REST Assured, We Manage your Microgrid

Alessandro Montanari, Yvonne-Anne Pignolet, Ettore Ferranti
ABB Corporate Research
Segelhofstrasse 1K, 5405 Baden, Switzerland
name.surname@ch.abb.com

***Abstract*—A Site Management System (SMS), controlling and coordinating domestic, commercial and industrial sites has an inherently dynamic computer network. Devices can enter and leave the network at any moment because of malfunctions, mobility or new components. Moreover, the size of the network continuously increases, thanks to adding networking capabilities to new ranges of appliances, from electrical plugs to PV panels. In these environments the burden of administration quickly exceeds any potential benefit if the devices require explicit configuration to work together. Furthermore, embedded systems used in these scenarios have severe requirements in terms of costs and size, offering limited resources to applications running on them.**

This paper presents a SMS platform based on IPv6 protocols and RESTful web services that requires no human intervention for the configuration and simplifies the addition of new nodes. The platform uses inexpensive devices with low power consumption and constrained resources. It lets SMSs discover and interact with new devices automatically, by leveraging efficient protocols like mDNS/DNS-SD and the IETF Constrained Application Protocol. From the communication perspective, the platform offers robustness and versatility. The use of an extended version of the RPL routing protocol, specifically tailored for sensor networks, permits to combine on the same device more than one communication technology, i.e. IEEE 802.15.4 and Ethernet. Thus, the platform can be applied in a large variety of settings where an adaptive and reliable communication infrastructure is needed. To evaluate its performance, the platform has been integrated in a real SMS.

I. Introduction

In modern buildings, industrial and commercial sites it is common to employ Site Management Systems (SMS) with the aim of ensuring the operational performance and energy efficiency of the facility as well as the comfort and safety of building occupants. One of the driver of this evolution is represented by a recent trend in the energy sector, where local sites are encouraged to build their own microgrids and coordinate building automation with energy production and consumption. SMS often grow organically, hence they need to be able to integrate large numbers of heterogeneous devices, based on both legacy and novel tools and systems. Currently, many standards are used for both control (e.g., KNX, BACNet) and communication (e.g., WiFi, Ethernet). Although these are quite widespread, they are limited in terms of interoperability and flexibility.

From a flexibility point of view, such systems often require technical personnel for installation, calibration and configuration, and are not designed to be modified frequently to adapt to changes in the site, or to changes in the management rules. Therefore, the users are discouraged from changing the settings of the system or from adding or updating devices, by lack of technical knowledge and by high costs.

Usually different systems are most suited for different aspects of building automation and microgrid tasks, but since these systems are usually not designed to interoperate, at the end only one of them is chosen. Often custom protocols are used that limit the integration of different technologies. By contrast, dealing with various needs require complementary technologies. To accomplish this using established SMS, large investments are necessary (e.g., to acquire all the gateways and bridges and to manage and train the needed personnel). We envision that in the future these systems will not only be used to interconnect appliances in a single site but also between different sites and cities. In this scenario the mentioned problems become even more significant.

Clearly, the above mentioned observations limit the work of a system designer that has to choose only one technology and can not take advantage of an heterogeneous mix of them. Similarly, the final user is constrained in buying devices that support exactly the technology used in his building and can not reconfigure the system as needed without calling an expert. In other words, a system that makes adding new devices as easy as possible and still offers the flexibility to tailor the system to its specific use is needed.

To tackle these problems, we propose a zero-configuration sensor network platform for building automation based on standard IPv6 protocols. The platform is designed so as to let a node in the network advertise what it can do (e.g. which sensors and actuators it can use) and expose a standard interface to interact with it by making use of RESTful Web Services. This enables the system to recognise new devices when they are turned on automatically, and secondly, to use them without any manual intervention or configuration. In order to keep the investment needed to deploy the proposed platform as low as possible, it has been designed targeting small and inexpensive devices with limited computational power and resources. This decision opened challenges regarding the choice of the most efficient protocols and mechanisms and how to fit them in the limited resources available. Moreover, the platform has been integrated and tested in a real SMS in order to assess its suitability.

