Platforms and Protocols for the Internet of Things

EAI Endorsed Transactions
on the Internet of Things
Research Article ICST.ORG

Platforms and Protocols for the Internet of Things

Chiara Pielli[1,*], Daniel Zucchetto[1], Andrea Zanella[1], Lorenzo Vangelista[1], and Michele Zorzi[1]

[1]Department of Information Engineering, University of Padova, Italy

Abstract

Building a general architecture for the Internet of Things (IoT) is a very complex task, exacerbated by the extremely large variety of devices, link layer technologies, and services that may be involved in such a system. In this paper, we identify the main blocks of a generic IoT architecture, describing their features and requirements, and analyze the most common approaches proposed in the literature for each block. In particular, we compare three of the most important communication technologies for IoT purposes, i.e., REST, MQTT, and AMQP, and we also analyze three IoT platforms: openHAB, Sentilo, and Parse. The analysis will prove the importance of adopting an integrated approach that jointly addresses several issues and is able to flexibly accommodate the requirements of the various elements of the system. We also discuss a use case which illustrates the design challenges and the choices to make when selecting which protocols and technologies to use.

Keywords: Internet of Things, IoT architecture, IoT protocols, REST, MQTT, AMQP, IoT middleware

Received on 20 October 2015; accepted on 21 October 2015; published on 26 October 2015

doi:10.4108/eai.26-10-2015.150599

1. Introduction

The Internet of Things is a communication paradigm in which sensors and microcontrollers are extended into the world of everyday objects: machines, buildings, vehicles, plants, people themselves, etc. This technology shift is deemed to be the next stage of the information revolution after the massive spreading of the Internet in every field, and its impact is expected to be much heavier than that caused by the integration of the Internet in our lives through smartphones and other mobile devices. In fact, the IoT shall therefore be able to seamlessly incorporate a large number of heterogeneous end systems, while providing open access to selected subsets of data for the development of digital services [1]. The integration of potentially any object into the Internet allows for new forms of interactions between human beings and devices, or directly among devices, according to what is commonly referred to as the Machine-to-Machine (M2M) communication paradigm [2].

A recent investigation from Juniper Research has revealed that the number of IoT connected devices is predicted to be 38.5 billion in 2020 and up from 13.4 billion already by the end of 2015, a rise of over 285% [3] whereas other studies [4] foresee 26 billions of non-phone interconnected devices against 5 billions of phone devices by 2020. According to the McKinsey Global Institute analysis [5] the potential economic impact of IoT is going to increase from $4 trillions up to $11 trillions by 2025.

However, the heterogeneity of both end devices and applications complicates the already challenging development of the IoT [6] which needs to cope with massive access to the transmission channels, security issues and energy efficiency problems, which are stressed by the use of constrained end devices. To cope with these issues, the ongoing research in the scientific community addresses all layers of the protocol stack, from physical transmission up to data representation and service composition.

Although it is not straightforward to describe a unified scheme for the various IoT applications, it is

*Corresponding Author. Email: chiara.pielli@patavinatech.com

possible to pinpoint the basic blocks that make up every IoT architecture (Figure 1) [7]:

- *Edge Technology layer*: it is the hardware layer that represents the *things* part of the IoT and consists of sensors and actuators. The main function of this layer consists in collecting information from an environment or a system and processing this information. The end devices must be able to communicate with the Access Gateway layer in order to transmit the collected observations and to receive feedback from the upper layers. Several solutions have been proposed for efficiently managing communication among the end devices, which typically are severely constrained in terms of computational and storage capabilities, and energy capacity;

- *Access Gateway layer*: it represents the point of access to the Edge Technology layer and basically revolves around data handling, i.e., forwarding the information generated by the end devices to the middleware layer and sending data produced by the latter back to the devices. It must provide all the functions that the constrained peripheral nodes cannot bear and must support protocols for communicating with the IP world, whilst coping with the large variety of devices that may be present in the network;

- *Middleware layer*: it is the intermediate layer between the *things* and the Internet and is mainly responsible for filtering and storing the data received from the end devices. It is also responsible for enforcing the security policy in the IoT network. This layer must be able to cope with the device heterogeneity and hide it to the IoT applications in order to facilitate their access to sensor data;

- *Application layer*: it is the layer responsible for presenting the information to the final user. It provides a high level management of all involved devices in an integrated way, ensuring scalability, high availability, and the reliable and secure execution of the requested functionalities from the devices.

To cope with the intrinsic constraints of IoT scenarios, several challenges must be addressed at every layer and the development of the whole architecture also has to efficiently integrate every element without interfering with the rest of the system. The goal of this paper is to provide an introductory overview of the protocols and technologies that can be employed in the communication between the various elements of an IoT system.

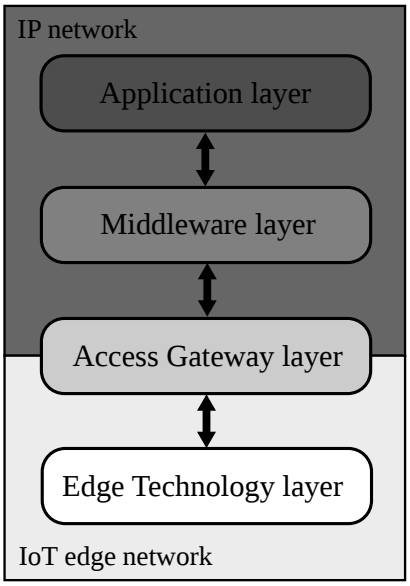

Figure 1. IoT architecture

The rest of the paper is organized as follows. In Section II we present a quick overview of the protocols and technologies used for physical communication in the constrained sensors networks. Section III discusses the most popular IoT messaging protocols, which are regularly used for communication between the Access Gateway layer and the Middleware layer, highlighting the strenghts and weaknesses of each approach. The role of IoT middleware is discussed in Section IV, together with the description of three existing middleware solutions and a comparison of their features. Section V provides a quick overview of the security aspects in IoT systems. Section VI introduces a use case; more specifically we describe a complete IoT architecture built for Smart Cities that makes use of LoRa™, a Low-Power Wide Area Network (LPWAN) solution as the physical communication protocol in the sensor network, MQTT as the messaging protocol, and Sentilo as the middleware platform. Conclusions are drawn in Section VII.

2. IoT wireless technologies

The basis of an IoT system is to be found in the physical connection between the devices in the IoT network. Most connections between *things* use wireless technologies, since the number of devices to be connected is usually large and, in many scenarios, devices need to be mobile, e.g., in the case of wearable technologies or for tracking purposes.

Multi-hop short-range wireless technologies have been the first enabler of IoT networks. Things connected to these networks usually run on dedicated protocol stacks specifically designed to satisfy the device constraints, although at least one of these devices must

be connected to an IP network. Devices connected to both the IoT and an IP network act as access gateways, as they allow users to communicate with *things* via traditional devices like PCs and smartphones. In this group, the most prominent technologies are the IEEE 802.15.4 family, including ZigBee™and 6LoWPAN, and ZWave™. These kinds of networks operate in the 2.4 GHz and 868/915 MHz unlicensed industrial, scientific and medical (ISM) frequency bands, with devices connected using a mesh topology. The distance between these devices ranges from few meters up to roughly 100 m, depending on the specific technology used and on the surrounding environment. One of the downsides of using a multi-hop technology is the need, for nodes in the network, to keep the radio circuitry on in order to forward the messages coming from other nodes, thus reducing the power efficiency of the network and reducing the battery life of mobile nodes. These technologies have also proven to be inadequate in scenarios where the network must provide a large coverage range, as in Smart City applications.

