This is the peer reviewed author accepted manuscript (post print) version of a published work that appeared in final form in:

Esposito, Christian, Castiglione, Aniello, Palmieri, Francesco, Ficco, Massimo, Choo, Kim-Kwang Raymond 2016 'A publish/subscribe protocol for event-driven communications in the Internet of Things', Proceedings - 2016 IEEE 14th International Conference on Dependable, Autonomic and Secure Computing, DASC 2016, 2016 IEEE 14th International Conference on Pervasive Intelligence and Computing, PICom 2016, 2016 IEEE 2nd International Conference on Big Data Intelligence and Computing, DataCom 2016 and 2016 IEEE Cyber Science and Technology Congress, CyberSciTech 2016, DASC-PICom-DataCom-CyberSciTech 2016, pp. 376-383

This un-copyedited output may not exactly replicate the final published authoritative version for which the publisher owns copyright. It is not the copy of record. This output may be used for non-commercial purposes.

The final definitive published version (version of record) is available at:
https://doi.org/10.1109/DASC-PICom-DataCom-CyberSciTec.2016.79

Persistent link to the UniSA Research Outputs Repository:
https://researchoutputs.unisa.edu.au/11541.2/123732

General Rights

Copyright and moral rights for the publications made accessible in the UniSA Research Outputs Repository are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognize and abide by the legal requirements associated with these rights.

- Users may download and print one copy for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the persistent link identifying the publication in the UniSA Research Outputs Repository

If you believe that this document breaches copyright, please contact us and provide details. We will remove access to the work immediately and investigate your claim.

A Publish/Subscribe Protocol for Event-Driven Communications in the Internet of Things

Christian Esposito*, Aniello Castiglione*, Francesco Palmieri*,
Massimo Ficco†, and Kim-Kwang Raymond Choo‡

*Department of Computer Science at the University of Salerno,
Via Giovanni Paolo II 132, I-84084 Fisciano, SA - Italy
christian.esposito@dia.unisa.it, castiglione@ieee.org and fpalmieri@unisa.it

†Department of Industrial and Information Engineering, Second University of Naples,
Via Roma 29, I-81031 Aversa, CE - Italy
massimo.ficco@unina2.it

‡ School of Information Technology and Mathematical Sciences, University of South Australia,
Mawson Lakes Campus, Adelaide SA 5001 - Australia
raymond.choo@fulbrightmail.org

Abstract

Internet of Things is a futuristic vision of networked objects able to take smart decisions and to assist humans in their daily activities. The founding factor of such a vision is the capacity of objects to exchange data by means of wireless networks, and the applied communication pattern is event-driven and multicast. Publish/subscribe services are a promising communication technology for building solutions according to the Internet of Things' vision. The available solutions consist in the adaptation of widely-known products for publish/subscribe services taken from other contexts and application domains, without considering the key peculiarities of the Internet of Things. Our driving idea is to consider beaconing as the basic communication means, and to build a publish/subscribe service based on it. Our solution can be used stand-alone for event-driven communications and/or integrated within standardized protocols to provide basic communication capabilities. We present a preliminary set of experiments showing the efficiency of our solution.

***Index Terms*—Internet of Things; Publish/Subscribe Services; Multicasting; Ad Hoc Networks**

I. Introduction

The vision of the Internet of Things (IoT) [1] is to have each object surrounding us capable of connecting among each other and of performing intelligent monitoring, collecting and processing of data. Radio Frequency IDentification (RFID) and sensor network technologies are the key enabling capabilities that are making more real such a vision by invisibly embedding information and communication capabilities within our daily objects. The IoT was envisioned at the beginning for the supply chain management [2], but in the past decade it has evolved in more inclusive aspects by covering other application areas, spanning from healthcare and assisted living to transport and critical infrastructures. Such a pervasive characteristic of IoT in every day activities is among the key factors of its success and widespread diffusion [3].

