sensors

MDPI

Article

Design and Implementation of a Framework for Smart Home Automation Based on Cellular IoT, MQTT, and Serverless Functions

Marco Esposito , Alberto Belli , Lorenzo Palma and Paola Pierleoni *

Department of Information Engineering (DII), Università Politecnica delle Marche, 60131 Ancona, Italy
* Correspondence: p.pierleoni@univpm.it; Tel.: +39-0712204847

Abstract: Smart objects and home automation tools are becoming increasingly popular, and the number of smart devices that each dedicated application has to manage is increasing accordingly. The emergence of technologies such as serverless computing and dedicated machine-to-machine communication protocols represents a valuable opportunity to facilitate management of smart objects and replicability of new solutions. The aim of this paper is to propose a framework for home automation applications that can be applied to control and monitor any appliance or object in a smart home environment. The proposed framework makes use of a dedicated messages-exchange protocol based on MQTT and cloud-deployed serverless functions. Furthermore, a vocal command interface is implemented to let users control the smart object with vocal interactions, greatly increasing the accessibility and intuitiveness of the proposed solution. A smart object, namely a smart kitchen fan extractor system, was developed, prototyped, and tested to illustrate the viability of the proposed solution. The smart object is equipped with a narrowband IoT (NB-IoT) module to send and receive commands to and from the cloud. In order to evaluate the performance of the proposed solution, the suitability of NB-IoT for the transmission of MQTT messages was evaluated. The results show how NB-IoT has an acceptable latency performance despite some minimal packet loss.

Keywords: internet of things; home automation; smart appliance; narrowband IoT; MQTT; serverless

check for updates

Citation: Esposito, M.; Belli, A.; Palma, L.; Pierleoni, P. Design and Implementation of a Framework for Smart Home Automation Based on Cellular IoT, MQTT, and Serverless Functions. *Sensors* **2023**, *23*, 4459. https://doi.org/10.3390/s23094459

Academic Editors: Antonio Puliafito, Giovanni Pau and Ilsun You

Received: 2 March 2023
Revised: 12 April 2023
Accepted: 29 April 2023
Published: 3 May 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Completely automated smart homes are on the way to becoming a well-established reality. The ever increasing number of smart objects in the smart home environment requires the definition of standardized and flexible protocols for commands and state information exchange, besides optimal strategies for processing a large number of commands. New technologies such as serverless computing and ad hoc communication protocols can be leveraged to manage a large fleet of smart objects, while also ensuring good accessibility and intuitive user interfaces. Serverless computing is an emerging cloud-computing paradigm that represents a promising development for many application fields, from micro- and web-services [1], to the the internet of things (IoT) [2,3], edge computing [4] and artificial intelligence (AI) [5], or even for domain-specific scientific applications [6,7]. Serverless computing operates at the crossing point of different models that have emerged alongside cloud computing, such as function as a service (FaaS) and pay-per-use models, and technologies such as event-driven programming, virtualization, and containerization [4]. Serverless computing comes with automated scalability and resource provisioning, meaning that resources assigned by the serverless platform scale depending on the number of functions that are running and the resource demand of each of those functions. These characteristics make serverless particularly appealing for applications that need to manage a large pool of users or, in the field of the internet of things and smart objects, a large number of devices [2].

The pioneering commercial platform for serverless computing was Lambda [8], developed by Amazon Web Services (AWS), the Amazon cloud platform. Lambda remains one of the most popular platforms for the development of serverless applications. A recent work by the authors of this paper [9] showed how AWS satisfies many key performance indicators in IoT applications, also compared to other competitors such as Microsoft Azure and Google cloud Platform, both of which now also offer serverless and FaaS services. Lambda also comes with extensive integration with Amazon Alexa [10], the voice assistant on Amazon Echo. Voice interfaces are seeing a wider use in different application fields, from voice assistants to home automation, as they constitute a more natural and intuitive interface to interact with smart objects and environments, as well as improving accessibility [11].

This paper proposes and experimentally evaluates a framework that represents a synergistic union of different cloud-computing and IoT-related technologies that can enable accessible and automated home environments. The framework is easy to replicate, is scalable, and is transparent to the underlying smart object or appliance that uses it due to the use of serverless technologies and a custom-defined command exchange protocol based on the message queue telemetry transport (MQTT) protocol. MQTT [12] is the main middleware service used in most cloud platforms for the IoT, therefore simplifying the interfacing of IoT devices and smart objects with the cloud. Compared to other protocols such as HTTP, which is also employed in many serverless solutions [3], MQTT messages have a small overhead that makes the protocol suitable for machine-to-machine applications. Furthermore, MQTT does not impose a format for its payload, making it extremely flexible [13], and the use of topics adds a further degree of freedom for the implementation of custom applications.

As the introduction of new communication technologies in such applications requires careful evaluation in terms of the overall application latency, the aim of this paper is to evaluate the performance of a framework for smart home appliances equipped with narrowband IoT (NB-IoT) modules to send and receive messages to/from the cloud. NB-IoT is a low-power wide area network (LPWAN) protocol by the Third-Generation Partnership Project (3GPP), designed to provide wide connectivity to a huge number of IoT devices in massive deployment scenarios and in scenarios that require good radio coverage and lower power consumption, such as smart metering [14,15]. NB-IoT devices can reach nominal life times of around 10 years with small and infrequent messaging [16], with an overall good power consumption performance compared to other communication technologies, both cellular and not [17]. Cellular technologies such as NB-IoT can be employed to embed additional services that require internet connectivity in home and kitchen appliances, adding negligible power consumption to their normal mode of operation and advancing the machines as a service (MaaS) model [18,19] and servitization of the products [20].

The framework proposed in this paper includes a smart IoT device making use of the aforementioned MQTT-based command protocol, and a serverless application on the cloud to manage devices and commands received through Amazon Alexa or through a dedicated application. The NB-IoT protocol was tested to determine if it is suitable to dispatch MQTT messages between MQTT clients. Automatic speech recognition (ASR) software already adds a delay to the overall application latency, and the execution of Lambda functions also adds a (minimal) latency, but the message exchange delay might severely degrade the user experience if it hinders the prompt execution of commands. For the demonstration of the proposed framework viability, a smart extractor fan system was designed, prototyped, and tested as a practical use case. An experimental test-bed with a NB-IoT node sending and receiving MQTT messages was therefore set up to evaluate the latency performance of the protocol in this context.

This work is organized as follows. Section 2 explains the main technologies employed in the proposed system, and illustrates some recent works in the field of home automation and vocal interfaces for smart homes. Section 3 goes into detail about the proposed system and the NB-IoT evaluation test-bed, while Section 5 gives the results of the evaluation. Lastly, Section 6 draws conclusions about the work and contributions of this paper.

2. Background

Many smart home automation papers propose APIs or practical implementations of smart device use cases, employing protocols ranging from HTTP to MQTT to CoAP, which has also shown potential for home automation applications [21]. Crisan et al. [22] proposed and developed an API for smart home automation called qtoggle to control any IoT device in the smart home environment with a TCP/IP stack via HTTP requests, regardless of the type of smart object or device. Sarkar et al. [23] proposed a framework to manage IoT devices in smart buildings using the serverless paradigm. Their proposed system consists of gas and humidity sensors and smart bulbs, and the serverless platform of choice was the OpenFaas platform. Froiz-Míguez et al. [24] proposed a framework for fog computing in smart homes called Ziwi, which enables the coexistence of Zigbee and Wifi devices on the same smart home framework. MQTT is used for its small code footprint and for addressing compatibility between smart devices. The increasing usage of MQTT in machine-to-machine applications has been stressed by Biswajeeban et al. [25], stemming from its lightweight design and its publish/subscribe architecture. Most frameworks employing this protocol require the implementation of security mechanisms (such as TLS) that add some extra overhead to the communication. As research on the topic of security in MQTT is increasing [26,27], various recent papers have also focused on the implementation of new security mechanisms designed specifically for MQTT-based smart home environments [28,29].