To overcome these shortcomings, new technologies have been proposed. They can be grouped in two families: cellular IoT networks and LPWANs and, unlike multi-hop short-range wireless technologies, they enable a *place-&-play* connectivity [8], i.e., any device can be connceted to the IoT network by simply placing it in the desired location and switching it on. In particular, the Third Generation Partnership Project (3GPP), which is the body that developed the specifications for the most popular cellular technologies, is attempting to revamp GSM (Global System for Mobile Communications) to support IoT devices, thus implementing the Cellular IoT (CIoT) architecture [9]. A possible issue that arises in these types of networks is the massive number of devices that need to access the transmission channel. Since cellular technologies were not designed to provide machine-type services to a huge number of devices, the signaling and control traffic may become the bottleneck of the system [6].

A possible alternative is represented by LPWANs, which combine the use of dedicated protocol stacks tailored to the device constraints, with a great coverage range. In these kinds of networks, the end devices are connected to a central aggregator, generally referred to as *gateway*, which provides bridging to the IP world in a fashion similar to the access gateway in multi-hop networks. The gateway coverage range is in the order of kilometers, making it possible to serve an entire city with a limited number of gateways. A limit of these networks is the low bitrate that, however, is expected to be sufficient for many IoT services.

Table 1. Comparison among LPWAN radio technologies.

Technology	SIGFOX™	Ingenu™	LoRa™
Coverage range [km]	rural: 30–50 urban: 3–10	≈ 15	rural: 10–15 urban: 3–5
Frequency bands [MHz]	868 or 902	2400	various, sub-GHz
Data rate [Kbps]	0.1	0.01–8	0.3–37.5
Nodes per BS	$\approx 10^6$	$\approx 10^4$	$\approx 10^4$

The first LPWAN technology proposed in the IoT market is SIGFOX™,[1] founded in 2009. The SIGFOX™physical layer uses an Ultra Narrow Band (UNB) modulation coupled with sub-GHz bands to ensure a great coverage range. SIGFOX™, which acts as an operator for IoT services, already deployed its nation-wide access networks in many European countries, including France, Spain and the Netherlands, thanks to the great coverage range of their gateways, claimed to be 30–50 km in rural areas and 3–10 km in urban areas [10].

A further LPWAN technology is LoRa™, designed and patented by Semtech Corporation [11], which also manufactures the chipsets. Its physical layer uses a derivative of Chirp Spread Spectrum, operating in the unlicensed sub-GHz bands. LoRa systems are being deployed by telecommunication providers like Orange and Bouygues Telecom in France, Swisscom in Switzerland, and KPN in the Netherlands. While the physical layer of LoRa™ is proprietary, the rest of the protocol stack, known as LoRaWAN™ [12], is being developed by the LoRa™Alliance[2], an association of industry partners dedicated to the development of LoRa™ solutions.

Ingenu™,[3] a trademark of On-Ramp Wireless, is another example of LPWAN technology. Ingenu™ networks, unlike most of the other LPWAN technologies, operate in the 2.4 GHz band, but thanks to the use of the patented Random Phase Multiple Access technology [13], can still work over long distances. In collaboration with Meterlinq, Ingenu™ is deploying a nationwide network in Italy to enable smart water and smart gas monitoring, with the long-term goal to scale the network to include additional IoT applications. Also, a nationwide network in the USA is planned to be completed by the end of 2017.

Table 1 shows a comparison of these LPWAN radio technologies, highlighting, in particular, the differences in bitrate and declared coverage range [8].

[1] http://www.sigfox.com/
[2] https://www.lora-alliance.org/
[3] https://www.ingenu.com/

3. Communication protocols

The interaction with the specific wireless transmission technologies discussed in the previous section is typically realized by means of standard Application Program Interfaces (APIs) and communication protocols that can be logically placed on the *Access Gateway layer* of Figure 1. The goal of this layer is to abstract the specificities of the lower layers and provide common ways to access the data collected by the IoT nodes. In this section we describe the most important protocols that are being proposed for this purpose. In particular, we focus on REST, MQTT and AMQP, as all of them are widely used and provide a comparable set of features. However, these communication protocols have been developed starting from different requirements and with contrasting use cases in mind, thus providing dissimilar performance in various scenarios.

REST. The Representational State Transfer (REST) is a software architecture style for building scalable web services, typically over the Hypertext Transfer Protocol (HTTP) [14], and originated from the Ph.D. thesis of Roy Fielding in the year 2000 [15]. For a service to be identified as RESTful, the following five constraints must be respected.

- *Client-server*: a RESTful service follows a client-server model, with separation of concerns.
- *Stateless*: at the server side, no information about session and client context is retained and each request is an independent transaction that is unrelated to any previous request. So, each client request needs to contain all the information necessary to serve the request and only the client holds the session state. In this way servers are simpler and scalability is enforced.
- *Layered system*: client and server may not be directly interconnected. Intermediary servers may improve system scalability by enabling load balancing and providing shared caches, and may also enforce security policies.
- *Cacheable*: clients and intermediaries can cache responses, allowing an improvement in scalability and performance.
- *Uniform interface*: the uniform interface between client and server allows each part to evolve independently. This constraint is based on two notions: first of all, individual resources must be identified in the requests and, secondly, these resources can be manipulated according to the CRUD pattern: Create, Retrieve, Update and Delete.

A further optional requirement is that servers shall be able to transfer executable code to clients.

The concept of resource is central in RESTful services: every resource is globally and uniquely identified by a Uniform Resource Identifier (URI) and is considered as an abstract entity disconnected from its representation. Since REST APIs are used almost exclusively over HTTP, in this work we will consider only REST over HTTP.

MQTT. The Message Queuing Telemetry Transport (MQTT) protocol is a lightweight event and message oriented protocol allowing devices to asynchronously communicate across constrained networks to remote systems. MQTT, version 3.1.1, has recently been standardized by the Organization for the Advancement of Structured Information Standards (OASIS) consortium [16] and has been submitted to the International Organization for Standardization (ISO) in order to become an International Standard [17]. The MQTT protocol has been initially designed to communicate telemetry data in a M2M scenario, and therefore can work in unreliable networks with small bandwidth and high latency. The size of the message header can be as small as 2 bytes, since, in IoT and M2M scenarios, messages are typically short and control information may easily become the predominant part of the communication. The protocol has a client-server architecture: the server part is represented by a central broker that acts as intermediary among the clients, i.e., the entities that produce and consume the messages. MQTT revolves around the concept of topics, which are UTF-8 (Unicode Transformation Format, 8 bit) strings used by the broker to filter messages for each connected client. Topics are used by clients for publishing messages and for subscribing to the updates from other clients. This pub/sub mechanism avoids the need for consumer entities to continuously poll the data producers for new messages: through a topic subscription, an MQTT client receives all the messages published by other clients for that topic. MQTT libraries have been provided for all major IoT development platforms, for the two major mobile platforms, i.e., Android and iOS, and for several programming languages (Java, C, PHP, Python, Ruby, Javascript).

