Polluino: An Efficient Cloud-based Management of IoT Devices for Air Quality Monitoring

Giovanni B. Fioccola*, Raffaele Sommese, Imma Tufano, Roberto Canonico* and Giorgio Ventre*
*Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione
Università degli Studi di Napoli "Federico II"
Napoli, Italy
Email: giovannibattista.fioccola@unina.it, raffaele.sommese@protonmail.ch, immatufano@yahoo.it,
roberto.canonico@unina.it, giorgio.ventre@unina.it

***Abstract*—The Internet of Things paradigm originates from the proliferation of intelligent devices that can sense, compute and communicate data streams in a ubiquitous information and communication network. The great amounts of data coming from these devices introduce some challenges related to the storage and processing capabilities of the information. This strengthens the novel paradigm known as Big Data. In such a complex scenario, the Cloud computing is an efficient solution for the managing of sensor data. This paper presents Polluino, a system for monitoring the air pollution via Arduino. Moreover, a Cloud-based platform that manages data coming from air quality sensors is developed.**

***Keywords*—IoT, cloud computing, air pollution monitoring.**

I. Introduction

The *Internet of Things* (IoT) is a paradigm that has recently become very attractive in the modern wireless telecommunications context. This paradigm indicates *"a world-wide network of interconnected objects uniquely addressable, based on standard communication protocols"* [1]. The fundamental idea of the IoT is the distribution of ubiquitous "objects" or "things", which collect and exchange data in order to reach a common objective by means of mutual interactions [2]. The Internet of Things makes these objects responsive, adaptive and omnipresent around our lives. The IoT is inspired by the success of the *Radio Frequency IDentification* (RFID), *Near Field Communication* (NFC), and *Sensor Network* (SN) technologies, which allow to integrate communication and information systems in the environment. Physical entities are provided with local intelligence and connectivity to the Internet, making them *smart*. Hence, a smart object is defined as a *cyber-physical system* or an *embedded system*, which consists of a physical device and a component that processes sensor data and guarantees a wireless communication to the Internet. Therefore, the vision of the Internet of Things can be built from smart objects, which are physical objects that have also sensing, processing and networking capabilities. In detail, these embedded systems are able to sense, log and understand all the interactions between themselves and the outer world. The connection of these physical objects to the Internet provides access to remote sensor data, so that it is possible to control the physical world from a distance [3]. This opens new opportunities for the *Information and Communication Technologies* (ICT) sector, by introducing services and applications that exploit recent technologies in order to create additional business opportunities. The real innovation introduced by the IoT is not related to the functional characteristics of the aforementioned smart objects; indeed, a lot of embedded systems were previously able to connect to the Internet. The novelty is the distribution of a great number of interconnected devices and the diversification of both systems and applications, thus creating pervasive computing environments. IoT solutions are capable of making a valuable contribution to several application fields and market sectors, while improving the quality of our lives. In particular, IoT technologies might play a leading role in the following areas: smart home/smart building management, smart cities, environmental monitoring, health-care, smart business/inventory and product management, security and surveillance. An IoT application area that has been gaining popularity over the last few years is the *Smart Environment* [4] [5]. In this context, IoT objects are used to sense natural phenomena and environmental conditions in a distributed manner and by integrating heterogeneous data sources, with the aim of monitoring and detecting possible anomalous conditions in the environment. The small size of IoT sensing objects allows monitoring of critical areas and environment data in real-time, preventing health problems.

To gather, store and analyze data generated by IoT devices, the availability of Cloud computing services represents the ideal complement to IoT [6]. A fundamental aspect of the Internet of Things is the integration with the Cloud infrastructure, which hosts interfaces and web-based applications that enable the communication with sensors and external systems. Therefore, the Cloud computing infrastructure might provide data access and management features, with the aim of collecting and managing data made available by smart objects. As a result, novel services are provided to users. In a *Smart Environment*, IoT objects and Cloud services are integrated, and new services are created by combining resources provided by different stakeholders. Thing-driven approaches use protocols that are specifically designed for the communication between different devices, such as MQTT, REST API over HTTP/HTTPS, WebSockets, XMPP and CoAP. IoT objects generate significant amount of information, because of the extensive coverage of sensors and the continuous sensing of data. Gartner report[1] estimated that 2.9 billion of connected things were in use in the consumer sector in 2015 and will reach over 13 billion in 2020, while by 2019 there will be 780 million of wearable devices and 2.2 billion of smartphones. As a result, the real-time data processing has become a new challenge, since one of the fundamental aspects