Different recent works explored the potentiality of integrating Amazon Alexa or, more generally, voice services into the smart environment to enable interactive and accessible interfaces. Iliev et al. [11] proposes a distributed framework for smart homes that uses natural language processing (NLP) to develop a human/smart environment interface. Their solution introduces a fog node for the deployment of intelligent user interface (IUI) software to reduce the voice interface response delay and to decrease bandwidth load. Krishnamurthi et al. [30] analyzed different data fusion and techniques for work offloading in IoT applications, such as fog and edge computing, which can find use in different IoT fields, including smart homes. The interface is designed to understand domain-specific natural language to support under-resourced languages that lack available ASR software, such as Amazon Alexa. Lee et al. [31] explored the possibility of using Amazon Alexa to build smart home services for the elderly or to enhance the effectiveness of existing services such as "carebots". Bodgan et al. [32] proposed a framework that integrates voice assistants into smart offices for various tasks such as ambient control and interaction with various services.

This paper proposes a very flexible framework for smart home automation that embeds a voice interface, while in addition evaluating the use of cellular technologies for the dispatching of MQTT messages for commands and status updates. The introduction of cellular communication in these frameworks, if carefully evaluated, can aid in the implementation of different services in home and kitchen appliances, enabling, for example, predictive maintenance and remote monitoring services to be built on top of said frameworks. The flexibility of the serverless model and the use of MQTT, which is the main middleware in the cloud technologies described above, allowed for the integration of the different services provided by AWS, including voice interaction services. Furthermore, NB-IoT has been evaluated to test its efficiency in serving a framework based on MQTT. The remaining part of this section describes the technologies used for the design, testing, and evaluation of the proposed framework, embedding a smart extractor fan prototype. The details of the implementation and fusion of the aforementioned technologies are detailed in Section 3.

2.1. Narrowband IoT

NB-IoT is a cellular technology based on long-term evolution (LTE) introduced by 3GPP. NB-IoT reduces the LTE protocol stack functionalities to make it suitable for IoT applications. A brief comparison of NB-IoT with other LPWAN communication protocols aimed at comparable applications is in Table 1. Compared to protocols such as LoRaWAN and Sigfox, NB-IoT works in LTE bands and does not require the use of a gateway. It provides extended coverage to delay-tolerant, low-cost, low-energy IoT applications, reaching good energy performance even in the worse-case, continuous transmission scenarios when compared to other communication protocols [17]. It represents a valuable choice for application fields that require massive deployments, good coverage and low data rates, including smart metering [33], smart grids [34], smart cities [35], healthcare [36–38], and industrial applications [39]. To the best of the authors' knowledge, it has still been seldom applied in home automation [40], mostly in home surveillance or metering applications [41,42], and rarely using MQTT. Cellular connectivity provides opportunities for enterprises to embed additional services into home appliances [20]. Furthermore, NB-IoT will be used in the machine-to-machine communication use case in 5G networks and will coexist with other services in the 5G environment [43]. While some works exist on the application layer performance over NB-IoT [44,45], this work mainly focuses on latency issues for the smart home use case, with the goal of ensuring a good user experience for the vocal interaction subsystem.

Table 1. Comparison of communication protocols for IoT and machine-to-machine applications.

	NB-IoT	**LoRaWAN**	**Sigfox**	**LTE-M**
Spectrum	Cellular (LTE)	ISM	ISM	Cellular (LTE)
Data Rate [46,47]	∼200 kbps	∼0.3–100 kbps	∼100–600 bps	∼1 Mbps
Payload (bytes) [47,48]	1600	243	12	1000
Tx Power (dBm) [48]	23	13	14	23
Consumption [48]	Medium low	Low	Very Low	Medium

2.2. MQTT

MQTT is an OASIS standard messaging protocol designed to be lightweight and to have a small code footprint [12]. MQTT is one the main protocols that is used for the IoT and for the implementation of middleware services, alongside COAP, AMQP and HTTP [49]. It is nowadays used in many application scenarios, such as smart homes and smart buildings [50,51], smart cities [52], smart grids and energy management [53,54], smart agriculture [55], environmental monitoring and early warning [56], and so on. Furthermore, the main cloud platforms for the IoT use MQTT as their middleware service and brokering protocol [9].

MQTT is a publish/subscribe message protocol that decouples data producers and data consumers through the use of a central entity called "broker", a server responsible for dispatching messages among MQTT clients. brokers contain a list of topics, which are strings that are organized hierarchically. The MQTT client can either subscribe to a topic, publish a message on a topic, or both. When a client publishes a message on a certain topic, the broker will forward it to every client that has subscribed to that topic. MQTT includes a series of security mechanisms, including password and username authentication, and authentication with transport layer certificate exchange. Data encryption can be guaranteed with protocols such as the transport layer security protocol (TLS), as is performed in AWS, or it can be implemented at the application layer to provide end-to-end encryption.

2.3. Amazon Web Services

Amazon Web Services is one of the most famous and oldest cloud platforms, offering services ranging from databases and data analytics to internet of things and mobile computing, besides many services dedicated to enterprises [57]. The smart appliance system proposed in this paper makes use of AWS IoT core, AWS Lambda, and two data storage services, AWS S3 and DynamoDB. Furthermore, Amazon Cloudwatch, a service used to monitor AWS resources and applications, is used to keep track of the performance and cost of each deployed service.

2.3.1. AWS IoT Core

AWS IoT core is the AWS platform for the internet of things. It allows devices to connect to the cloud and other AWS services. The architecture of AWS IoT core is described in Figure 1. The central component of the architecture is the message broker, which dispatches messages to and from things (i.e., IoT devices) and other AWS services and user applications. AWS IoT core supports MQTT and MQTT over Websockets, but clients can also publish messages on the broker using HTTP. A rules engine is available to automatically process messages and to generate triggers. Each IoT device has an associated "shadow" to keep track of its status, and shadows are stored in a corresponding registry. Various security and identity mechanisms are employed to guarantee secure communication between the broker and devices, including TLS data encryption [58] and secure authentication with X.509 certificates [59].

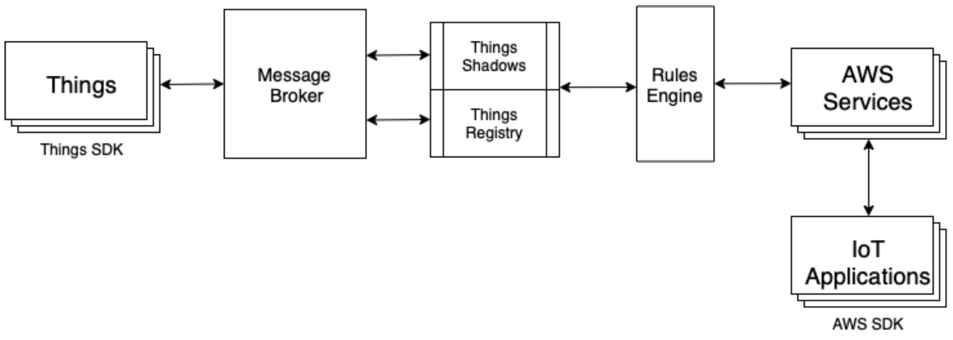


Figure 1. Architecture of AWS IoT core (adapted from [9]).

2.3.2. AWS Lambda

AWS Lambda is a serverless computing platform that lets users deploy applications and run code without managing servers, delegating to Lambda all the server and operating system maintenance, including capacity provisioning and automatic scaling of resources, while leaving the code as the sole responsibility of the developer. Lambda can be used to build IoT backends and countless other applications. Lambda functions are event-driven, meaning that they are instantiated to manage events. Each function is usually linked to an event source that generates triggers and will contain a series of handlers to manage those triggers. Various AWS services are event sources and can be configured to trigger Lambda functions, including Amazon Alexa skills and AWS IoT core rules.

2.4. Amazon Alexa

Amazon Alexa skills employ ASR and natural language understanding technology [60] for the development of custom applications that take vocal commands as input, convert them into text, interpret them to execute various actions, and possibly return a vocal response to the user that invokes the application. Alexa skills can be configured to send messages to various endpoints, including Lambda functions, essentially turning vocal commands into Lambda triggers.

The components of an Alexa skill invocation are the following:

- A wake word, such as "Alexa", to turn on the device;
- An invocation name, the de facto name of the Alexa skill that is being invoked;
- An utterance, telling the Alexa skill what action to execute, or, in Alexa skill jargon, which "intent" is to be called;
- Optionally, a series of custom "slots", which are additional variables that are being passed to the intent.

Intents produce a message in JSON format that is then forwarded to a Lambda endpoint. The Lambda function will parse the JSON message content, call a dedicated handler based on the intent, execute the handler's routine, and lastly produce a JSON response message that is sent back to the Alexa skill.