AMQP. The Advanced Message Queuing Protocol (AMQP) is an open Internet protocol for message exchange. AMQP version 1.0 has been standardized by the OASIS consortium [18] and successively by the ISO, as ISO/IEC 19464:2014 [19]. Its original goals were to enable communication between systems of different vendors, support messaging semantics needed in the financial service industry, be extensible to new queuing and routing strategies, and allow complete configuration of the message routing. The initial development of AMQP started from the initiative of financial institutions that needed to reliably exchange data between heterogeneous systems. Since then, this

protocol has been successfully used to exchange messages in M2M scenarios. In an AMQP system, the entities that produce and consume messages over the network are linked to central messaging servers, called brokers. At the broker, inbound messages are put in different queues, waiting to be collected by message consumers. The message routing is very flexible, as it allows, e.g., to send messages in broadcast, to direct them to a single entity, or to use a topic-based pub/sub mechanism, as in MQTT.

In the rest of this section, we compare the protocols performance with a focus on the IoT scenario. Hence, we consider the following aspects: support of different IoT traffic patterns, data encoding and manipulation, reliability, and security.

3.1. Support of different IoT traffic patterns

In an IoT network many devices are linked to a central aggregator that monitors and operates them. An IoT network can be used to serve several purposes, which may differ in the way the messages are exchanged between network nodes. It is possible to categorize the message exchanges of an IoT network in four different communications patterns (Figure 2): *telemetry*, *notifications*, *inquiries*, and *commands* [20]. The characteristics of each such traffic patterns, and the suitability of the different communication protocols to support them are discussed next. The result of such a discussion is summarized in Table 2.

Telemetry: in the telemetry pattern, the device autonomously sends data to the central aggregator with a fixed time period or at the occurrence of some events. The aggregator, upon data reception, simply stores them for further analysis. Data messages are usually small, with lengths of some tens of bytes, but frequent. Depending on the use case, the average interval between the data messages could range from hours (e.g., for environmental monitoring) to fractions of a second (e.g., for car telemetry). The HTTP protocol, being ASCII-oriented, is too verbose for short messages, since each optional header greatly increases the size of the message. For this reason, the REST approach is not very efficient in this case. MQTT, instead, was designed to have a low overhead and to be very efficient in case of short messages. AMQP, due to its many features, has a larger header than MQTT, however it provides a flow control mechanism, not present in HTTP and MQTT, to slow down the source in case the destination is unable to keep up to the rate of messages.

Notifications: refer to the central aggregator sending messages to the devices to notify them about an event. It has many characteristics in common with the telemetry pattern, but it also requires all the end devices, or their associated access gateway, to be reachable from the central aggregator. Hence, if using the REST architecture over HTTP, special attention is required to connect nodes through Network Address Translation (NAT) and firewall gates. Furthermore, each device must host an HTTP server to be able to receive the notifications, adding complexity to the device, which is usually constrained in terms of memory and computing power. None of these issues arise for MQTT and AMQP because, in those protocols, connections are always initiated by the client, so that only the message broker needs to be publicly reachable. Furthermore, to receive and send messages, nodes only need the MQTT or AMQP client, which can run in a constrained environment.

Inquiries: in this pattern the end device (or the element in charge of connecting the end device to the IP world) sends requests to the central aggregator, which successively answers with the required information. This is exactly the use case addressed by the REST architecture, since it can be seen as a traditional request-response pattern. Here the HTTP server must be placed in the central aggregator, while the end devices (or the access gateways) are equipped with an HTTP client, which is less demanding in terms of computing resources than the HTTP server. Instead, the MQTT protocol does not formalize a way to exchange messages in a request-response pattern, so that the parties must agree beforehand on pair topics for this pattern: a topic for publishing requests and another for publishing responses. However, the OASIS MQTT Technical Committee is currently working on a mechanism to formally enable the request-response messaging pattern in MQTT. AMQP, instead, already supports a mechanism to enable the request-response message exchange, thus providing the flexibility needed to operate in this scenario.

Commands: in this case, the central aggregator sends a message to the end device to trigger an action and then waits for the reply by the end device containing the outcome of the action. Besides the reachability issue described in the notifications pattern, the REST architecture also fails to manage the case in which the end device is temporarily offline: messages sent while the device is unavailable are simply lost. MQTT addresses this problem with the introduction of the optional retain flag in the published message, forcing the message to be sent to all clients that, in the future, will subscribe to the corresponding topic. As mentioned before, it is also important to remark that MQTT lacks a formalized request-response mechanism to correlate the request to the end device with its response. Regarding AMQP, its message broker always saves incoming messages to the message queues, allowing them to be retrieved at a later time even if the recipients are temporarily unavailable. Furthermore,

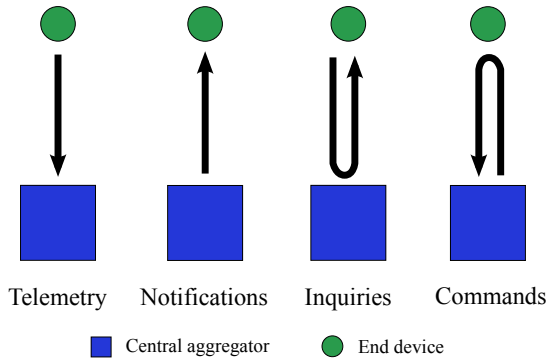

Figure 2. Communication patterns

AMQP introduces a refinement that consists in a Time-to-Live indication, to remove stale messages from the queues.

3.2. Data encoding and manipulation

Usually, message recipients elaborate the received data depending on the type of the message content. AMQP features a rich set of metadata to describe the transmitted data, including a complete type system. Actually, AMQP type system defines some primitive types, togeteher with constructs to extend them in order to allow the association of an AMQP value with an external type that is not present as an AMQP primitive. This feature is not present in REST nor in MQTT. Actually, the REST architecture only allows to specify the type of message content through the HTTP *Content-Type* header, but lacks a more comprehensive metadata set. The message content is, instead, completely opaque to MQTT, which does not even allow the indication of generic information such as the *Content-Type*. In order to elaborate the message content, the parties must therefore agree beforehand to the exact format of the message, and a change in message format requires the communicating parties to be manually updated.

3.3. Reliability

In the IoT field, reliability refers to the absence of communication errors in the transmitted data and the guarantee that transmitted messages have been delivered to the recipients. REST over HTTP relies only on the underlying Transmission Control Protocol (TCP) to provide reliability of message exchanges, while MQTT and AMQP offer more flexible mechanisms to provide additional levels of reliability assurance [21]. In MQTT, Quality of Service (QoS) is an attribute of the individual message being published. However, due to device or link contraints, a subscribing client can set the maximum quality of service a broker can use to send messages to it, hence each message will be delivered with a QoS value that is the minimum between the value of the QoS attribute in the message and the maximum QoS accepted by the client. The QoS attribute can take three possible values: *at-most-once* (QoS level 0), in which no acknowledgement is needed, *at-least-once* (QoS level 1), which requires the transmission to be acknowledged, and *exactly-once* (QoS level 2), that requires a more sophisticated acknowledgement mechanism which involves the exchange of three acknowledgement messages. QoS level 2 is the only level that can be used for non-idempotent messages that must be delivered reliably, since it guarantees that the message is not delivered multiple times, unlike QoS level 1. However, it is to be noted that, with the highest QoS level, the overhead is large and sending messages at a high rate may degrade the system performance. AMQP has QoS properties similar to those of MQTT, supporting message queuing and delivery semantics that cover at-most-once, at-least-once and exactly-once deliveries. Furthermore, the AMQP specification also describes an optional transaction mechanism with a multiphase commit sequence, to ensure that each message is delivered as intended, regardless of failures or reboots.