The rest of the paper is organized as follows: in Section II we point to related works and compare them with our approach, in Section III we introduce the SMS which has been used for the integration and validation of our platform. In Section IV we introduce our solution for a zero-configuration sensor network. In Section V we present the results obtained during the testing phase before concluding the paper in Section VI.

II. Related Work

In the literature, different approaches have been proposed for auto-configuring sensor networks. Since one of the main objectives of this work is the adherence to Internet standards, solutions based on custom protocols or non-IP networks are omitted intentionally.

978-1-4799-4934-2/14/$31.00 ©2014 IEEE

Edwards describes that there are many discovery systems in use with characteristics appropriate for different needs but not necessarily all of them are suitable for constrained nodes [12]. In fact, some of them were designed with the intention to be used in enterprise environments and the exposed features reflect this choice: managed service directories running in centralized servers, powerful search mechanisms and robust security. Two examples are Apache River [1] (before known as Jini) and the Service Location Protocol [14]. Obviously these systems do not fit the requirements for a low-power sensors.

The current approaches adopted in constrained networks are oriented towards the use of two main technologies: Universal Plug and Play (UPnP [4]) and a combination of the protocols multicast DNS (mDNS [9]) and DNS Service Discovery (DNS-SD [8]). Mitsugi et al. propose a bridging system between pure UPnP devices and sensor nodes because it is not feasible to accommodate the protocol in each sensor node [21]. The authors describe the use of an extended version of the CoAP protocol inside the sensor network and a gateway that translates the CoAP messages into UPnP commands. Jara et al. propose an adaption protocol to connect together IPv6 and non-IP networks [18]. They employ DNS-SD in the gateway between the two networks to store information about the sensors, their properties and functionality in a service-oriented way. These approaches are substantially different from the work presented in this paper where no external gateway is used to relay discovery messages from and to the sensor network. By giving the devices the ability to use the discovery protocol directly it is possible to increase the flexibility of the system and achieve a real zero-configuration architecture. Klauck et al. present an implementation of the two protocols for the Contiki Operating System [19]. Schor et al. present another example of the use of mDNS and DNS-SD in sensor networks [24]. In this case, the two protocols are used directly inside the constrained devices to periodically advertise the presence of a RESTful interface on the device. This approach is similar to the one presented here, but only periodically advertises services and is not able to answer queries. Sending periodical messages could lead to congestion problems inside the network and is not efficient in general because many messages will be sent even if they are not strictly necessary.

After the discovery of the nodes and services, another important aspect for a zero-configuration network is the possibility for the devices to interact with each other spontaneously. Therefore an homogeneous way of interaction is needed. Nowadays the trend focuses on web technologies, in particular on RESTful web services. The simplicity and lightweight properties of RESTful web services in combination with the CoAP protocol, specifically designed as a replacement for HTTP in constrained devices, make them perfect for sensor networks. Guinard et al. present two demonstrator applications built on top of RESTful sensor networks: one to monitor and control the energy consumption in houses and the second to combine the usage of sensor and actuator devices using a web interface [13]. Castellani et al. propose a real world implementation of a RESTful sensor network deployed across several university buildings [7]. Schor et al. developed a web application to monitor and control a prototypical sensor network based on RESTful web services, mDNS and DNS-SD [24]. The aforementioned systems all use RESTful principles but they do not use CoAP as application protocol. In contrast, this paper presents a RESTful sensor network based on CoAP able to use several communication media.

Colitti et al. compare HTTP and CoAP in terms of power consumption and demonstrates the better performance of the CoAP protocol [10]. Moreover, the authors present an end-to-end IP based architecture integrating a CoAP sensor network with an HTTP based application. Kovatsch et al. envision an architecture where the devices expose only their elementary functionalities using RESTful web services over CoAP and the actual application logic is running on external application servers [20]. Ishaq et al. describe an architecture that simplifies the deployment of sensor networks by making use of RESTful Web Services and the CoAP protocol [17]. The work presented here, similarly to the three previously cited, uses CoAP as application protocol but it also employs mDNS and DNS-SD protocols for the device discovery phase.