The elements composing the overall architectural scheme of IoT can be classified in three main classes, as illustrated in Figure 1: (i) the sensors, made of hardware able to sense the environment, to communicate by means of wireless radio technologies, and to actuate proper commands; (ii) an analytic and visualization service, developed in a centralized or distributed manner, for collecting the information coming from the sensors and for properly processing them in order to give back commands and/or

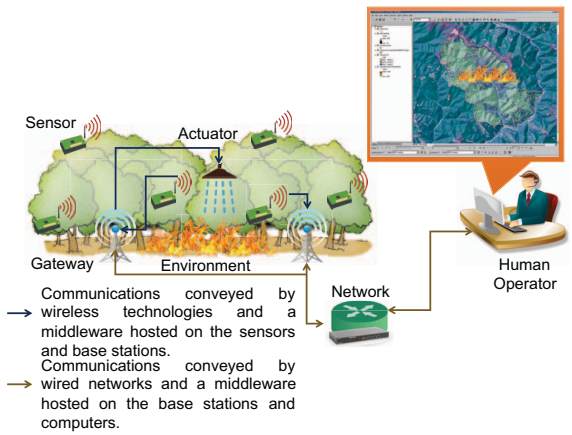


Fig. 1: Example of IoT Application for forest fire detection.

to visualize them in a proper manner to the users; (iii) a middleware abstraction that is built on top of the communication technologies available at the sensors and used to let the sensors communicating among each others and/or with the analytic and visualization service, and provides the needed data sharing and storage capabilities. The communication patterns implemented by the middleware in IoT are based on an event-driven model [4], where the specification and/or determination of the destination of a given data is not based on an address included in the exchanged messages, but on the data content itself that may, or not, match the interests expressed by the potential destinations. This makes request/reply communication pattern not suitable in IoT and calls for more advanced patterns such as the publish/subscribe one [5]. Such a pattern enforces data-centric communications where the reception of a data is based on the expression of interest in a topic characterizing the data. Such an interest clusters the nodes of IoT, implementing the so called group-based communications. Actually, there are several enabling middleware solutions for implementing data-centric and group-based communications in IoT, as surveyed in [6].

We have noticed that within the context of IoT, the available solutions implementing the publish/subscribe communication pattern require the presence of brokers running the required logic to glue together the entities acting as publishers and/or subscribers, and/or consist in adapting standard-based publish/subscribe services, designed for other contexts and application domains, to the case of IoT, with a scarse consideration of the peculiarities of such novel networks. In our work, we have started from the consideration that, despite the heterogeneity in the supporting hardware and software, each entity within the IoT is always able to perform a very basic operation: beaconing. Such an operation consists of using the intrinsic multicasting feature of a wireless communication by sending a message to all the devices reachable by the signals from another device. Based on such an operation, we propose a publish/subscribe protocol that does not require any broker, and can be used as a building block in any standard publish/subscribe implementation for IoT.

The remaining of the paper is structured as follows. Section II introduces some basic concepts of publish/subscribe services, their usage within the context of IoT and available related work within the current literature. Section III presents our solution, while Section IV contains a preliminary assessment we have made of such a solution by using a set of simulations. We conclude the paper in Section V with some final remarks and future work.

II. Background and Related Work

Publish/subscribe services [5] are middleware solutions characterized by two types of processes: *Publishers*, which produce notifications, and/or *Subscribers*, which consume the notifications they are interested in, where such an interest is indicated by means of *Subscriptions*. The basic architectural schema of this service also comprises the *Notification Service* (NS), as depicted in the upper part of Figure 2, which is a proper abstraction that plays the role of mediator between publishers and subscribers by giving strong decoupling features and offering the following functionalities: (i) storing subscriptions; (ii) receiving and buffering events; and (iii) dispatching received notifications to the interested subscribers. The lower part of Figure 2 shows the main entities and how they interact so as to produce an event and to consume it later on when the notification service has forwarded it from the publisher to the interested subscribers. Specifically, a subscriber has to be registered on the notification service, to define a proper submission predicate and to submit it to the notification service. On the other hand, a publisher has to register an advertisement and to encapsulate an event into a notification, which is passed to the notification service for dissemination. The notification service will route the received notifications based on the submitted subscriptions. If a notification satisfies a subscription submitted by a certain subscriber, it will be delivered to this subscriber, which will consume the notification by extracting the event contained in it. Before terminating a subscriber, all its subscriptions will be removed from the notification service; the same is done by publishers with their active advertisements. Summarizing, the five most important operations that publishers and subscribers have to invoke during the life are the following ones:

1) subscribe - it is used by subscribers to express an interest in a class of events so as to receive notifications related to them;