3.4. Security

REST over HTTP, MQTT and AMQP can all be placed on top of Transport Layer Security (TLS) [22], which provides confidentiality of the data exchanged. TLS also supports authentication of the server, which is the message broker in case of MQTT and AMQP, or the HTTP server in case of REST. While TLS could also be used to authenticate clients, this is not commonly employed because it involves the generation and management of a certificate for each client that must be able to connect to the server. Instead, client authentication is typically implemented in the protocol running on top of TLS. With the REST architecture it is possible to use HTTP Basic or Digest authentication mechanisms. MQTT, instead, allows the clients to specify the username and, optionally, a password while connecting to the broker. AMQP does not provide an authentication mechanism itself, but allows the use of the Simple Authentication and Security Layer (SASL) framework.

To conclude, the choice of the protocol to use depends on the specific use case. The most important factors to consider are: the rate at which new messages are generated, the underlying network performance, the reliability of links, the necessity of extensibility and message semantic, and the quality of service required. According to this analysis, in Section 6 we explain the choices we made for a specific use case.

Table 2. Features of the various communication patterns and their support by the considered protocols. (✓: fully supported; ∼: partially supported; ✗: not supported.)

Features	Telemetry			Notifications			Inquiries			Commands		
	REST	MQTT	AMQP	REST	MQTT	AMQP	REST	MQTT	AMQP	REST	MQTT	AMQP
Small overhead	✗	✓	∼	✗	✓	∼	✗	✓	∼	✗	✓	∼
Flow control	✗	✗	✓	✗	✗	✓	Not required			Not required		
Reachability of nodes behind NAT or firewall gates	Not required			✗	✓	✓	Not required			✗	✓	✓
Support for constrained devices	✗	✓	✓	✗	✓	✓	✗	✓	✓	✗	✓	✓
Request-response pattern	Not required			Not required			✓	✗	✓	✓	✗	✓
Publish-subscribe pattern	✗	✓	✓	✗	✓	✓	Not required			Not required		
Support for temporarily offline devices	Not required			✗	∼	✓	Not required			✗	∼	✓
Overall fit	✗	✓	✓	✗	✓	✓	✗	∼	✓	✗	∼	✓

4. Between the Things and the Internet: the middleware

IoT systems often deal with different types of devices, each with its own communication protocol and different requirements, that need to somehow interact with the final user. In order to meet this demand, IoT architectures require a software platform, called middleware (see Figure 1), which represents an intermediate layer between the Internet and the *things* and acts as a bond joining mixed applications communicating over heterogeneous interfaces. The middleware is also in charge of masking the system complexity that is faced when interacting with the end devices, so that even the average technology-inexperienced user is able to enjoy IoT services effortlessly.

The development of a middleware in the IoT context requires the support of various functionalities. The following list summarizes the crucial issues that the middleware must address [23] [24]:

- *Interoperability*: the conceived middleware must cope with the great heterogeneity of the smart objects. Interoperability aims at device abstraction and is threefold: technical, syntactic and semantic. According to the European Telecommunications Standards Institute (ETSI) [25], technical interoperability is defined as the association of hardware or software components, systems and platforms that enable M2M communication to take place. Syntactic interoperability deals instead with data formats and asks for an agreed upon and well-defined common syntax for messages. Finally, semantic interoperability is associated with the ability of computer systems to exchange data with unambiguous and shared meaning, understandable to humans.

- *Device discovery and management*: bootstrapping is a crucial phase in the IoT as it prepares the smart objects to join the network and to interact with the other end devices, detecting all their neighbours and making their presence known. Moreover, the middleware must be aware of the context in order to work in smart environments, as the smart objects may move in a random fashion causing rapid changes in the network topology. An IoT middleware must be able to update routing information in an efficient way without affecting the overall network performance and indipendently of the routing protocol used. Another issue related to device management involves actuators, which may be accessed simultaneously by different applications in a contradictory way: the middleware is in charge of solving such conflicts.

- *Security and privacy aspects*: security is a key point in IoT architectures, which often deal with sensitive data. Thus the middleware must ensure authentication, confidentiality, data integrity and non repudiation and must be able to manage different roles and privileges.

- *Application abstraction*: the middleware should provide an interface for both high-level applications and end users to interact with the end devices without prior knowledge about the physical network and the implementation details.

- *Data management*: the IoT is leading to an explosion of data exchanges, thus the middleware

needs to cope with enormous volumes of data. It is also necessary to have historical data stored, which allow the end user to retrieve old observations and to display time-series graphs.

Other useful features concern modularity, i.e., the possibility to add functionalities without altering the existing core, which is essential to customize the platform in a plug-and-play fashion for accomodating missing features, and the capability of supporting downlink traffic towards the end devices to enable actions from the final users. Many challenges need to be addressed in order to build an efficient, robust, scalable and real-time platform. For these reasons, it may be preferable not to develop a custom middleware from scratch, but rather to use an existing and well tested platform and adapt it to fulfill the specific system requirements, if needed. There exist many implemented middlewares; we selected and analyzed three different frameworks, namely openHAB, Sentilo and Parse, which are intended to be used along with an access gateway to provide the described functions. Their characteristics are detailed in the rest of this section.

4.1. openHAB

The software platform openHAB[4] targets home automation and was born in 2008 from the need of its creator, Kai Kreuzer, to integrate sensors and actuators in his own house in Darmstädt (Germany).

OpenHAB is extensible through a plug-and-play principle and interoperable thanks to the use of modules to support different communication protocols and mechanisms. Many modules have already been implemented, such as the MQTT binding, a component that allows openHAB to act as an MQTT client and hence to support a pub/sub mechanism for seamlessly interacting with the nodes. End devices are registered in openHAB as *items*. An item is a data-centric functional atomic building block: all openHAB resources are represented using this abstraction, which is independent of the technology used. In this way the final user does not need to be aware of the physical network technology employed, since he/she only needs to dialogue with openHAB via HTTP. Historical data can be stored in relational, NoSQL or round-robin databases, in IoT cloud services or in simple log files, according to the user needs. For what concerns security, TLS can be enabled for all the protocols that support it and user authentication is also possible. However, openHAB targets home automation and therefore it is designed to be used by a limited number of users, all with complete access to the available information. The implementation of user differentiation according to given roles is on the future work list and will make it possible to assign different read and write permissions to users.

The strenghts of the openHAB platform are its high modularity and the presence of bindings that support different protocols. Its main shortcomings, instead, concern the lack of a user conditional access and the internal items implementation, since items cannot be bound to a specific time and geospatial context nor customised according to the users needs. This abstraction, for example, makes it impossible to store location information to track mobile end devices; also, past measures cannot be inserted at a later time, since data cannot contain a custom timestamp.

The openHAB project gave rise to Eclipse SmartHome, a flexible framework for the smart home. Eclipse SmartHome will be the basis for the next iteration of the openHAB project, namely openHAB 2, which is still in its early stages of development.