III. Background

A step towards simplified Site Management Systems is represented by the Smart Grid Demo Lab [5], [11], [27] which has been developed at the ABB Corporate Research Center in Baden-Dättwil, Switzerland. It connects devices of many different vendors and enables interactions between them. Yet, an analysis of this and other such systems revealed the following problems:

- The offered API does not provide a uniform method to read or write a specific property of a device.
- There is no way to maintain semantic information regarding devices and their properties. For example it is not possible to know which devices are active and what their properties are, which operations can be performed, which physical objects they represent or control, or where in the building they are.
- The use of new devices involves the modification of a configuration file and the restart of the relative agent, a task that requires human intervention. Moreover, the format used for the configuration file is not uniform across different device families, for example, a CSV file is used for the KNX devices and a custom text file for Smart Meters and Plugwise.

Regarding the first issue, it is impossible to easily create a system that automatically uses devices without a clear and uniform interface to interact with them. The second problem is related to the management of the devices. In fact, a system can adopt different approaches to control heterogeneous devices. For example if a device is able to accept only integer values, the system should know this and convert values accordingly before sending them to the device. To accomplish this in an automatic manner, all information need to be well structured and the system has to know where to get this information for each device. Moreover, as mentioned in the third item above, the system must be able to detect a new device, add its information to the collection of the available devices and use it without experiencing downtimes.

Hence, in addition to the development of a zero-configuration building automation platform, one contribution of this paper is the extension of the Smart Grid Demo Lab architecture in order to tackle the aforementioned problems.

IV. Zero-Configuration Sensor Network

The main contribution of this paper is the design and implementation of a building automation platform that does not require any human intervention, in the sense that the

platform is able to include new devices without requiring manual configuration of them.

The main objective during the design of the zero-configuration platform is to build an architecture that permits to discover and use new devices directly without the need of intermediate components like, bridges or gateways. This choice is fundamental to keep the complexity of the architecture low and to increase the flexibility. In fact, the use of a gateway to relay messages from the sensor network to a standard network and vice versa involves several problems. First of all, a standard gateway that can be used directly may not be available therefore it is necessary to develop it from scratch. On the one hand, this limits the flexibility of the resulting system because it would enforce to use that specific gateway in order to interact with the sensor platform. On the other hand, it introduces issues regarding the development, implementation and maintenance of the gateway that has to be done in-house. Even if the gateway for the chosen technologies was available, its use increases the complexity of the resulting platform, because for each installation of the platform, issues regarding the setup, the configuration and the availability of the gateway must be taken into consideration.

A key decision in order to simplify the design is to divide the overall discovery process into two phases. During the first phase, the devices available in the network have to be discovered and the necessary information to communicate with them (IP address and port number) have to be retrieved. The second phase has to handle the discovery of the resources (sensors and actuators) hosted by the device along with all the information needed for their usage. The following sections describe the design of the two phases.

A. Node Discovery

The first step towards a zero-configuration sensor network involves the ability to discover new devices in an automatic and standard fashion as soon as they are available. As stated in the previous sections, clearly, a solution based on custom protocols would not be acceptable, therefore only the available standard protocols for IP networks have been taken into consideration. As described in Section II, there are many systems able to discover new nodes inside a network, but not all of them are suitable for constrained devices. Therefore, the focus has been directed towards mDNS/DNS-SD [8], [9], the most used in the literature in the sensor networks field.

The mDNS and DNS-SD protocols have several benefits for constrained devices. First of all, the protocols are simple, which facilitates their implementation to be adapted to memory-constrained microcontrollers. The protocol message overhead is small, which makes the protocols suitable for low-power radio links. The drawbacks are that the protocols require link-local multicast for all queries and that the architecture does not provide any caching agent. Thus the scalability of the architecture can be a problem in large-scale networks. Despite these problems, the use of mDNS and DNS-SD protocols has been evaluated to be the best solution available, because of their efficiency and ease of implementation. In fact, Klauck and Kirsche [19] describe an implementation of the two protocols for Contiki which we used as a starting point, reducing the development time.