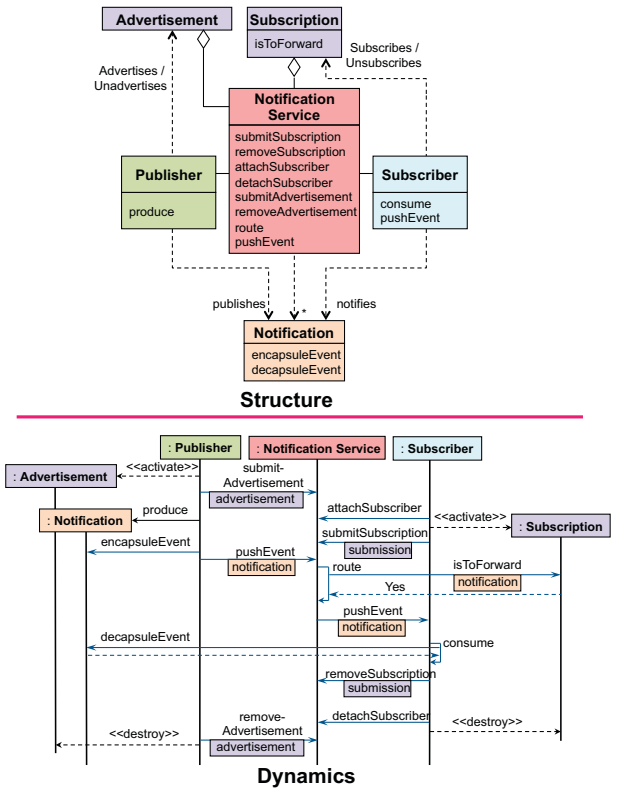


Fig. 2: Structure with the involved entities, and dynamics of a generic Publish/subscribe service.

2) unsubscribe - it is used by subscribers to communicate the conclusion of an interest;
3) advertise - it is used by publishers to express the intent of publishing notifications for a class of events;
4) unadvertise - it is used by publishers to communicate the termination of a publishing intent;
5) publish - it is performed by publishers to forward a notification to all the interested subscribers.

As mentioned, subscribers only receive events in which they are interested. Specifically, there are different ways of specifying such an interest in a subset of the published events, which affect the architecture and algorithm adopted to implement the publish/subscribe platform [5]. The most common one, which will be adopted within the context of this work, is indicated as *topic-based*, with publishers tagging outgoing notifications with a topic string, while subscribers using string matching to detect their events of interest. A different one is the *content-based*, where subscribers express complex predicates on the content of events, and only those events that satisfy such predicates are delivered. Furthermore, the Notification Service can be implemented according to two main approaches [7]: (i) Direct Delivery, *i.e.*, each publisher takes the duty of delivering notifications to interested subscribers or (ii) Indirect Delivery, *i.e.*, the notification duties are shifted from the publishers to one or more networked brokers in order to improve scalability. In the context of this work, we have sensors behaving in a promiscuous manner by being jointly as brokers and subscribers, *i.e.*, implementing an indirect delivery without dedicated brokers.

Publish/subscribe services are key solutions for implementing event-driven multicasting, and are suitable within the context of the IoT for their decoupling properties, which simplify the issue of connecting all the entities composing the network supporting an IoT implementation. Moreover, the selective forwarding of messages that is implicitly realized by such kind of middleware is a valuable method for reducing the communications, and extending the lifespan of the batteries in the resource-constrained nodes of an IoT. The current literature of the application of publish/subscribe services within the context of IoT is quite recent, but already rich of past experiences. Most of them consist in the application of a widely-known publish/subscribe service by adapting it to the specific requirements of IoT in terms of mobility, energy-efficiency, and wireless communications. Here we summarize some of such past experiences:

- Authors in [8] propose the application of products based on the OMG specification for publish/subscribe services named as Data Distribution Service (DDS) [9]. OMG DDS specifies the API for broker-less topic-based services equipped with content-based filtering of the incoming notifications, and such a middleware is combined with Software Defined Networking (SDN) [10] with proper gateways to manage mobility of smart objects.
- In [11], another standard for publish/subscribe services, named as Extensible Messaging and Presence Protocol (XMPP) [12], has been used as the basic communication protocol. XMPP is facing increasing success with networks of resource-constrained devices due to its lightweight implementations and the possibility of omitting intermediaries between the interacting devices, thanks to the XEP-0174 specification [13] and using Bonjour/Zeroconf as a discovery protocol.
- An extension of the publish/subscribe protocol named as Message Queuing Telemetry Transport (MQTT) [14] for sensor networks called MQTT-S [15]. MQTT is a lightweight broker-based protocol, running on embedded devices with limited processing, memory or battery resources.