4.2. Sentilo

Sentilo[5] is the product of a project started in November 2012 by the Barcelona City Council and conceived to make Barcelona a reference point in the field of Smart Cities. The name Sentilo was chosen because it means *sensor* in Esperanto, underlying the intention of openness and universality in the use of a platform.

Sentilo is an extensible open source platform that offers a REST API over HTTP, supporting all the communication patterns described in Section 3. However, Sentilo does not support other communication protocols: it is necessary to implement a bridging module for every other protocol to be used in the architecture, in order to properly translate its messages into their REST equivalent. Sensors and actuators are registered in Sentilo as uniquely identifiable items and are organized according to a hierarchical structure. It is also possible to track the location of mobile sensors. For what concerns access control, Sentilo features a token-based authentication system to identify the petitioner of the request, coupled with a privilege policy based on roles. Also, to provide confidentiality, the REST API can be used over the secure HTTPS channel.

To recap, the selling points of Sentilo are the possibility of extending its functionalities in a plug-and-play fashion, the presence of a hierarchical and slightly customizable item representation and the implementation of an authorization and role-based permission mechanism that facilitates the interaction in the same context of multiple users with different roles. Sentilo's most important drawback, instead, is its weak

[4] http://www.openhab.org/

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

interoperability, as it natively supports only the REST API and cannot communicate via other protocols.

4.3. Parse

Parse[6] is a cloud-based data management system that allows people to quickly develop web and mobile apps. More specifically, it is a Backend as a Service (BaaS) solution, a turnkey service that adds user authentication, push notifications, social media integration, location data, and data analytics into any app. It was acquired by Facebook in 2013 with the aim of adding Mobile BaaS capabilities to the existing platform and, as IoT backends are the logical extension of mobile backends, at the end of March 2015 Facebook announced Parse for IoT. Parse for IoT is a collection of Software Development Kits (SDKs) for connected devices, such as Arduino Yún, a microcontroller board with built-in WiFi capabilities. Parse SDKs are directly deployed on hardware platforms and provide a simple REST API. Such SDKs make devices able to receive push notifications, save data, and take advantage of the Parse Cloud. All Parse resources are represented as Parse Objects, uniquely identified and customizable. Particular objects are the *roles*, which group users with common access privileges in order to support role-based access control. Even data storage on Parse is built around a Parse Object: there is no need to explicitly create databases or tables to use Parse, since data will be automatically stored in the cloud. Finally, it is possible to extend the functionalities of Parse for the IoT by creating the so called Cloud Modules.

To sum up, the strengths of Parse are the great customization available for Objects and the presence of a solid permissions and roles structure to control user access. The major weaknesses are instead the need of installing the SDK on each device and the inability to communicate in a way different from REST. Moreover, being Parse for IoT so recent (it was officially announced just a few months ago), this tool is not widely deployed, so a proof of its real-world performance is still missing.

4.4. Platforms comparison

The above descriptions highlight the great effort required to develop a complete middleware that simultaneously accomplishes all the listed requirements.

All three platforms represent valuable middleware solutions for the IoT, but at the same time all of them lack some useful features. They all provide modularity, data management and application abstraction, which is typically achieved by implementing a REST API designed to be used by the user interface. Security is achieved by means of authentication and message encryption, but, unlike the others, openHAB does not offer user differentiation according to roles. However, openHAB is the only middleware, among the three described in this paper, capable of supporting different communication protocols simultaneously. We deem openHAB to be suitable for small deployments, where there is no need to distinguish among end users, whereas Sentilo, which supports user roles, may be employed in wider deployments, although it may require the implementations of protocol bridging bindings. Parse for IoT is not the best suited platform for large existing IoT networks, since it requires the SDK installation on all the hardware devices, and therefore depends on the particular sensors and the network topology used.

Table 3. Comparison among the platforms

PLATFORM	OPENHAB	SENTILO	PARSE
modularity	✓	✓	✓
application abstraction	✓	✓	✓
multiple protocols support	✓	×	×
semantic and syntactic interoperability	✓	✓	✓
data management	✓	✓	✓
data storage	✓	✓	✓
items custom representation	×	✓	✓
user conditional access	×	✓	✓
add timestamp to transmitted data	×	✓	×
security	✓	✓	✓
growing community	✓	✓	✓

5. Security in IoT systems

A central aspect of every IoT application is security, which must be guaranteed at every level of the system. IoT security, in particular, revolves around the concepts of identification, confidentiality, integrity and availability, and needs to meet the new requirements implied by the pervasive presence of the Internet in any aspect of daily life. Internet-facing services are in fact under continual attack and this does not bode well for the IoT, which relies on it and also incorporates many constrained devices for which it is hard to apply security mechanisms such as frequency hopping comunication and public key encryption [26]. But as the IoT also touches many sensitive areas, security represents a challenge that cannot be neglected: attacks and malfunctionings would just outweigh any of the IoT benefits. Security experts are currently investigating

[6] https://parse.com

whether current security mechanisms can be integrated in the IoT or new designs are required to accomplish security goals. What mainly introduces new threats is the distributed nature of IoT architectures and the use of fragile technologies, such as limited-function embedded devices in public areas where they are accessible by anyone and may be physically harmed [27]. As sensors are typically simple low power devices, they cannot even support ordinary security measures: network firewalls and protocols can manage the high-level traffic flowing through the Internet, but the protection of the endpoint devices with limited resources available to accomplish it raises new challenges and demands for revolutionary solutions.

Key features for gaining security are the following.

- *Identification*: the *things* must be uniquely identified, indipendently of their underlying mechanisms, e.g., the IP address they are associated to. Assigning a unique identifier to devices is the basis for the authentication step and the consequent authorization phase.
- *Confidentiality*: it refers to the guarantee that information is not made available or disclosed to unauthorized individuals, entities, or processes. Confidentiality is fundamental in an IoT scenario, in which a plethora of devices transmit messages, leading to an explosion of data. Access to these data must be controlled mainly by means of cryptographic mechanisms and users access lists.
- *Integrity*: to maintain the consistency, accuracy, and trustworthiness of data over its entire life cycle, data must not be changed in transit or altered by unauthorized people.
- *Availability*: for any information system to serve its purpose, the information must be available when it is needed. Availability may be hindered by legitimate users too, if they flood the network with requests that exhaust network resources, interrupting services available to other legitimate users.

Clearly, the IoT is prone to be more susceptible to attacks than the rest of the Internet, since billions of devices will be producing and consuming a large number of different services. From a network perspective, the sensors should open a secure communication channel with more powerful devices exploiting cryptographic algorithms and using an adequate system for exchanging the keys. A safe transmission over TCP/IP connections can be achieved by enabling Transport Layer Security (TLS), which asks the parties to authenticate themselves and provides message encryption. At the application level, security needs for different application environments are different, although data privacy, access control and disclosure of information are likely common requirements. In [7] the authors stress the crucial role of security and privacy and highlight how the public acceptance of the IoT will happen only when strong security and privacy solutions will be in place. In fact, when the Internet first appeared, no security infrastructure had been built for it. But, when the first security problems came out, the only viable solution to solve them was to treat security and privacy as add-on features. In the IoT, instead, security has to be intrinsic, hence we must find new fundamental solutions, shared among all interested parties, for addressing this challenge.