Thus every node in the LoWPAN network was equipped with an implementation of these protocols in order to be able to publish a service used to advertise the IP address and the port

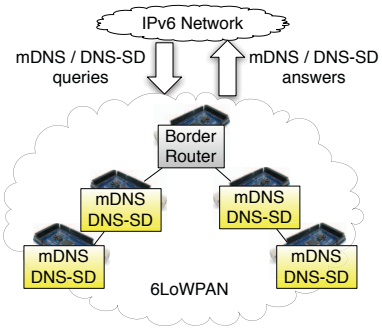

Fig. 1: Direction of mDNS/DNS-SD queries and answers.

number necessary to interact with the web service interface available on that node. This way the constrained devices can be discovered from any IPv6 network using standard protocols (Figure 1). More precisely, a service is configured on each node with the `Service Type` defined to be `_coap._udp` in order to specify both the transport layer protocol and the application layer protocol. The `Instance Name` on the other hand is not defined to a specific value, thus it can be a descriptive name of the device. An example of a complete service name is:

```
Office1Blinds._coap._udp.local.
```

This way a host on the standard IPv6 network can browse the LoWPAN network for the service type `_coap._udp` and obtain a list of devices that offer this service. Using this list, the host can then interact with the devices using the CoAP protocol and the RESTful interface. Since uIPv6 does not provide a way to forward multicast packets by default, we decided to use the protocol described in [15] which applies a Trickle algorithm to forward multicast messages without the need for topology maintenance. The implementation of the protocol used in this paper is based on [3] (some modifications were necessary to work properly with multiple network interfaces).

B. Physical Resource Discovery

After a device is discovered, the second phase of the discovery process is to understand what the device can do (i.e. which sensors and actuators are available) and how to interact with it. In other words a homogeneous way of interaction is needed.

Since one of the requirements is to facilitate the interoperability with existing systems, the design was directed towards web services. In fact, given the prevalence of the Web and its associated technologies, web services have seen a tremendous adoption in the general purpose IT world in the past couple of years. All major programming languages provide libraries tailored to build web-service-oriented applications. By using web service technology for embedded applications, existing web-service-oriented systems, programming libraries, and knowledge can be directly applied. This makes it possible to integrate embedded applications into enterprise systems without any intermediaries, thus reducing the complexity of the system as a whole. For industries, embedded applications can be built using off-the-shelf technology without any customized interfaces or translators.

In particular, nowadays the trend focuses on RESTful web services. RESTful web services have many advantages over *arbitrary web services* [6] (i.e., SOAP), such as less overhead, less parsing complexity, statelessness and therefore perform

better on wireless sensor nodes with limited resources. Hence, sensors and actuators are exposed as RESTful resources.

In order to keep the RESTful web service implementation efficient, the CoAP protocol is used at the application layer. CoAP has been specifically designed as a replacement for HTTP on constrained devices. Moreover, three features of CoAP are fundamental for the realization of a zero-configuration sensor network: reliable communication over UDP, block-wise transfers and push notifications. In a sensor network composed of constrained devices, messages can be lost easily and this can cause lower performances. CoAP provides reliability even without the overhead of TCP because *confirmable* messages are retransmitted until the recipient sends an acknowledgment or the maximum number of retransmissions is reached. The block-wise transfer functionality is useful to fragment a piece of data that can not be sent in a single packet. This functionality is already present in 6LoWPAN, but in CoAP it is combined with the retransmission mechanism to create a reliable communication. At the last, the publish/subscribe mechanism implemented in CoAP allows to reverse the classical polling scheme where the hosts periodically send queries to the sensor nodes. In fact, using CoAP, a host can subscribe to a particular resource of a certain device and the device will send new available data to all subscribers, avoiding the transmission of unnecessary packets.