[1]http://www.gartner.com/newsroom/id/2905717

of the IoT is the ability to respond to a trigger in real-time. Object unique addressing, storage and representation of the exchanged data are the most problematic topics to be addressed. Indeed, massive amounts of data must be stored, processed and presented in a readable and easily interpretable way for smart monitoring. Hence, appropriate languages and standardized formats, models and metadata must be used, in order to extract suitable information from data and guarantee interoperability among several applications. In this way, IoT applications might benefit from automated reasoning, a key feature for the large-scale distribution of this novel technology. However, IoT devices have limited performance in terms of both computational resources and energy autonomy. Minimizing the energy spent for communication and computing purposes is also a primary concern, because devices are typically battery equipped [7]. New solutions must be developed in order to optimize energy usage, while maintaining *Quality of Service* (QoS) at an acceptable level. The QoS management is complicated also by the distribution of heterogeneous devices. Hence, resource efficiency, energy management, robustness and scalability problems must be taken into account when proposing IoT solutions [8].

In the *Smart Environment* context, IoT might be used to address the air pollution problem, which has both social and economical relevance. Ground-level ozone and particulate matter might cause asthma and respiratory diseases. People with asthma are more sensitive to the effects of air pollutants such as sulphur dioxide, nitrogen oxides or airborne particles. One of the main causes of air pollution is the emission of polluting gases from vehicles. Indeed, urban traffic contributes to the air quality degradation and *Greenhouse Gas* (GHG) emissions. A real time monitoring of the presence and the concentration of pollutants is necessary, in order to detect air quality status and trends. By allowing a continuous real-time monitoring of outdoor pollutant levels, IoT might help city managers and health agencies to take the most suitable and effective decisions in case the environmental conditions become not compatible with the public health.

In this paper, we present *Polluino*, a system for monitoring the air pollution based on Arduino. We have implemented a prototype of this system and have deployed it in Marigliano, Napoli, Italy. Moreover, we present a Cloud-based platform that manages data collected from gas sensors. A comparison between two Cloud computing service models and between two IoT communication protocols is performed. Finally, an efficient approach to manage the power consumption of the IoT devices is proposed, based on the aforementioned Cloud-based analysis.

II. Environmental Monitoring

Air pollution means *"the presence in the earth's atmosphere of one or more contaminants in sufficient quantity to cause short- or long-term deleterious effects to human, animal, or plant life, or the environment"* [9]. These contaminants may be classified as being either anthropogenic (i.e. resulting from human activities), or caused by natural events (such as fires, eruption), or resulting by the decomposition of organic compounds. Regardless of their origin, pollutants can be divided into:

- **Primary pollutants**, such as carbon, nitrogen, sulfur, and halogen compounds. They are released directly into the atmosphere from sources, and have high health impacts.
- **Secondary pollutants**, such as nitrogen dioxide, hydrogen peroxide, ozone, sulfate and nitrate aerosols. They are not directly emitted. These pollutants are formed by atmospheric chemical processes acting upon primary pollutants, and event other gaseous species (non pollutant) in the atmosphere.

The environmental impact of the air pollutants is variable. Some compounds have a short lifetime (a few hours or days) and they mainly impact on a local scale, that is where they were produced or released. Other compounds are released over a long period of time, allowing them to spread over large areas, by producing an effect on the whole planet. A real time monitoring of the presence and the concentration of pollutants is necessary, in order to detect air quality status and trends. Indeed, air pollution might cause acute and chronic effects on human health, ecosystems and cultural heritage.