6. A use case: an IoT system for Smart Cities

We participated in the development of a complete IoT system that targets the Smart Cities context, a project carried out by Patavina Technologies s.r.l.[7] in the city of Padova, Italy.

LoRa™ has been chosen as the wireless technology. From the analysis carried out in [8] and summarized in Section 2, it results that, although the declared coverage range of LoRa is slightly lower than that of the other two technologies, the transmission data rate achievable with LoRa is higher. The LoRa™ network is typically laid out in a *star-of-stars* topology, where the end devices are connected via a single-hop LoRa™ link to one or many gateways which, in turn, are connected to a common Network Server (NetServer) via standard IP protocols [12]. The NetServer represents the point of access for the LoRa™ network as it can support other communication protocols.

We chose to make the NetServer communicate with the rest of the world using MQTT. As outlined in Section 3, MQTT is a lightweight protocol that meets IoT requirements thanks to its very small message header and the pub/sub mechanism, features not provided in HTTP. AMQP was discarded because it is very complex and all of the features missing in MQTT but provided in AMQP were not necessary for the implemented architecture. Therefore, setting up AMQP clients and broker would have required much more work with very little benefit. An advantage of AMQP over MQTT is the message queueing system that allows the clients not to miss messages arrived whilst they were unavailable. Anyway, when unmissable data is sent, it is still possible to use the simple retain mechanism available in MQTT, as explained in Section 3. Patavina Technologies also developed mechanisms for detecting the unavailability of any MQTT element and activate it again within a short time, assuring that no MQTT component is affected by an

[7] http://www.patavinatech.com/en

extended downtime. Request and response topics and the format of the message payload have been agreed beforehand.

The selected middleware is Sentilo. In fact, Parse for IoT demands the SDK installation on hardware devices, restricting the freedom of choice on the development environment, and mandating the use of REST interfaces on the devices, a choice not well suited for the constrained devices and network technology used in our solution. On the other hand, openHAB did not provide a mechanism for user differentiation according to roles. Hence, the overall architecture sees LoRa devices that communicate with the IP world through the NetServer, which supports MQTT. The messages sent by the NetServer are processed by an Application Server (AppServer) and the new human-readable MQTT messages are forwarded to Sentilo, which is in charge of registering the nodes and their data so that authorized users can access them. We also developed a friendly website for interacting with the nodes and displaying their readings.

The system deployment corroborated the choices made by Patavina Technologies. LoRa network connectivity has been tested by installing a private network in a large and tall building (19 floors), with nodes placed also in heavily shielded positions, e.g., inside elevators in order to put the connectivity condition under strain. Other experimental tests have been carried out in Padova with the purpose of assessing the worst case coverage in an urban environment. It turned out that, in harsh propagation conditions, the LoRa technology allows to cover a cell of about 2 km of radius and, even when assuming a radius of 1.2 km to take into account a reasonable margin for interference and link budget variations, the number of gateways needed for assuring coverage in the municipality of Padova is much lower than the number of sites required by one of the major cellular operators in Italy to provide mobile cellular access over the same area. MQTT has proved to be an excellent communication protocol: the pub/sub mechanism makes it possible to automatically receive updates from nodes avoiding the polling procedure of HTTP, while the extremely small header of MQTT messages (which is even smaller than in AMQP) affects the traffic intensity in a minimal way. However, the use of MQTT required the implementation of an additional bridging module in Sentilo for the conversion of MQTT messages into HTTP messages (and viceversa for downlink traffic). Using this setting, we evaluated the average delay experienced by a packet in uplink from the NetServer to the final application and the average uplink traffic intensity for a particular network setup. Both metrics have been calculated as the average values over more than 2.5 million packets. We did not consider downlink traffic, mainly constituted of sporadic commands targeting the sensors, as messages coming from the nodes are expected to be the predominant traffic in the considered scenario.

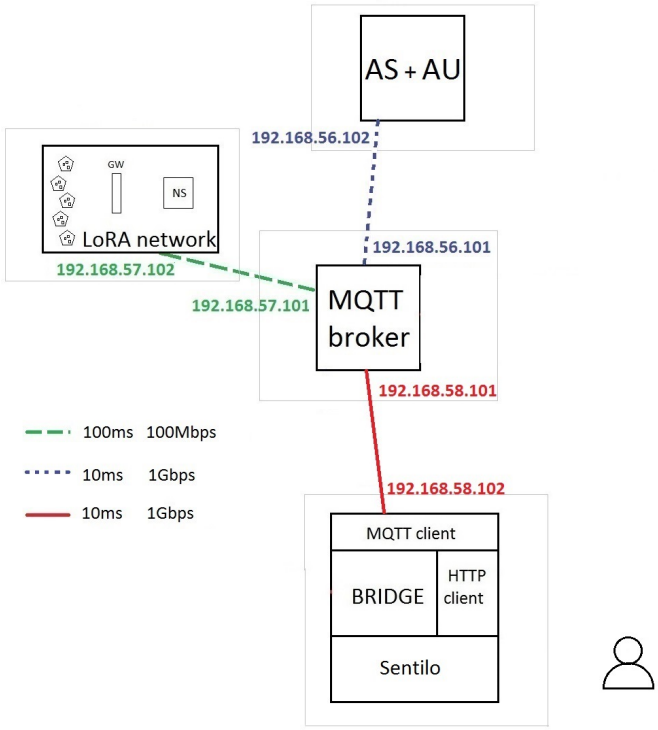

Figure 3. Simulation setup

The configuration we adopted is represented in Figure 3 and sees the AppServer, the MQTT broker and Sentilo deployed in a cloud service, hence with very fast connections between the elements, while the NetServer is placed close to the LoRa gateways in order to minimize the latency in the LoRa network, and is connected to the MQTT broker by means of an Asymmetric Digital Subscriber Line (ADSL) link. The propagation times are around 10 ms for the connections in the cloud and 100 ms for the ADSL link. It is worth noting that the use of TLS for security reasons increases the traffic intensity because of the preliminary authentication phase and the additional message headers. Moreover, we used an MQTT QoS level of 2 (see Section 3) in all exchanges, thus affecting both the traffic intensity - because of the acknowledgement messages - and the delay, since the broker needs to wait for the acknowledgement from the producer before sending the message to the subscribers of the involved topic. It turned out that, in addition to the LoRa traffic, each message generated by a peripheral node produces about 22 kByte of traffic resulting from the MQTT and HTTP messages exchanged in order to process the node observation and deliver it to the final user. About a quarter of such flow is comprised of HTTP

Table 4. Traffic intensity. Bytes exchanged at each link for one node observation

TRAFFIC [KBYTE]	$NetS \rightarrow brk$	$AppS \rightarrow brk$	$brk \rightarrow Sentilo$
NO TLS	0.6	3.9	11.1
TLS	0.8	6.7	13.9

Table 5. Transmission delays

DELAY [MS]	$t_{propagation}$ (all)	t_{broker}	t_{AS}	$t_{Sentilo}$
NO TLS	130	204	29	120
TLS	130	203	28	120

traffic for dialoguing with Sentilo. The traffic associated to each link is reported in Table 4, where *brk* refers to the broker, *NetS* to the NetServer and *AppS* to the AppServer. The global traffic highly depends on the number of nodes employed and on the rate at which they transmit their messages.

