Embedded Web Technologies for the Internet of Things

Walter Colitti, Nguyen Thanh Long, Niccolò De Caro and Kris Steenhaut

Abstract The adoption of open standard communication protocols based on embedded IP and Web technologies is a key building block for the realization of the Internet of Things (IoT) vision, where trillions of smart objects will be connected and will open up exciting possibilities for the creation of new and innovative services. Embedded Web protocols are interoperable, reduce the overall IoT architectural complexity, and deliver a long term value as it has already been demonstrated by the success of the Internet and of the Web. This chapter discusses on the adoption of IP and in particular of Web technologies in smart objects for the realization of the Web of Things (WoT). The work reviews the main protocols and discusses on how and why they improve interoperability, reduce the overall IoT architectural complexity and facilitate the integration between the IoT and Web applications.

1 Introduction

Recent market studies in telecommunications have estimated that in the near future billions or trillions of smart objects will connect physical environments to the Internet. A smart object can be defined as any physical entity enhanced with sensing/actuating, computation and communication capabilities. The concept of having physical entities connected to the Internet is commonly known as the Internet of Things (IoT). IoT will unleash exciting possibilities and challenges for a variety of application domains, such as smart metering, e-health, logistics, building and home automation [1, 2].

The choice of the communication protocols has been a major debate among the IoT community. The IoT has been largely dominated by proprietary and domain specific protocol stacks. This has been motivated by the need of performance optimization of severely constrained devices and by the vendor lock-in based market, which forces

W. Colitti (✉) · N. T. Long · N. De Caro · K. Steenhaut
Department ETRO-INDI, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium
e-mail: wcolitti@etro.vub.ac.be

customers to be locked into a long term relationship. The use of highly specialized proprietary protocols implies the need of sophisticated application gateways between the network of smart objects and the Internet. Application gateways complicate the overall architectural complexity, hamper the scalability and complicate the application development. In addition, proprietary and specialized siloed protocols are also an obstacle for the creation of new innovative services [3].

The fragmentation of communication technologies has been considered as one of the main obstacles hampering the realization of the IoT vision. The ubiquity of sensors and actuators will unlock huge opportunities for the Internet of Services (IoS). But a necessary requirement is that the IoT will be based on open standard communication technologies which enable interoperability and plug-and-play capabilities. Open standard communication protocols based on Internet and Web technologies are a way to deliver long term value [4].

The introduction of open standards based on IP and Web technology into smart objects has been considered as a way to overcome the IoT protocol fragmentation problem and consequently to improve interoperability, flexibility, scalability and facilitate the integration of the IoT with the Internet and the Web without complicating the architectural complexity. This is a common vision shared by several communities, such as academia, industrial players and standardization bodies. The IP for Smart Objects (IPSO) Alliance, a cluster of major IT/telecom players and wireless silicon vendors, promote the use of embedded IP into constrained devices. The Internet Engineering Task Force (IETF) has standardized IPv6 over Low power Wireless Personal Area Networks (6LoWPAN), which enables the transmission of IPv6 packets in networks of resource constrained devices and IPv6 Routing Protocol for Low-power and Lossy networks (RPL), which is an IPv6 based routing protocol for the IoT [5–7].

A low power embedded IPv6 networking layer opens up the possibility to embed Web services into smart objects based on the Representational State Transfer (REST) and on the widely known Hyper Text Transfer Protocol (HTTP) up to the sensor/actuator level. This is commonly known as the Web of Things (WoT) paradigm [8]. The use of HTTP on top of 6LowPAN based smart objects eliminates the need of application gateways which can be replaced by less complex 6LoWPAN enabled edge routers. This scenario preserves the end-to-end design principle when connecting the WoT to the Internet/Web [9].

HTTP based Web applications are based on design principles different than IoT applications. In addition, it has been demonstrated that HTTP is too cumbersome for constrained devices. The main reason is that it is based on the Transmission Control Protocol (TCP) which has a large overhead. Motivated by this problem, the IETF Constrained RESTful environments (CoRE) working group, has defined a web transfer protocol called Constrained Application Protocol (CoAP). CoAP is a REST based web transfer protocol which includes several HTTP functionalities re-designed for constrained devices. CoAP is based on the User Datagram Protocol (UDP) instead of TCP [6, 10, 11].