C. Mapping Physical Resources on RESTful Resources

To use the physical resources (sensors and actuators) available on a device they have to be mapped on RESTful resources in order to let a host read or change their values using CoAP methods (an example of a RESTful API is depicted Fig. 2). To simplify the development of the hosts that have to interact with the devices, a few rules on how to structure the resources have been defined.

The root resource of the API has to represent the device. All other resources that represent sensors or actuators of a device are placed in its sub-tree. For each sensor/actuator on the device there should be a resource representing it. These resources are the actual datapoints used to interact with the device. The interaction with the resources is implemented using the CoAP methods `GET` and `PUT`, to read and write a new value respectively. The new value that has to be written is specified through the URL parameter "val" (i.e. `/device1/actuators/lights?val=1`). The data exchanged for sensor/actuator resources is in plain text and includes only the requested values (i.e the value "25.3" for a temperature sensor). This data type has been chosen for two reasons: (1) to reduce the overhead of the transmission and send only the strictly necessary bytes and (2) it is not necessary to send other information because all the information needed to interpret correctly the value are already included in the description of the resource as described below. In the resource hierarchy there could be generic resources that do not have a specific meaning but are used to structure the API in a more user-friendly way (e.g. "sensors" and "actuators" in Figure 2).

Moreover a predefined resource called "/newdata" has been included in the RESTful API. It is the only observable resource and returns automatically new data when available to the subscribers of the resource. The memory constraints permit only one observable resource therefore the new data from different sensors have to be combined and sent together. The data pushed to the subscribers is

```
/device1
    /sensors
        /motion
        /brightness
    /actuators
        /blinds
        /lights
/newdata
```

Fig. 2: Example of REST resources.

a JSON object which contains the URIs of all resources that have a new value and their respective values (i.e. `{"/device1/sensors/brightness":150.1}`).

In order to use the resources in an automatic fashion, a host needs other information apart from the URI of the resource, including: which operations can be performed on the resource (read or write a value), the data type of the value returned by the resource (integer or floating point), the minimum and maximum values, ... This way, every system able to use RESTful web services can read the available resources and their description and use them without any previous configuration, because the description already provides the necessary configuration information. To include this information in the description of the resources the Constrained RESTful Environments Link Format [25] has been used because it is a standardized way of associating information to a URI. Following this format, it is possible to specify a list of attributes in the form `<attribute-name> = <attribute-value>` for each resource. Another strength of this format is its extensibility, in fact to add more information for each resource it is enough just to add more attributes and this does not break in any way the previous data or how the system process it.

The URI `/.well-known/core` is used to discover available resources [25]. A CoAP `GET` operation on that resource returns a collection of URIs that represents the available resources along with their attributes. The presence of this URI in the API provided by the devices is fundamental because it provides a standard "hook" that can be used by a host on the standard IPv6 network to discover all available resources.

D. Hardware and Communication

From a hardware point of view the platform consists of nodes based on Atmel AVR ATmega2560 microcontrollers and in particular the Arduino [2] Mega 2560 platform has been used during the prototyping phase. AVR microcontrollers are inexpensive components that contain all the necessary to build a sensing or actuating node: processor core, memory and input/output peripherals. The Arduino platform has been selected because its modular nature can be exploited to adapt it to the particular needs of the project. The hardware platform chosen has severely limited resources (256KB of Flash Memory and 8KB of SRAM) and this posed important challenges during the implementation of the platform.

Concerning the communication capabilities of the devices, the fundamental decision to simplify the interoperability is to base the solution on standard IPv6 protocols. Therefore, the Arduino boards were equipped with the IPv6 stack of the Contiki operating system (uIPv6), more precisely with a version supporting multiple interfaces [22], [23]. This enables the creation of a more reliable network where the failure of a communication link does not prevent the system to continue to work and where the performance is improved by the networking stack choosing the best link in a given situation.

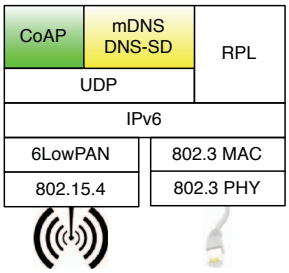

Fig. 3: Network Stack used on the Arduino boards.