IoT diverges from other kinds of sensor-based networks interconnected by means of wireless networks for its overwhelming scale, in terms of large network size and a massive number of notifications to be exchanged among interested destinations. Such a large scale motivates the

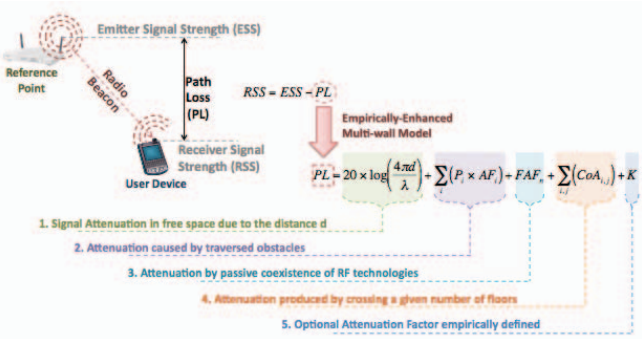


Fig. 3: Path loss determination based on distance and traversed obstacles [17].

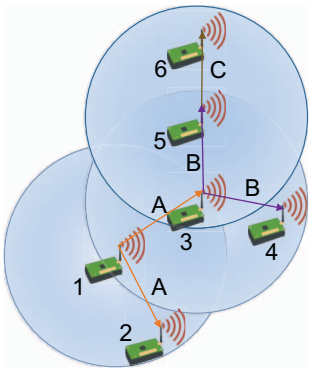


Fig. 4: Schematic description of notification distribution realized by the proposed approach.

use of a publish/subscribe service. In addition, the IoT is characterized by spontaneous interactions [16], which happen suddenly based on events among nodes within the network. To this aim, the IoT requires a zero infrastructure by letting nodes to discover each others without assuming a fixed infrastructure of brokers managing resource publications and node sampling. This is the main flaw that the above-mentioned related works have, and that is felt as an open issue in the current literature. Our work aims at filling such a gap by proposing a proper broker-less protocol with a simple discovery of the nodes of interest, and establishing of joining relations based on the mutual interest on given topics.

III. Approach

Our approach for event-driven communications according to a publish/subscribe communication pattern is based on the beaconing capability that each node using a wireless communication technology is able to perform, *i.e.*, the ability to delivery a message in broadcast to all the other nodes that are listening on a given radio frequency and reachable by the emitted signal. Specifically, each sensor of an IoT is equipped with an omnidirectional antenna, which radiates radio wave power uniformly in all directions in one plane. Such an antenna is characterized by a *Emitter Signal Strength* (ESS), *i.e.*, the output gain of the signals radiated by the antenna of the sensor. Along their path, signals are affected by a *Path Loss* (PL), *i.e.*, the attenuation experienced by the signal on its path from the emitter to the receiver. PL is the function of the distance from the emitter and the number and type of the traversed obstacles and floors. Figure 3 contains the mathematical formulation of the gain of a signal received by an antenna, namely *Receiver Signal Strength* (RSS), defined as the difference of the ESS and PL. The area of reachability for a beaconing message is characterized by a radius equal to the maximum distance from the emitter, placed at the center of this area, for which the obtainable RSS is greater than or equal to a given threshold. In fact, if the RSS is too low, the receiver is not able to decode the received signal.

Taking advantages of the intrinsic multicasting possibilities offered by beaconing, our solution consists in the selecting flooding based on the beaconing capability, as illustrated in Figure 4. The selection of the nodes replicating and beaconing an incoming notification is decided based on the number of interested nodes within their area of interest, and that may have not potentially received the notification. As a concrete example, let us consider the sensor numbered as 4 in the figure. After the reception of the message at the iteration B, it did not commence a novel beaconing round, because no more sensore were in its area of reachability. On the contrary, the sensor numbered as 5 encompassed sensor 6 in its area of reachability, which had not received a message, yet. For this reason, such a sensor decided to beacon the received message. Our solution consists in managing a consistent view of such a joining state per each area at each sensor. Therefore, we have a protocol composed by a first step, where the network is initialized, and three procedures to subscribe to a topic, unsubscribe to a previously-subscribed topic, and to publish a notification.

