MDPI

Article

Light Node Communication Framework : a new way to communicate inside a Smart Home

Valère Plantevin [1], Abdenour Bouzouane [2] and Sebastien Gaboury [3]

[1] UQAC; valere.plantevin1@uqac.ca
[2] UQAC; abdenour_bouzouane@uqac.ca
[3] UQAC; Sebastien_gaboury@uqac.ca
* Correspondence: valere.plantevin1@uqac.ca

Version November 7, 2017 submitted to Sensors; Typeset by LaTeX using class file mdpi.cls

Abstract: The Internet of Thing has profoundly changed the way we imagine information science and architectures and Smart Homes are an important part of this domain. Created a decade ago, the few existing prototypes use the technologies of the day forcing designers to create centralized and costly architectures that raise some issues concerning the reliability, the scalability and ease of access which cannot be tolerated in an assistance context. In this paper we briefly introduce a new kind of architecture where the focus was placed on the distribution and especially. More specifically, we answer the first issue we met by proposing a lightweight and portable messaging protocol. After running several tests, we observed a maximized bandwidth, no packets were lost and a good encryption was obtained. These results tend to prove that our innovation may be employed in a real context of distribution on small entities.

Keywords: Messaging protocol; IoT; SmartHome; Distributed Computing

1. Introduction

The evolution of our society towards the all-digital of the Internet of Things (IoT) profoundly remodeled our relationship with the science of information. In this new one, the smart home became the subject of numerous researches [1–3] and joins the recent current of thought stemming from the Ambient Intelligence (Amb. I). This last one refers to a tendency that wants us to miniaturize a set of electronic devices (sensors and effectors) in order to integrate them into any object of everyday life (lamp, refrigerator, etc.) in a transparent way for the person. The aim behind this idea is to supply punctual assistance to the occupants according to the gathered information and to the history of the accumulated data.

The vast majority of work in the smart home domain focuses on the activity recognition problem in order to assist the inhabitant with a potential dementia often caused by an advanced age[4–6]. Nevertheless, none of them seems to propose a standard architecture which provides both high-reliability and scalability capabilities at a relative low-cost. And still, high-reliability has to be a mandatory feature of such architecture since the assistance is vital for the inhabitant with a potential dementia. Moreover, as the disease can stay for decades, any work on architecture must take the scalability parameter into account since many sensors or improvements can be realized during the illness evolution. Finally, the low-cost aspect has to be taken into account as the vast majority of the aging population will be located in poor or developing countries by 2050 [7]. As far as we know, this paper is the first focusing on those three points in particular.

Here, we briefly introduce a new kind of smart home architecture providing both reliability and scalability based on low-cost smart sensors. To achieve this objective, the first issue we ran into is the difference between all the possible entities in the environment. In fact, our solution has to integrate

different operating systems (e.g. Linux or FreeRTOS) running on different hardware (e.g. computer or microcontroller) and using different communication technologies (e.g. Wi-Fi, ZigBee or 6LowPan). To answer these dissimilarities, we have to use a highly portable communication protocol using a broker less architecture to provide the highest reliability. This point will be the main concern of this paper. Even if many protocols exist like MQTT, RabbitMQ or ZeroMQ [8–10] none of them fully answer our requirements since the first two require brokers to work and the last one is based on POSIX sockets and cannot be embedded in some light systems. Consequently, the contribution we make in this paper is a new way to communicate that can be embedded in every system as soon as they implement an IP stack. Our solution provides discovery mechanism, security via AES encryption and two different channels in order to address the difference between configuration messages and data messages.

This paper is divided in four sections. The first one will present a state of the art about existing smart homes and their architectures. The second part will cover the technological breakthroughs that the embedded computing has experienced since the creation of the first Smart Homes. Then, the proposed solution will be explained and some tests on the messaging protocol will be presented in the third section. Finally, a conclusion and some future works will end this paper.

2. Existing architectures

Many smart habitations have been implemented in laboratories since the creation of the ambient intelligence [1,3,11,12]. Each of these projects use the technology of its day to create a testing environment in which the data accessibility was the main challenge. Here, we depict three of those starting with the LIARA and DOMUS, which share the same architecture [12]. We continue this review with the Gator Tech house [3] and CASAS [1]. Finally, we end this part by describing Software Defined Smart Home [13,14], a recently released architecture based on software defined networks.

2.1. LIARA and DOMUS

LIARA and DOMUS laboratories aim to study how smart homes can assist people with cognitive deficiencies. They both created a very similar architecture, represented in Figure 1, to test their algorithms and solutions [15]. Inherited from the industry, they use some islands, made of industrial grade hardware, to agglomerate transducers. Then, an automate is in charge of getting back, from the islands, the values of the sensors or changing the values of the effectors. To end this process, the automate will update a relational database hosted on a SQL Server in order to provide a simple interface for other systems like, in this case, an artificial intelligence.

These two smart homes present some interesting features we have to discuss. First, the use of industrial grade hardware means that all the components have been tested for a continuous use in a far much harder environment than just a house (e.g. production line in a factory). Therefore it demonstrates an excellent reliability even if the smart home has to operate at all times. The second main advantage of these environments comes from the highly centralized architecture itself. Indeed, all the values coming from sensors and all the actuator controls end up in the same database. As a result, it facilitates interaction with the home since this kind of storage offers an easy way to retrieve sensors values or interact with actuators.

Nevertheless, industrial material suffers from two main drawbacks. The first one is the introduction of black boxes in a research environment. As a matter of fact, the communications between all these pieces of hardware often rely on proprietary libraries that can impact future evolution. The second main disadvantage is the price of such an architecture. Based on the hardware presented by Bouchard *et al.* and the price of it, we were able to compute the total price of the chain Island-Automate-Main Server. With 2000 dollars each island [16], 1500 dollars the automate [16] and 4000 the server [17], the architecture reaches 13,500 dollars without any transducers or backbone structure (e.g. networking, cooling for the server, maintenance). Finally, the highly centralized architecture presented here creates many Single Points of Failure (SPoFs) like the automate, the Islands and the main server hosting both the AI and the SQL server. So if one of these SPoFs fails,

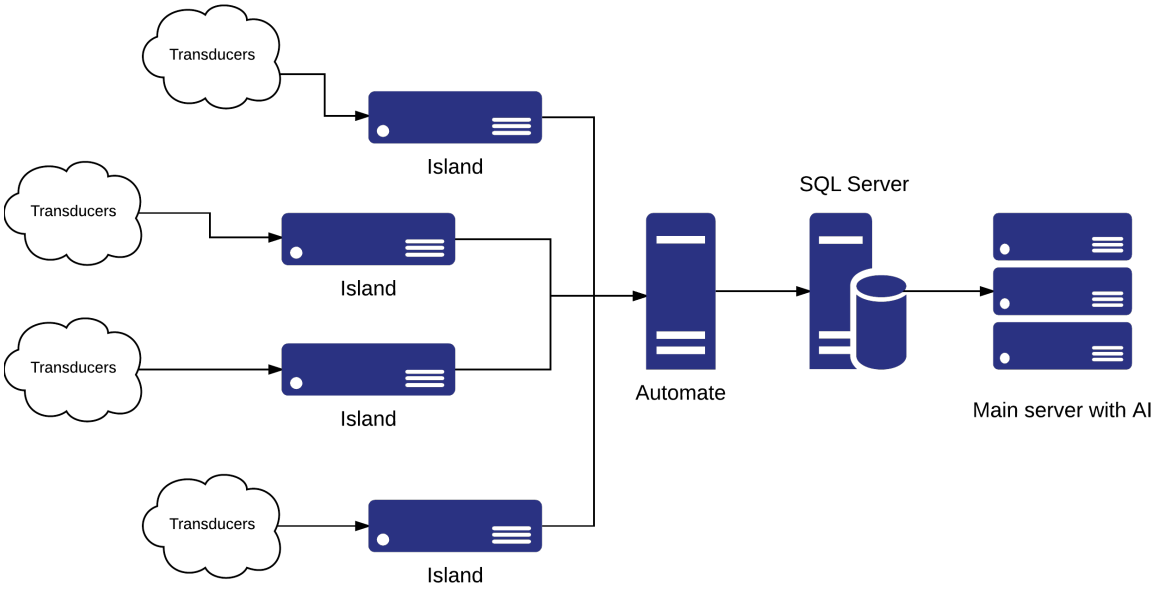

Figure 1. The LIARA and DOMUS architecture.

at least a quarter of the environment and its assistance will fail too. Moreover, these points represent some serious bottlenecks in the architecture preventing a real scalability.