III. Device Implementation for Air Quality Monitoring

The air pollution monitoring involves the collection of environmental data by using several external sensors. In this paper, we present Polluino, an Arduino-based sensor device that collects and transmits data collected by multiple sensor modules. In detail, the following environmental parameters are collected with the aim of measuring air pollution levels: *Carbon Monoxide* (CO), *Carbon Dioxide* (CO_2), *Nitrogen Dioxide* (NO_2), *Methane* (CH_4), *Hydrogen Sulfide* (H_2S), *Ozone* (O_3), *Ammonia* (NH_3), *Particulate Matter* (PM), *Benzene* (C_6H_6), *Ethanol* (C_2H_6O), *Toluene* (C_7H_8), *Propane* (C_3H_8). Moreover, other parameters like temperature, humidity, light intensity, presence of flames and rains are monitored. In this section, the implementation of the proposed framework is discussed. In particular, the multiple technologies involved in this work are detailed:

- Arduino is a flexible open source micro-controller that works with several communication and sensing technologies. This single-board development environment allows to read data coming from sensors and to control different devices. The Arduino board consists of an open hardware design with an ATmega2560 micro-controller. An on-board input/output support is also provided.
- ESP8266-01 is a low cost Wi-Fi module with an AT command library. It allows the micro-controller to connect to the Internet through a Wi-Fi connection. Moreover, ESP8266 has a full TCP/IP protocol stack integrated on the chip.
- Twelve gas sensors: MQ-2 (C_3H_8), MQ-3 (C_2H_6O), MQ-4 (CH_4), MQ-7 (CO), MQ-131 (O_3), MQ-135 (C_6H_6), MQ-136 (H_2S), MQ-137 (NH_3), MQ-138 (C_7H_8), MG-811 (CO_2), WSP-1110 (NO_2), Sharp GP2Y1010AU0F (PM).
- Five environmental sensors: DS18B20 Temperature, DHT11 Humidity, Photo-resistor, Flames, Raindrops.

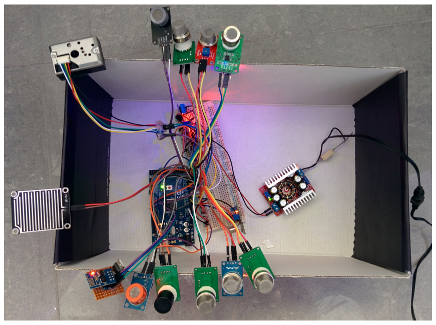


Fig. 1. Integration of Step-Down module, ESP8266-01, and Arduino board, which is equipped with air quality and environmental sensors

- Step-Down module. It is an integrated circuit that is useful for the design of a step-down switching regulator.

There are some limitations in terms of resolution. Indeed, the inputs coming from analog sensors operate by default at 10-bit resolution. This is the resolution of the *Analog-to-Digital Converter* (ADC) and provides a scale of 0 to 1023. The Step-Down DC-to-DC converter is a switch-mode power supply that has a power efficiency of 95%. This module is useful because we want to assemble a low-power device that can be exploited in several situations, both in places where there is a current source and in isolated places, where battery and solar sources are used. A mechanism is also implemented for the shutdown of the sensors when they do not produce useful data, with the aim of decreasing energy consumption. The on/off switching of the sensors is controlled remotely according to sensor-based data that are stored and managed directly in the Cloud infrastructure. The Arduino collects all the data from sensors and forwards them to the Cloud platform by using the Wi-Fi module ESP8266, which is connected to Arduino through an on-board serial port.
In this work, we consider a *Smart City* scenario. Typical characteristics of *Smart Cities* are the mobility and the variety of nodes (user mobile devices, sensors, and vehicles), which can be equipped with at least a network interface, and the widespread availability of free Wi-Fi access points [10]. Therefore, we suppose that a short-medium range communication technology like Wi-Fi is available. However, the standard Wi-Fi might not be always available and an alternative approach to transfer data must be considered. Therefore, we have also evaluated the possibility of using GSM/3G communications, but these are more expensive solutions. An experimental version of the proposed system has been implemented in practice (see Figure 1).