3. Materials and Methods

The design proposed in this paper is applicable to any kitchen or home appliance due to the flexibility of the MQTT protocol and the serverless paradigm. It provides a valuable framework for the management of smart appliances that can be easily replicated in any smart home system. It leverages technologies in the field of serverless and machine-to-machine communication to create interactive smart objects with dedicated on-demand services, with good accessibility due to the use of vocal interfaces.

A description of the proposed framework for a smart appliance system follows in this section, and in particular, the design of the hardware and the serverless application for a smart kitchen extractor fan are outlined. The framework is completely application agnostic and can be replicated with any kitchen appliance or smart home device. The cloud-developed application makes use of Amazon Alexa to build a vocal interface and provides touchless interaction with the smart object. A phone application acting as an MQTT publisher can also be used to send commands to the appliance. MQTT makes the framework extremely flexible, and it makes the integration with cloud platforms (in this case, with AWS) simple and secure due to the use of certificates securely stored on the device by the use of a cryptochip. MQTT is also used to receive status updates from the smart object. The serverless framework provides efficient replicability and scalability properties to the application, and most importantly, it rationalizes resource allocation on the cloud when managing a large number of devices.

3.1. Smart Extractor Fan Design

The kitchen extractor fan is controlled by a device whose block diagram is illustrated in Figure 2. It consists of a microcontroller (referred to as "micro-host") that interfaces with various peripherals and controls the ventilation motor and a set of lights. A NB-IoT chip is used to receive commands from the cloud in MQTT format over NB-IoT. The MQTT protocol guarantees a simple and secure interface toward the cloud through the native use of TLS encryption and authentication. The AWS IoT core requires authentication over MQTT with X.509 certificates. A cryptochip is used to securely connect and authenticate the AWS IoT core. Such a device can securely store certificates on the IoT device and authenticate the AWS IoT core.

The device receives messages from the cloud, parses them, and executes commands based on the content of their payload. It also sends state updates toward the cloud. Firmware updates can be received over the air using MQTT, simplifying the deployment of new firmware on large fleets of devices.

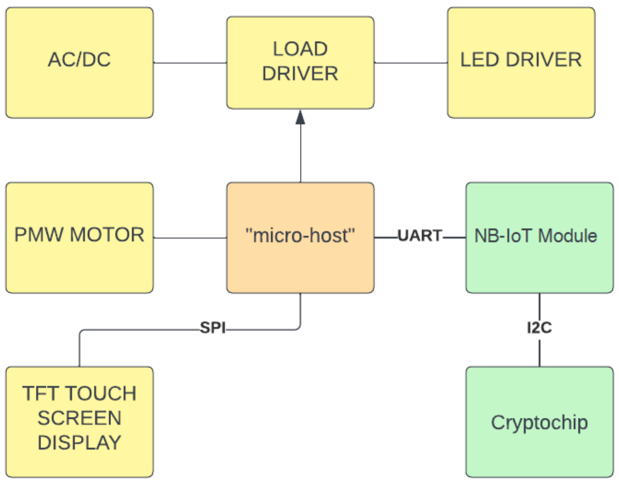


Figure 2. Block diagram of the smart extractor fan.

3.2. Architecture of the Serverless Cloud Application

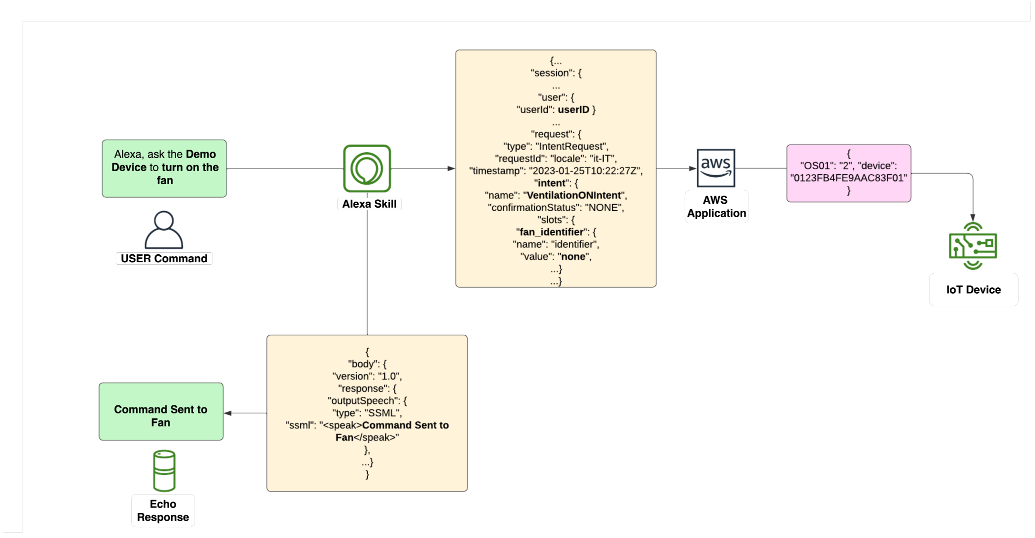
An application was designed for the management and control of smart objects. The application is developed using AWS Lambda and the MQTT protocol to provide flexible interaction between the users and the extractor fan. Amazon Alexa is used to implement a vocal interface for the control of the smart object. State updates are also sent by the fan using the custom commands protocol and are processed by a dedicated Lambda function on the cloud to keep track of the state of each device.

The application was developed on AWS, and the serverless platform of choice was Amazon Lambda. Two Lambda functions were implemented in NodeJS (JavaScript) using the AWS SDK: one to forward user commands to devices, either by an application or Alexa skill, and the other to handle state updates from the devices. JavaScript, being a high-level interpreted language, lends itself quite naturally to the development of serverless or FaaS applications and, alongside Python, is one of the most used languages for the development of serverless applications [3]. Lambda functions comprised one or more handlers that are used to manage triggers, which can be received by various outside services, including Amazon Alexa or AWS IoT core. Each handler was defined on the basis of the custom command protocol and of the intents that were implemented in the Alexa skill. This way, users are able to control and monitor all the admissible states of the devices.

The project uses two AWS storage services: Amazon S3 and Amazon DynamoDB. DynamoDB is used to store devices states and devices and user identifiers. Similarly to the approach used in [61] to store Lambda code, in this project, Amazon S3 is used to store the device firmware. This way, each device can download any firmware version at any moment, greatly simplifying software updates and the deployment of new code. A software update can be initiated by sending the url of the S3 bucket containing the new firmware inside an MQTT command. The device will then download the file from the S3 bucket and execute the firmware update in autonomy.

An Alexa skill has been developed to interact with the devices. The endpoint of the skill is Amazon Lambda. A Lambda function is triggered when a command is invoked by the user. The Alexa skill takes care of translating input vocal commands into text and then translates text into JSON inputs that can be parsed by Lambda functions. Each JSON input produced by the Alexa skill also contains a user identifier that is used by Lambda to retrieve (from DynamoDB) the device ID associated with the user that invoked the skill. This mechanism guarantees that Lambda functions are only able to control the fans corresponding to the user that invoked the Alexa skill from their personal account. Custom slots are used to define a series of extra identifiers to distinguish between different fans when a user owns or controls more than a single fan. Figure 3 illustrates the message flow of the application. Green boxes indicate vocal commands and responses, yellow boxes are messages in JSON format generated by the Alexa skill, and the pink box indicates the payload of the MQTT message generated by the AWS application and forwarded to the device.

The complete description of the smart appliance system is illustrated in Figure 4. When a user invokes the Alexa skill with a valid intent, i.e., an Alexa skill intent for which a handler was defined, the skill triggers Lambda and passes a JSON message to it, containing both the UserID and a string identifying the handler required to manage that request. The Lambda function queries DynamoDB for the identifier(s) of the device(s) owned by that user. If the user owns more than one device, then a custom slot in the Alexa intent is used to distinguish between their multiple devices and retrieve the correct identifier from DynamoDB. Once Lambda knows the device to send the command to, and on the basis of the intent, it publishes an MQTT message on AWS IoT core following the commands protocol for that specific device. The device, having subscribed to its individual topics at start-up, will receive the message, parse its payload, and execute a command according to its content. Whenever it receives a command, the device will also publish an update on its status on its dedicated OutState topic. It is also possible to configure a “refresh time” at the monitoring topics, i.e., a time at which the devices sends periodic updates on the topic. An AWS IoT core rule is implemented to handle state updates and to invoke a second Lambda function, which updates a DynamoDB table with the new state information. The state information is used to monitor devices and their activities, but it also allows users to ask the Alexa skill for the state of the fan, and it uses "feedback" commands that require the Lambda function to know the current state of the device (e.g., "lower the ventilation", "raise the ventilation", and similar commands). The implementation of feedback intents makes the communication between the smart appliance and the user feel more "natural". At the same time, periodic refreshes at short intervals amplify the application load on the cloud and the number of transmissions, which might not be suitable for certain devices or communication protocols. For such use cases, it might be required to set a longer refresh time or to only enable refreshes when there is a state update via outside command. The refresh option has been used for the experimental evaluation of the smart object work load on the cloud.