2.2. Gator Tech

Gator Tech [3] is a project funded in Florida. Its main goal is to prove the feasibility of a low-cost smart home where the integration of new transducers will be easy. In order to accomplish that the authors present an OSGI [18] based architecture sums up in Figure 2. In this last one, each transducer has a simple EEPROM memory containing the driver to communicate with it. Once powered, the transducer registers itself by sending its driver to an OSGI service definition. This last one will act as an abstraction layer to create basic services that allow the consumption of highly abstracted data (e.g. "Sunny" instead of 10 000 lumen for a light sensor) or the combination of basic services to a composite service (e.g. create a voice recognition service on all the different microphone services).All this architecture allows developers to create applications without any knowledge of the underlying communication and with only highly abstracted data which simplify the development.

This environment has some really good advantages. First of all, the automatic transducer registration really helps the scalability of such an environment (e.g. add new sensors or replace some of them). Secondly, the high abstraction of the data generates by this system greatly helps the application development. For example, it is straightforward to enable the air conditioning when the temperature is "Hot." However, it is more complicated when the decision is only based on the microcontroller value since this one depends on the hardware (e.g. the microcontroller itself, the temperature sensor or even the analog to digital chips). Finally, the price of such infrastructure is as low as possible as every transducer is designed to be the most affordable possible by using Atmega128 as the main processor unit which is a low-cost platform [19]. Moreover, because every transducer is wireless, there is no need of Islands or Automate as in LIARA and Domus homes. Despite all these great advantages, the use of OSGI on a unique server create a SPoF which is a big problem in a high reliability architecture.

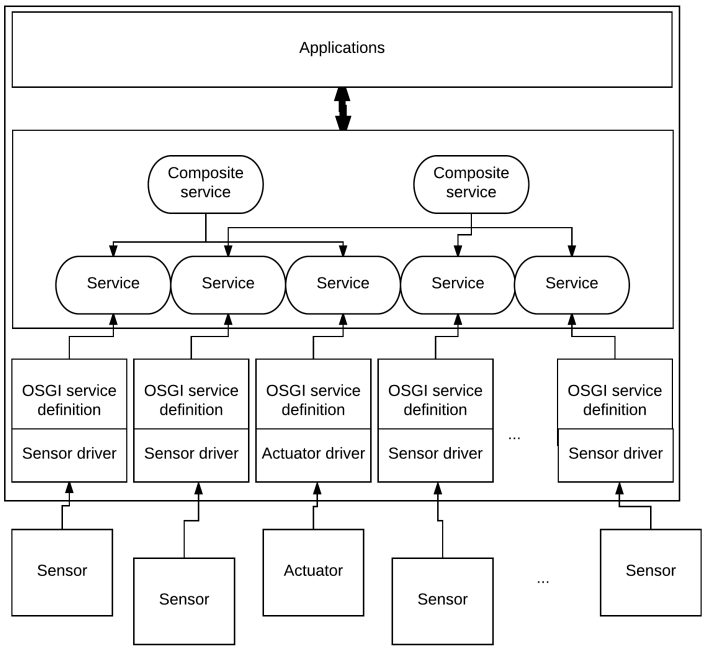

Figure 2. The Gator Tech architecture.

2.3. CASAS

CASAS [1], or the Smart Home in a box, is an infrastructure where the accent was put on the price and the ease of installation. Depicted in Figure 3, the architecture is divided in four main elements. The first one is the ZigBee mesh which represents all the transducers communicating between them in a network where every node relay the information to its neighbors by using the ZigBee protocol. This mesh sends events on a Publish/Subscribe (Pub/Sub) messaging service through a ZigBee bridge in charge of converting events to higher-level XMPP messages. The messaging service allows other applications to easily integrate the infrastructure and use the transducers. By default, there are two services which are the Storage and the Intelligence. The first archives all the events occurring in the environment by using the Scribe Bridge. As for the intelligence, it is in charge of energy monitoring and the discovery and recognition of any activity that can happen in the house.

CASAS has two main benefits : the price and the ease of installation. For the first, the authors present a detailed summary of the cost. They state that their solution cost only 2,765 dollars which is really a great achievement. Regarding the ease of installation, they demonstrate it by conducting a test on people aged from 21 to 62 and it requires only an hour to set up the whole environment. In spit of these qualities, CASAS, as the other architecture, suffer from the existence of many SPoFs like the ZigBee bridge or the Application Bridge which can stop the assistance or the Pub/Sub-messaging service which is a sensitive component.

2.4. Software Defined Smart Home

The works previously described are the old founders of the Smart Home architecture. However, some more recent papers exist in this particular domain [13,14]. One of them introduce the idea of Software Defined Smart Home or SDSH for short [13]. This concept is a new way to integrate heterogeneous smart appliances (e.g. Smart Light, Smart Flowerpot, etc.) in one homogeneous platform creating a Smart Home from the chaos of the different hardwares and communication protocols implemented by the companies who create these devices. To achieve such a goal, the authors propose a three layers architecture derived from software defined networks [20]. First, the Smart Devices layer includes all the different kinds of smart hardware in a home (a.k.a. the smart

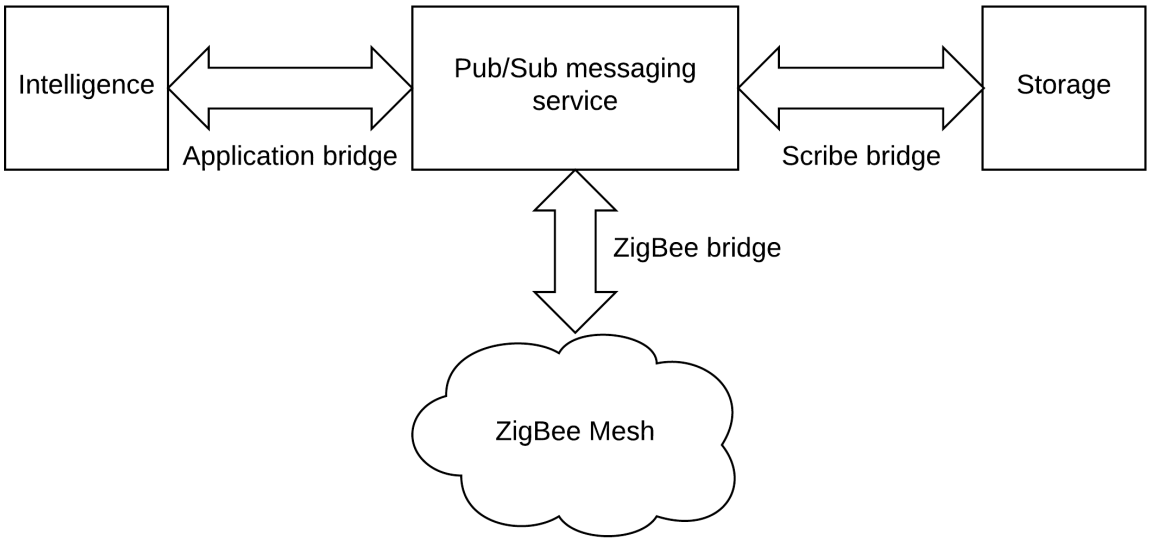

Figure 3. The CASAS architecture.

appliances). Next, the controller layer is a centralized management service locally implemented or deployed in the cloud. Its main goal is to hide the implementation details of the hardware layer, retrieve and analyze the user demands and manage the whole smart home. Moreover, it is in charge of encapsulates information extracted from the smart home and provide them to the last layer : the external service layer. This last one uses the smart home resources to provide some smart services like home security or medical attention.