IV. Polluino Management Platform

In this section, we illustrate the software platform we have designed to manage a set of Polluino devices, which have been deployed for air monitoring. Preliminarily, we discuss the variety of Cloud services that can be used to deploy such a platform, namely: *Sensing as a Service*, *Platform as a Service* (PaaS), and *Infrastructure as a Service* (IaaS).

A. Cloud Computing Service Models for Sensor Management

A sensor management platform has three main actors: sensor owners, Cloud providers and data consumers. In the *Sensing as a Service* approach, the owners of the sensors are able to publish data collected from all their available devices, with the aim of getting a return on investment [11]. In details, sensors will be used over the Internet either for free or by paying a fee. Consumers are able to select sensors from which collect data within a given time period. By using this approach, companies acquire more accurate data in real-time, while avoiding the deployment of sensors and the manual collection of data from users. This model is built on top of the IoT infrastructure and services. Several servers manage sensing requests received from different users, collect data from sensors and return them to the users. The infrastructure and the front-end reporting tools are granted by the Cloud provider. Privacy and security issues must be taken into account when designing and implementing a *Sensing as a Service* solution. In the PaaS approach, the Cloud provider offers runtimes, databases, middlewares, frameworks, brokers and other services for IoT device communications. One of the most important software programs that have been used in this context is Node-RED.
In the IaaS approach, the monitoring infrastructure is built by sensor owners, who deploy sensors in the environment and create the management infrastructure by relying on Cloud provider IaaS services. Large amounts of sensor data can be stored and analyzed by using processing, storage, networks, and other fundamental computing resources.

B. A Selection of Suitable Cloud Services

An example of *Sensing as a Service* model is provided by *ThingSpeak*, which uses a REST API approach for the communication between IoT devices and the Cloud. The key element of this platform is the *channel*, which stores and retrieves data generated from sensors via REST API. A channel includes the following elements: (i) 8 fields for storing data; (ii) 3 location fields useful to store latitude, longitude and elevation; (iii) 1 status field that allows to describe data stored in the channel. The *ThingSpeak* public channel has been used to analyze and visualize collected data with the built-in "Analysis" and "Visualizations" MATLAB apps. In Figure 2, an example of relative humidity data is shown. Data has been captured by the DHT11 sensor on a Polluino device, and displayed in real-time with *ThingSpeak*.

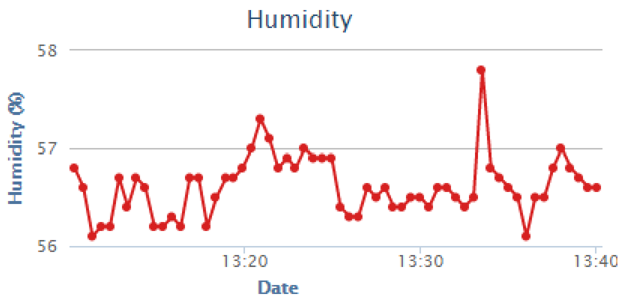


Fig. 2. Relative humidity graph

ThingSpeak allows the computation of statistical metrics, such as average, median, min, max, sum, timescale and so forth. Finally, JavaScript-based charts, read/write API key management, and a time-zone management are also provided. A commercial *Platform as a Service* solution is IBM's *Bluemix*. *Bluemix* provides a PaaS platform that supports several runtimes like Node.js, PHP, Go, and so on. As for the *Infrastructure as a Service* approach, there is a wide variety of commercial Cloud providers, such as Amazon AWS, Microsoft Azure, Google Cloud, and so on.