The IPv6 protocol suite represents the next standard for the Internet communication systems and hence guarantees a high level of interoperability with next generation systems. However, since IPv6 has not been designed to operate on severely resource-constrained devices, specific mechanisms and protocols have been chosen in order to improve its performances on the available devices. In particular the protocol RPL [26] (Routing Protocol for Low-Power and Lossy Networks) has been employed because it enables the automatic creation of a self-healing network and it is specifically designed for constrained devices. Its use enables the creation of a network where the nodes can enter and leave it at any time and simplifies significantly its administration (e.g. no need for manual routes configuration). The use of this routing protocol enables the platform to adapt to many environments where complex multi-hop topologies are required. Another mechanism employed to improve the performances is 6LoWPAN [16]. This layer is used to optimize the transmission of IPv6 packets over the radio link. It permits to compress and fragment the packets to ensure that they fit into 802.15.4 frames (127 bytes). In addition, in order to reduce the overhead at the transport layer and to reduce the memory consumption, the UDP transport protocol is used for all communication.

Summarizing the network stack used on the Arduino nodes is depicted in Figure 3. At the physical layer two interfaces are used: the first one, with a low power radio module (XBee 802.15.4, Digi International), to communicate wirelessly and the second one with Ethernet, suitable for situations where a more robust communication medium is necessary. For the radio interface the stack is configured with 6LoWPAN enabling its fragmentation capability, in order to send packets exceeding the maximum allowed payload for an 802.15.4 frame. In the upper layers the stack includes the RPL protocol, the UDP protocol, the CoAP protocol, the mDNS and DNS-SD protocols.

V. Validation

In multi-hop sensor networks an important evaluation aspect is the delay introduced by each hop. Usually these networks are composed of constrained devices and if the transmissions are too slow, the network will not be of much use. In order to evaluate the relation between the number of hops and the delay time, we measured the following:

- Time interval between the connection of a node to the sensor network and the moment when it is discovered by the Smart Grid Demo Lab .
- Execution time of the `GET` and `PUT` CoAP methods.

The first measurement is useful to evaluate the implementation of the mDNS/DNS-SD protocols and how the multicast packets are forwarded in the network. Moreover, it tests the block-wise functionality of CoAP. When a new device is discovered, the resource discovery phase is started. At this point, the new device has to send around 500 bytes in total. The test is repeated 20 times for each node.

The execution time of `GET` and `PUT` operations measures the delay introduced by the nodes and the reliability of the network because by counting how many messages are lost. One hundred get/put requests have been sent to each device, one every 30 seconds. The test is repeated 10 times in order to compute the average of the results.

There are five nodes involved in the measurements plus a border router node: it forwards messages from the sensor network to the Smart Grid Demo Lab and vice versa. The topology used for the test is depicted in Figure 4. We deployed a chain multi-hop network where the number of hops between each node and the border router is well defined and static. The nodes communicate with one another using their radio module. In addition the node pairs (1,2) and (3,4) are connected using an Ethernet cable, creating a heterogeneous network (radio and Ethernet). A MAC layer address filter is enabled to ensure that the nodes communicate wirelessly only with the intended neighbors and not with every node in the network.

The average time and standard deviation to discover a new device and its resources, after it is turned on, are depicted in Figure 5. The second time refers to the moment the device is discovered and not to the moment it is started. E.g., on average Node 1 is discovered 11 seconds after it was started and its resources are discovered after 26 seconds, therefore a total time of around 37 seconds is needed to process a new device. From the two plots it is possible to notice that the total time needed to discover a new device and its resources is around two minutes for the first four nodes and it is more than four minutes for the fifth node. The significant difference between the time needed to discover the fifth node compared to the time needed for the previous ones is due to the nature and implementation of the Trickle algorithm. This algorithm floods the network in a controlled fashion and, especially in the linear topology, this leads to a busy network that contributes in delaying the DNS-SD multicast messages. However, since the objective of this work is to build a sensor network for monitoring and control sites, the fact that the Node 5 becomes usable after four minutes is acceptable.