SDSH offers some great features. First of all, as Gator Tech, this architecture offers a strong separation between raw sensors and final services via its controller layer. Next, it uses OpenFlow [21] as its main protocol which is a well-known protocol widely implemented in software defined network. Finally, it uses smart appliances already in the market and standardize the access to their data. Unfortunately, the centralized controller depicted as one of the main advantages of this work is also a great default because, if it allows to configure the whole Smart Home at the same place, it represents a severe single point of failure. To answer this problematic, the authors introduced visualization techniques but these one require either an Internet cloud connection (which cannot be tolerated in some applications) or a server strong enough to deal with many systems started on the same hardware which can be very expensive for a single house.

2.5. Conclusion on existing architectures

All these existing architectures have some common points. First the majority of their components are transducers. Moreover, according to the Gator Tech and CASAS cases, it seems that embed some intelligence and communication abilities in them helps to reduce costs and ease installation and scalability. Finally, we have to point out the common problem in all these architectures: the centralization. This weakness creates single points of failure which can lead to a complete stop of the assistance. In corporate computing and Web domains, this particular issue has been solved ten years ago by using redundancy, clusters and distributed computing [22–25].

The ideal architecture appears to be composed of many smart transducers easing the scalability of such architecture. This specific attribute bring our environment closer to another computer science domain which is the Internet of Things (IoT). In this last one, already used in the Smart Home [26,27], a multitude of smart objects communicate in a uniform manner and generates a huge amount of data often associate to "Big Data" [28]. In this last case, it is not conceivable to handle the information in a centralized way any more, even if we use server clusters. It is more appropriate to use decentralized

methods relocating the intelligence as close as possible to the units composing this huge data pool [28,29].

3. Technological breakthroughs

The transducers (i.e. sensors and actuators) are the essential basis of every Smart Home architecture. CASAS and Gator Tech case studies proved that embed intelligence and communication in these entities allow to reduce costs while improving the ease of implementation and the scalability. In this part, we are going to study the concept of a smart transducer such as designed by the standards. Then, we will review some hardware evolution realized since the creation of the first Smart Homes.

A smart transducer is clearly defined in the IEEE 1451.2 standard [30]. To sum up, it is an entity providing more features than the one's mandatory to generate a good representation of the controlled quantity. Some of these attributes can be sensor identification, a process to simplify the installation or the maintenance, network interfaces or the coordination and synchronization with other entities [31]. In order to guide the community, Lewis [31] proposed three objectives for these transducers. The first one is to move the intelligence closest to the sensing point. The second one is to make the installation easier, and the maintenance of massive distributed sensor networks less expensive. The last one is to facilitate the interfacing of many different sensors. Now that the concept of a smart transducer is well defined, we can work at the different technological breakthroughs that occur during the last ten years and allow us to finally design and build inexpensive smart sensors for the smart environments.

Many prototypes of smart environment have emerged during the last decade (e.g. Gator Tech in 2005 or LIARA/DOMUS in 2009 [3,12]. They have been built on top of existing technologies, which for the most part are anterior to great innovations made recently. One of these is the apparition and especially the democratization of System on Chip (SoC) which are full systems integrated on a single substrate providing all the elements to run an application (e.g. processor, memory, radio). The SoCs are the cutting edge of the modern electronic and can be found everywhere from smart sensors to nano-computers and drive the price and power consumption reduction in all the modern devices [32].

To illustrate the growth in power and integration, we propose to make a quick comparison between two microcontrollers platforms and some nano computers. The first two are the Arduino USB, easily accessible at the time of the creation of the first smart homes, and the latest released ESP 32 from the Espressif company. Concerning the nano computers, we chose the evolution of the Raspberry Pi since its creation. The attributes we compare are the released year, the processor frequency, the memory available, the connectivity, the relative size and the price. Tables 1 and 2 both represents the different values for the attributes retain respectively for the micro controllers and the Raspberry Pi. The first thing that jump out from these tables is the increase in both processor frequency and memory with 16 MHz and 1 kB of RAM for the Arduino USB to a dual core 240 MHz with 512 kB of RAM for the ESP 32. And the phenomenon is the same for the different Raspberry Pi with 700 MHz and 512 MB of RAM for the Pi1 to 1 GHz and the same amount of memory but with half the size of the Pi Zero W and four cores at 1.2GHz with 1 GB of RAM for the Pi3 but with the same form factor. Moreover, it is pretty obvious that the embedded connectivity became a must in this period with the integration of Wi-Fi and both Bluetooth and BLE on the ESP 32 and Wi-Fi/BLE for the Pi Zero W and Pi3. Finally, it must be noted that the price of these platform stay the same of decrease drastically even if the platforms increase in power and connectivity.

Technological evolution since the beginning of the 2000s was impressive. The democratization of the SoC permits an increase of power for such piece of hardware while reducing costs and power consumption. In parallel, SoC integrate much more advanced features like Wi-Fi and Bluetooth communication. Subsequently, it seems now possible to create powerful applications on embedded hardware and one of these applications is the creation of more intelligent transducers as depicted in the IEEE 1451 standard.

Table 1. Arduino USB and ESP 32 comparison

	Arduino USB	ESP32 Thing
Released Year	2005	2016
Processor Frequency	16 MHz	2 x 240 Mhz
Memory	1 kB	512 kB
Connectivity	None	Bluetooth + BLE and WiFi
Relative size	1	0.5
Price (USD)	35	7

Table 2. Raspberry Pi platform over time

	Pi 1	Pi Zero W	Pi 3
Released Year	2012	2017	2016
Processor Frequency	700MHz	1 GHz	4 x 1.2GHz
Memory	512 MB	512 MB	1 GB
Connectivity	None	BLE/WiFi	BLE/WiFi
Relative size	1	0.5	1
Price (USD)	35	9	35

4. Proposed solution

We saw that existing smart homes had some weaknesses in both reliability and scalability. Yet these kinds of weak points are not bearable in the assistance domain. Here, we propose a new kind of architecture using the latest technological advances to provide a reliable and scalable distributed environment to safely run the assistance.

The main concept behind our solution is that the only non-removable elements of a smart environment are the transducers themselves. And if we think about it, they represent a vast number of distributed entities. With the latest hardware innovations it is feasible to equip each of them with an intelligent entity with both communication and processing capabilities creating a huge network with highly distributed computation potential and no single point of failure. In this vision, the generic smart entities have to answer the three main objectives firstly formulated by Lewis [31]. It means that they must allow to move the artificial intelligence to the closest sensing point, provide methods to easily install, configure and maintain this smart network and finally ease the interfacing between many different sensors. The first issue that such an architecture has to deal with is the difference between all the intelligent entities we can use. Indeed, if we want our solution to be the most generic possible we have to cope with the most different hardware and operating systems. Thus, we want to make feasible the integration of sensors based on different operating systems (e.g. Linux or FreeRTOS) implemented on different hardware but also to be able to interface different communication protocols (e.g. ZigBee, Wi-Fi or BLE). In order to answer this problem, we had to think of a new way to communicate between all these entities.