C. Software Platform Design

Having considered the complex processing requirements of the sensor management platform, we opted for the PaaS approach. In particular, we decided to deploy a Node-RED application on top of the Node.js runtime in Bluemix. Node-RED is a browser-based visual tool for software development that allows to wire together IoT devices. It is a generic event-processing engine that is included in the IBM "Bluemix IoT starter application". By using the PaaS approach, data collected from air quality sensors can be used to trigger automated actions. Once emergency events are detected, selected town council members will receive an email alert in a timely and reliable manner. Alert messages can be easily configured through PushingBox API. For example, a notification is automatically sent when threshold limit values for contaminant gases are exceeded.
The existing correlation between air pollutants and meteorological parameters, which will be discussed in Section V, persuaded us to evaluate the possibility of saving energy. To this end, gas sensors can be turned off during periods in which their performance and accuracy are affected by adverse weather conditions. Moreover, gas sensors are also turned off during time slots in which a lower concentration of greenhouse gases in the atmosphere has been observed. The proposed Node-RED application is able to selectively turn off sensors, while keeping the Arduino board switched on.
Data management in the Cloud infrastructure is also important. In the IoT context, the best approach for data storage and retrieval is the use of a *NoSQL* or a *Time Series* (TS) database, which is optimized for managing arrays of numbers indexed by time. All the data are provided in the *JSON* format, and can be retrieved and plotted. In our context, data collected from sensors are stored in Cloudant, a NoSQL *Database as a Service* (DBaaS) that handles a wide variety of data types, including JSON files.

V. Power-saving Strategy

In this section, we present an energy-efficient approach that allows to turn off gas sensors in a smart way, so as to reduce power consumption. A calibration procedure has been preliminarily defined in order to improve the accuracy of the air quality sensors. To this end, calibrations are performed by taking a number of readings in clean air, more specifically in Monteforte Irpino (AV), a city located in a leafy semi-rural landscape with low-density population. This procedure provides a baseline of clean air against which accurate air quality levels can be measured.
The experimental evaluation consists of three sets of trials. The first one was started on the 18th of April 2016. This set of trials focuses on a correlation analysis between greenhouse gas concentrations and daily values of some meteorological parameters (temperature, relative humidity and rainfall). All the gas and environmental sensors were used. The data was collected over one week and results are shown in Figure 3. For the sake of simplicity, only the influence of rainfall on the PM concentration is illustrated.

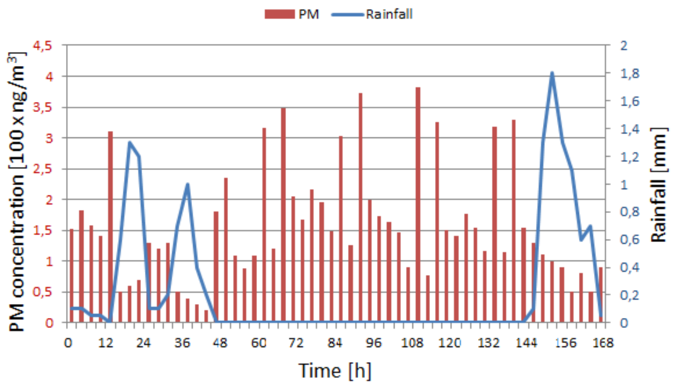


Fig. 3. Hourly variation of rainfall and PM emissions observed during a week

The influence of temperature over PM depends on the fact that atmospheric dispersion conditions mainly occur under warm air than cold air masses. Moreover, the particulate matter is efficiently scavenged by precipitations, resulting in a removal mechanism of the PM from the atmosphere. An increase in precipitation causes a decrease in PM. Finally, changes in humidity also affect PM, because increasing humidity also increases PM water content and the uptake of semi-volatile components.
The second set of trials, started on the 25th of April 2016, involved the use of gas sensors only. The data was collected over one week, so as to analyze the daily concentration trend of air pollutants. Figure 4 shows the hourly variation of CO, O_3, NO_2, and PM emissions observed during that week.

