Received: 14 December 2021 | Revised: 23 February 2022 | Accepted: 28 February 2022

DOI: 10.1111/exsy.12994

ORIGINAL ARTICLE

Expert Systems WILEY

Flex-request: Library to make remote changes in the communication of IoT devices

Karol Mateusz Ciok[1] | Jordán Pascual Espada[1] | Rubén González Crespo[2]

[1]Department of Computer Science, University of Oviedo, Asturias, Spain

[2]UNIR International University of La Rioja, Madrid, Spain

Correspondence
Jordán Pascual Espada, Department of Computer Science, University of Oviedo, Asturias, Spain.
Email: pascualjordan@uniovi.es

Abstract

In recent years, Internet of Things (IoT) systems have changed the way we live, work and do businesses in many areas, even those that until recently seemed unlikely. Some areas that can benefit from their application are, among others, healthcare, smart cities, industrial automation, smart agriculture, intelligent transportation systems, smart logistics, and emergency response. This research work proposes a novel alternative that allows the creation of IoT systems capable of making remote changes in devices' communication in a fast and agile manner. Our proposal gives way to some of the most common changes in communication made during the development and maintenance phase in IoT systems, like changing the destination of data transmission, sending the data to multiple destinations, and changing the frequency of sending data. Our solution, which is used in the programs, is loaded on the device. When the device starts, it connects to a configuration server in the background and listens for changes. The changes are sent to the configuration server using specified commands. When a change is detected, the command is processed, and the change in communication is applied without stopping the running program. We designed experiments to evaluate the complexity of the programs developed using our proposal and the actions needed to make a change.

KEYWORDS
Internet of Things, MQTT protocol, remote configuration, sensors, software update

1 | INTRODUCTION

In recent years, Internet of Things (IoT) systems have changed the way we live, work and do business in many areas, even those that until recently seemed unlikely. These IoT systems are defined as the interconnection of devices sensing and actuating with the real world, providing the ability to share information and coordinate themselves to carry out tasks (Gubbi et al., 2013; Kumper & Tonjes, 2011). Some areas that can benefit from their application are, among others, healthcare, smart cities, industrial automation, smart agriculture, intelligent transportation systems, smart logistics, and emergency response (Kumar et al., 2020; Razzaque et al., 2016; Zhang et al., 2016). However, the number of fields that can accommodate an IoT solution is vast and continues expanding as more devices achieve internet connection capabilities. By 2020, more than 50 billion devices are expected to be connected to the Internet of Things (García et al., 2019; Vaquero & Rodero-Merino, 2014).

IoT Devices usually obtain data from their near environment using the sensors. Devices also can perform actions using actuators like relays. In IoT systems, devices are distributed to networks to exchange data and perform tasks together. Communication between devices and services is one of the key points of IoT systems. However, when the number of devices and services increases, so does the difficulty of making changes in communications.

In many cases, IoT systems are made up of many small-scattered devices. Because of that, some modifications may involve many updates. For example, if it is required to change the IP of the centralized service, developers might have to update the software of devices. Despite the potential benefits of using IoT devices, the uptake by the industry of innovative large-scale sensor/actuator applications is slow. One of the main reasons is the lack of built-in support for the maintenance of such devices. Although the standards provide many options to configure network protocols, it is difficult to change the configuration settings at run-time; this implies that the entire network needs to be reconfigured offline if changes are required (Ruckebusch et al., 2016). Due to this, it is interesting for developers to make changes in communication in an efficient manner from the start of the software's development for those devices. Those changes could bring advantages to the developers in different phases of the design and implementation of the IoT system, such as the ability to get rapid prototyping and deploy alternatives to find the most suitable one for the system's needs. IoT systems can be based on many communication topologies, like client–server, trees, meshes, P2P, etc. In many cases, IoT systems are innovative solutions, but the creation of the system carries a high degree of uncertainty and restrictions, so it is often necessary to try different approaches and architectures to find an appropriate way to create the IoT system.

The complexity of the software in IoT devices could be very low as these devices usually do not require doing complex tasks. For example, some devices are limited to getting data through a sensor and sending it to a server. However, the complexity of the device software could be conditioned by a few factors, like the programing language, the code length, and the structure and particularities of the framework used to access the device's hardware and communications (Antinyan et al., 2017; Espada et al., 2015). IoT tends to have some uncertainty, and writing less lines of code enables programmers to make more effective changes, which allows them to test more alternatives and choose the one that's most suitable for their needs (Chiang & Lu, 2011; Mukhtar et al., 2009).

Another critical aspect of making changes when the system is already running is system downtime. Applying the changes could require stopping the current program and executing the new one. This process is slower when the new software must be sent remotely to a device, such as an Over-the-Air (OTA) update. Because of this, there is some time when the device is neither collecting nor sending the sensor data. For example, for a power plant, collecting and analysing this data is critical to safely manage the plant and avoid potential important data loss.

Nowadays, there are certain alternatives to resolve the problem of remotely reconfiguring some aspects of IoT devices. Some of the most common alternatives can be grouped in the following categories:

- *Devices consulting its communication parameters from external services*. In this category, we can include services like Firebase Remote Config.[1] This involves developing the software in a specific and slightly more complex way so that it integrates with the remote configuration service. The main disadvantage of these services is that they enable little changes in the configuration parameters.
- *Devices which use software modular software*. Some examples could be the *node-supervisor*[2] or *hotswap*[3] node module. These solutions also have some downsides. *Node-supervisor* restarts the program when one of the files it is watching changes. On the other hand, *hotswap* reloads the modules when it detects a change without stopping the program. However, in this case, the programs must be done in modular way, forcing the developer to foresee all possible future changes and maintaining the modules small because reloading bigger modules can cause some performance issues.
- *Devices managed by an IoT platform that uses a proxy or a gateway server*. Some examples of these platforms could be Ruckebusch et al. (2016) and Jin and Kim (2018). These kinds of platforms tend to need a big architecture setup.
- *Full or partial software update based on OTA* (*Over the Air update*). Although there are some solutions that try to minimize the size of the update (Mukhtar et al., 2009) or (Chiang & Lu, 2011), many OTA updates still need to restart the device in order to apply the update.

The main objective of this research work is to provide an alternative that leads to the creation of IoT systems capable of making remote changes to devices' communication in a fast and agile way. This will allow some of the most common changes in communication to be made during the development and maintenance phase in IoT systems, like changing the destination of data transmission, sending the data to multiple destinations, and changing the frequency of sending data. To be considered appropriate, the solution should comply with the following aspects:

- The complexity of implementing software capable of remote configuration must be reduced, not involving highly additional costs for developers compared to developing an IoT software without this feature.
- The cost and complexity of doing a change in a device communication configuration must be reduced. Developers should be able to make changes quickly and efficiently compared to current alternatives for making changes in IoT devices software.
- Finally, the proposal should minimize the time that the IoT device is off or not working correctly after making a change in communication. In most IoT systems, it is common that after a change or update in the device software, it remains inactive for a while. This inactivity time must be short. It could be critical for many IoT systems because they could lose relevant data during this time.

The remainder of this paper is structured as follows: Section 2 presents the background and related work on software update problems on IoT devices in literature. Section 3 describes the proposed solution. Section 4 describes some use cases of the proposal. Section 5 describes the evaluation process and the obtained results. Section 5 concludes the study and proposes future works.

2 | BACKGROUND

A microcontroller is a single integrated circuit that, as a minimum, contains the necessary elements of a complete computer system and can contain additional peripheral modules, such as serial and timer units. A microcontroller must have more than just a CPU, a program, and a data memory to be able to solve real-world problems. In addition to that, it must contain hardware allowing access to information from the outside world. Once it gathers information and processes the data, it must also be able to affect change on some portion of the outside world (Bannatyne & Viot, 1998).