4.1. Communication protocol

There are many ways to communicate by using messages in the literature or industry. As far as we know the most popular ones are MQTT, RabbitMQ and ZeroMQ [8,9,33]. The first of them is mainly used in the Internet of Thing application by its high portability and its reduce footprint in terms of memory and power. It's a publish/subscribe protocol where clients connect to a centralized instance named broker. It supports different type of quality of service which affects the reliability of communication (message is delivered at most once, at least once or exactly once). Finally, it can support a "Last will and testament" (LWT) which allows to send a specific message on a specific subject when the entity disconnect in an abnormal way from the network. RabbitMQ, on the other hand, is a leading messaging protocol mainly use in distributed architectures. It implements the

Advanced Message Queuing Protocol (AMQP) and consequently has a broker architecture which provides ease of development in favor of scalability and speed since the broker adds latency and treatment and the message exchanged are pretty big. Finally, ZeroMQ is a messaging system which allows developers to create themselves the architecture including brokerless ones. The main problem with this approach is the portability since ZeroMQ relies on POSIX sockets which are only supported in Unix and Windows operating systems. To conclude, none of the leading messaging protocol fit in our application since we want one without a centralized unit like a broker and heavily portable in order to deal with the most part of the possible entities in a Smart Home which can be composed of embedded systems running on top of different Real Time Operating Systems (RTOS) like FreeRTOS or RiotOS.

The contribution we make in this paper is a new communication protocol with two main characteristics. First, it can be embedded on any device from computers to microcontrollers as long as they implement an IP stack (over Wi-Fi, 6LowPan or ZigBee IP). Second, our protocol does not have the need for any main server also known as a broker. This last point was an issue in the most popular solutions (e.g. MQTT, NATS, etc.). To build our solution, we made two basic assumptions. The first one is that all our messages will stay in the smart home network. The second is that UDP is the minimum requirement for any device that wants to communicate over a network as it is the base of many network configuration protocols (e.g. DHCP or DNS).

One of the first issues we ran into is the fundamental difference between configuration and data streams. The first one has to be based on a reliable delivery system allowing point to point communication without the urge of the highest data speed. The second one, have to be able to stream a huge quantity of information in a minimum of time without the highest reliability to many different listeners. In order to answer this problematic, we propose to use two different channels like the FTP protocol [34]. We will now explain how these two channels work in order to offer all the features we want.

4.1.1. Configuration channel

As said sooner, the configuration of a smart entity has to be distributed over a reliable communication. In order to achieve this objective, we propose to use CoAP, a well-known IoT protocol, already implemented in many platforms [35]. It allows us to use HTTP-like request to get or change values represented by URI in a fail-safe manner based on an acknowledgment system for important messages (i.e. messages with high reliability). We propose to use this URI representation for the entity configuration. In order to facilitate the understanding of such a concept, we present a simple example in Table 3. In this table, each line represents a possible configuration variable with its CoAP URI, the methods allowed in order to get or set the information and the type of data that is asked by the entity. The first one is pretty obvious and represents the update frequency used by a potential sensor. It is a simple integer, represents by the URI /rate and that can be obtained or modified by using respectively GET or POST request. The usage of a HTTP-like protocol allows us to define read-only values like the version which is a string that is only reachable via a GET request on the /version URI. Moreover, we can exchange much more complex data types like JSON to clearly define hardware configurations and interaction. Another interesting feature of CoAP that we use in our configuration sample is the block-wise extension of the protocol that permit the transfer of large binary file like updates for the embedded software. Finally, the last feature of CoAP we use in this configuration channel is the ability to encrypt the communication by the usage of DTLS. To demonstrate the utility of such a feature we propose to change symmetric encryption keys on the device. This kind of operation is critical since it has to be highly secured in order to guarantee the fact that nobody can intercept these keys to listen and speak over a secure network. With our method, we simply exchange keys by using CoAP protected by SSL which guarantees both confidentiality and

Table 3. A configuration example based on CoAP

	URI	Methods allowed	Data type
Update rate	/rate	GET/POST	Integer
Version	/version	GET	String
Hardware	/hardware	GET/POST	JSON
Update	/update	POST	Binary
Encryption key	/keys	POST	Binary

authentication. Now that the configuration channel operation is explained, we can think about how to exchange information through the data channel.

4.1.2. Data channel

In addition to a reliable communication channel to ensure the good configuration of any device, we have to provide a method to exchange data messages between the smart entities in our architecture. We want to be able to transfer huge data in a minimal amount of time, to discover smart sensors connected to the network and to give the ability to encrypt sensitive information. To attain these aims, we propose a simple publish/subscribe messaging protocol based on UDP multicast chosen for its ability to transfer to one or many listeners at once in addition to its low-latency. To sum up the protocol work-flow, users join the multicast group, they register to any topic, represented by a simple character string, and will receive any messages labeled by this topic. They may also ask for any sensors connected to the network group by sending a discovery packet with a request in it (e.g. sensors with service "temperature") and those sensors will answer with their IP address and the different topics they expose. We are now going to explain the three different modes of our messaging protocol which are data, discovery and encrypted data.

4.1.3. Data message

Figure 4a represents a single and unencrypted data message in our solution. Every packet begins with an options byte, divided in two parts. The first three most significant bits (MSB) compose the version number of this packet and is all set to 0 for the version we present here. The other bits are reserved and unused except for the two least significant bits (LSB) who represent flags used in discovery and encrypted mode and have to be set to 0. These options are followed by the topic length which can go from 1 to 255 characters encoded on a single byte. Any message without topic has to be considered as an error and forget. Next come the topic with a variable length defined previously followed by the data length encoded on two bytes (the protocol tolerate empty data packages) and the data associated. Finally, a checksum is computed with the whole packet by using the CRC-32 algorithm already used in the Ethernet frame which is a fast and lightweight hash algorithm. It has to be noted that the maximum size of one packet is limited by the theoretical limit of UDP which is 65,535 bytes. When a client receives a packet, the first operation he has to do with it is the checksum validation. If this fails, the packet has to be dropped as the protocol does not have any mechanisms to send the packet again. Otherwise, the packet can be split by using the different sizes and the data and topic can be easily read.

4.1.4. Discovery message

A really interesting feature we have to provide is a mechanism to discover entities in the network. To achieve this, each of them can register custom key-value pairs in addition to any readable (i.e. allowing GET method) configuration values that will be exposed to any discovery request. Figure 4b