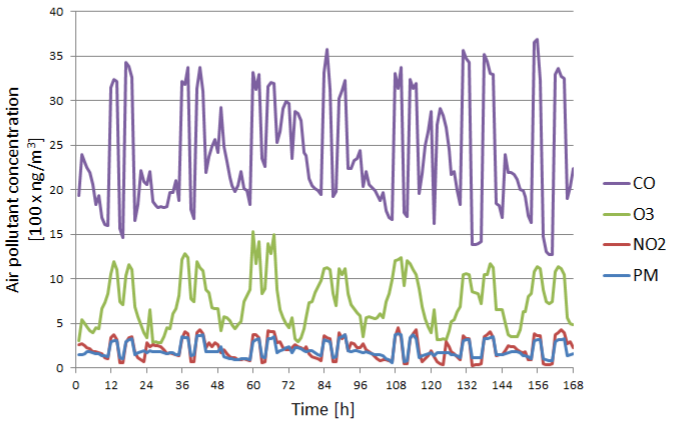


Fig. 4. Hourly variation of CO, O_3, NO_2, and PM emissions observed during a week

The main purpose of the proposed system is to alert town council members when the air quality values exceed the threshold range. Therefore, gas sensors are turned on during time slots in which, according to data shown in Figure 4, a higher concentration of greenhouse gases in the atmosphere

has been observed. Hence, our strategy consists in waking up the gas sensors only for the time needed to acquire a new set of samples, and turning them off afterwards. A MOSFET IRL540 is used for each sensor in order to turn it on/off. The MOSFET acts as a switch that can disconnect and reconnect circuits when needed, controlled via the digital output pins of the Arduino board. These actions are triggered based on data being processed by the IoT Cloud platform. In this way, power consumption is optimally managed through a periodic sensing approach, and the number of samples (hence the amount of data to be processed) is also reduced. A better approach would require an adaptive sensing strategy, in order to dynamically adapt the on/off of the sensors to the dynamics of the environment. But each sensor is characterized by a wake-up latency, which is *"the time required by the sensor to generate a correct value once activated"* [12]. Gas sensors used in this research work are simple and cost effective, requiring more than 12 hours before reading data. The collected data is not significant if the reading is performed before the wake-up latency time, hence these sensors are not suited for power-cycling. Therefore, we are not able to follow a more efficient approach, which for example involves the activation of the sensors whenever fixed threshold limit values for contaminant gases are exceeded. In the marketplace, more technologically advanced gas sensors are available. They require a shorter warm-up time, but this makes them even more expensive. Instead, our system consists of devices of much lower cost (the *Polluino* project costs 150 euros), and this also enables future large-scale distributions.
The third set of trials was started on the 2th May 2016. The city of Marigliano (NA) was selected to collect data. All the gas and environmental sensors were used. The data was collected over one month. By taking into account our previous remarks and Figure 4, sensors have been powered on during time slots shown in Table I.

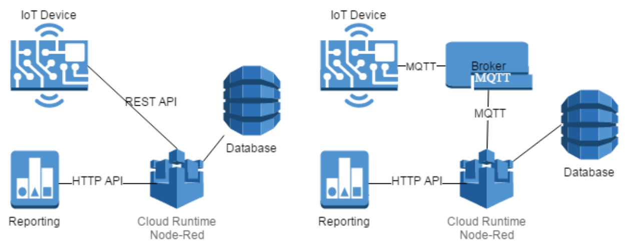
TABLE I. GAS SENSOR ACTIVITY PLANNING

	Mon.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.
Week 1	12-14			17-19		18-21	11-13
Week 2		12-14			17-19	18-21	11-13
Week 3	12-14			17-19		18-21	11-13
Week 4		12-14			17-19	18-21	11-13

When all the sensors are on, the average power consumption of the described system is 9.8 W, distributed as follows: Arduino/ESP 1.5 W, gas sensors 8 W, environmental sensors 0.3 W. This consumption has been computed by using a PC-based wattmeter, plugged in for 24 hours to get an accurate reading. The appliance's monthly electricity consumption in constant use is 6.6 kWh, while the monthly electricity consumption by considering our approach is 3.5 kWh. Our approach has led to savings of 47%. Further savings have been obtained by turning off the gas sensors during heavy rainy days, according to the previous considerations, for a total of 22 hours. As a result, additional savings of 0.20 kWh have been reached.

VI. A Comparison of IoT Communication Protocols

In the PaaS model, several protocols might be used in order to establish connections and communications among different IoT subsystems. In this context, a comparison between REST API (see Figure 5a) and MQTT (see Figure 5b) is performed.