The average delay experienced by a packet can be expressed as the sum of the various propagation times and the processing times, namely the transmission times from the NetServer to the broker, from the broker to the AppServer and vice versa, from the broker to Sentilo and from Sentilo to the final user, the time needed by the broker for handling the QoS level and identifying the interested subscribers, the processing times of the AppServer and Sentilo. Table 5 shows the average propagation and processing times we evaluated in our simulations, resulting in an average transmission time of about 480 ms. Notice that this value strongly depends on the propagation times of the connection links, especially that of the ADSL, that represents the bottleneck of the transmission, whereas the processing times of the AppServer and Sentilo are in the order of just a few milliseconds.

Security in the Edge Technology layer is granted by the LoRa™ technology: data frames are encrypted with the scheme described in IEEE 802.15.4/2006 Annex B [28] using the AES algorithm with a key length of 128 bits. For each end device there is a specific application session key which is used by both the NetServer and the end device to encrypt and decrypt the payload field of application-specific data messages. Security on the MQTT connections is granted by enabling both authentication and TLS encapsulation: the MQTT broker provides username and password authentication and limits access to topics by using access control lists, whereas TLS ensures confidential transmission. Finally, Sentilo REST API is used over the secure HTTPS channel and Sentilo validates all HTTP requests according to the AAA architecture: Authentication, Authorization and Accounting. This means that the platform first identifies the petitioner of the request, then checks whether it is authorized to perform the requested action over the requested resource, and it finally traces the request by auditing the action and who performed it. Authentication is enabled by the mandatory use of an identification field in the HTTP headers, resulting in the so-called token-based authentication, which also allows for the authorization of a request by simply looking up the privileges associated to the involved token.

However, a weakness in Sentilo's security framework is that tokens, which are necessary to guarantee a secure and controlled access to resources, were stored in the database in clear and, although the access to the databases requires authorization, it is always a good habit not to store passwords as they are, as a malicious attacker may find a way to access the database. There exist many hashing techniques commonly used for storing passwords, such as the MD5 algorithm and the family of the Secure Hash Algorithm (SHA). We opted to use *bcrypt*, a cryptographic hash function (i.e., a one-way hash function, practically impossible to invert) which aims at being slow, or, more precisely, as slow as possible for the attacker while not being intolerably slow for the honest systems. It is derived from the Blowfish block cipher which uses look up tables to generate the hash, thus requiring a significant amount of memory space. This discourages attacks based on Graphics Processor Unit (GPU), which excels at doing simple manipulations on a large set of data, as it will become cumbersome to generate the hashes due to memory restrictions.

Another leak of Sentilo concerns the generation mechanism of tokens, based on the hashing of some knowable values, namely, a prefix retrievable in the source code, the name of the entity for which the token is being generated, and the creation time of this entity with a millisecond accuracy. Tokens are generated in two steps: firstly, the three mentioned elements are concatenated in a single string, and then the hash function of such string is computed according to the SHA-256 algorithm. The resulting token is a string made of 26 hexadecimal numbers. Basically, knowing the timestamp of the creation of a specific entity in Sentilo, it is possible to calculate the token associated to that entity. We tried to perform a brute-force attack to retrieve the token associated to a particular Sentilo role, and succeeded in a reasonable amount of time. We implemented a non optimized single-threaded attack that runs on Central Processing Unit (CPU) rather than on GPU. We ran our brute-force attack on an Intel® Core™ i7-2600 quad core processor and we found that about 4 ms were needed for a single attempt. Thus, knowing the creation day of the entity for which the token has been generated, the token is retrievable in 4 days in the worst case.

The original token generation procedure is clearly unsafe and represents a big leak in the security of Sentilo. We hence decided to change it with a completely random token generator. If a brute-force attack is perpetrated by trying all possible strings of 26 hexadecimal characters, the average time for determining the correct token increases considerably. There are $16^{26} \simeq 2.03 \cdot 10^{31}$ possible combinations, but the number of tries before a success cannot be represented as a geometric random variable as the attempts, despite being independent of each other, are not identically distributed: after k wrong attempts, $16^{26} - k$ combinations remain. If we model the probability of needing at least k attempts before succeeding with a random variable, it is possible to estimate the lower and upper bounds of its cumulative distribution function by assuming identically distributed tries with the minimum and maximum probabilities, respectively. Considering that $p_{j+1} \leq p_j \forall j \geq 0$, the minimum probability of a try is that of the first attempt, i.e., $p_0 = p = 16^{-26}$, whereas the maximum probability of k tries is $p_k = 1/(1/p - k)$. We evaluated that for $k = 2 \cdot 10^{30}$ the probability is still low, about 0.1. With an average computing time of 4 ms per attempt, about 10^{20} years would be needed for trying $k = 2 \cdot 10^{30}$ combinations. It is evident that using a random token certainly improves security against brute-force attacks.

The described use case shows the extent of elements that must be considered in an IoT system and the effort needed for integrating them in a solid, robust and secure system. Currently, there exist many solutions in the literature and since there is no well-established and widely-acknowledged best practice, developers should analyze the available strategies and protocols to identify those that best meet their requirements.

7. Conclusions

In this paper we identified the fundamental parts of a generic IoT system, independently of its final application. We outlined the requirements and features of the different modules that compose an IoT architecture and described the current solutions, highlighting pros and cons of each approach. We also depicted the security needs of any IoT system, stressing the importance of security and the variety of aspects that must be taken into consideration. Finally, we reported a use case in which a complete IoT architecture has been developed in the context of Smart Cities. We found that the used solutions have great performance, but still have several issues regarding, in particular, the security aspects. Furthermore, integration of the different layers in an IoT architecture is difficult due to the different interfaces, which required bridging modules to translate messages between them.

Future work may focus on a more comprehensive performance evaluation of the architecture described in Section 6, both for what concerns LoRa™ and the IP network. For instance, further experimental tests in even harsher environments may allow to sharpen the values about LoRa™ coverage range. Moreover, by deploying the system for a longer observation time and in a wider area, we could carry out perfomance measures concerning, e.g., scalability, reliability, robustness and energy efficiency.

Acknowledgement. The authors would like to thank Nicola Bressan and Ivano Calabrese from Patavina Technologies, for their fundamental contributions in the implementation of the LoRa™ system.

References

[1] Zanella, A., Bui, N., Castellani, A., Vangelista, L. and Zorzi, M. (2014) Internet of Things for Smart Cities. *IEEE Internet of Things Journal* **1**(1): 22–32.

[2] Anton-Haro, C. and Dohler, M. (2015) *Machine-to-Machine (M2M) Communications: Architecture, Performance and Applications* (Woodhead Publishing Ltd.), 1st ed.

[3] (2015) *Internet of Things' connected devices to almost triple to over 38 billion units by 2020*. Tech. rep., Juniper Research. URL http://www.juniperresearch.com/press/press-releases/iot-connected-devices-to-triple-to-38-bn-by-2020.

[4] (2015) *Ericsson mobility report. On the pulse of the networked society*. Tech. rep., Ericsson. URL http://www.ericsson.com/res/docs/2015/ericsson-mobility-report-june-2015.pdf.