There are hundreds of different microcontrollers with a variety of features. Some examples of those are Siemens S, Mitsubishi FX, Kunbus PR, Arduino, Raspberry Pi, and Intel Galileo (Al-Fuqaha et al., 2015; Singh & Kapoor, 2017). They usually fashion an array of GPIO (General Purpose Input/Output) pins to allow them to connect peripherals.

These microcontrollers are the fundamental basis for the next step in the ladder: smart objects. Smart objects are autonomous physical/digital objects augmented with sensing, processing, and network capabilities. They carry chunks of application logic that let them make sense of their local situation and interact with human users. They sense, log, and interpret what is occurring within themselves and the world, act on their own, intercommunicate with each other, and exchange information with people (Kortuem et al., 2010).

Smart objects can be very different and can be classified in many different ways, based on various features like computing capacity, communication skills, intelligence level, and purpose (Razzaque et al., 2016).

Managing IoT devices, especially if its number is high, entails some challenges like heterogeneity, scalability, interoperability, security, and privacy (Sarkar et al., 2015). Also, there is a lack of build-in possibility to reconfigure the device dynamically without shutting down the device (Ruckebusch et al., 2016). There are many commercial IoT management platforms like Balena.io,[4] Thethings.io, Cloudino,[5] Google Cloud IoT Core,[6] Microsoft Azure IoT Central[7] and AWS IoT Device Management.[8] These platforms allow adding or removing devices and have configured OTA updates. There are also other services like Firebase Remote Config (Alsalemi et al., 2017) that enable setting some specific application parameters on the cloud and offers an API to update those parameters in the application if they change on the cloud. Some research works focus on changing some specific configuration parameters in medical devices (Barron-Gonzalez et al., 2013; Barron-Gonzalez et al., 2014; Kumper & Tonjes, 2011), in home devices (Nikolaidis et al., 2006), or in remote controls (Seo et al., 2013). There are also AT commands for configuring WiFi routers and Bluetooth devices.

There are many solutions based on OTA updates that try to eliminate or minimize some negative aspects of these updates. Some propose updates based on diff (Chiang & Lu, 2011; Mukhtar et al., 2009) to minimize the size of the update that get sent. Others reduce the time offline (Chang et al., 2013; Jurković & Sruk, 2014) or energy consumption (Jurković & Sruk, 2014). Some solutions focus on heterogeneity (Jin & Kim, 2018) and open standards (Dalipi et al., 2017). Some proposed solutions use Proxy to reinforce security (Su et al., 2020) or to analyse the network traffic (Kayode & Tosun, 2019).

There are different alternatives in terms of topology and communication patterns of IoT devices. Some research works use a central gateway or Proxy to manage the IoT devices network. These solutions enable dynamic adding and removing of IoT devices and also modification of some communication parameters (Dalipi et al., 2017; Ruckebusch et al., 2016). The most common update mechanism using that approach consists of installing the update on all the devices registered in the gateway. Other researchers present a more decentralized approach. Some enable the simultaneous updates of multiple distributed nodes involved in a running service to prevent mismatching communication (Weisbach et al., 2017). Others focus on node failure tolerance, basing their solution on moving the aspect of the update process to the devices and leaving the central server only responsible for indicating when the updates are available (Kolomvatsos, 2018; Raptis et al., 2018).

As IoT systems bring many challenges (Sarkar et al., 2015), one important aspect when working with IoT Systems is fast and efficient prototyping. Some research works that try to reduce development time by unifying the access to the most common IoT features (Datta & Bonnet, 2016), abstracting the business logic so that it can be written only once for different devices (Lee et al., 2017), using containers (Mazzei et al., 2016) using a macro programming framework (Patel & Cassou, 2015) and graphic programming languages (Kikuchi et al., 2018; Nepomuceno et al., 2018).

3 | PROPOSED PLATFORM

This research work proposes a new library, which can be used in the device software. This library allows modifying dynamically the communication characteristic of the software executed on the device without altering the software code. This library can be used as a standard library to send requests over the Internet and allows developers to change dynamically and quickly some of the main aspects of the communication, such as:

- Changing the destination IP of the requests
- Allowing resending requests to other devices or servers, in addition to the main destination

- Changing the frequency of requests being sent

These changes do not imply changes in the software implementation, nor running a new program or restarting the device.

The proposed platform has two parts. The main part is the communication library, which is used in the device's program, and the other part is the configuration server. This server is a simple MQTT broker that exposes a topic where the configuration updates can be published for the IoT devices to capture and apply the change. The configuration changes are sent using the MQTT protocol which is a Publish-Subscribe protocol.

The communication library is used in programs loaded on IoT devices. When the program starts, the library connects to the configuration server in a background process. This process is automatic and imperceptible to the developer. The connection listens to the communication configuration changes, and when it detects a new configuration change, it updates the local internal configuration of the library and applies the necessary changes.

The communication scheme between the library and the MQTT broker is depicted in Figure 1.

1. IoT devices using the library have specified in the program code the services where they send the recorded data.
2. A developer can make a change in communication configuration by sending a command to the Configuration Server.
3. The library inside the IoT devices detects the configuration change that had been sent by the developer to the Configuration Server through an MQTT Broker, processes the command, and applies the communication change.
4. Once the received command is processed by the device, its communication changes – for example, it starts sending the recorder data to a new service.

The proposed library defines a *request* method that accepts the following arguments:

url [*optional*] – url or IP of the destination server. Must include the protocol. When passed to the call, it is parsed with URL of WebAPI and overwritten by properties explicitly set in the *options* object.

options [*required*] – object with the following request options:

- **id** [*required*] – request id. Used to differentiate the different uses of the library inside the same program.
- **data** [*required*] – a function that returns a promise with the data to be sent in the request.
- **host** [*optional*, *defaults to* '*localhost*'] – host of the destination server.
- **protocol** [*optional*, *defaults to* '*http:*'] – protocol used.
- **method** [*optional*, *defaults to* '*GET*'] – http method.
- **port** [*optional*, *defaults to 80*] – destination port.
- **requestFrecuency** [*optional*, *defaults to 1000*] – frequency (in milliseconds) of sending request.
- **forwardUrls** [*optional*, *defaults to empty array*] – array of URLs to forward the request sending.

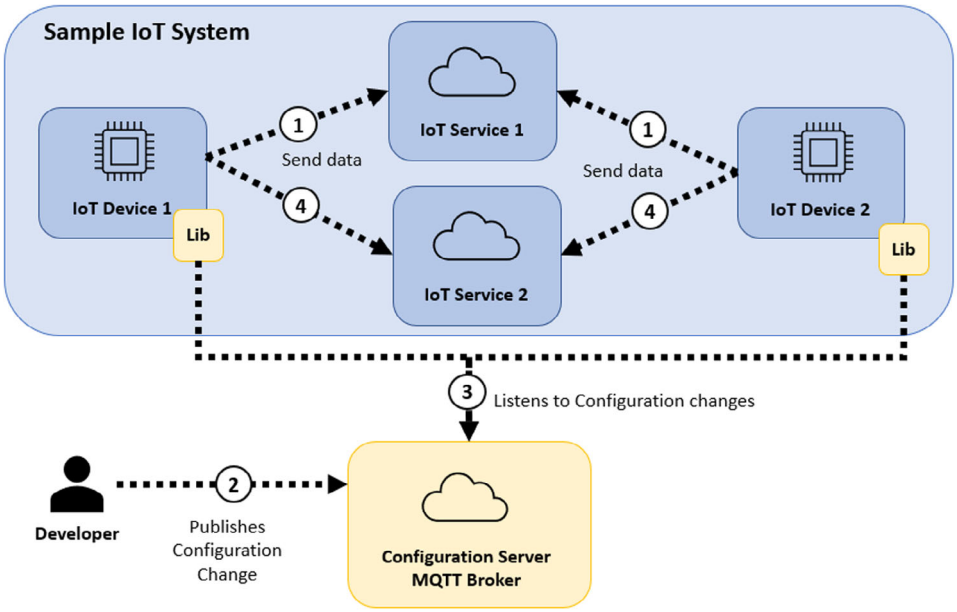

FIGURE 1 Proposal conceptual scheme

- **callback** – the function to be executed when the response is received.