At the beginning, the nodes must be initialized with some preliminary information related to the other nodes within the IoT. Such an initialization is not made manually by the deployer of the network, but can be performed in an automatic manner. Specifically, the nodes broadcast their unique identifier to all reachable nodes, which will do the same afterwards. This is useful to let the node knowing which are the neighboring nodes in its area of reachability. Such a knowledge is stored in a list of neighboring nodes, which is called $neighbors_list$.

During the life of a node, before publishing a notification and/or consuming some notifications of interest, the node must announce its interest towards certain classes

Algorithm 1 Execution of a joining operation

Require: $routing_topic_table$
1: Function join(topic)
2: **if** ∃ routing_topic_table(topic) **then**
3: addRow(routing_topic_table, topic);
4: **end if**
5: row ← routing_topic_table(topic);
6: row.isInterested ← true;
7: row.nodes_info ← {this, 1};
8: msg ← new join_msg;
9: msg.topic ← topic;
10: msg.info ← row.nodes_info;
11: sendMessage(msg);
12: EndFunction

of events. Such operation is called $subscribe$ when performed by subscribers, and $advertise$ when done by the publishers. In both cases, a node intends to join a group of nodes sharing an interest in a particular class of events identified by a topic. To this aim, it sends in broadcast a special message, called $join_msg$, expressing such an interest, and stores in its memory a data structure, called $routing_topic_table$ containing the following fields:

- the unique identifier of the topic;
- a list of neighboring nodes with the same interest in the given topic
- the number of nodes reachable by a neighboring node with the interest in the same topic.
- a list of an integer values, called $finger_id$ that indicates the unique identifier of the received notification for the given topic;
- a list of integer values, each called $sequence_id$ that indicates the sequential identifier of the last-received datagram, relative to the a notification identified by $finger_id$ and resulting from the fragmentation of a given notification in datagrams.
- a boolean value indicating the interest in such a topic.

At the beginning, we have valued only the first element, but when a node receives a $join_msg$, it updates its internal $routing_topic_table$ relative to the given topic, and replies with a message, called $update_msg$, with a snapshot of the updated $routing_topic_table$. When a node receives a $update_msg$, it updates its own $routing_topic_table$ with the information contained in the received message. This is described in Algorithm 1 and Algorithm 2, and continues until all the nodes have updated their internal $routing_topic_table$ and no more messages are exchanged, so as to let the network reaching a stable state. The $join_msg$ is not only processed by nodes interested to the same topic expressed in the message, but also by uninterested message. This is because the set of interested node may not be uniformly distributed within the IoT network, but it is possible to have two sub-groups, or even more, that to do not have direct connections among their

Algorithm 2 Reaction of a node at the reception of a $join_msg$ and a $update_msg$ from a neighboring node

Require: $neighbors_list$, $routing_topic_table$
1: Function processJoinMsg(msg)
2: **if** msg.type == join_msg **then**
3: **if** sender(msg) ∈ neighbors_list && sender(msg) ≠ this **then**
4: topic ← msg.topic;
5: **if** ∃ routing_topic_table(topic) **then**
6: addRow(routing_topic_table, topic);
7: **end if**
8: row ← routing_topic_table(topic);
9: update(row, msg);
10: **end if**
11: **end if**
12: discard(msg);
13: EndFunction

Require: -
1: Function update(row, msg)
2: isUpdated ← update (row, msg);
3: **if** isUpdated **then**
4: newMsg ← new update_msg;
5: newMsg.topic ← msg.topic
6: newMsg.nodes_info ← row.nodes_info;
7: sendMessage(newMsg);
8: **end if**
9: EndFunction

Require: $neighbors_list$, $routing_topic_table$
1: Function processUpdateMsg(msg)
2: **if** msg.type == update_msg **then**
3: **if** sender(msg) ∈ neighbors_list && sender(msg) ≠ this **then**
4: topic ← msg.topic;
5: **if** ∃ routing_topic_table(topic) **then**
6: row ← routing_topic_table(topic);
7: update(row, msg);
8: **end if**
9: **end if**
10: **end if**
11: discard(msg);
12: EndFunction

members. To this aim, it is necessary that uninterested noded may forward the messages among the different distant sub-groups.