Alexa, ask the Demo Device to turn on the fan
USER Command
Alexa Skill
AWS Application
{ "OS01": "2", "device": "0123FB4FE9AAC83F01" }
IoT Device
Command Sent to Fan
Echo Response

```
{...
"session": {
...
"user": {
"userId": userID }
...
"request": {
"type": "IntentRequest",
"requestId": "locale": "it-IT",
"timestamp": "2023-01-25T10:22:27Z",
"intent": {
"name": "VentilationONIntent",
"confirmationStatus": "NONE",
"slots": {
"fan_identifier": {
"name": "identifier",
"value": "none",
...}
...}
```

```
{
"body": {
"version": "1.0",
"response": {
"outputSpeech": {
"type": "SSML",
"ssml": "<speak>Command Sent to Fan</speak>"
},
...}
}
```

Figure 3. Messages flow from the user command up to the MQTT message generated by AWS that is forwarded to the device.

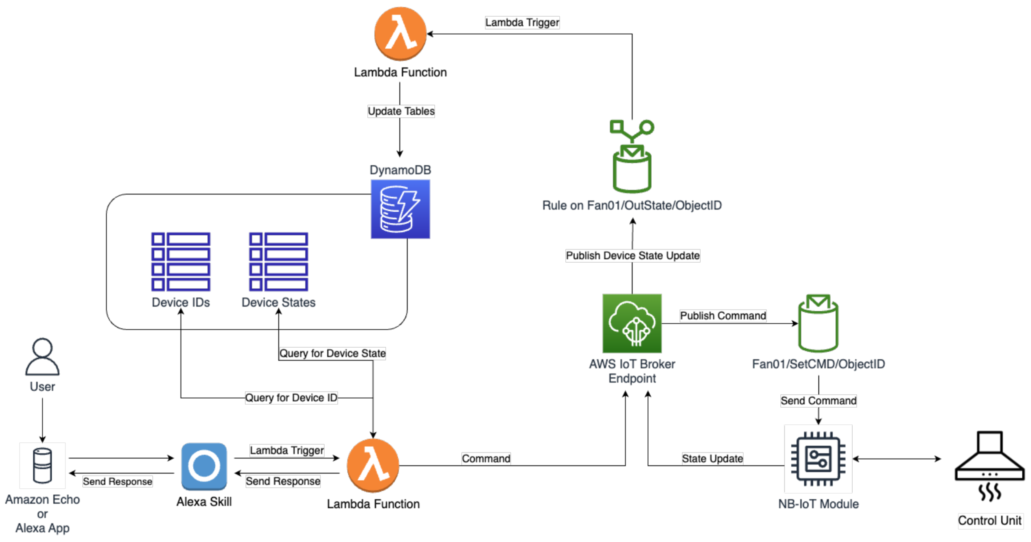


Figure 4. Architecture of the proposed smart fan application.

4. Experimental Evaluation

An experiment was carried out to evaluate the use of narrowband IoT to dispatch MQTT messages across MQTT clients. The goal of the experiment was to provide an estimation of the time needed to forward a message to/from an MQTT client over NB-IoT to show whether the protocol is suitable for applications, such as the one developed in this paper, that require an acceptable latency performance to ensure a good user experience.

The experiment involves the following nodes

- An MQTT Client (Client 1) that publishes messages on a dedicated topic named Topic1;
- An MQTT broker;
- A NB-IoT module subscribed to Topic1 and publishing on a dedicated topic named Topic2;
- A second MQTT Client (Client 2) subscribed to Topic2.

This experimental setup is shown in Figure 5. The two MQTT clients might represent cloud applications sending or receiving messages from a smart object, such as vocal commands or state updates, while the NB-IoT device stands for the smart object itself. This setup reflects the interaction between users and the smart extractor fan system, where client 1 might be a Lambda function publishing a command on the broker to control the NB-IoT module/smart fan, which will then immediately publish a status update on the OutState topic once it sets the correct output variables.

At the start of the evaluation, the board registers to the NB-IoT network, connects to the MQTT broker, and it subscribes to Topic1. At the same time, the two clients also establish a connection to the MQTT broker, and Client 2 subscribes to its assigned topic. After waiting a short time to ensure that each MQTT client has successfully connected to the broker, Client 1 will publish a first message on Topic1, containing a timestamp $T1$ representing the start of the communication. The broker then forwards the message to the evaluation board. The board will then publish a new message on Topic2 containing $T1$ and the time elapsed between reception of the first message from the broker and the transmission of the second message. The broker will forward this message to Client 2, which will save the timestamp of reception ($T2$). This information, alongside the one obtained from the first client, the board, and the broker, is used to evaluate latency at different sections of the communication. The procedure is repeated multiple times at intervals of 10 s between each message.

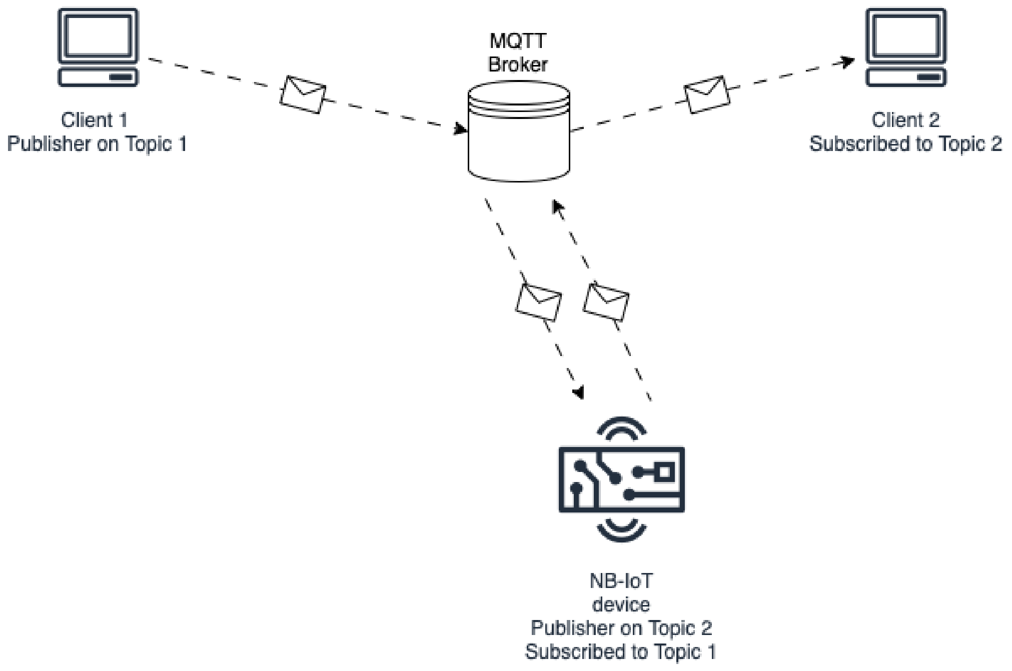


Figure 5. Diagram of the evaluation testbed used to evaluate NB-IoT latency for dispatching MQTT messages.

The two MQTT clients were implemented on a computer using the Paho-MQTT [62] Python library. The MQTT broker was deployed on a different machine in the same network. The broker of choice was the Eclipse' Mosquitto [63] broker. The MQTT clients are synchronized using the network time protocol. The NB-IoT module is connected to a mobile IoT network, which provides acceptable network coverage and reliability. In debug mode, Mosquitto does not provide a precision for packets logging below the second. However, the time difference between the transmission of the packet and its reception on the broker is not higher than a second, as verified on the Mosquitto log file. Therefore, considering symmetrical links and a negligible delay for the communication between the clients and the broker on the same network, the communication delay between the broker and the smart object was estimated as

$$T_{NB-IoT} = \frac{T_2 - T_1 - \Delta}{2},$$

in which Δ represents the time elapsing between the reception of the message on the board and the transmission of a message toward the broker.