The aim of this chapter is to give an overview on the introduction of standard IP/Web technologies in networks of smart objects. We discuss on the benefits of embedded IP and on how this technology improves IoT interoperability and architectural complexity. We describe embedded Web technologies, giving special attention to CoAP which is the protocol we have adopted in our IoT system development, and we review the main state-of-the-art work on Web standards. We also discuss on the main architectural issues for the integration of networks of web based smart objects with web applications. We do so by describing a concrete example of a REST/CoAP based Wireless Sensor Network (WSN) we have developed for greenhouse monitoring.

The rest of the chapter is organized as follows. Sections 2 and 3 discuss on embedded IP and on the embedded Web, respectively. Section 4 focuses on CoAP and on the main differences with HTTP. Section 5 describes the main building blocks which enable the integration of a REST/CoAP based WSN with a web application. Section 6 discusses on the main advantages of the adoption of open standard Web based technologies. Section 7 concludes the chapter.

2 Embedded IP Networking Technologies

The realization of a WoT requires the use of embedded IP technologies. This section gives an overview of 6LowPAN and discusses on how it brings advantages in terms of interoperability, architectural complexity and scalability.

The use of IP technology in smart objects is not straightforward. IP was designed for relatively stable networks and for non-constrained nodes. Sensor/actuator motes are severely constrained in terms of power, memory, communication bandwidth and energy. As a consequence, networks of smart objects like WSNs can be considered an instance of Low Power and Lossy Networks (LLN) [12, 13], in which the communication protocols need low power consumption and in which the packet loss may be deteriorated by the physical environments (fading, interferences, bit errors, etc.). This problem has motivated IETF's standardization activity in defining 6LoWPAN so that to enable the use of IPv6 in networks of resource constrained devices, such as the ones based on IEEE 802.15.4 radio links.

6LoWPAN defines an adaptation layer which provides encapsulation and header compression mechanisms allowing IPv6 packets to be transferred between resource constrained devices. IP packets in general have a significantly large packet header resulting in a large overhead which increases the motes' energy consumption and the packet loss probability. The header compression in the 6LowPAN adaptation layer eliminates redundant fields of the packet header. The adaptation layer also provides fragmentation and reassembly mechanisms needed to transfer IPv6 packets in multiple 127 bytes long IEEE 802.15.4 frames [5].

A major advantage of the use of embedded IP based technologies is the interoperability, which is one of the main reasons for the success of IP. IP has been designed to work with different link layers (wired, wireless, etc.). This is of critical importance when considering the heterogeneous nature of the IoT. IoT consists of tiny constrained devices which communicate wirelessly but also of more powerful

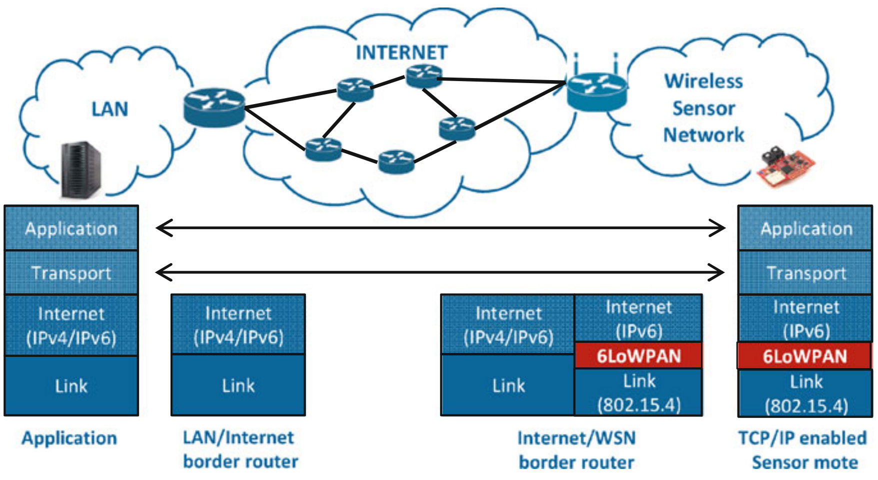

Fig. 1 Embedded IP technologies decrease the architectural complexity by replacing application gateways with border routers

devices connected to the network with wired communication. The layered architectural principle of IP facilitates the communication between these heterogeneous devices. In addition, IP has been designed based on the end-to-end principle, so that to decouple the networking technology from the upper application protocols which could evolve independently. The network is transparent to the application logic which is understood only at the end points. Without the end-to-end principle, application functionalities should have been pushed into the network with consequent increase of the architectural complexity. This would have hampered the huge application potential of the Internet and would have prevented IP to become the preferred networking protocol for a variety of applications and data, such as text, voice, video, etc.