In order to use the library, the developers have to include the library inside the program first. The library implementation has to be compatible with the used programming language. Then, developers have to configure the communication with the configuration server for the entire program. After that, we can use the method offered by the library to send requests over the Internet. Each use of the request method must be identified with an id to allow independently modifying each use. Figure 2 represents an example of a program using the proposed library.

When this program starts, it will do two things. In the foreground, it will read the sensor's temperature and send it to http://192.168.0.27:6001 every second, and read the sensor's humidity and send it to http://158.54.0.100:7000 every second as well. When a response is received, it will be logged to the console. In the background, as explained earlier, it will connect to the MQTT broker and listen for configuration changes. When a configuration change is detected, it will parse the command received and update the local internal state. Then each request method whose id is equal to the one received in the command will change its communication configuration. That way, we may have many devices where the request method has the same id, and when the configuration change is made to that id, all devices will change their configuration.

In order to change a communication parameter, the administrator has to send an MQTT request to the configuration server with a specific command that enables the required change. The configuration server publishes the command to all the devices that are listening. In order to cover the communication changes, the following commands have been designed:

- set <*id*> <*destinationURL*> – changes the URL of the destination server of the request with <*id*>. That command can be useful to change the server to which the data is sent in a Client–Server scheme because it has been decided to change the current server to a new one. Another use may be to change the device with which the communication is done
  - *master-request* – is a reserved id for referring to the MQTT broker URL
- forward <*id*> <*forwardURL1*> <*forwardURL2*> – forwards the send request with <*id*> to the URLs form <*forwardURL*> arguments. It can have as many forward URLs as one wants, separated by spaces. This command can be used if there is a need to separate the data processing onto different servers, for instance.
- set <*id*> frequency <*frequencyValue*> – changes the frequency of the request sending to the <*frequencyValue*> (in milliseconds). This can be used to reduce the impact on network traffic as there might not be a need to send the data every second all the time.

The current commands were designed to be used in what may seem like the common cases where there are a set of IoT devices that send the captured data to a central service. However, the proposal may be extended to support other communication changes. In many cases, IoT systems have other topologies – a mesh, for instance, where an IoT device can communicate with one or more other IoT devices.

Our proposal could be used to help modify those interactions remotely. For example, if we have one detector device that is communicating with one actuator, we could change the actuator for another one or make the detector interact with two actuators.

Another case where our proposal may be useful is to add new devices to the IoT systems (Figure 3). We could make the devices that we already have to start interacting with the ones we want to add to our mesh. For example, if we want to add a new light controller and we already have a presence detector running, we could make the detector send the data to that light controller in addition to the destination where it is currently sending the data.

4 | USE CASES

Below we present some use cases of the main functionalities of our proposal: changing the destination server IP V4, forwarding a request to another server, and changing the sending frequency.

In order to evaluate the use cases, we designed a small sample with limited functionality, just enough to show the usage of the library. This sample simulates the recording of the temperature and humidity inside a warehouse. The designed system has various devices with internet connectivity that are equipped with temperature and humidity sensors. Those devices capture the temperature and humidity in multiple locations of the warehouse and send the recorded data to a centralized service where it is analysed and stored. Those generated analyses are used by the warehouse managers to assess changes in infrastructure, the position of machinery and doors, the placement of refrigeration equipment, and so forth.

That use case is one possible use of our proposal. We believe our proposal may be applied in many different environments. For example, it may be used in distributed systems without centralized topologies where IoT devices communicate with each other.

```
01: const sl = require("node-dht-sensor");
02: const FlexRequest = require("flex-request");
03: const factory = FlexRequest({
04:   masterServerUrl: "mqtt://192.168.0.27:5000",
05: });
06:
07: // Sending the temperature
08: factory.request(
09:   {
10:     id: 1,
11:     protocol: "http:",
12:     host: "192.168.0.27",
13:     port: 6001,
14:     data: new Promise((resolve, reject) => {
15:       resolve({
16:         temperature: sl.read(11, 4).temperature,
17:       });
18:     }),
19:     method: "POST",
20:   },
21:   (err, res, body) => {
22:     if (err) {
23:       return console.log(`Error: ${err.message}`);
24:     }
25:     console.log(`${body}`);
26:   }
27: );
28:
29: // Sending the humidity to another server
30: factory.request(
31:   {
32:     id: 2,
33:     protocol: "http:",
34:     host: "158.54.0.100",
35:     port: 7000,
36:     data: new Promise((resolve, reject) => {
37:       resolve({
38:         humidity: sl.read(11, 4).humidity,
39:       });
40:     }),
41:     method: "POST",
42:   },
43:   (err, res, body) => {
44:     if (err) {
45:       return console.log(`Error: ${err.message}`);
46:     }
47:     console.log(`${body}`);
48:   }
49: );
```