5. Results

The performance of the developed smart fans was monitored using Amazon Cloudwatch and forcing a situation with frequent calls to Lambda. By setting a refresh time of 1 min for the OutState topic, the Lambda function responsible for updating the state database is called once every minute by the AWS IoT rule that handles messages received on the OutState topic. The function simply parses the JSON message forwarded to it by AWS IoT core, containing the new states and the identifier of the smart object, and updates a DynamoDB table with this new information, also increasing a usage atomic counter on the table to keep track of the usage of each smart device. Such a frequent refresh time might be incompatible with NB-IoT, given the high transmission load compared to the relatively low throughput expected of applications using this protocol, but more suitable to WiFi implementation. Furthermore, configuring the device so that the OutState topic is refreshed only when there is a state variable change is an implementation choice with an acceptable load for NB-IoT, and possibly a more realistic work condition for the smart home object. In the case of the Cloudwatch evaluation, the goal was to evaluate a situation with frequent calls to the serverless Lambda function; therefore, the number of state updates was incremented.

The results of the evaluation are shown in Figures 6 and 7. The device was left on for about 24 h and the Cloudwatch Lambda insights service was used to keep track of various parameters. Figure 6 shows the median, 90th percentile, and maximum duration of Lambda function calls, computed in intervals of 5 min. Figure 7 shows the maximum CPU usage time and network load during 24 h, computed in 5-min intervals.

The spikes in duration correspond to "cold starts" that were also recorded by Cloudwatch. A cold start is the time required to allocate cloud resources to instantiate a Lambda function environment for the first time. Each Lambda function is only kept "warm" for a certain amount of time, requiring a new, prolonged startup to instantiate a new execution environment once the execution environment turns "cold". Setting up provisioned concurrency can help mitigate the effect of cold starts when there are multiple devices invoking the Lambda function.

The results of the experimental evaluation carried out to evaluate the use of narrowband IoT to send MQTT messages through MQTT clients are shown in Figure 8. The delay has a mean value of 0.447 s, with occasional spikes of up to 11.41 s. The 90th percentile is 0.527 s, and the standard deviation is 0.938 s. The contribution of Δ is negligible and has minimal variations across multiple repetitions.

To investigate the reason behind the spikes, the experiment was repeated keeping track of packet numbers and MQTT acknowledgment (ACK) messages received by the board. Having set the MQTT quality of service to 1, the MQTT broker forwards an ACK message toward the board whenever a message has been published correctly. The NB-IoT board firmware was therefore slightly modified so that, after an ACK is received from the broker, a second MQTT message containing the current ACK number and Δ is also immediately published on Topic2. Client 2 therefore keeps track of the packet numbers, their associated timestamps, and the ACK numbers, besides computing the time difference between the timestamp of reception and the timestamp sent by Client 1, as in the previous experimental setup. The new experiment flow is detailed in Figure 9.

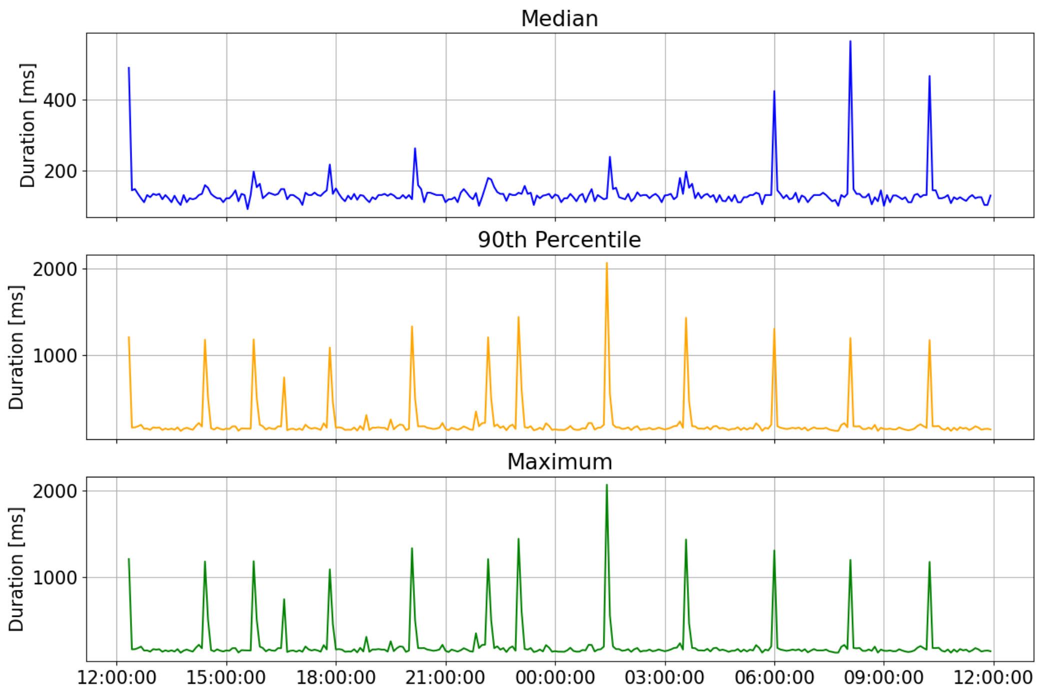


Figure 6. Median, 90th percentile, and maximum duration of a lambda function calls, computed in intervals of 5 min.

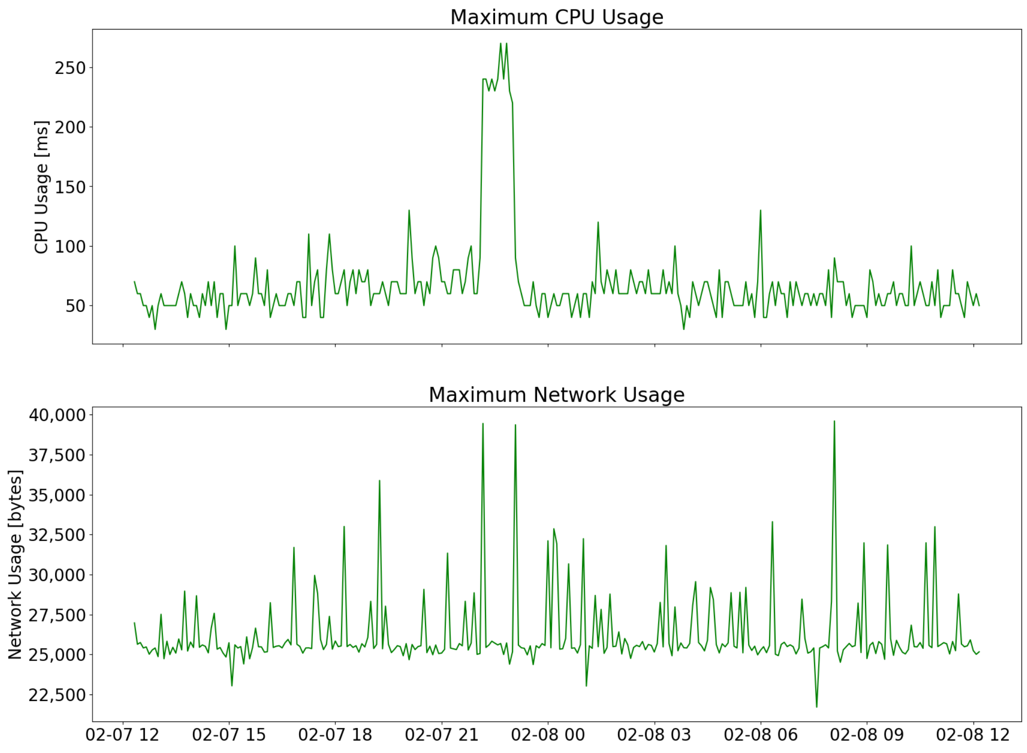


Figure 7. Maximum CPU (**top**) and network (**bottom**) usage during the Lambda function testing, computed in intervals of 5 min.