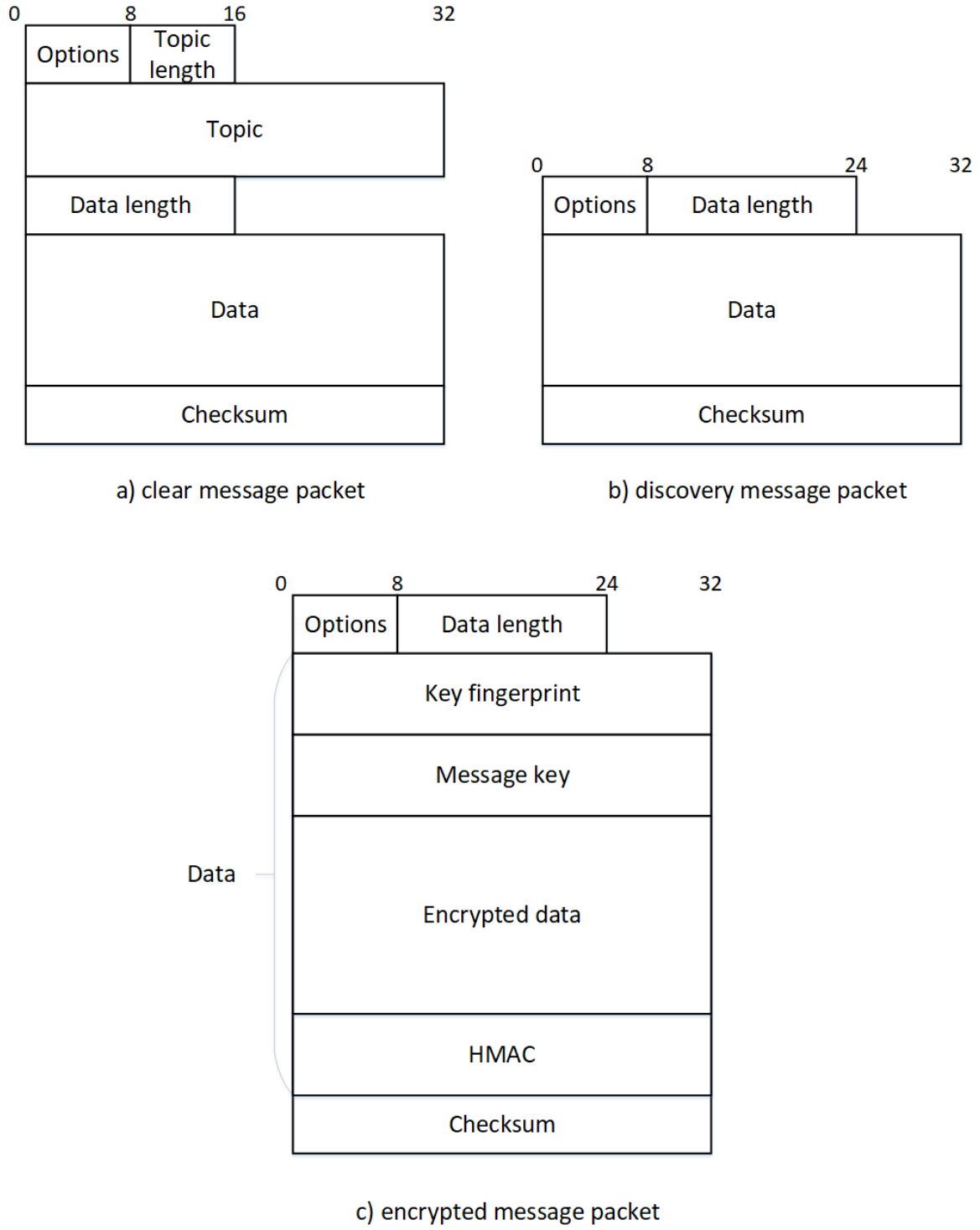

Figure 4. Representation of the three packets present in our protocol.

Table 4. An example of the discovery decision process

Key	Requested	Exposed	Conclusion
version	"1.0.1"	"1.0.1"	OK
name	"temp2"	"temp32"	NOK
units	"C"	["C", "F"]	OK
units	"K"	["C", "F"]	NOK
units	["C", "F"]	"C"	OK
units	["C", "F"]	"K"	NOK
units	["C", "F"]	["C", "K"]	OK
units	["C", "F"]	["K", "R"]	NOK

represents a single discovery packet. As the data packet, the options come first. The only difference is the LSB of this particular byte which is set to 1 representing a discovery packet. Next come the length of the data containing the request, a simple key-value data structure represented in JSON format. For security reasons, data has to be set in order to limit the number of answers. Consequently, any discovery packets with a zero-length data have to be ignored and consider as an error. When the frame is received, receivers will have to decide if they have to answer or not. To achieve this, they check every key in the request if they do not have one of them they can drop the packet they must not answer. Regarding the associated values, two cases are possible : either the key contains a single value or an array of values. Consequently, four situations can occur depending on the combination of two different data type on the receiver and the sender. Examples of each case are presented in Table 4. The first case shown here is when both requested and exposed are single value. In this case, the two values have to be exactly the same as presented in the first two lines of the table. Next, come the case where one has an array and the other a single value here, the single value has to be in the array like in the lines 3 to 6 in the Table 4. Finally, when both have arrays as values, at least one value in the requested array has to be present in the exposed one.

4.1.5. Encrypted data message

As we deal with sensitive information inside a smart environment, our protocol has to provide an easy way to secure the communications. We adapt a well-known secure chat found in the Telegram [36] application to achieve this goal.

The encryption process starts by adding random padding to the message. Indeed, Advanced Encryption Standard (AES), the encryption algorithm we use, is a block cipher so our data have to be a multiple of the block size B_s. The number of random bytes P_s we have to add is defined by the formula 1. We always add B_s bytes in order to always put some randomness in the original message then, we add enough bytes to be a multiple of B_s (16 in the case of AES). In formula 1 we use $DataLength + 2$ because the last step in the packet preparation is the prepending of the data size a two-byte long number.

$$P_s = B_s + (B_s - (DataLength + 2)\%B_s) \tag{1}$$

When the packet is ready to be encrypted, we create the message key used to generate the AES key and IV by computing the SHA1 of the packet to encrypt. This hash is given to a Key Derivation Function (KDF) with the Secret Key preconfigured by using the configuration channel. The KDF presented in algorithm 1 is a sequence of different SHA1 hash and will result in two values, the 128-bit AES Key and AES IV used to encrypt the packet with AES. Next, in order to identify the secret key used to encrypt, we compute a unique fingerprint of it by using the Base64 representation of the key SHA256. Finally, the hash message authentication code (HMAC) is computed by using the secret key with the SHA1 hashing algorithm.

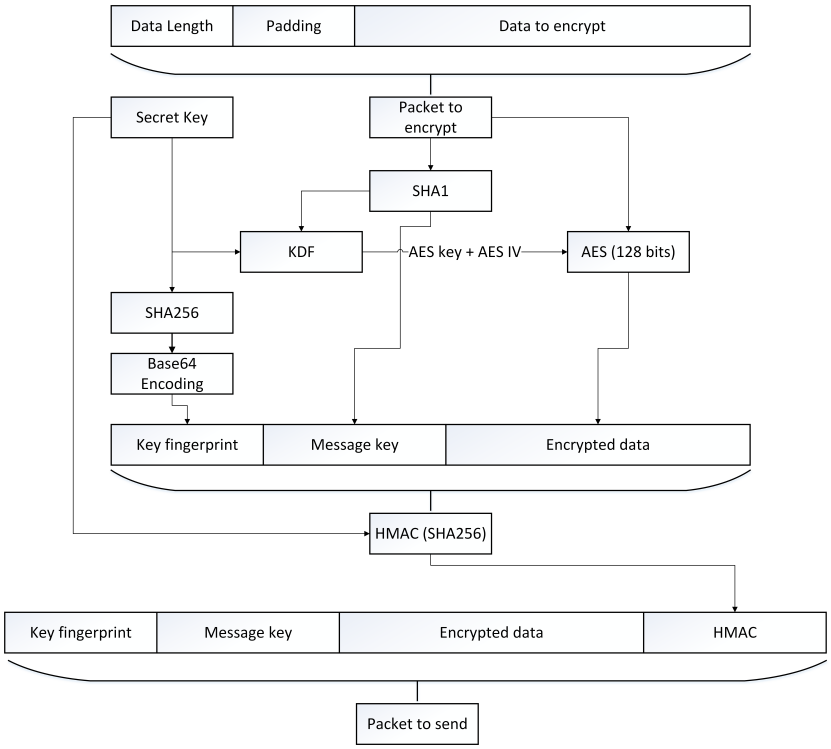

Figure 5. The encryption process.