FIGURE 2 Example implementation for a device software

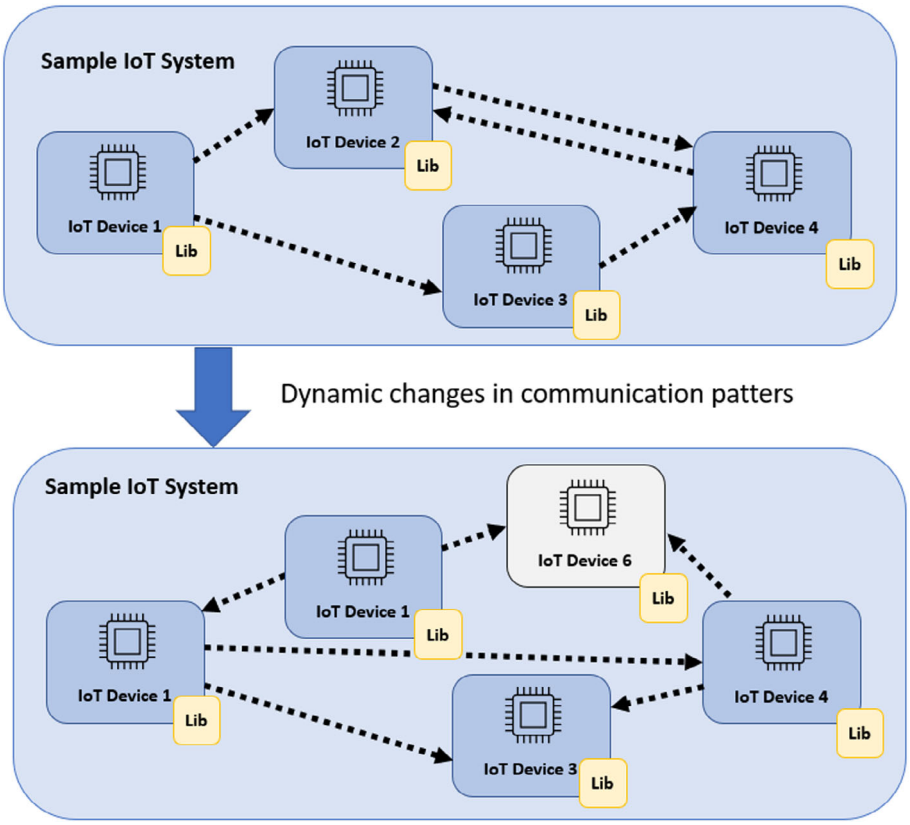

FIGURE 3 Dynamic changes in an IoT system with mesh topology

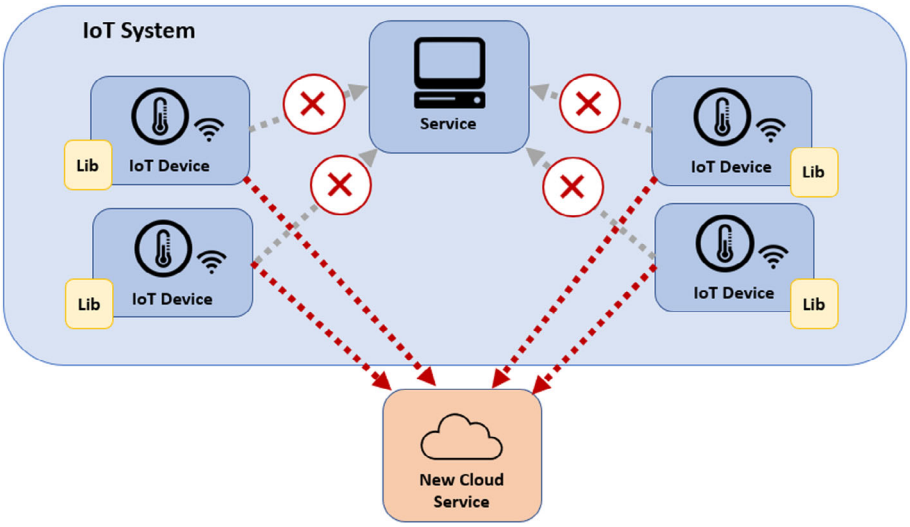

FIGURE 4 Changing the clients' destination server scheme

4.1 | Use case 1: Change the destination server of the clients

The company that owns the warehouse decided to change its centralized service for another cloud computing provider (Figure 4). That change involves reconfiguring all the IoT devices that record the temperature and the humidity to send the data to the new service. The company wants to try various cloud computing providers in order to choose the one that best covers their needs.

Once the program is loaded onto the IoT device and is running, it starts sending requests to the destination server http:192.168.0.27:6001. In order to change the destination IP, we have to send an MQTT request to the MQTT broker with the following command: *set 1* http://192.168.

0.100:7071 (Figure 5). When the library that is listening for configuration changes detects a new change, it updates the destination server IP and starts to send it to the new destination.

4.2 | Use case 2: Forward the requests to another server

In another use case, the company detects a point inside the warehouse where temperature management is critical. In order to overcome that problem, the company acquires a new device that can manage the air conditioning according to temperature. It is necessary that the devices that are close to that critical point record and send the temperature to that new device in addition to sending the data to the centralized service (Figure 6).

In this case, to make that change, we have to send the following command: *forward 1* http://192.168.0.200:4044. Like in the previous use case, when the library detects a new configuration, it saves the forward address locally and starts forwarding each request sent to this new address.

4.3 | Use case 3: Change the sending frequency

With the recent increase in the workforce and the new machinery, it has been detected that temperature changes are much more pronounced and faster than before. Those fast temperature changes require increasing the frequency of the temperature being sent to the centralized service. The company determines that this sending frequency increase is necessary even though it means higher processing costs for the server and higher consumption for the devices (Figure 7). In order to change the frequency using our solution, the responsible system administrators have to send the following command: *set 1 frequency 500*. In this command, '500' is the number of milliseconds between each data sent.

5 | EVALUATION

In the evaluation section, three changes will be made in the communication of devices. Those changes correspond to those made in the use cases.

- *Change the destination server of the clients*. The initial version of the system has an IoT device that sends data to a service. After the change, the device must send the data to another different service.

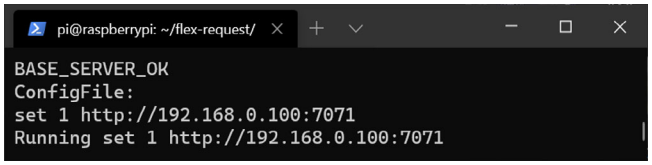

FIGURE 5 Configuration change received and applied by the IoT device

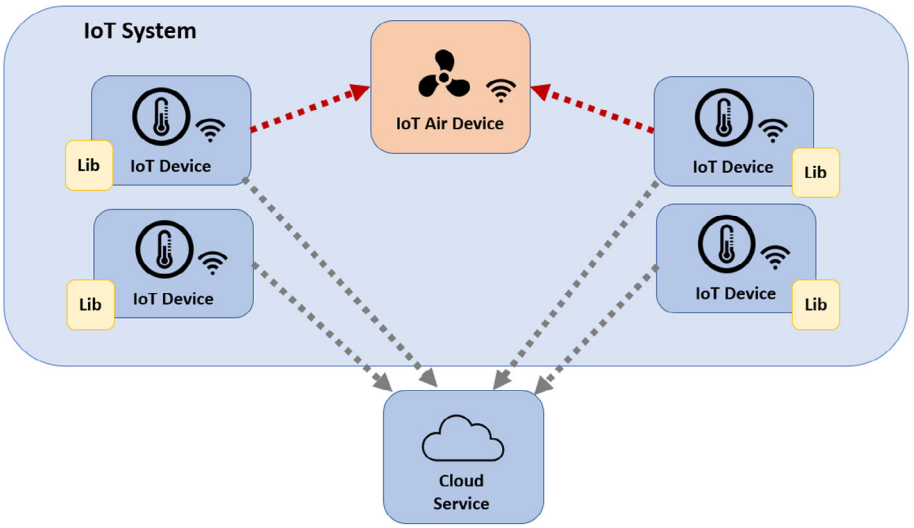

FIGURE 6 Forwarding the data sending to another device scheme

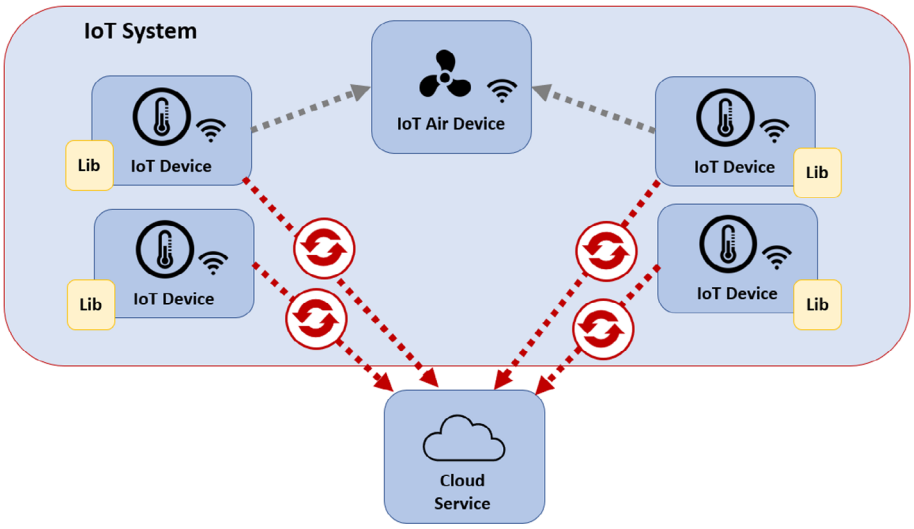

FIGURE 7 Changing the frequency of data sending scheme

- *Forward the requests to another server*. The initial version of the system has an IoT device that sends data to a service. After the change, the device must send the data to another new additional service as well.
- *Change the sending frequency*. The initial version of the system has an IoT device that sends data to a service every second. After the change, the device must send the data every 0.5 s.

Each one of the three use cases will be evaluated with the following alternatives.