The dual operation of leaving the group, which consists in the unsubscribe or unadvertice operations, described in Algorithm 3, consists in the negate of the boolean value indicating the interest and the distribution of a $leave_msg$ message that forces an update of the $routing_topic_table$ at the receiving nodes, and the consequent exchange of $update_msg$ messages, but not the removal of the $routing_topic_table$, so as to let the node to be a forwarder for such a topic. Such a data structure is removed only when a node do not have any neighboring node with an interest or acting as forwarder for the given topic.

The remaining aspects to describe are the publication, and the consequent consumption, of notifications for a

Algorithm 3 Execution of a leaving operation for a given topic

```
Require: routing_topic_table
Function leave(topic)
if ∃ routing_topic_table(topic) then
    row ← routing_topic_table(topic);
    row.isInterested ← false;
    row.nodes_info(this)–;
    msg ← new leave_msg;
    msg.topic ← topic;
    msg.info ← row.nodes_info;
    sendMessage(msg);
end if
EndFunction
```

given topic of interest. When an event occurs at a node acting as publisher, a notification is built with all the details of the occurred event, the unique identifier of the topic and the dissemination procedure is performed. Specifically, the node with a message for a given notification (published or received) checks the existence $routing_topic_table$ of the relative topic and if there is at least one neighboring nodes with a non-null number of interested nodes. If such a check is passed, then another constraint must be verified in order to avoid loops in the dissemination paths: if the couple of $finger_id$ and $sequence_id$ are not present in the $routing_topic_table$. In the affirmative case, the message of interest can be assumed new, and not redundant, and can be forwarded by means of a beaconing. On the contrary, the node has already seen the message and it is discarded. The last operation is the eventual consumption of the message if the boolean value in the $routing_topic_table$ is true. Such a behavior is detailed in Algorithm 4.

Algorithm 4 Publishing and consuming messages

```
Require: routing_topic_table
Function processNotification(msg)
if msg.type == notification then
    topic ← msg.topic;
    if ∃ routing_topic_table(topic) then
        row ← routing_topic_table(topic);
        hasInterest ← false;
        for all pair ∈ row.nodes_info do
            if pair.int_val > 0 then
                hasInterest ← false;
                break;
            end if
        end for
        if hasInterest then
            notReceived ← false;
            for all pair ∈ row.not_info do
                if pair.finger ≠ msg.finger then
                    notReceived ← true;
                else
                    if pair.sequence < msg.finger then
                        notReceived ← true;
                    end if
                end if
            end for
            if notReceived then
                newMsg ← new(msg);
                sendMessage(newMsg);
            end if
        end if
    end if
end if
EndFunction
```

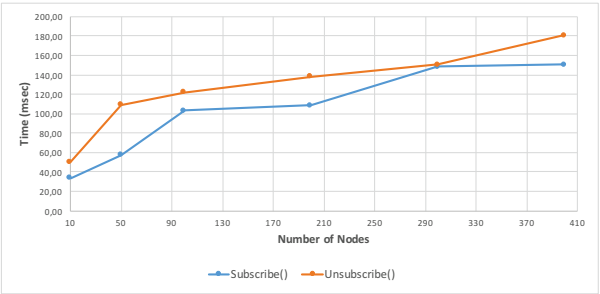
IV. Experimental Assessment

This section has the aim of describing a preliminary experimental assessment of our solution for a publish/subscribe protocol tailored to IoT characteristics and needs. To achieve this aim, we implemented our solution by using the OMNET++[1] simulator, within which we have used the library for modeling for mobile and fixed wireless networks called MiXiM[2]. Such a library provides detailed models for each aspects to be modeled when simulating a wireless network, such as radio wave propagation, interference estimation, radio transceiver power consumption and wireless MAC protocols. Specifically, we have (i) used the communication protocol compliant to the 802.15.4 standard, upon which ZigBee has been specified, (ii) adopted the BaseNetworkLayer model, which implements beaconing as communication means, (iii) disabled any networking fault for which the network may be vulnerable, and (iv) deployed the sensors according to a grid with a cell length of 50 cm. We have set the time to exchange a message ugual to 200 msec. We have also considered the block size equal to 1472bytes, that is the size of MTU in Ethernet, so that a single event is fragmented in only 1 block. Finally, without loss of generality, we considered a system with 1 publisher and a varying number of subscribers, all subscribed to the same topic (in our experiments we have the coexistence of three distinct groups with each node joining only two groups out of the available ones). We have published 1000 events for each experiment, executed each experiment 3 times and reported the average (we did not observe any standard deviations above 5% of the reported values, thus they are not plotted on the curves).