Data: secret_key[16], msg_key[20]
Result: aes_key[16], aes_iv[16]
sha1_a = sha1(msg_key + secret_key[0...3]);
sha1_b = sha1(secret_key[4...5] + msg_key + secret_key[6...7]);
sha1_c = sha1(secret_key[8...11] + msg_key);
sha1_d = sha1(msg_key + secret_key[12...15]);
aes_key = sha1_a[0...3] + sha1_b[0...7] + sha1_c[4...7];
aes_iv = sha1_a[12...15] + sha1_b[12...19] + sha1_d[0...3];

Algorithm 1: The Key Derivation Function

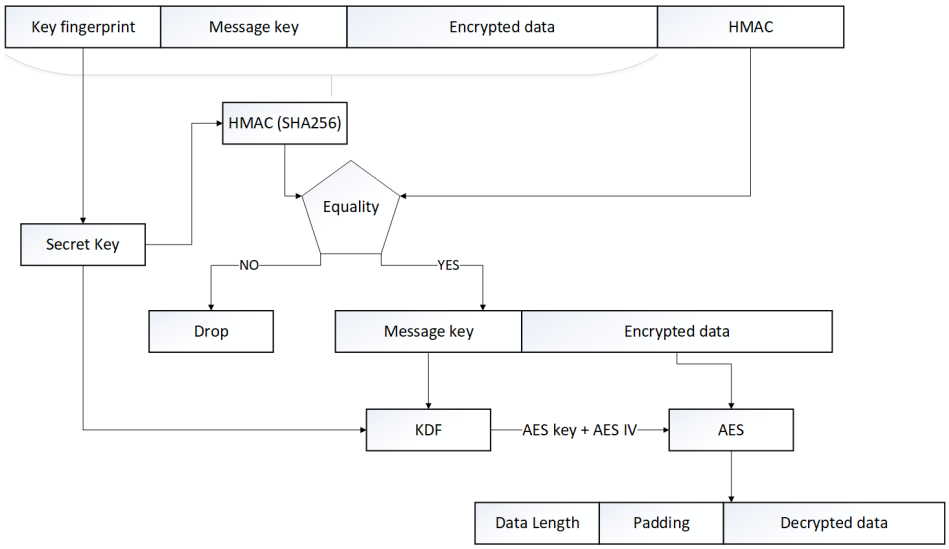

Figure 6. The decryption process.

In order to decrypt an encrypted packet (identified by a 1 at position 1 in the option byte of the header) the receiver has to do some operations sum up in Figure 6. The first task to accomplish is to verify the HMAC to ensure the authentication and the integrity of the packet. To accomplish this, we first have to retrieve the preconfigured secret key used to encrypt identified by its fingerprint. Next, we can compute the HMAC of the packet (without the sender HMAC) and check for equality. If the two hash message authentication codes are different, the packet must be dropped otherwise, the decryption process can begin. First, we have to reconstruct the AES key and IV used to encrypt the message by passing the preconfigured secret key and the message key received in the KDF algorithm presented earlier. Next, we can use AES 128 algorithm in decrypt mode with the generated key and IV to finally decrypt the whole message. The final step is to remove the padding bytes which can be easily done with the message size store in the first two bytes.

4.2. Tests and discussion

In order to validate the proposed protocol, we have made three tests on it. The first one is about bandwidth and try to maximize the number of messages per second exchanged to demonstrate the speed of our protocol. The second one answers a question about the UDP protocol. Indeed, UDP does not provide safety mechanisms about lost, corrupted or disordered network packet. Consequently we decide to show the number of packets we did not receive because of this lack. Finally, we want to demonstrate the fact that even with the same message and secret key, our encrypted packet is always totally different in order to prevent semantic attack since our Initialization Vector is not randomly generated. To accomplish that, we will compute a similarity measure between encrypted packets containing the same message with the same encryption key. The hardware used in our tests was a laptop (MSI GT62VR), a Raspberry Pi 3 and a Raspberry Pi Zero W. The first one was connected to a Gigabit wireless router (LinkSys WRT1900AC) through its Gigabit wired and wireless (AC Wi-Fi) card and the other ones over a simple Wi-Fi connection. Concerning the implementation of our solution, we used C++ with the libraries Boost ASIO (for the network) and Crypto++ (for the encryption algorithms).

In order to test the bandwidth capabilities of our protocol, we first generate 9 random messages with different sizes from 16 bytes to 60 kibibytes (kiB). Then, we bound a listening process on the wired network card on the laptop in order to monitor the packets transiting through the network while we use wireless connections to send 20 000 packets for each different sizes with 10 000 encrypted and another 10 000 not. Results from this test are summed up in Figures 7 to 9 where the upper plot

represents the messages per second (Msg/s) and the lower one the data rate in mebibytes per second (MiB/s) transmitted by our protocol both depending on the message size. Figure 7, 8, 9 respectively show the laptop, Raspberry Pi 3 and Raspberry Pi Zero W results. The first thing we can note is the important drop in both data rates and message per second occurring for packet larger than 2kiB on every platform and for both encrypted and non-encrypted messages. With respectively 10.9, 13.1, 3.8, 4.6, 2.2 and 2.78 times fewer messages sent depending on the platform and encryption, our solution seems to present a limit concerning high-frequency large messages (greater than 1000 per second). We investigate the reason for such a decreasing in performances and it seems to be fragmented Ethernet packets that appear when the length of the transported data is greater than the maximum data size of an Ethernet packet which is 1522 bytes. The second observation is pretty obvious and is the fact that message rate and data rate decreased when using encryption. This is due to the computation of different cryptographic algorithms like AES or SHA but we can say that for a packet under the 2kiB limit the encryption process does not impact too much our protocol. Finally, we can say that our protocol does not overload the Raspberry Pi Zero W, the smallest platform, since both encrypted and clear tests give nearly the same results in terms of data rates which can be explained by a fully used network adapter.

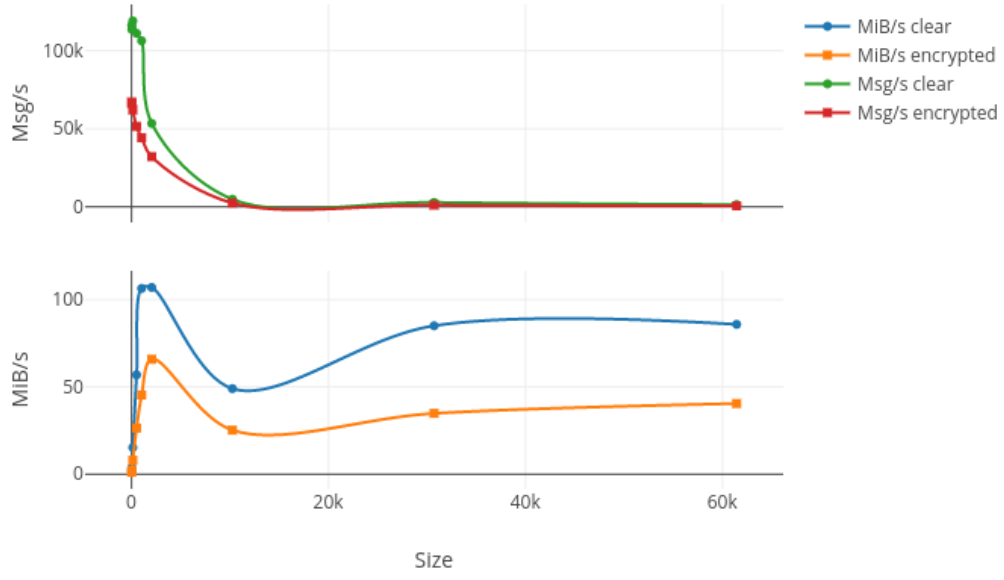

Figure 7. Bandwidth results in terms of Msg/s and MiB/s for 10,000 send on the laptop of an increasing message size in both encrypted and clear mode

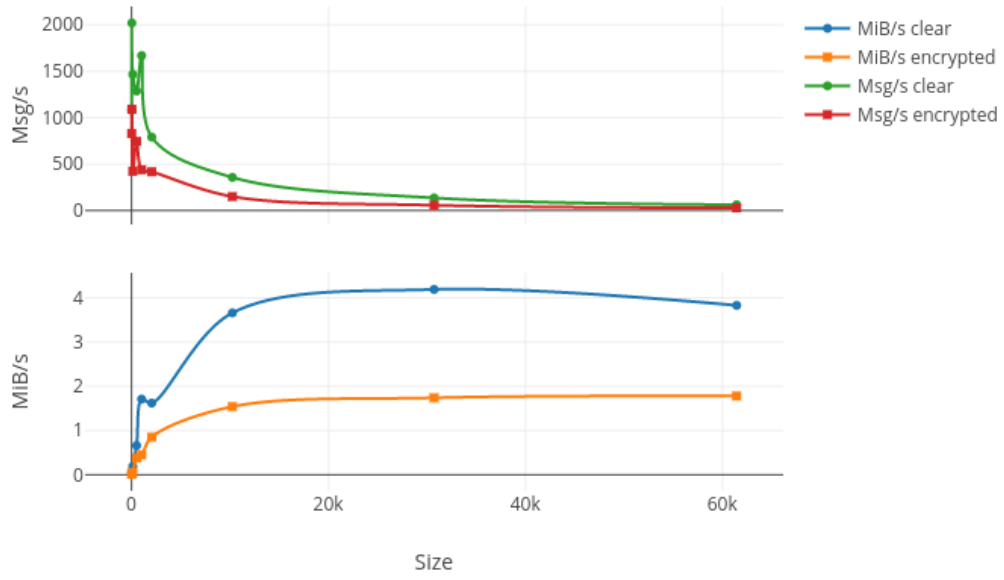

Figure 8. Bandwidth results in terms of Msg/s and MiB/s for 10,000 send on the Raspberry Pi 3 of an increasing message size in both encrypted and clear mode