[5] (2015) *The Internet of Things: mapping the value beyond the hype*. Tech. rep., McKinsey Global Institute. URL http://www.mckinsey.com/insights/business_technology/the_internet_of_things_the_value_of_digitizing_the_physical_world.

[6] Biral, A., Centenaro, M., Zanella, A., Vangelista, L. and Zorzi, M. (2015) The challenges of M2M massive access in wireless cellular networks. *Digital Communications and Networks* **1**(1): 1–19.

[7] Tan, L. and Wang, N. (2010) Future Internet: The Internet of Things. In *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, **5**: 376–380.

[8] Centenaro, M., Vangelista, L., Zanella, A. and Zorzi, M. (submitted for publication) Long-Range Communications in Unlicensed Bands: the Rising Stars in the IoT and Smart City Scenarios. *IEEE Wireless Communication* URL http://arxiv.org/abs/1510.00620.

[9] Vodafone Group Plc. (2014) *New Study Item on Cellular System Support for Ultra Low Complexity and Low Throughput Internet of Things*. Tech. rep. gp-140421, 3GPP TSG GERAN.

[10] *M2M and IoT redefined through cost effective and energy optimized connectivity*. Tech. rep., SIGFOX.

URL http://www.sigfox.com/static/media/Files/Documentation/SIGFOX_Whitepaper.pdf.
[11] Sforza, F. (2013), *Communications system*, US Patent 8,406,275.
[12] (2015), LoRaWAN™ Specification V1.0. URL https://www.lora-alliance.org/portals/0/specs/LoRaWAN%20Specification%201R0.pdf.
[13] Myers, T. (2010), *Random phase multiple access communication interface system and method*, US Patent 7,782,926.
[14] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. (1999) *Hypertext Transfer Protocol–HTTP/1.1*. Ietf rfc 2616.
[15] Fielding, R.T. (2000) *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. thesis, University of California, Irvine.
[16] (2014), OASIS: MQTT Version 3.1.1. URL http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.
[17] (2015), ISO/IEC DIS 20922 Information technology–Message Queuing Telemetry Transport (MQTT) v3.1.1. URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=69466.
[18] (2012), OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0. URL http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf.
[19] (2014), ISO/IEC 19464:2014 Information technology–Advanced Message Queuing Protocol (AMQP) v1.0 specification. URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=64955.
[20] Patierno, P. (2014) Internet of Things: protocols war! In *Betterembedded 2014* (Florence, Italy).
[21] Foster, A. (2013) *Messaging Technologies for the Industrial Internet and the Internet of Things*. Tech. rep., PrismTech.
[22] Dierks, T. and Rescorla, E. (2008) *The Transport Layer Security (TLS) Protocol Version 1.2*. Ietf rfc 5246.
[23] Fersi, G. (2015) Middleware for Internet of Things: a study. In *IEEE International Conference on Distributed Computing in Sensor Systems*.
[24] Bandyopadhyay, S., Sengputa, M., Maiti, S. and Dutta, S. (2011) Role of Middleware for Internet of Things. *International Journal of Computer Science & Engineering Survey* **2**(3): 94–105.
[25] (2012) *Electromagnetic compatibility and Radio spectrum Matters (ERM); Short Range Devices (SRD); Radio equipment to be used in the 25 MHz to 1000 MHz frequency range with power levels ranging up to 500 mW; Part 1: Technical characteristics and test methods*. Tech. Rep. EN 300 220-1 V2.4.1, ETSI.
[26] Suo, H., Wan, J., Zou, C. and Liu, J. (2012) Security in the Internet of Things: A Review. In *2012 International Conference on Computer Science and Electronics Engineering (ICCSEE)*, **3**: 648–651.
[27] Roman, R., Najera, P. and Lopez, J. (2011) Securing the Internet of Things. *IEEE Network* **44**(9): 51–58.
[28] (2012), IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 3: Physical Layer (PHY) Specifications for Low-Data-Rate, Wireless, Smart Metering Utility Networks.

Chiara Pielli received the Bachelor's degree in Information Engineering in 2013 and the Master's degree in Telecommunication Engineering in 2015, both from the University of Padova, Italy, where she is currently pursuing her Ph.D degree since October 2015. Her research interests include the Internet of Things, with particular focus on security and energy efficiency.

Daniel Zucchetto He received the Bachelor's degree in Information Engineering in 2012 and the Master's degree in Telecommunication Engineering in 2014, both from the University of Padova, Italy. Since October 2015 he is a Ph.D. student at the Department of Information Engineering of the University of Padova, Italy. His research interests include Low Power WAN technologies and next generation cellular networks (5G), with particular focus on their application to the Internet of Things.

Andrea Zanella (S'98-M'01-SM'13) He received the Laurea degree in computer engineering and Ph.D. degree in electronic and telecommunications engineering from the University of Padova, Padova, Italy, in 1998 and 2000, respectively. He was a Visiting Scholar at the Department of Computer Science, University of California, Los Angeles (UCLA), Los Angeles, CA, USA, in 2000. He is an Associate Professor with the Department of Information Engineering (DEI), University of Padova. He is associate editor of the IEEE Internet of Things Journal. His long-established research activities are in the fields of protocol design, optimization, and performance evaluation of wired and wireless networks.

Lorenzo Vangelista (S'93-M'97-SM'02) He received the Laurea and Ph.D. degrees in electrical and telecommunication engineering from the University of Padova, Padova, Italy, in 1992 and 1995, respectively. He subsequently joined the Transmission and Optical Technology Department, CSELT, Torino, Italy. From December 1996 to January 2002, he was with Telit Mobile Terminals, Trieste, Italy, and then, until May 2003, he was with Microcell A/S, Copenaghen, Denmark. In July 2006, he joined the Worldwide Organization of Infineon Technologies as Program Manager. Since October 2006, he has been an Associate Professor of Telecommunications with the Department of Information Engineering, Padova University. His research interests include signal theory, multicarrier modulation techniques, cellular networks and wireless sensors and actuators networks.

Michele Zorzi (S'89-M'95-SM'98-F'07) He received his Laurea and PhD degrees in electrical engineering from the University

of Padova in 1990 and 1994, respectively. During academic year 1992-1993 he was on leave at UCSD, working on multiple access in mobile radio networks. In 1993 he joined the faculty of the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy. After spending three years with the Center for Wireless Communications at UCSD, in 1998 he joined the School of Engineering of the University of Ferrara, Italy, where he became a professor in 2000. Since November 2003 he has been on the faculty of the Information Engineering Department at the University of Padova. His present research interests include performance evaluation in mobile communications systems, random access in wireless networks, ad hoc and sensor networks, Internet-of-Things, energy constrained communications protocols, cognitive networks, and underwater communications and networking. He was Editor-In-Chief of IEEE Wireless Communications from 2003 to 2005 and Editor-In-Chief of the IEEE Transactions on Communications from 2008 to 2011, and is the founding Editor-In-Chief of the IEEE Transactions on Cognitive Communications and Networking. He has also been an Editor for several journals and a member of the Organizing or the Technical Program Committee for many international conferences, as well as guest editor for special issues in IEEE Personal Communications, IEEE Wireless Communications, IEEE Network and IEEE Journal on Selected Areas in Communications. He is an IEEE Fellow, served as a Member-at-Large of the Board of Governors of the IEEE Communications Society from 2009 to 2011, and is currently its Director of Education and Training.