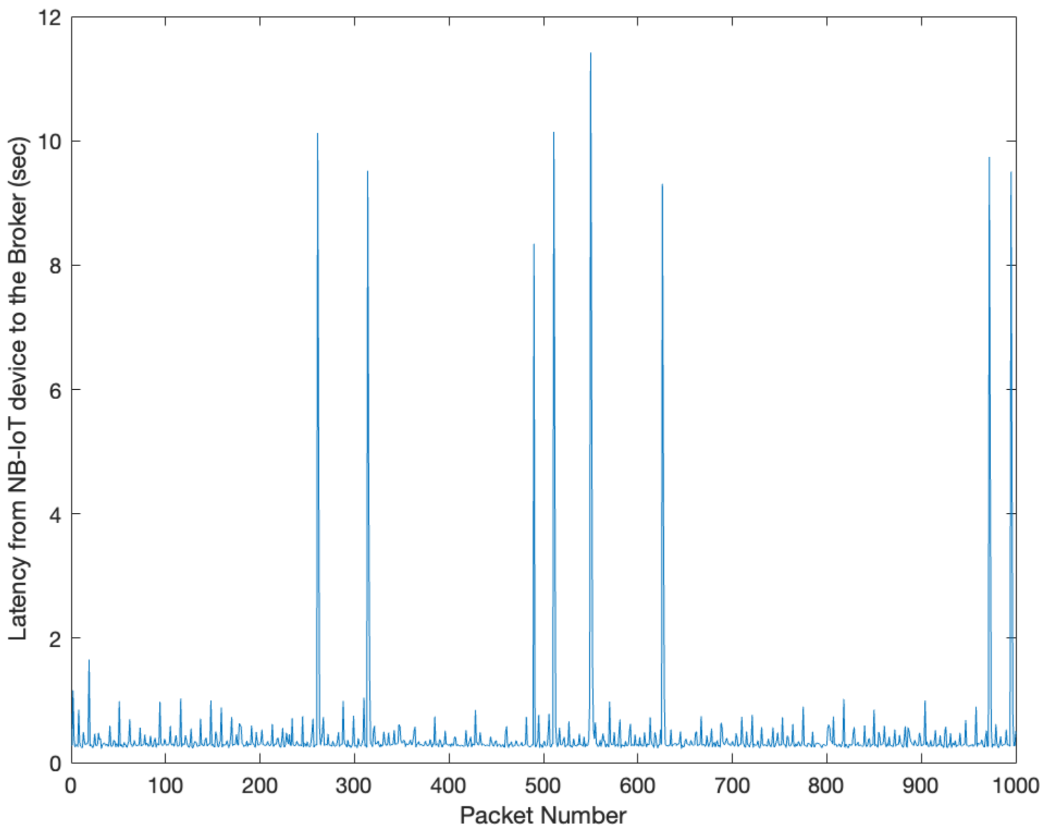


Figure 8. Estimated latency from the NB-IoT device to the MQTT broker.

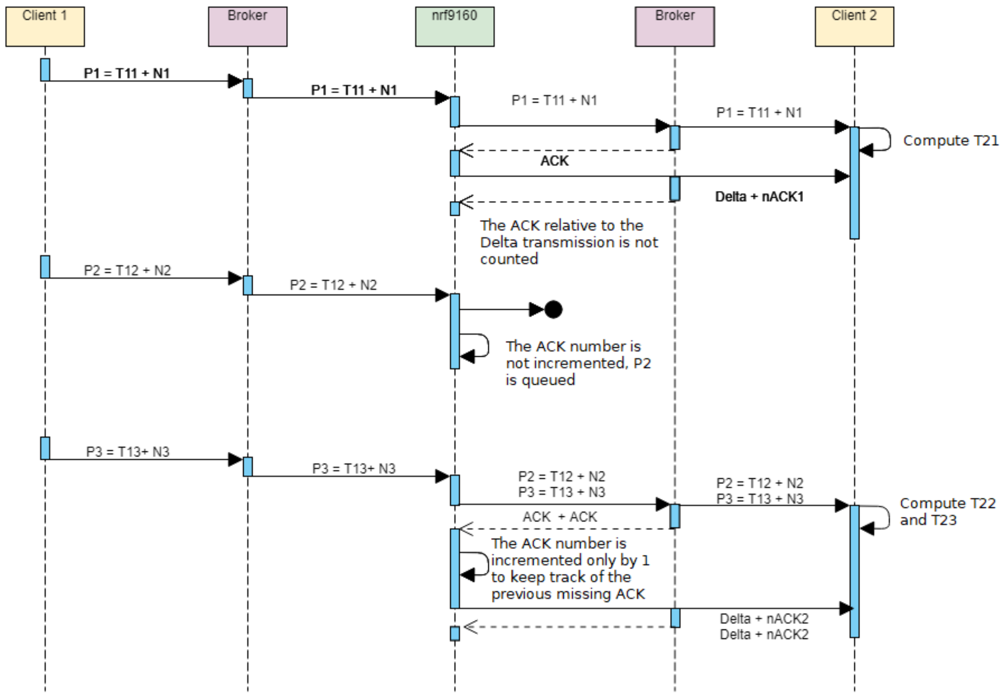


Figure 9. Sequence diagram of the experiment used to investigate latency spikes by keeping track of packet numbers and ACK messages.

The iteration of the experiment described in the Figure comprises the following steps. At first, a packet P1 containing a timestamp T1 and packet number $N1 = 1$ is sent to the broker. P1 is then forwarded by the NB-IoT board and is correctly acknowledged; therefore, nACK1 is also set to 1 and sent to Client 2. The difference between T1 and the timestamp of reception (T2) is computed by Client 2 and is referred to as T21 in the figure. The delay in this case is below a reasonable value, i.e., not corresponding to a latency spike. During the following transmission, no ACK is received for packet P2, and thus, nACK is not increased and P2 is reinserted in the output queue. At the third transmission, the board sends both P2 and the new P3 packet and receives an ACK for both messages. nACK for P3 is not set to 3 in this case to keep track of the previously lost ACK message, meaning that both P2 and P3 have an associated $n\text{ACK} = n\text{ACK2} = 2$. By looking at log entries on Client 2 where a packet has an associated packet number that is different from its ACK number, it is therefore possible to find packets for which the previous ACK message was not received by the board.

These packets also correspond to spikes in latency in the resulting latency graph, meaning that the spikes are most likely due to the development of an output queue in the NB-IoT board. The loss of an ACK message could be because the publish message did not reach the broker or the ACK message from the broker was lost in transmission. In this last case, since MQTT QoS is set to 1, there is no guarantee that each message will not be forwarded twice toward Client 2. Client 2 does not receive any duplicate message, meaning that the reason for a lack of ACK messages on the NB-IoT board is most likely because the original publish message traveling on the NB-IoT network was lost. The average loss across the two experiments is about 0.8%. The latency for the second experiment was in line with the previous one, even if slightly lower (0.3845 s).

6. Conclusions

In this paper, a generalized and flexible framework for smart object applications in a smart home environment was proposed, leveraging MQTT, the serverless computing platform Amazon Lambda, Amazon ASK, and NB-IoT. The framework consists of smart objects receiving messages from the cloud through a dedicated message-exchange protocol and interactive vocal interfaces, with serverless functions deployed on the cloud to monitor and control the objects. A practical use case based on this framework was also developed, prototyped, and tested. The smart object, a smart kitchen fan, is equipped with a NB-IoT module and makes use of a custom command exchange protocol based on MQTT. A vocal interface was developed using Amazon Alexa and Lambda, enabling vocal control of all the smart object's controllable states. The suitability of NB-IoT for MQTT messages dispatching was tested and showed a good latency performance, excluding some peaks that were evaluated and traced back to the formation of MQTT queues on the sender due to lost packets. Excluding cold starts, Lambda execution times are below 200 ms, positively contributing to an overall quasi-real time interaction between users and the smart object. Such a connected object could integrate additional customized services such as predictive maintenance and consumption control, enabled by cellular connectivity and simplified by resource provisioning on the cloud.

Author Contributions: Conceptualization, M.E., L.P., A.B. and P.P.; methodology, L.P., A.B. and P.P.; validation, M.E., L.P., A.B. and P.P.; formal analysis, M.E., L.P., A.B. and P.P.; investigation, M.E., L.P. and A.B.; writing—original draft preparation, M.E., L.P. and A.B.; writing—review and editing, M.E., A.B. and L.P.; visualization, M.E.; supervision, L.P., A.B. and P.P.; project administration, P.P.; funding acquisition, P.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by Marche Region in the implementation of the financial program POR MARCHE FESR 2014-2020, project "Miracle" (Marche Innovation and Research Facilities for Connected and Sustainable Living Environments), CUP B28I19000330007. The authors are grateful to Gitronica S.p.A. (Via Virgilio Guzzini, 12 62010 Montelupone (MC) Italy) for the opportunity to collaborate in this project.