- OTA Update. Over the Air update is one of the most common ways to update an IoT device software. It consists of remotely sending a new version of the software. This new software must replace the current version. Most IoT devices with some computing capacity can use operating systems that allow this type of update. Many OTA updates require restarting the device to apply the update.
- Partial OTA. An optimized version of the previous OTA. This process allows generating a package with a partial update, which is small and can be done more quickly. This process is supported only for devices that run complex operating systems. It often requires restarting the device to apply the update.
- Partially updateable software. This is software that has been particularly designed since its inception to support partial updates of some of its components. In this case, the software update depends on itself, not on the device's operating system. This solution is highly dependent on the software implementation strategy, but always requires a more complex implementation than software without updates periodically. This type of update may or may not require restarting the device.
- Proxy. An intermediate proxy makes all devices communicate with the Proxy, therefore only by modifying the proxy implementation could a developer modify some communication aspects of the entire IoT system.
- Firebase. Services like Firebase allow the inclusion of remote configurations, which could be obtained by the device's software. Developers usually store configuration settings in Firebase (like connection strings, configuration values, etc.). The device's software must know how to get and process those configuration values. Like 'Partially updateable software', this kind of solution requires a more complex implementation of the device software.
- The proposal of this paper. As explained earlier, it uses a specialized command to change some aspects of the communication of the devices.

To evaluate our solution, we used one Raspberry Pi 3 Model B (WiFi 802.11ac double band) as an IoT device and a PC acting as a server. On the Raspberry Pi, we installed Raspbian GNU/Linux 10 (buster) operating system and Node.js version 12.16.1 as an execution platform. The PC has Windows 10 Pro x64 version 1909 operating system with an Intel Core i5-9600KF 3.70GHz CPU and 16 GB of RAM. In order to access the Raspberry device and easily transfer the programs used in the evaluation of the experiment, we used WinSCP.

Based on the research objectives, the proposal should allow the following:

1. Changes in the communication of the devices, implying a low implementation cost. For this reason, during the evaluation process, we will estimate the implementation complexity in each case.

2. Changes in communications must be made quickly and easily. For this reason, during the evaluation process, we will measure how complex it is for developers to make changes in communication.
3. Minimize the time that the device is not working correctly after a change. For this reason, during the evaluation process, we will measure how long it takes for a device to apply the change and start sending with the new configuration.

Additionally, we should analyse the network traffic in the IoT system. The evaluated alternatives could have different impacts on network traffic. The objective of this research is not to optimize network traffic, but this could be an important factor in some IoT systems.

5.1 | Implementation complexity

This is the first measurement of all. It tries to estimate the initial implementation complexity of the IoT system before applying the configuration change, which will modify the initial communication scheme. In this evaluation process, we will estimate the implementation complexity of the initial IoT system for the three experiments using the six alternatives.

Measuring the complexity of program implementation is a complicated task. The complexity depends on the number of aspects considered. The simplest estimates of complexity are based solely on the size of the code; in this case, we have started from a complexity estimation system used in previous research works (Cueva-Fernandez et al., 2014; Espada et al., 2015). This system considers the following:

- C – Size: number of the program characters, spaces included
- L – Number of lines of code in the program
- F – Number of functions used but not implemented by the programmer
- NM – Number of standard modules included in Node.js distribution
- NE – Number of modules not included in Node.js distribution and that had to be installed with npm

Each aspect measured was assigned with the following weights: C = 1, L = 2, F = 3, NM = 4, and NE = 5. The global score for each evaluated alternative is the sum of the weights for each aspect.

Figure 8 shows the results of the implementation complexity for each alternative in the three experiments. Firebase has the highest complexity in all experiments because, for that, additional alternative code must be added to connect to the Firebase service. OTA Update, Partial OTA, and Partially Updateable Software have close implementation complexity with score differences no larger than 100 points between them in each experiment. Proxy has the same complexity in the first two experiments. Because the IoT device is the one that sends the request and the Proxy only redirects or forwards the request to another server, we determined that we cannot use Proxy to implement the sending frequency change. Our proposal has the lowest complexity in all experiments.

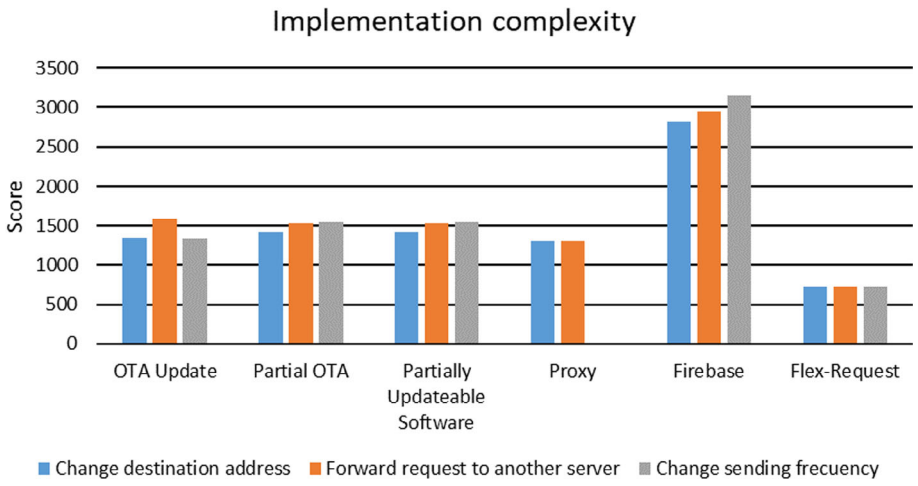

FIGURE 8 Implementation complexity of each alternative in each experiment

5.2 | Configuration change complexity

This is the second measurement. It measures how complex it is to make a change over the initial version of the system.

Unlike the implementation of the software, the application of changes over the communication scheme could not only involve changes in the code, but sometimes it requires using configuration tools, commands, and so forth. Due to these particularities, we have included in this complexity estimation all actions performed by the developer with a mouse and keyboard.

We define configuration change complexity as the actions that must be done to apply a configuration change. To measure this complexity, the following actions were measured:

- K – Number of keystrokes
- MLB – Number of mouse's left buttons clicks
- MRB – Number of mouse's right button clicks
- MDC – Number of double clicks
- MWS – Number of mouse's wheel scroll
- MM – Mouse movement in centimetres

All these aspects were measured with Mousotron software. For each aspect, we associated the following weights: K = 1, MLB = 2, MRB = 1, MDC = 1, MWS = 1, and MM = 0.5. The global score for each evaluated alternative is the sum of the weights for each aspect. For every change that involves sending a request over the Internet to change the configuration, we made a script. Therefore, the action of sending the request with the change translates to writing the command to execute the script.

The configuration change measurements from Figure 9 show that Proxy has the lowest complexity for changing the destination address and forwarding requests. That is because the Proxy was on the same PC used to run the experiments. Therefore, those changes only imply changing and saving local files without connecting to other machines over the Internet. We determined that Proxy cannot be used in the third experiment as the sending frequency have to be modified on the IoT devices that send the requests. The lower change complexity for the third experiment is the OTA Update as it only needs to change the value of the property that stores the sending frequency and execute the script that sends the update. OTA Update, Partial OTA, and Partially Updateable Software have significantly bigger change complexity in the second experiment than the other ones. This is due to all the code that needs to be added in order to forward the request to another server. Our proposal has the second-lowest change complexity for the second experiment and the third-lowest change complexity for the other two.

5.3 | Time the device is inactive

This is the third measurement, and it evaluates how long it takes until the device starts sending data with the new configuration after the configuration change had happened.