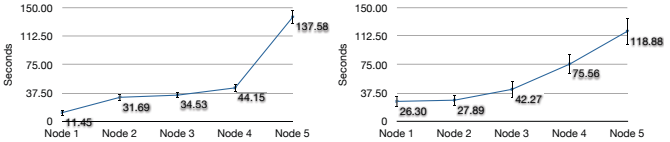

Fig. 5: Average Device and Resource Discovery Time.

The average round trip times and the number of lost messages for the `GET` and `PUT` operations are reported in Figure 6. The results show that the system is very robust as the percentage of lost messages is around 2-3%. It should also be considered that the tests performed here represent the worst case scenario where messages are sent repeatedly to a specific node, in a real environment this will most likely not be the case. Regarding the average round trip times, the values are almost the same for the first four nodes, but then the time for the Node 5 increases significantly. These values were expected because the packets sent to a specific node have to pass through all the other nodes before reaching the actual destination and then come back to reach the standard PC. Therefore, it it is

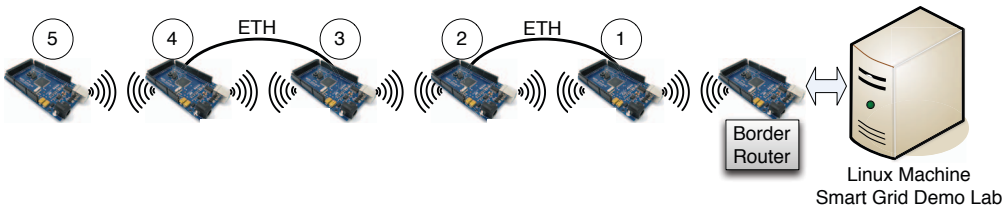

Fig. 4: Setup for the integration testing of the Zero-Configuration sensor network in the Smart Grid Demo Lab.

expected that for nodes further away from the border router the round trip times would be longer. The response time is in the order of seconds or tens of seconds (around 12 for Node 5) and this result is satisfactory, because the applications envisioned for this kind of platform, monitor and control small sites, do not require real-time responses or responses in the order of milliseconds. In fact, it is not expected to employ this platform in critical environments where important delays could lead to physical harm or financial losses. Moreover, the devices will push new information automatically through the observable resource (`newdata`), therefore there is no need to send frequent `GET` operations in order to know the status of the devices, they will communicate any changeautonomously.

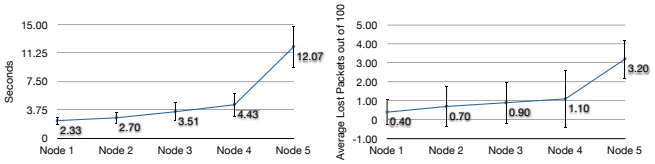

Fig. 6: Average RTT and packet loss - get/put Operations.

VI. Conclusions and Future Work

In this paper we presented a zero-configuration sensor network platform. The platform is built on top of Arduino boards that have been equipped with two different communication interfaces, one based on XBee radio modules and the other one based on Ethernet. The zero-configuration capabilities have been added to the boards making use of standard IPv6 protocols and RESTful web services allowing the discovery and the use of devices without any human intervention. Moreover, the new platform has been integrated into the Smart Grid Demo Lab project being executed at the ABB Corporate Research Center extending its ground for smart grid development.

With a functional validation it was shown that the platform works reliably. Moreover, its integration in the Smart Grid Demo Lab project proved that the design based on standard protocols represents a good solution to simplify the interoperability between different technologies. The results presented in the previous section show that the use of the platform for building automation tasks is feasible but it could not be appropriate for other kind of applications with more strict constraints regarding the response time of the devices.

References

[1] Apache River Website. http://river.apache.org/.

[2] Arduino Website. http://www.arduino.cc/.