Informed Consent Statement: Not applicable.

Acknowledgments: The authors extend their appreciation to Marche Region in the implementation of the financial program POR MARCHE FESR 2014-2020, project "Miracle" (Marche Innovation and Research Facilities for Connected and sustainable Living Environments), CUP B28I19000330007. The authors are grateful to Gitronica S.p.A. (Via Virgilio Guzzini, 12 62010 Montelupone (MC) Italy) for the opportunity to collaborate in this project.

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

Abbreviations

The following abbreviations are used in this manuscript:

IoT	Internet of Things
AI	Artificial Intelligence
FaaS	Function as a Service
OS	Operating System
ASK	Amazon Skills Kit
ASR	Automatic Speech Recognition
MQTT	Message Queue Telemetry Transport
HTTP	Hypertext Transfer Protocol
NB-IoT	Narrowband Iot
LPWAN	Low-Power Wide Area Network
3GPP	Third-Generation Partnership Project
MaaS	Machine as a Service

References

1. Eismann, S.; Scheuner, J.; Eyk, E.V.; Schwinger, M.; Grohmann, J.; Herbst, N.R.; Abad, C.L.; Iosup, A. A Review of Serverless Use Cases and their Characteristics. *arXiv* **2020**, arXiv:2008.11110.
2. Wang, I.C.; Qi, S.; Liri, E.; Ramakrishnan, K.K. Towards a Proactive Lightweight Serverless Edge cloud for internet-of-Things Applications. In Proceedings of the 2021 IEEE International Conference on Networking, Architecture and Storage (NAS), Riverside, CA, USA, 24–26 October 2021; pp. 1–4. [CrossRef]
3. Cassel, G.A.S.; Rodrigues, V.F.; da Rosa Righi, R.; Bez, M.R.; Nepomuceno, A.C.; André da Costa, C. Serverless computing for internet of things: A systematic literature review. *Future Gener. Comput. Syst.* **2022**, *128*, 299–316. [CrossRef]
4. Aslanpour, M.S.; Toosi, A.; Cicconetti, C.; Javadi, B.; Sbarski, P.; Taibi, D.; Assuncao, M.; Gill, S.S.; Gaire, R.; Dustdar, S. Serverless Edge Computing: Vision and Challenges. In Proceedings of the 2021 Australasian Computer Science Week Multiconference, Dunedin, New Zealand, 1–5 February 2021.
5. Paraskevoulakou, E.; Kyriazis, D. ML-FaaS: Towards exploiting the serverless paradigm to facilitate Machine Learning Functions as a Service. *IEEE Trans. Netw. Serv. Manag.* 2023, *Early Access*. [CrossRef]
6. Bebortta, S.; Das, S.K.; Kandpal, M.; Barik, R.K.; Dubey, H. Geospatial Serverless Computing: Architectures, Tools and Future Directions. *ISPRS Int. J. Geo-Inf.* **2020**, *9*, 311. [CrossRef]
7. Malawski, M.; Gajek, A.; Zima, A.; Balis, B.; Figiela, K. Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda and google cloud functions. *Future Gener. Comput. Syst.* **2020**, *110*, 502–514. [CrossRef]
8. AWS Lambda Overview. Available online: https://aws.amazon.com/lambda/?nc1=h_ls (accessed on 25 January 2022).
9. Pierleoni, P.; Concetti, R.; Belli, A.; Palma, L. Amazon, Google and Microsoft Solutions for IoT: Architectures and a Performance Comparison. *IEEE Access* **2020**, *8*, 5455–5470. [CrossRef]
10. Alexa Developer Documentation. Available online: https://developer.amazon.com/en-US/docs/alexa/documentation-home.html (accessed on 25 January 2022).
11. Iliev, Y.; Ilieva, G. A Framework for Smart Home System with Voice Control Using NLP Methods. *Electronics* **2023**, *12*, 116. [CrossRef]
12. MQTT Specification. Available online: https://mqtt.org/mqtt-specification/ (accessed on 25 January 2022).
13. Desbiens, F. MQTT. In *Building Enterprise IoT Solutions with Eclipse IoT Technologies: An Open Source Approach to Edge Computing*; Apress: Berkeley, CA, USA, 2023; pp. 67–101. [CrossRef]
14. Mwakwata, C.B.; Malik, H.; Mahtab Alam, M.; Le Moullec, Y.; Parand, S.; Mumtaz, S. Narrowband internet of things (NB-IoT): From Physical (PHY) and Media Access Control (MAC) Layers Perspectives. *Sensors* **2019**, *19*, 2613. [CrossRef]
15. Hwang, S.H.; Liu, S.Z. Survey on 3GPP Low Power Wide Area Technologies and its Application. In Proceedings of the 2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS), Singapore, 28–30 August 2019; pp. 1–5. [CrossRef]
16. Mekki, K.; Bajic, E.; Chaxel, F.; Meyer, F. A comparative study of LPWAN technologies for large-scale IoT deployment. *ICT Express* **2019**, *5*, 1–7. [CrossRef]
17. Díaz Zayas, A.; Rivas Tocado, F.J.; Rodríguez, P. Evolution and Testing of NB-IoT Solutions. *Appl. Sci.* **2020**, *10*, 7903. [CrossRef]