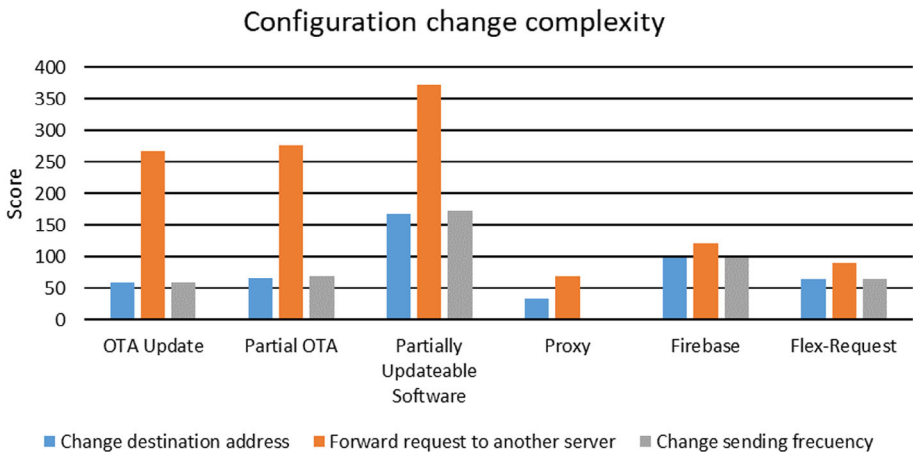

FIGURE 9 Configuration change complexity for each alternative in each experiment

Applying an update on a running software means that during the period when the device update is being applied, the device may face different issues. It can stop, run but not send data or run but send data using the old configuration. Because of that, we measure the time it takes for the device to apply the new configuration and start sending data using it.

The results of measuring the time it takes for the device to apply the change are shown in Figure 10. The only alternative that required entirely stopping the running program was the OTA Update. The times the program was not running for each experiment are 346.30 ± 8.49 ms, 321.30 ± 7.85 ms, and 356.10 ± 20.59. Other alternatives did not require the program to be stopped. Even if the change was not yet applied, it could enable, for example, storing the data and sending it after the update was done. OTA Update has the longest change apply time in each experiment. In the third experiment, the time is significantly lower because, as explained, we measure the time until the device starts sending data using the new configuration. Therefore, in the case of the OTA Update and Firebase, where the change times are the longest, the frequency of sending the data has an impact on the obtained results. In the third experiment, we change the frequency from 1 s to 0.5 s, and because of that, the first data sent with a new configuration occurs earlier. As explained earlier, in the third experiment, we determined that the proxy solution cannot be used. Our proposal had no significant difference in experiments 1 and 2 with Partial OTA, Partially Updateable Software, and Proxy, however in the third experiment, our proposal had the lowest change time.

5.4 | Network traffic

This is the last measurement. It evaluates how much network traffic consumes an IoT system developed with the different alternatives in a time interval. First, analyse a period of operation without doing configuration changes. Second, an equivalent period, but this time doing a configuration change. This will allow us to see the usual network traffic for IoT systems developed with different technologies and the cost of doing a change in the communication scheme.

To measure the network traffic, we used Wireshark software. We measured the total size (in bytes) of the packets sent between the IoT device and the main PC in a 5-min time interval. For evaluations in which the configuration was changed There are also some small typographical errors, the change was always done 1 min from the start of the measurement, because adding request forwarding to other servers or changing the sending frequency implies the modification of the total request sent and therefore the total size of the packets sent.

Figure 11 shows network traffic measurements. We measured the traffic usage of each alternative in a 5-min period without doing configuration changes, and then we measured again the same period making a configuration change. In the first case, without changes, our proposal has the highest traffic usage and is followed by Proxy and OTA Update and then the rest of the alternatives. Even though our alternative has the highest traffic usage, the relative difference with other alternatives is only between 4% and 12%.

In the case where configuration change was performed, our proposal still has the highest traffic usage. However, it is closely followed by Firebase and Partially Updateable Software, which have significantly higher network traffic relative to the case without configuration changes. The relative difference in network traffic to our proposal is between 0.4% and 9%.

Figure 12 shows the relative increase in network traffic for each alternative between measurements without changes and with changes. Firebase has the lowest traffic usage without changes with just 344,405 bytes; however, the relative increase in traffic for the case with changes is 11.30%, the highest one. Partially Updateable Software has a relative increase of 6.45%, it may be because in order to make the change, we have

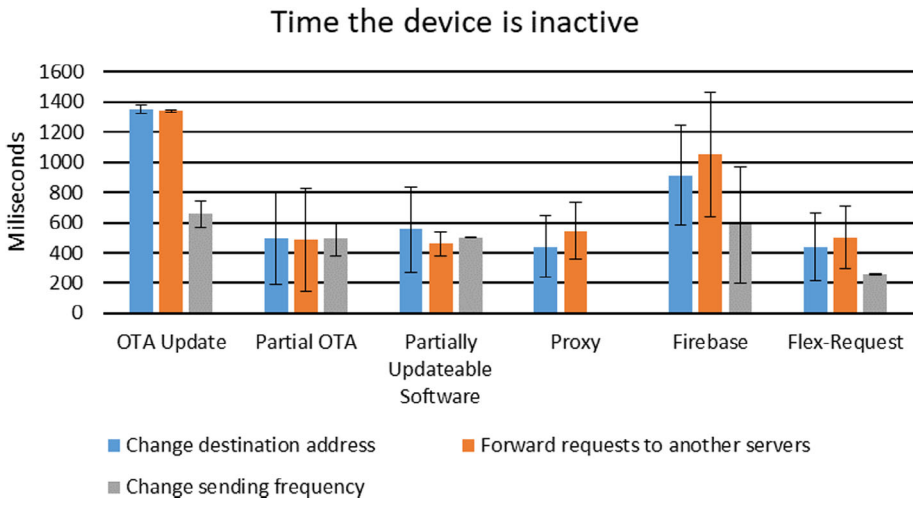

FIGURE 10 Time it takes for the device to apply the change

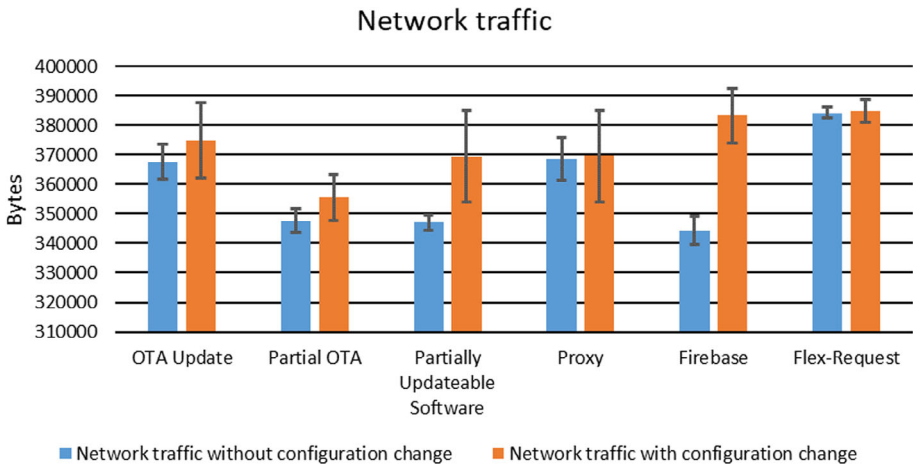

FIGURE 11 Traffic network with and without configuration change

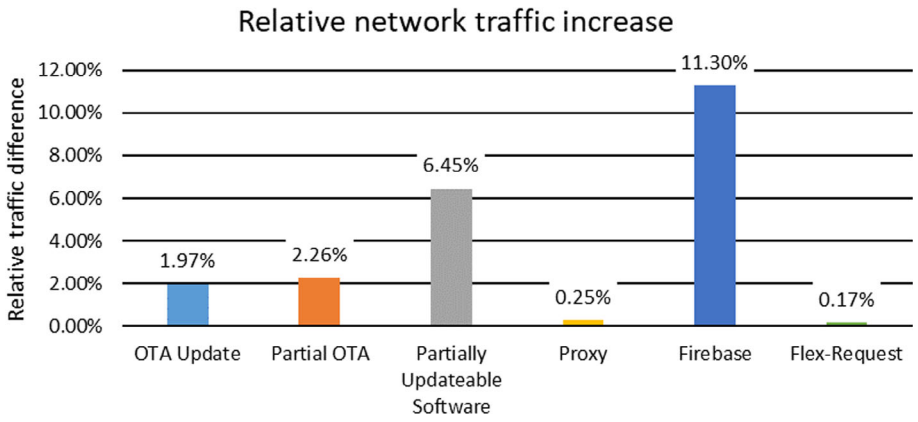

FIGURE 12 Relative network traffic increase

to connect directly to the device through ssh. Although our proposal has the highest network traffic – 38,430,140 without configuration change and 38,495,260 with changes, the relative traffic increase is the lowest one, with just 0.17%.