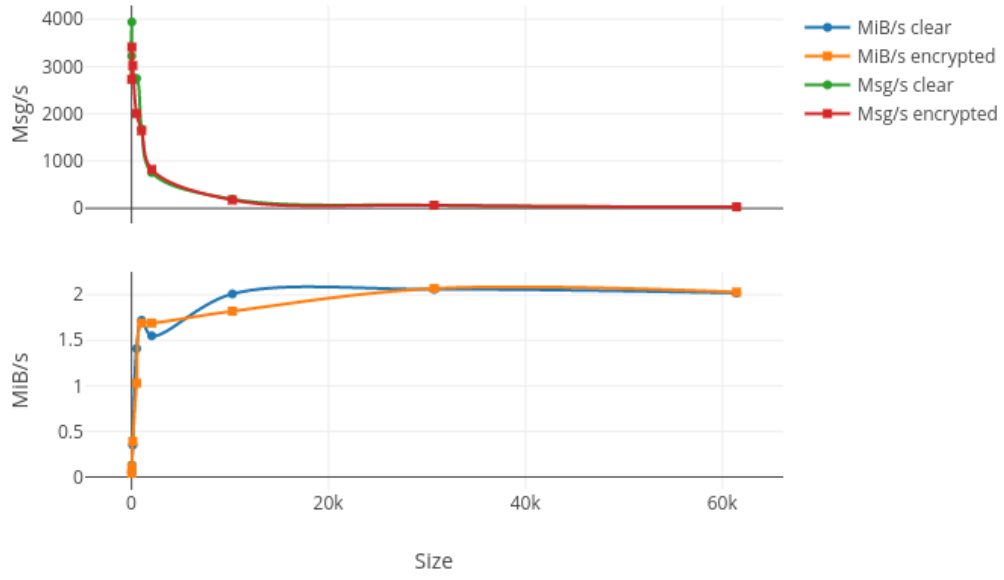

Figure 9. Bandwidth results in terms of Msg/s and MiB/s for 10,000 send on the Raspberry Pi Zero W of an increasing message size in both encrypted and clear mode

UDP is a protocol without any reliability mechanisms. It means that packets can be lost or corrupt and the protocol does not support any means to retrieve this packet unlike TCP. As our work is based on UDP and does not provide such mechanisms either, we wanted to show and quantify the risk of data loss. To do so, we create 10 000 messages of 1kiB composed by a two-bytes long sequence number and some random padding. While the wired interface on the laptop monitor the packet arrival (order, corruption using the embedded checksum or loss) the wireless connections send 10 000 packets in order. The results of this test are summed up in Table 5 where each line represents a different platform (laptop, raspberry Pi 3 and raspberry Pi zero W) and the three columns are respectively the number of lost packet, corrupted ones and finally the number of sequencing problems. As we can see, on the same network, we did not lose or receive corrupted packets but we had some problem in the sequence (2 for the laptop and Raspberry Pi 3 and 3 for the Raspberry Pi Zero W). That means that over 10 000 messages only 2 or 3 arrive before the previous one a result that tends to prove the relative reliability of our protocol even if we use UDP as our transport protocol.

Table 5. Number of lost, corrupted or misplaced packets over 10,000 send of a 1kiB message

Sender	Packets lost	Packets corrupted	Sequence problem
Laptop	0	0	2
Pi 3	0	0	2
Pi Zero W	0	0	3

The last test we did is a similarity test. Indeed, while we ensure the confidentiality of the data in encrypted mode, we did not randomly generated our Initialization Vector (IV) for the AES encryption. Instead, we use an algorithm (KDF described in algorithm 1) to be able to compute the IV based on information found in the encrypted packet and the secret key. And yet, the randomization of this IV guarantees the semantic security of AES. Consequently we wanted to know if our algorithm using random padding bytes is random enough to ensure a non-similarity between encrypted packets with the same secret key and containing the same data. To validate this point, we compute the Euclidian distance between 1000 secured packets containing the same 1kiB message of random data. The results of this test are summed up in table 6. In this last one, the first line represents the maximum distance we can obtain by having one packet with all bytes set to 0 and the other 255 this maximum distance will be used as a reference to compute the percentage of difference between packets. Next comes the mean distance computed on all the packets, with a value of 3458.79, together with a maximum of 3775.54 and a minimum of 3175.23, we can say that our algorithm guarantee a relative non-similarity between those packets. Finally, we report the mean difference percentage of 42.39% in the last line of the results table. This means that for the exact same packet encrypted with our algorithm we generate encrypted packets that are on average 42.39% different. Consequently, we can say that our encryption process, even if it does not use a full random one, compute sufficiently different IV for the same packet in order to protect it from semantic and similarity attacks.

Table 6. Distance between 1000 packets containing the same message of 1kiB on the same topic and encrypted with the same encryption key

Variable	Value
Maximum theoretical distance	8160
Mean distance	3458.79
Maximum distance	3775.54
Minimum distance	3175.23
Mean difference percentage	42.39%

In conclusion, we can say that our tests were divided in three. The first one was to compute the speed capabilities of our protocol. In this case the results demonstrate that for data under 2kiB we

assure a very high speed with nearly a thousand messages per second on the lightest platform we executed the test on. Another conclusion that we made thanks to this test is the relative low-impact of the encryption even on light platform. The second test was realized to ensure the reliability of UDP multicast on a single network. Indeed, this communication protocol does not provide reliability insurance mechanisms and we wanted to put a number on the risks inherent in its usage. With 0 packet lost or corrupted and a maximum of 3 order problems over 10 000 messages sent we can safely say that even if we use UDP as our base communication protocol, our method seems to be reliable enough for data streaming inside a Smart Home. Finally, the last test we executed was to compute the similarity between secured packets containing the same message and encrypted with the exact same secret key. Indeed as our Initialization Vector is computed instead of randomly generated we had to prove that the semantic security of AES is guaranteed. Our results show that our algorithm to derive the AES key and IV from the message itself and the secret pre-shard key is random enough to an average of 42.39% difference between 1000 encrypted packets containing the same data.

5. Conclusions and future works

In this paper, we introduced a new distributed way to communicate between smart entities distributed in an environment. Unlike MQTT or RabitMQ, well-known protocols, we don't need to have a centralized broker instance and unlike ZMQ, a well-known framework to implement messaging protocols, we don't rely on POSIX sockets which are hard to embed on tiny devices without a Linux or Windows operating system. Our protocol, like FTP, relies on two channels. The first one is for the configuration of every entity in the network and is based on COAP, a well-known protocol in the field of the Internet of Thing, for its reliability. The other one is a data channel, based on multicast messages with a protocol entirely define in this paper. This last one permits to send discovery requests to the network as well as messages encrypted or not. For the encrypted way, we adapt the Telegram protocol a well-known secure instant messaging protocol in order to work on tiny devices and provide an easy authentication with a HMAC.