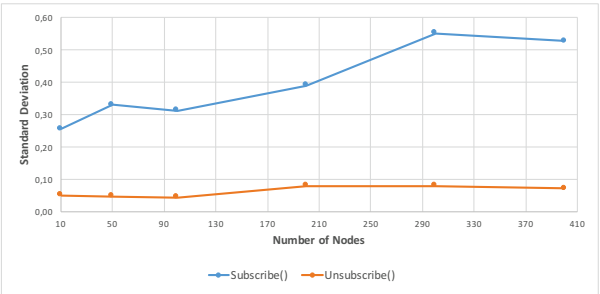
Figure 5 presents the results we have obtained when monitoring the time required to being joined to the set of other nodes interested to a given topic and to leave a given topic. We have measured the average time, shown in Figure 5(a), where we can notice that such a time increases as the number of nodes augments, and a similar trend is observable for the time to leave a topic. We can also notice

[1] www.omnetpp.org

[2] http://mixim.sourceforge.net/

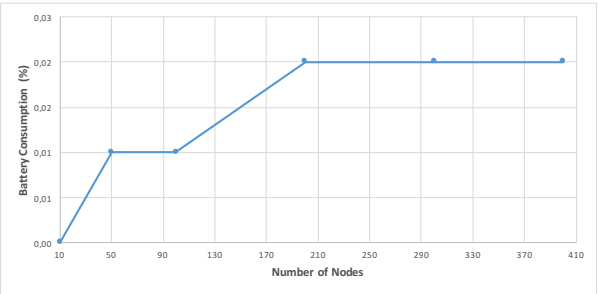


(a)

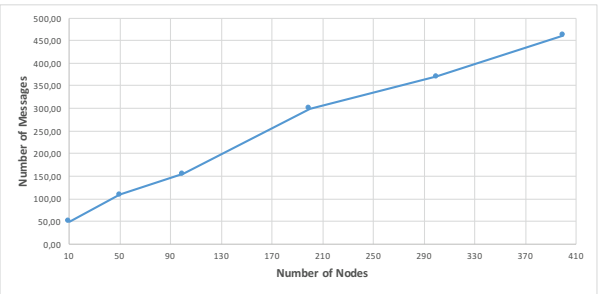


(b)

Fig. 5: Time to join and to leave a topic-based group.



(a)



(b)

Fig. 6: Battery consumption and number of messages to join a topic-based group.

that it is quicker to join a topic rather than to leave it, since when leaving there are some additional messages to be exchanged so as to keep the nodes interested to the same topic connected among each others. Figure 5(b) contains the percentage of the standard deviation for these two measures, and presents some interesting results. On the one hand, we have that the time to join a topic has a string variability as the number of nodes increases, since such a value is higher has the nodes are farer then the publishing node. On the other hand, such a variability is not observable for the time to leave a topic, since the messages related to a leaving node, are exchanged only among the neighboring nodes.

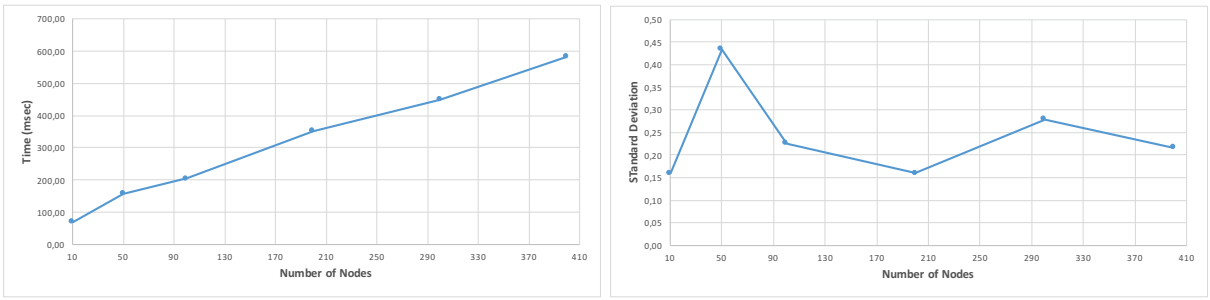
Figure 6 depicts the battery consumption and the number of messages required to join a topic, which supports the considerations we have done previously. In fact, the number of messages to join a topic increases when the total number of nodes grows, as illustrated in Figure 6(b). Figure 6(a) shows that the battery consumption is quite limited, even if it grows when the number of nodes is increased.