6 | CONCLUSIONS AND FUTURE WORK

This research work focuses on providing an alternative that allows the creation of IoT systems capable of remotely modifying devices' communication configurations in a fast and agile way. This feature will facilitate prototyping and attempt different architectures during the development phase of an IoT system. Enabling agile changes is especially useful in systems with uncertainty. During the use and exploitation phase of IoT systems, changes in execution can also be frequent, due to the introduction of new devices, services, changes in the physical environment, etc.

The proposal to address this problem was the specification of a library to be used in the software of the devices. A specific implementation of this library has been developed in Node.js. This library allows the developer to do configuration changes remotely using commands; these changes do not require updating the device software.

The proposal should not increase the implementation complexity of software for IoT devices. Based on the use cases evaluated, a software developed using the proposal could have significantly less implementation complexity than software developed using other alternatives. In the cases analysed, implementation complexity was reduced by around 40%–50%.

Once an IoT system was running, the cost of making changes to the device communication was measured. Among the analysed alternatives, the one which led to changes in the device communication with less cost was the Proxy. Still, it did not have enough functionality to make complex changes, as changes specified in use case 3. The cost of making changes for the proposal had no significant difference in comparison to three of the alternatives, and this cost was much smaller than the last of the alternatives.

The proposal should minimize the time that the IoT device is off or not working properly after doing a change in communication and the time it takes to make the change. Only one of the studied alternatives requires the device to be rebooted (OTA) after the update. The time that our proposal took to apply the change had no significant differences with other alternatives in use case 1 and 2. However, in the third use case, the time to apply the change of our proposal was between 45% and 60% shorter.

The proposal got an increase in network traffic compared to the other alternatives in the cases evaluated. This increase was between 4% and 12% during a period of normal use, without changes in communication. When these changes were done, the increase of the traffic network for the proposal was reduced to 0.4%–9%. According to the results, the proposal always has more consumption of network traffic than other alternatives, but the relative increase in traffic network for each change made is just 0.17%, much less than in other alternatives.

Results show that this proposal could be very suitable for many IoT systems which require applying changes in their devices' communication.

Future research lines will be related to include the possibility of changing data protocols (CoAP, AMQP, MQTT, etc.), communication schemes, and reducing the impact of the proposal over the traffic network.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

ORCID

Jordán Pascual Espada https://orcid.org/0000-0001-8076-282X
Rubén González Crespo https://orcid.org/0000-0001-5541-6319

ENDNOTES

[1]Firebase Remote Config – https://firebase.google.com/docs/remote-config.

[2]Node-supervisor – https://github.com/petruisfan/node-supervisor.

[3]Hotswap – https://github.com/rlidwka/node-hotswap.

[4]https://www.balena.io/.

[5]http://www.cloudino.io/.

[6]https://cloud.google.com/iot-core.

[7]https://azure.microsoft.com/es-es/services/iot-central/.

[8]https://aws.amazon.com/es/iot-device-management/.

REFERENCES

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A survey on enabling technologies, protocols, and applications. *IEEE Communication Surveys and Tutorials*, *17*, 2347–2376. https://doi.org/10.1109/COMST.2015.2444095

Alsalemi, A., Al Homsi, Y., Al Disi, M., Ahmed, I., Bensaali, F., Amira, A., & Alinier, G. (2017). Real-time communication network using firebase cloud IoT platform for ECMO simulation. In *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)* (pp. 178–182). IEEE.

Antinyan, V., Staron, M., & Sandberg, A. (2017). Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time. *Empirical Software Engineering*, *22*, 3057–3087. https://doi.org/10.1007/s10664-017-9508-2

Bannatyne, R., & Viot, G. (1998). Introduction to microcontrollers. I. In *Northcon/98. Conference Proceedings (Cat. No.98CH36264)* (pp. 238–248). IEEE.

Barron-Gonzalez, H. G., Martinez-Espronceda, M., Led, S., Serrano, L., Fischer, C., & Clarke, M. (2013). New use cases for remote control and configuration of interoperable medical devices. In *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (pp. 4787–4790). IEEE.

Barron-Gonzalez, H. G., Martinez-Espronceda, M., Trigo, J. D., Led, S., & Serrano, L. (2014). Proposal of a novel remote command and control configuration extension for interoperable personal health devices (PHD) based on ISO/IEEE11073 standard. In *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society* (pp. 6312–6315). IEEE.

Chang, Y. C., Chi, T. Y., Wang, W. C., & Kuo, S. Y. (2013). Dynamic software update model for remote entity management of machine-to-machine service capability. *IET Communications*, *7*, 32–39. https://doi.org/10.1049/iet-com.2012.0459

Chiang, M. L., & Lu, T. L. (2011). Two-Stage Diff: An efficient dynamic software update mechanism for wireless sensor networks. In *Proc – 2011 IFIP 9th Int Conf Embed Ubiquitous Comput EUC 2011* (pp. 294–299). IEEE. https://doi.org/10.1109/EUC.2011.74

Cueva-Fernandez, G., Espada, J. P., García-Díaz, V., García, C. G., & Garcia-Fernandez, N. (2014). Vitruvius: An expert system for vehicle sensor tracking and managing application generation. *Journal of Network and Computer Applications*, *42*, 178–188. https://doi.org/10.1016/j.jnca.2014.02.013

Dalipi, E., Van Den Abeele, F., Ishaq, I., Moerman, I., & Hoebeke, J. (2017). EC-IoT: An easy configuration framework for constrained IoT devices. In *2016 IEEE 3rd World Forum Internet Things, WF-IoT 2016* (pp. 159–164). IEEE. https://doi.org/10.1109/WF-IoT.2016.7845483

Datta, S. K., & Bonnet, C. (2016). Easing IoT application development through DataTweet framework. In *2016 IEEE 3rd world forum on internet of things (WF-IoT)* (pp. 430–435). IEEE.

Espada, J. P., Díaz, V. G., Crespo, R. G., Martínez, O. S., G-Bustelo, B. C. P., & Lovelle, J. M. C. (2015). Using extended web technologies to develop Bluetooth multi-platform mobile applications for interact with smart things. *Information Fusion*, *21*, 30–41. https://doi.org/10.1016/j.inffus.2013.04.008

García, C. G., Núñez-Valdéz, E. R., García-Díaz, V., G-Bustelo, B. C. P., & Lovelle, J. M. C. (2019). A review of artificial intelligence in the internet of things. *International Journal of Interactive Multimedia and Artificial Intelligence*, *5*, 9–20. https://doi.org/10.9781/ijimai.2018.03.004

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*, 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

Jin, W., & Kim, D. H. (2018). IoT device management architecture based on proxy. In *Proc 2017 6th Int Conf Comput Sci Netw Technol ICCSNT 2017 2018-Janua* (pp. 84–87). IEEE. https://doi.org/10.1109/ICCSNT.2017.8343663

Jurković, G., & Sruk, V. (2014). Remote firmware update for constrained embedded systems. In *2014 37th Int Conv Inf Commun Technol Electron Microelectron MIPRO 2014 – Proc* (pp. 1019–1023). IEEE. https://doi.org/10.1109/MIPRO.2014.6859718

Kayode, O., & Tosun, A. S. (2019). Analysis of IoT traffic using HTTP proxy. In *ICC 2019–2019 IEEE international conference on communications (ICC)* (pp. 1–7). IEEE.