18. Soldani, D.; Pentikousis, K.; Tafazolli, R.; Franceschini, D. 5G networks: End-to-end architecture and infrastructure [Guest Editorial]. *IEEE Commun. Mag.* **2014**, *52*, 62–64. [CrossRef]
19. Tran, V.; Lenssens, B.; Kassab, A.; Laks, A.; Rivière, E.; Rosinosky, G.; Sadre, R. Machine-as-a-Service: Blockchain-based management and maintenance of industrial appliances. *Eng. Rep.* **2022**, *2022*, e12567. [CrossRef]
20. Paiola, M. Digitalization and servitization: Opportunities and challenges for Italian SMES. *Sinergie Ital. J. Manag.* **2018**, *36*, 11–22.
21. Tariq, M.A.; Khan, M.; Raza Khan, M.T.; Kim, D. Enhancements and Challenges in CoAP—A Survey. *Sensors* **2020**, *20*, 6391. [CrossRef]
22. Stolojescu-Crisan, C.; Gal, J. A Home Energy Management System. In Proceedings of the 2022 International Symposium on Electronics and Telecommunications (ISETC), Timisoara, Romania, 10–11 November 2022; pp. 1–4. [CrossRef]
23. Sarkar, S.; Wankar, R.; Srirama, S.N.; Suryadevara, N.K. Serverless Management of Sensing Systems for Fog Computing Framework. *IEEE Sens. J.* **2020**, *20*, 1564–1572. [CrossRef]
24. Froiz-Míguez, I.; Fernández-Caramés, T.M.; Fraga-Lamas, P.; Castedo, L. Design, Implementation and Practical Evaluation of an IoT Home Automation System for Fog Computing Applications Based on MQTT and ZigBee-WiFi Sensor Nodes. *Sensors* **2018**, *18*, 2660. [CrossRef]
25. Mishra, B.; Kertesz, A. The Use of MQTT in M2M and IoT Systems: A Survey. *IEEE Access* **2020**, *8*, 201071–201086. [CrossRef]
26. Husnain, M.; Hayat, K.; Cambiaso, E.; Fayyaz, U.U.; Mongelli, M.; Akram, H.; Ghazanfar Abbas, S.; Shah, G.A. Preventing MQTT Vulnerabilities Using IoT-Enabled Intrusion Detection System. *Sensors* **2022**, *22*, 567. [CrossRef]
27. Buccafurri, F.; De Angelis, V.; Nardone, R. Securing MQTT by Blockchain-Based OTP Authentication. *Sensors* **2020**, *20*, 2002. [CrossRef]
28. Munshi, A. Improved MQTT Secure Transmission Flags in Smart Homes. *Sensors* **2022**, *22*, 2174. [CrossRef]
29. Baek, J.; Kanampiu, M.W.; Kim, C. A Secure internet of things Smart Home Network: Design and Configuration. *Appl. Sci.* **2021**, *11*, 6280. [CrossRef]
30. Krishnamurthi, R.; Kumar, A.; Gopinathan, D.; Nayyar, A.; Qureshi, B. An Overview of IoT Sensor Data Processing, Fusion, and Analysis Techniques. *Sensors* **2020**, *20*, 6076. [CrossRef]
31. Lee, E.; Vesonder, G.; Wendel, E. Eldercare Robotics—Alexa. In Proceedings of the 2020 11th IEEE Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), New York, NY, USA, 28–31 October 2020; pp. 0820–0825. [CrossRef]
32. Bogdan, R.; Tatu, A.; Crisan-Vida, M.M.; Popa, M.; Stoicu-Tivadar, L. A Practical Experience on the Amazon Alexa Integration in Smart Offices. *Sensors* **2021**, *21*, 734. [CrossRef] [PubMed]
33. Pennacchioni, M.; Di Benedette, M.G.; Pecorella, T.; Carlini, C.; Obino, P. NB-IoT system deployment for smart metering: Evaluation of coverage and capacity performances. In Proceedings of the 2017 AEIT International Annual Conference, Cagliari, Italy, 20–22 September 2017; pp. 1–6. [CrossRef]
34. Li, Y.; Cheng, X.; Cao, Y.; Wang, D.; Yang, L. Smart Choice for the Smart Grid: Narrowband internet of things (NB-IoT). *IEEE Internet Things J.* **2018**, *5*, 1505–1515. [CrossRef]
35. Kadusic, E.; Zivic, N.; Ruland, C.; Hadzajlic, N. A Smart Parking Solution by Integrating NB-IoT Radio Communication Technology into the core IoT Platform. *Future Internet* **2022**, *14*, 219. [CrossRef]
36. Zhang, H.; Li, J.; Wen, B.; Xun, Y.; Liu, J. Connecting Intelligent Things in Smart Hospitals Using NB-IoT. *IEEE Internet Things J.* **2018**, *5*, 1550–1560. [CrossRef]
37. Daraghmi, Y.A.; Daraghmi, E.Y.; Daraghma, R.; Fouchal, H.; Ayaida, M. Edge/Fog/cloud computing Hierarchy for Improving Performance and Security of NB-IoT-Based Health Monitoring Systems. *Sensors* **2022**, *22*, 8646. [CrossRef]
38. Thedy, J.; Liao, K.W.; Tseng, C.C.; Liu, C.M. Bridge Health Monitoring via Displacement Reconstruction-Based NB-IoT Technology. *Appl. Sci.* **2020**, *10*, 8878. [CrossRef]
39. Dangana, M.; Ansari, S.; Abbasi, Q.H.; Hussain, S.; Imran, M.A. Suitability of NB-IoT for Indoor Industrial Environment: A Survey and Insights. *Sensors* **2021**, *21*, 5284. [CrossRef]
40. Han, C.; Zhang, W.; Li, M.; Tian, Y. Design of Smart Home System Based on Nb-Iot. *J. Phys. Conf. Ser.* **2022**, *2254*, 012039. [CrossRef]
41. Li, T.; Hou, P. Application of NB-IoT in Intelligent Fire Protection System. In Proceedings of the 2019 International Conference on Virtual Reality and Intelligent Systems (ICVRIS), Jishou, China, 14–15 September 2019; pp. 203–206. [CrossRef]
42. Sisavath, C.; Yu, L. Design and implementation of security system for smart home based on IOT technology. *Procedia Comput. Sci.* **2021**, *183*, 4–13.
43. Muteba, K.; Djouani, K.; Olwal, T. 5G NB-IoT: Design, Considerations, Solutions and Challenges. *Procedia Comput. Sci.* **2022**, *198*, 86–93.
44. Khan, B.; Pirak, C. Experimental Performance Analysis of MQTT and CoAP Protocol Usage for NB-IoT Smart Meter. In Proceedings of the 2021 9th International Electrical Engineering Congress (iEECON), Pattaya, Thailand, 10–12 March 2021; pp. 65–68. [CrossRef]
45. Larmo, A.; Ratilainen, A.; Saarinen, J. Impact of CoAP and MQTT on NB-IoT System Performance. *Sensors* **2019**, *19*, 7. [CrossRef] [PubMed]
46. Khanh, Q.V.; Hoai, N.V.; Manh, L.D.; Le, A.N.; Jeon, G. Wireless communication technologies for IoT in 5G: Vision, applications, and challenges. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 3229294. [CrossRef]

47. Wang, S.Y.; Chang, J.E.; Fan, H.; Sun, Y.H. Comparing the performance of NB-IoT, LTE Cat-M1, Sigfox, and LoRa for IoT end devices moving at high speeds in the air. *J. Signal Process. Syst.* **2022**, *94*, 81–99. [CrossRef]
48. Khalifeh, A.; Aldahdouh, K.A.; Darabkh, K.A.; Al-Sit, W. A Survey of 5G Emerging Wireless Technologies Featuring LoRaWAN, Sigfox, NB-IoT and LTE-M. In Proceedings of the 2019 International Conference on Wireless Communications Signal Processing and Networking (WiSPNET), Chennai, India, 21–23 March 2019; pp. 561–566. [CrossRef]
49. Naik, N. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In Proceedings of the 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, Austria, 11–13 October 2017; pp. 1–7. [CrossRef]
50. Macheso, P.; Manda, T.D.; Chisale, S.; Dzupire, N.; Mlatho, J.; Mukanyiligira, D. Design of ESP8266 Smart Home Using MQTT and Node-RED. In Proceedings of the 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), Coimbatore, India, 25–27 March 2021; pp. 502–505. [CrossRef]
51. Pierleoni, P.; Conti, M.; Belli, A.; Palma, L.; Incipini, L.; Sabbatini, L.; Valenti, S.; Mercuri, M.; Concetti, R. IoT Solution based on MQTT Protocol for Real-Time Building Monitoring. In Proceedings of the 2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT), Ancona, Italy, 19–21 June 2019; pp. 57–62. [CrossRef]
52. D'Ortona, C.; Tarchi, D.; Raffaelli, C. Open-Source MQTT-Based End-to-End IoT System for Smart City Scenarios. *Future Internet* **2022**, *14*, 57. [CrossRef]
53. Manowska, A.; Wycisk, A.; Nowrot, A.; Pielot, J. The Use of the MQTT Protocol in Measurement, Monitoring and Control Systems as Part of the Implementation of Energy Management Systems. *Electronics* **2023**, *12*, 17. [CrossRef]
54. Veichtlbauer, A.; Heinisch, A.; von Tüllenburg, F.; Dorfinger, P.; Langthaler, O.; Pache, U. Smart Grid Virtualisation for Grid-Based Routing. *Electronics* **2020**, *9*, 1879. [CrossRef]
55. Vitali, G.; Francia, M.; Golfarelli, M.; Canavari, M. Crop Management with the IoT: An Interdisciplinary Survey. *Agronomy* **2021**, *11*, 181. [CrossRef]
56. Esposito, M.; Palma, L.; Belli, A.; Sabbatini, L.; Pierleoni, P. Recent Advances in internet of things Solutions for Early Warning Systems: A Review. *Sensors* **2022**, *22*, 2124. [CrossRef]
57. Gupta, B.; Mittal, P.; Mufti, T. A Review on Amazon Web Service (AWS), Microsoft Azure, Google cloud Platform (GCP) Services. In Proceedings of the 2nd International Conference on ICT for Digital, Smart, and Sustainable Development, ICIDSSD 2020, Jamia Hamdard, India, 27–28 February 2020.
58. Alexa Developer Documentation. Available online: https://docs.aws.amazon.com/iot/latest/developerguide/data-protection.html (accessed on 25 January 2022).
59. Alexa Developer Documentation. Available online: https://docs.aws.amazon.com/iot/latest/developerguide/authentication.html (accessed on 25 January 2022).
60. Këpuska, V.; Bohouta, G. Next-generation of virtual personal assistants (Microsoft Cortana, Apple Siri, Amazon Alexa and Google Home). In Proceedings of the 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 8–10 January 2018; pp. 99–103. [CrossRef]
61. George, G.; Bakir, F.; Wolski, R.; Krintz, C. NanoLambda: Implementing Functions as a Service at All Resource Scales for the internet of things. In Proceedings of the 2020 IEEE/ACM Symposium on Edge Computing (SEC), San Jose, CA, USA, 12–14 November 2020; pp. 220–231. [CrossRef]
62. Eclipse Paho Pyhthon Client. Available online: https://www.eclipse.org/paho/clients/python/ (accessed on 24 October 2022).
63. Light, R.A. Mosquitto: Server and client implementation of the MQTT protocol. *J. Open Source Softw.* **2017**, *2*, 265. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.