(a) REST API (b) MQTT

Fig. 5. Platform as a Service approach based on REST API and MQTT

The *Constrained Application Protocol* (CoAP), which is a lightweight RESTful protocol based on UDP that inherits the same client/server paradigm adopted in HTTP [13], has not been considered in this work. The reason is that ESPDUINO, which is an ESP8266 wrapped in an Arduino board, contemplates only a native MQTT/RESTful client library for ESP8266, while a CoAP implementation has not been developed yet. REST is an approach that leverages the HTTP protocol, by using standardized methods (e.g. GET, PUT, POST, DELETE, etc.) to deal with resources. In regard to the *Message Queuing Telemetry Transport* (MQTT) protocol, all the Internet of Things objects connect as clients to the MQTT broker on the Cloud. This broker dispatches the exchanged messages between the different devices. MQTT is a lightweight publish/subscribe messaging protocol for *Wireless Sensor Networks* (WSN). This protocol is designed for easily connecting IoT objects to the Internet in a *Machine-to-Machine* (M2M) scenario. MQTT is able to support low-bandwidth, high-latency or unreliable network environments. Therefore, it is an optimal solution for mobile applications, because exchanged packets are characterized by small size headers and low-power usage. Clients subscribe to be notified of incoming messages that belong to specific topics, and other clients publish on those topics. A topic is an UTF-8 string, which is used by the broker to filter messages for each connected client. MQTT is efficient and event-driven, indeed a client constantly receives all the messages sent to that topic by other clients. Instead, in the REST API approach a continuous polling is needed every-time a consumer entity wants to check for status updates. MQTT is used in the Facebook Messenger application for status updates, with the aim of saving the battery life of devices. MQTT is characterized by three levels of QoS:

- At most once (0): this is the minimal QoS level and it provides a best effort delivery. A message is sent to the receiver only once, and there is not acknowledgment.
- At least once (1): a message is sent more than once, until the sender gets an acknowledgement.
- Exactly once (2): a message is received only once, and whenever a packet gets lost, the sender is responsible for resending the last message.

In our scenario, the QoS values 0 and 1 are acceptable, while in a domotic scenario a QoS 2 value is more advisable in order to control specific loads.

A. Performance Evaluation

In this section, a comparison between MQTT and REST API protocols is presented. The evaluation of each solution is performed in terms of latency, defined as the time elapsed from the moment the client sends a request until the moment it receives the response. As expected, MQTT guarantees the best performance, because of the latency introduced by the three-way-handshake used by TCP to establish and close the HTTP connection. Instead, MQTT maintains session state information (see Figure 6). The tests have been repeated with a mobile connection 3G, and the same differences between MQTT and REST-API protocols were revealed.

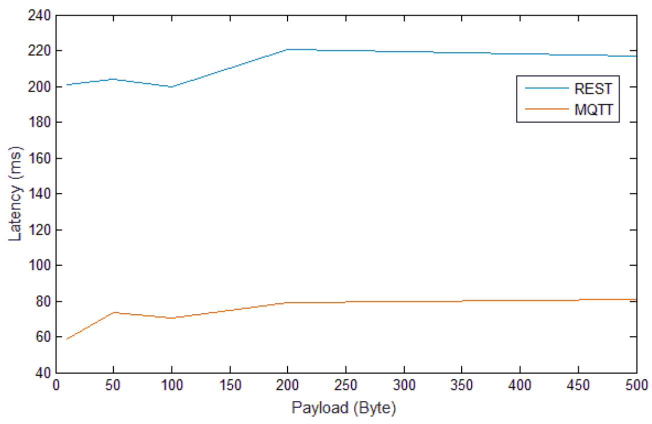


Fig. 6. Latency results according to the payload size

Finally, protocol overhead has been evaluated. The overhead is caused by packet headers, both at the application layer and transport layer. In the REST approach, the handshake packets have been considered too. Results in Figure 7 show that the REST architecture has a high overhead, which affect the performance and the cost of a mobile-network based IoT solution.