Kikuchi, S., Thomas, I., Jallouli, O., Doerr, J., Morgenstern, A., Baccelli, E., & Schleiser, K. (2018). Orchestration of IoT device and business workflow engine on cloud. In *2018 3rd Cloudification of the Internet of Things (CIoT)* (pp. 1–2). IEEE.

Kolomvatsos, K. (2018). An intelligent, uncertainty driven management scheme for software updates in pervasive IoT applications. *Future Generation Computer Systems*, *83*, 116–131. https://doi.org/10.1016/j.future.2018.01.036

Kortuem, G., Kawsar, F., Sundramoorthy, V., & Fitton, D. (2010). Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, *14*, 44–51. https://doi.org/10.1109/MIC.2009.143

Kumar, S., Solanki, V. K., Choudhary, S. K., Selamat, A., & González-Crespo, R. (2020). Comparative study on ant colony optimization {(ACO)} and K-means clustering approaches for jobs scheduling and energy optimization model in internet of things (IoT). *International Journal of Interactive Multimedia and Artificial Intelligence*, *6*, 107–116. https://doi.org/10.9781/ijimai.2020.01.003

Kumper, D., & Tonjes, R. (2011). Remote configuration and deployment of sensor drivers for a medical bluetooth sensor gateway. In *2011 IEEE international symposium on a world of wireless, Mobile and multimedia networks* (pp. 1–6). IEEE.

Lee, G., Heo, S., Kim, B., Kim, J., & Kim, H. (2017). Rapid prototyping of IoT applications with Esperanto compiler. In *Proceedings of the 28th International Symposium on Rapid System Prototyping Shortening the Path from Specification to Prototype – RSP '17* (pp. 85–91). ACM Press.

Mazzei, D., Baldi, G., Montelisciani, G., & Fantoni, G. (2016). A full stack for quick prototyping of IoT solutions. In *2016 Cloudification of the internet of things (CIoT)* (pp. 1–5). IEEE.

Mukhtar, H., Kim, B. W., Kim, B. S., & Joo, S. S. (2009). An efficient remote code update mechanism for wireless sensor networks. In *IEEE Military Communications Conference Proceedings – MILCOM*. IEEE. https://doi.org/10.1109/MILCOM.2009.5379862

Nepomuceno, T., Carneiro, T., Carneiro, T., & Martin, A. (2018). A GUI-based platform for quickly prototyping server-side IoT applications. In *Smart SysTech 2018; European Conference on Smart Objects, Systems and Technologies* (pp. 1–9). IEEE.

Nikolaidis, A. E., Papastefanos, S. S., Stassinopoulos, G. I., Drakos, M. P. K., & Doumenis, G. A. (2006). Automating remote configuration mechanisms for home devices. *IEEE Transactions on Consumer Electronics*, *52*, 407–413. https://doi.org/10.1109/TCE.2006.1649657

Patel, P., & Cassou, D. (2015). Enabling high-level application development for the internet of things. *Journal of Systems and Software*, *103*, 62–84. https://doi.org/10.1016/j.jss.2015.01.027

Raptis, T. P., Passarella, A., & Conti, M. (2018). In K. R. Chowdhury, M. Di Felice, I. Matta, & B. Sheng (Eds.), *Distributed path reconfiguration and data forwarding in industrial IoT networks* (pp. 29–41). Springer International Publishing.

Razzaque, M. A., Milojevic-Jevric, M., Palade, A., & Clarke, S. (2016). Middleware for internet of things: A survey. *IEEE Internet of Things Journal*, *3*, 70–95. https://doi.org/10.1109/JIOT.2015.2498900

Ruckebusch, P., Van Damme, J., De Poorter, E., & Moerman, I. (2016). Dynamic reconfiguration of network protocols for constrained internet-of-things devices. In *Lecture notes of the Institute for Computer Sciences* (pp. 269–281). Social-Informatics and Telecommunications Engineering, LNICST.

Sarkar, C., Nambi, S. N. A. U., Prasad, R. V., Rahim, A., Neisse, R., & Baldini, G. (2015). DIAT: A scalable distributed architecture for IoT. *IEEE Internet of Things Journal*, *2*, 230–239. https://doi.org/10.1109/JIOT.2014.2387155

Seo, B., Kim, S. H., & Choi, H. (2013). The remote command and control system for outdoor robot. In *2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)* (pp. 303–304). IEEE.

Singh, K. J., & Kapoor, D. S. (2017). Create your own internet of things: A survey of IoT platforms. *IEEE Consumer Electronics Magazine*, *6*, 57–68. https://doi.org/10.1109/MCE.2016.2640718

Su, M., Zhou, B., Fu, A., Yu, Y., & Zhang, G. (2020). PRTA: A proxy re-encryption based trusted authorization scheme for nodes on CloudIoT. *Information Sciences (NY)*, *527*, 533–547. https://doi.org/10.1016/j.ins.2019.01.051

Vaquero, L. M., & Rodero-Merino, L. (2014). Finding your way in the fog. *ACM SIGCOMM Computer Communication Review*, *44*, 27–32. https://doi.org/10.1145/2677046.2677052

Weisbach, M., Taing, N., Wutzler, M., Springer, T., Schill, A., & Clarke, S. (2017). Decentralized coordination of dynamic software updates in the Internet of Things. In *2016 IEEE 3rd World Forum Internet Things, WF-IoT 2016* (pp. 171–176). IEEE. https://doi.org/10.1109/WF-IoT.2016.7845450

Zhang, D., Wan, J., (Robert) Hsu, C.-H., & Rayes, A. (2016). Industrial technologies and applications for the Internet of Things. *Computer Networks*, *101*, 1–4. https://doi.org/10.1016/j.comnet.2016.02.019

AUTHOR BIOGRAPHIES

Karol Mateusz Ciok is a Research scientist at Computer Science Department of the University of Oviedo. He received BSc from the University of Oviedo in Computer Science Engineering. His research interests include the Internet of Things, robots, ubiquitous computing and emerging technologies, particularly mobile and Web applications.

Jordán Pascual Espada received a PhD in Computer Science from the University of Oviedo (Spain) in 2011. He is a professor at Computer Science Department of the University of Oviedo (Spain). He received BSc in Computer Science Engineering and a MSc in Web Engineering from

the University of Oviedo. His research interests include the Internet of Things, exploration of new applications and associated human computer interaction issues in ubiquitous computing and emerging technologies, particularly mobile and Web applications. He has been author of published papers in several journals and recognized international conferences.

Rubén González Crespo, PhD, is the deputy director of the School of Engineering at the Universidad Internacional de La Rioja. Professor of Project Management and Engineering of Web sites. He is honorary professor and guest of various institutions such as the University of Oviedo and University Francisco José de Caldas. Previously, he worked as Manager and Director of Graduate Chair in the School of Engineering and Architecture at the Pontifical University of Salamanca for over 10 years. He has participated in numerous projects I + D + I such as SEACW, GMOSS, eInkPlusPlusy among others. He advises a number of public and private, national and international institutions. His research and scientific production focuses on accessibility, web engineering, mobile technologies and project management. He has published more than 80 works in indexed research journals, books, book chapters and conferences.

How to cite this article: Mateusz Ciok, K., Pascual Espada, J., & González Crespo, R. (2022). Flex-request: Library to make remote changes in the communication of IoT devices. *Expert Systems*, e12994. https://doi.org/10.1111/exsy.12994