In this paper, we realized three different tests to ensure our capabilities in terms of speed, reliability and security. In the light of the results, we can say that for data under 2kiB we ensure a very high speed with nearly a thousand messages per second on the lightest platform we executed the test on. Moreover, with no packet lost or corrupted and a maximum of 3 order problems over 10 000 messages sent we can safely say that our method seems reliable enough for data streaming. Finally, our results show that our algorithm to derive the AES key and IV from the message itself and the secret pre-shard key is random enough to generate an average of 42.39% difference between 1000 encrypted packets containing the same data ensuring the security against semantic attacks.

Ultimately, we can say that our protocol with its two channels allows to make a difference between configuration values which need a high reliability but won't be changing every millisecond and data values or streaming which need higher data rates but less reliability. Moreover, our protocol support a native encryption mode which provides security for sensitive information we can easily find in a Smart Home. In addition, we designed our innovation to be both brokerless, which is the main difference between many existing messaging protocol, and easily portable as it only relies on UDP, a communication protocol found in every network applications. Lastly, as a future work, we want to realize more tests including tests on embedded environment, like a Real Time OS on a microcontroller and tests with a lot more entities in the network.

Abbreviations

The following abbreviations are used in this manuscript:
AES: Advanced Encryption Standard
AI : Artificial Intelligence
Amb. I: Ambiant Intelligence

AMQP: Advanced Message Queuing Protocol
BLE: Bluetooth Low-Energy
CoAP : Constrained Application Protocol
CRC: Cyclic Redundancy Check
EEPROM : Electrically Erasable Programmable Read-Only Memory
GHz: gigahertz
HMAC : keyed-Hash Message Authentication Code
HTTP: Hypertext Transfer Protocol
IEEE: Institute of Electrical and Electronics Engineers
IoT: Internet of Things
IV: Initialization Vector
KDF : Key Derivation Function
LNCF: Lght Node Communication Framework
LIARA: Laboratoire d'Intelligence Ambiante pour la Reconnaissance d'Activités
LSB : Least Significant Bit
LWT: Last Will Testament
kB: kilobyte
kiB: kibibyte
kiB/s: kibibyte/second
MHz: megahertz
MiB: mebibyte
MiB/s: mebibyte/second
MSB : Most Significant Bit
Msg/s: messages/seconds
OSGI: Open Service Gateway Initiative
RAM: Random Access Memory
RTOS : Real Time Operating System
SDSH: Software Defined Smart Home
SHA: Secure Hash Algorithms
SoC: System on Chip
SPoF: Single Point of Failure
UDP: User Datagram Protocol
XMPP : eXtensible Messaging and Presence Protocol
ZMQ : ZeroMQ

Author Contributions: All the authors contributed equally to this work.

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

Bibliography

1. Cook, D.J.; Crandall, A.S.; Thomas, B.L.; C., K.N. CASAS: A Smart Home in a Box **2012**. *100*, 130–134.
2. Ghayvat, H.; Mukhopadhyay, S.; Gui, X.; Suryadevara, N. WSN- and IOT-based smart homes and their extension to smart buildings. *Sensors (Switzerland)* **2015**, *15*, 10350–10379.
3. King, J.; Jansen, E. The Gator Tech Smart House. *Computer* **2005**, *38*, 50–60.
4. Patterson, D.J.; Liao, L.; Fox, D.; Kautz, H. Inferring High-Level Behavior from Low-Level Sensors. International Conference on Ubiquitous Computing, 2003.
5. Augusto, J.C.; Nugent, C.D. *Designing smart homes: the role of artificial intelligence*; Vol. 4008, Springer, 2006.
6. Roy, P.C.; Bouchard, B.; Bouzouane, A.; Giroux, S. Ambient Activity Recognition in Smart Environments for Cognitive Assistance. *International Journal of Robotics Applications and Technologies* **2013**, *1*, 29–56.
7. United Nations.; Department of Economic and Social Affairs.; Population Division. World Population Ageing 2015. Technical report, 2015.

8. Hunkeler, U.; Truong, H. MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. *systems software and ...* **2008**.
9. Videla, A.; Williams, J. RabbitMQ in action: distributed messaging for everyone **2012**.
10. Zeromq, 2016.
11. Cook, D.J.; Youngblood, M.; Heierman, E.; Gopalratnam, K.; Rao, S.; Litvin, A.; Khawaja, F. MavHome: an agent-based smart home. *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003. (PerCom 2003).* **2003**, pp. 521–524.
12. Giroux, S.; Leblanc, T.; Bouzouane, A.; Bouchard, B.; Pigot, H.; Bauchet, J. The Praxis of Cognitive Assistance in Smart Homes. *BMI Book* **2009**, pp. 183–211.
13. Xu, K.; Wang, X.; Wei, W.; Song, H.; Mao, B. Toward software defined smart home. *IEEE Communications Magazine* **2016**, *54*, 116–122.
14. Patel, S.M.; Kanawade, S.Y. Internet of Things Based Smart Home with Intel Edison. Proceedings of International Conference on Communication and Networks. Springer, 2017, pp. 385–392.
15. Bouchard, K.; Bouchard, B.; Bouzouane, A. Guidelines to Efficient Smart Home Design for Rapid AI Prototyping: A Case Study. *PETRA* **2012**.
16. Advantech. Automation Controllers & I/Os, 2016.
17. Dell. Dell PowerEdge Rack Servers, 2016.
18. OSGi^{TM} Alliance – The Dynamic Module System for Java, 2016.
19. Drumea, A.; Popescu, C.; Svasta, P. GSM solutions for low cost embedded systems for industrial control. 28th International Spring Seminar on Electronics Technology: Meeting the Challenges of Electronics Technology Progress, 2005. IEEE, 2005, pp. 240–244.
20. Li, C.S.; Liao, W. Software defined networks. *IEEE Communications Magazine* **2013**, *51*, 113–113.
21. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* **2008**, *38*, 69–74.
22. Chu-Sing, Y.; Mon-Yen, L. Realizing Fault Resilience in Web-Server Cluster. ACM/IEEE SC 2000 Conference (SC'00). IEEE, 2000, pp. 21–21.
23. Lu, F.; Parkin, S.; Morgan, G. Load balancing for massively multiplayer online games. Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games - NetGames '06; ACM Press: New York, New York, USA, 2006; p. 1.
24. Mon-Yen, L.; Chu-Sing, Y. Constructing zero-loss Web services. Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213). IEEE, 2001, Vol. 3, pp. 1781–1790.
25. Schroeder, T.; Goddard, S.; Ramamurthy, B. Scalable Web server clustering technologies. *IEEE Network* **2000**, *14*, 38–45.
26. Atzori, L.; Iera, A.; Morabito, G. The internet of things: A survey. *Computer networks* **2010**.
27. Liu, B.; Cao, S.G.; He, W. Distributed data mining for e-business. *Information Technology and Management* **2011**, *12*, 67–79.
28. Chen, M.; Mao, S.; Liu, Y. Big data: A survey. *Mobile Networks and Applications* **2014**, *19*, 171–209.
29. Hey, A.; Tansley, S.; Tolle, K. The fourth paradigm: data-intensive scientific discovery **2009**.
30. IEEE Standard for a Smart Transducer Interface for Sensors and Actuators, 1998.
31. Lewis, F. Wireless sensor networks. In *Smart Environments: Technology , Protocols, and Applications*; 2005; chapter 2.
32. Martin, G.; Zurawski, R.; Philips, C. Trends in embedded systems Opportunities and challenges for System-on-Chip and Networked Embedded Systems technologies in industrial automation. *ABB Review* **2006**, *2*.
33. Hintjens, P. *ZeroMQ: messaging for many applications*; 2013.
34. Postel, J. User Datagram Protocol. *RFC* **1980**.
35. Shelby, Z.; Hartke, K.; Bormann, C. The constrained application protocol (CoAP) **2014**.
36. Telegram. Telegram Protocol, 2013.