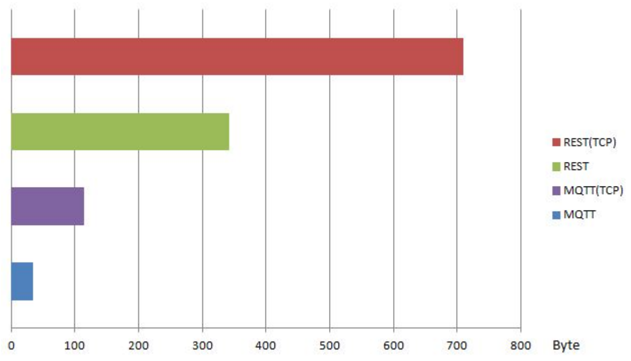


Fig. 7. Protocol overhead analysis

Ultimately, the MQTT approach is recommended when there are latency constraints and expensive data plans for IoT applications, which have an individual monthly fixed data allowance.

VII. Conclusion

In this paper, a system for monitoring the air pollution via Arduino is presented. Based on IoT principles, a communication of the air quality sensors with the Cloud platform is established. A comparison between two Cloud computing service models and between two IoT communication protocols is performed. Finally, an efficient approach to manage the power consumption of the IoT devices is proposed. Our approach has led to power savings of 47%. According to the results achieved and the ease of implementation, the prototype of Polluino is up-and-coming in terms of both costs and performance. The proposed approach might be used to find an optimal way of controlling traffic lights, in order to improve traffic while minimizing air pollution levels. Moreover, our future work will target the comparison between MQTT and CoAP protocols, and the use of a battery source instead of a permanent power supply.

References

[1] D. Bandyopadhyay and J. Sen, "Internet of Things: Applications and Challenges in Technology and Standardization," *Wirel. Pers. Commun.*, vol. 58, no. 1, pp. 49–69, May 2011.

[2] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A Survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, October 2010.

[3] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Boston, MA: Springer US, 2011, ch. Internet of Things, pp. 307–323.

[4] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, and T. Razafindralambo, "A Survey on Facilities for Experimental Internet of Things Research," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 58–67, November 2011.

[5] X. Li, R. Lu, X. Liang, X. Shen, J. Chen, and X. Lin, "Smart community: an internet of things application," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 68–75, November 2011.

[6] A. Botta, W. de Donato, V. Persico, and A. Pescapé, "On the integration of cloud computing and internet of things," in *Proceedings of the 2014 International Conference on Future Internet of Things and Cloud*, ser. FICLOUD '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 23–30.

[7] A. Biral, M. Centenaro, A. Zanella, L. Vangelista, and M. Zorzi, "The Challenges of M2M Massive Access in Wireless Cellular Networks," *Digital Communications and Networks*, vol. 1, no. 1, pp. 1 – 19, 2015.

[8] J. Kim, J. Lee, J. Kim, and J. Yun, "M2M Service Platforms: Survey, Issues, and Enabling Technologies," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 61–76, January 2014.

[9] D. E. Painter, *Air Pollution Technology*. Reston Publishing Company Inc., 1974.

[10] A. Morelli, C. Stefanelli, M. Tortonesi, and N. Suri, "Mobility Pattern Prediction to Support Opportunistic Networking in Smart Cities," in *Proceedings of the 2013 International Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, ser. MOBILWARE '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 166–175.

[11] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing As a Service Model for Smart Cities Supported by Internet of Things," *Trans. Emerg. Telecommun. Technol.*, vol. 25, no. 1, pp. 81–93, January 2014.

[12] C. Alippi, G. Anastasi, M. D. Francesco, and M. Roveri, "Energy management in wireless sensor networks with energy-hungry sensors," *IEEE Instrumentation Measurement Magazine*, vol. 12, no. 2, pp. 16–23, April 2009.

[13] G. Tanganelli, C. Vallati, and E. Mingozzi, "Coapthon: Easy development of coap-based iot applications with python," in *Proceedings of the IEEE 2nd World Forum on Internet of Things (WF-IoT)*, December 2015, pp. 63–68.