Figure 7 describes the average delivery time for notifications and the relative standard deviation. Such a time in average increases as the total number of nodes in the network grows, which is not surprising, but we also have that the variability of such a time, *i.e.*, the standard deviation, is not subject to such a linear increasing trend, since it is not function of the total number of nodes in the network.

V. Final Remarks

In this paper we have presented our solution for a broker-less publish/subscribe protocol for IoT that make use of the peculiar beaconing communication means provided by RFID. We have provided a preliminary simulation-based assessment to show the effectiveness and efficiency of such a solution. Our plans for future work is threefold:

1) We intend to compare our solution with some protocols available in the literature in order to better point out the quality of our proposal. Furthermore, we will implement our protocol within the context of some concrete IoT platforms. Specifically, we have plans to implements it for Arduino board, and mote sensors based on TinyOS and the nes-C programming language.
2) A key property that an IoT must satisfy is the security of the communications, in terms of confidentiality and privacy of the exchanged data. Currently, we are working on a group-based encryption mechanism within our publish/subscribe protocol by jointly protecting the communications and minimizing the relative costs to be paid, in terms of battery consumption, delivery time and traffic workload.



(a) (b)

Fig. 7: Time to distribute a notification within a topic-based group.

3) IoT, as any wireless network, is affected by network faults, such as message losses, that can compromise the effectiveness of the communication protocols. We are working on applying and optimizing a retransmission-based recovery action, called gossiping [18], so as to tolerate message losses with limited performance worsening.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29:1645–1660, 2013.

[2] K. Ashton. That "internet of things" thing. *RFiD Journal*, 2009.

[3] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54:2787–2805, 2010.

[4] T.R. Sheltami, A.A. Al-Roubaiey, and A.S.H. Mahmoud. A survey on developing publish/subscribe middleware over wireless sensor/actuator networks. *Wireless Networking*, October 2015.

[5] P.Th. Eugster, P.A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many Faces of Publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.

[6] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communication Surveys & Tutorials*, 17(4):2347–2376, Fourth Quarter 2015.

[7] R. Meier and V. Cahill. Taxonomy of Distributed Event-Based Programming Systems. *The Computer Journal*, 48(4):602–626, June 2005.

[8] A. Hakiri, P. Berthou, A. Gokhale, and S. Abdellatif. Publish/subscribe-enabled software defined networking for efficient and scalable iot communications. *IEEE Communications Magazine*, 53(9):48–54, September 2015.

[9] OMG. Data Distribution Service (DDS) for Real-Time Systems, v1.2. www.omg.org. Accessed: September 2012.

[10] A. Hakiri, A. Gokhale, P. Berthou, D.C. Schmidt, and T. Gayraud. Software-defined networking: Challenges and research opportunities for future internet. *Computer Networks*, 75, Part A:453–471, December 2014.

[11] M. Kirsche and R. Klauck. Unify to bridge gaps: Bringing xmpp into the internet of things. *Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 455–458, March 2012.

[12] IETF. RFC 6120: Extensible Messaging and Presence Protocol (XMPP). http://tools.ietf.org/html/rfc6120. Accessed: March 2016.

[13] P. Saint-Andre. XEP-0174: Serverless Messaging. http://www.xmpp.org/extensions/xep-0174.html. Accessed: March 2016.

[14] D. Locke. MQ Telemetry Transport (MQTT) V3.1 Protocol Specification. http://www.ibm.com/developerworks/webservices/library/ws-mqtt/. Accessed: March 2016.

[15] A. Stanford-Clark and H. L. Truong. MQTT for sensor networks (MQTT-S). http://www.mqtt.org/MQTTs_Specification_V1.0.pdf. Accessed: March 2016.

[16] K. Paridel, E. Bainomugisha, Y. Vanrompay, Y. Berbers, and W. De Meuter. Middleware for the internet of things, design goals and challenges. *Electronic Communications of the EASST*, 28, 2010.

[17] C. Esposito and M. Ficco. Deployment of rss-based indoor positioning systems. *International Journal of Wireless Information Networks*, 18(4):224–242, December 2011.

[18] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. Epidemic Information Dissemination in Distributed Systems. *IEEE Computer*, 37(5):60–67, May 2004.