[3] Multicast Packets Forwarding using the Trickle Algorithm - Implementation. https://github.com/g-oikonomou/contiki-sensinode/tree/mcast-forward.

[4] Information technology - UPnP Device Architecture. Technical Report ISO/IEC 29341-x, International Organization for Standardization, Geneva, Switzerland, 2008.

[5] D. Adolf. Agent Based Architecture for Smart Grids. Semester thesis, ETH Zürich, 2010.

[6] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web services architecture. *W3C*, 2004.

[7] A. P. Castellani, N. Bui, P. Casari, M. Rossi, Z. Shelby, and M. Zorzi. Architecture and protocols for the internet of things: A case study. In *PERCOM Workshops*. IEEE, 2010.

[8] K. M. Cheshire, S. DNS-Based Service Discovery, RFC 6763, 2011.

[9] S. Cheshire and M. Krochmal. Multicast DNS, RFC 6762, 2011.

[10] W. Colitti, K. Steenhaut, and N. De Caro. Integrating wireless sensor networks with the web. *IP+SN*, 2011.

[11] D. Adolf, E. Ferranti and S. Koch. SmartScript - a domain-specific language for appliance control in Smart Grids. In *SmartGridComm*, 2012.

[12] W. K. Edwards. Discovery systems in ubiquitous computing. *Pervasive Computing, IEEE*, 5(2), 2006.

[13] D. Guinard, V. Trifa, and E. Wilde. A resource oriented architecture for the web of things. *Internet of Things*, 2010.

[14] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2, RFC 2608, 1999.

[15] J. Hui and R. Kelsey. Multicast Forwarding Using Trickle - draft-ietf-roll-trickle-mcast-00. 2011.

[16] J. Hui and P. Thubert. Compression format for ipv6 datagrams over ieee 802.15.4-based networks, rfc 6282. 2011.

[17] I. Ishaq, J. Hoebeke, J. Rossey, E. De Poorter, I. Moerman, and P. Demeester. Facilitating Sensor Deployment, Discovery and Resource Access Using Embedded Web Services. *IMIS*, 2012.

[18] A. J. Jara, M. A. Zamora, and A. Skarmeta. Glowbal IP: An adaptive and transparent IPv6 integration in the Internet of Things. In *Mobile Information Systems*. IOS Press, 2012.

[19] R. Klauck and M. Kirsche. Bonjour Contiki: a case study of a DNS-based discovery service for the internet of things. In *Ad-hoc, Mobile, and Wireless Networks*. Springer, 2012.

[20] M. Kovatsch, S. Mayer, and B. Ostermaier. Moving Application Logic from the Firmware to the Cloud: Towards the Thin Server Architecture for the Internet of Things. *IMIS*, 2012.

[21] J. Mitsugi, S. Yonemura, H. Hada, and T. Inaba. Bridging UPnP and ZigBee with CoAP: protocol and its performance evaluation. In *ACM IoTSP*, 2011.

[22] Y. Pignolet, I. Rinis, D. Dzung, and A. Karaagac. Multi-Interface PLC/Wireless Network Simulation. *arXiv preprint arXiv:1209.6182*, 2012.

[23] B. Scheidegger, Y.-A. Pignolet, and E. Ferranti. Smart eagle - a bird's-eye view on heterogeneous networks for commercial and industrial sites. In *SmartGridComm*. IEEE, 2013.

[24] L. Schor, P. Sommer, and R. Wattenhofer. Towards a zero-configuration wireless sensor network architecture for smart buildings. In *ACM BuildSys*, 2009.

[25] Z. Shelby. Constrained RESTful Environments (CoRE) Link Format, RFC 6690, 2012.

[26] T. Winter, P. Thubert, A. Brandt, T. H. Clausen, J. W. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and J. Vasseur. RPL: IPv6 Routing Protocol for Low power and Lossy Networks. *IETF RFC 6550*, 2012.

[27] D.-Y. Yu, E. Ferranti, and H. Hadeli. An intelligent building that listens to your needs. In *ACM SAC*, 2013.