys".

The edge stations have a dedicated channel with the aggregator that is based on LoRa to enable long-range transmissions with low power consumption. On top of LoRa we designed a reliable transport protocol called "LoRaCTP" (LoRa Content Transfer Protocol) that allows to transfer blocks of bytes ("content") adapting to the quality of the channel.

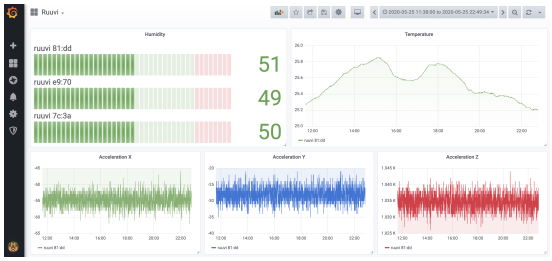
Some details of the implementation aspects were described along with some preliminary results. The initial tests of the architecture indicated that is flexible and robust enough to provide a good platform for the deployment of edge services in remote or rural context.

More work and evaluation are obviously required to determine for example whether this architecture can be adapted to other edge computing solution like for example rCUDA[24].

7 ACKNOWLEDGMENT

This work was partially supported by the "Ministerio de Ciencia, Innovación y Universidades, Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad, Proyectos I+D+I 2018", Spain, under Grant RTI2018-096384-B-I00.

[7] https://grafana.com/
[8] https://ruuvi.com/



(a) Monitoring data from the Ruuvi devices.

(b) Monitoring system data from the FUDGE devices.

Figure 4: Grafana examples for a monitoring app.

REFERENCES

[1] H. Gedawy, K. A. Harras, K. Habak, and M. Hamdi, "Femtoclouds beyond the edge: The overlooked data centers," *IEEE Internet of Things Magazine*, vol. 3, no. 1, pp. 44–49, 2020.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 13–16.

[3] C. C. Byers, "Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for fog-enabled iot networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 14–20, 2017.

[4] C. R. Banbury and et al, "Benchmarking TinyML systems: Challenges and direction," in *arXiv 2003.04821*, 2020.

[5] S. Verma, Y. Kawamoto, Z. M. Fadlullah, H. Nishiyama, and N. Kato, "A survey on network methodologies for real-time analytics of massive iot data and open research issues," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1457–1477, 2017.

[6] W. Ayoub, A. E. Samhat, F. Nouvel, M. Mroue, and J. Prévotet, "Internet of mobile things: Overview of lorawan, dash7, and nb-iot in lpwans standards and supported mobility," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1561–1581, 2019.

[7] B. S. Chaudhari BS, Zennaro M, "LPWAN technologies: Emerging application characteristics, requirements, and design considerations," *Future Internet*, vol. 12, no. 3, 2020.

[8] A. S. Bharadwaj, R. Rego, and A. Chowdhury, "Iot based solid waste management system: A conceptual approach with an architectural solution as a smart city application," in *2016 IEEE Annual India Conference (INDICON)*, 2016, pp. 1–6.

[9] S. Spinsante, G. Ciattaglia, A. Del Campo, D. Perla, D. Pigini, G. Cancellieri, and E. Gambi, "A lora enabled building automation architecture based on mqtt," in *2017 AEIT International Annual Conference*, 2017, pp. 1–5.

[10] S. Penkov, A. Taneva, V. Kalkov, and S. Ahmed, "Industrial network design using low-power wide-area network," in *2017 4th International Conference on Systems and Informatics (ICSAI)*, 2017, pp. 40–44.

[11] M. Niswar, S. Wainalang, A. A. Ilham, Z. Zainuddin, Y. Fujaya, Z. Muslimin, A. W. Paundu, S. Kashihara, and D. Fall, "Iot-based water quality monitoring system for soft-shell crab farming," in *2018 IEEE International Conference on Internet of Things and Intelligence System (IOTAIS)*, 2018, pp. 6–9.

[12] A. Huang, M. Huang, Z. Shao, X. Zhang, D. Wu, and C. Cao, "A practical marine wireless sensor network monitoring system based on lora and mqtt," in *2019 IEEE 2nd International Conference on Electronics Technology (ICET)*, 2019, pp. 330–334.

[13] C. Paolini, H. Adigal, and M. Sarkar, "Upper bound on lora smart metering uplink rate," in *2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC)*, 2020, pp. 1–4.

[14] A. Lachtar, T. Val, and A. Kachouri, "Elderly monitoring system in a smart city environment using lora and mqtt," *IET Wireless Sensor Systems*, vol. 10, no. 2, pp. 70–77, 2020.

[15] A. Bhawiyuga, K. Amron, R. Primanandha, D. P. Kartikasari, H. Arijudin, and D. A. Prabandari, "Lora-mqtt gateway device for supporting sensor-to-cloud data transmission in smart aquaculture iot application," in *2019 International Conference on Sustainable Information Engineering and Technology (SIET)*, 2019, pp. 187–190.

[16] M. Nunes, R. Alves, A. Casaca, P. Póvoa, and J. Botelho, "An internet of things based platform for real-time management of energy consumption in water resource recovery facilities," in *Internet of Things. Information Processing in an Increasingly Connected World*, L. Strous and V. G. Cerf, Eds. Cham: Springer International Publishing, 2019, pp. 121–132.

[17] L. Angrisani, A. Amodio, P. Arpaia, M. Asciolla, A. Bellizzi, F. Bonavolontà, R. Carbone, E. Caputo, G. Karamanolis, V. Martire, M. Marvaso, R. Peirce, A. Picardi, G. Termo, A. M. Toni, G. Viola, and A. Zimmaro, "An innovative air quality monitoring system based on drone and iot enabling technologies," in *2019 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*, 2019, pp. 207–211.

[18] P. Boccadoro, B. Montaruli, and L. A. Grieco, "Quakesense, a lora-compliant earthquake monitoring open system," in *2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2019, pp. 1–8.

[19] C. Stracener, Q. Samelson, J. Mackie, and M. Ihaza, "The internet of things grows artificial intelligence and data sciences," *IT Professional*, vol. 21, no. 3, pp. 55–62, 2019.

[20] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.

[21] A. Akbar, A. Khan, F. Carrez, and K. Moessner, "Predictive analytics for complex iot data streams," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1571–1582, 2017.

[22] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[23] M. Mobasheri, Y. Kim, and W. Kim, "Toward developing fog decision making on the transmission rate of various iot devices based on reinforcement learning," *IEEE Internet of Things Magazine*, vol. 3, no. 1, pp. 38–42, 2020.

[24] C. Reaño, J. Prades, and F. Silla, "Exploring the use of remote gpu virtualization in low-power systems for bioinformatics applications," in *Proceedings of the 47th International Conference on Parallel Processing Companion*, ser. ICPP '18. New York, NY, USA: Association for Computing Machinery, 2018.