The advantages of the end-to-end design principle will be reflected in the IoT if IP technology will be adopted as much as possible. Application developers can design applications independently of the networking mechanisms and of the internal functionality of embedded devices. In addition, IP technology enables the development of applications and systems which are based on widely used protocols such as HTTP, Simple Network Management Protocol (SNMP) and Dynamic Host Configuration Protocol (DHCP) [9].

Figure 1 gives an idea of the simplification of the network architectural complexity when embedded IP based technology is deployed in the IoT. The figure shows a simple case in which an application running in a certain Local Area Network (LAN) needs to access a 6LowPAN enabled WSN via the Internet. The WSN is connected to the Internet via a WSN border router. On the WSN side, the border router is equipped with a low power wireless data link interface, such as IEEE 802.15.4, which is typical in WSNs. On top of the data link layer, the 6LowPAN adaptation layer compresses/decompresses IPv6 packets going from/to the Internet. With IP being the networking protocol in the WSN, the edge device connecting the Internet to the IoT is nothing else than a router with a 6LowPAN adaptation layer. The figure also shows that the application server and the sensor being accessed have the same TCP/IP stack and consequently the end-to-end principle can be preserved across the three network segments (LAN, Internet and IoT). If the sensor had a different non TCP/IP based stack, the edge device connecting the IoT to the Internet would not be a border router but an application gateway. It is straightforward to understand that application gateways know the application logic and consequently the end-to-end principle would not be preserved between the Internet and the IoT. In addition application gateways decrease the network flexibility and hamper the scalability. Whenever there is a protocol upgrade or a new application is developed, the change has to be reflected in the application gateways [9].

3 Embedded Web Technologies

The previous section has discussed on the benefits of introducing standard IP networking into the IoT. This section gives an overview of how also the Web concept and technologies have been considered to be ported to the IoT for the realization of the WoT.

Web Services have been considered as a solution to facilitate the integration of the IoT with the Web. Web services allow application developers to rely on platform-independent and reusable software, which drastically simplifies the application development process. In addition, IoT monitoring and actuation functionalities can become part of the mashup process which has been one of the key success factors of the Web 2.0. Developers can seamlessly mash up functionalities provided by traditional web services with the ones offered by embedded devices. This opens up the possibility to develop innovative and creative applications [14, 15].

There are two major Web service paradigms: *WS-** and *Representational State Transfer (REST)*.

In the WS-* architecture, the service functionality and interfaces are declared in a Web Service Description Language (WSDL) file. It uses Simple Object Access Protocol (SOAP) messages with an Extensible Markup Language (XML) payload and an HTTP based transport protocol to provide remote Procedure Calls (RPC) between clients and servers. There has recently been an effort to adapt the paradigm to constrained devices [16]. A lightweight version of WS-*, called Device Profile for Web Services (DPWS) has also been proposed [17].

REST is an architectural style based on the concept that everything is modeled as a resource. Resources are abstractions controlled by the server and identified by a Universal Resource Identifier (URI). Resources are accessed by clients in a synchronous request/response fashion using an application protocol. REST is not tied to a particular application protocol. However, the vast majority of REST architectures nowadays use HTTP. HTTP manipulates resources by means of its methods GET, POST, PUT, etc. One of the main advantages of REST is that the resources are decoupled from the services and therefore can be arbitrarily represented by means of various formats. These means that more compact formats with a lower overhead compared to XML can also be used, such as Java Script Object Notation (JSON). JSON has a compact data representation but does not have the flexibility of XML. A better compromise between the compact representation and the flexibility of XML is a binary representation of XML called Extensible XML Interchange (EXI). In [6] other binary representations of XML are described, such as Fast Infoset, a standard defined by International Telecommunication Union Telecommunication Standardization Sector (ITU-T), and Binary Extensible Markup Language (BXML), defined by the Open Geospatial Consortium (OGC). The author in [6] shows that EXI provides a more efficient encoding up to 97 % smaller than the equivalent XML. The efficiency of EXI has also been demonstrated in [18].

REST has widely been accepted as the preferred architecture for the WoT [8, 19, 14]. Firstly, REST is based on loosely-coupled stateless interactions and consequently Web services based on RESTful Application Programming Interfaces (APIs) can be reused and combined in a straightforward manner. Secondly, REST has a more lightweight structure than WS-* which makes it more appropriate for networks of constrained devices. The better performance of REST compared to WS-* when applied to constrained devices has been demonstrated in [20] in terms of overhead, memory footprint and completion time. Thirdly, the World Wide Web is largely dominated by REST based Web services and therefore porting the REST principle to the IoT would simplify its integration with the Web.

Various research papers have already demonstrated the feasibility of applying REST/HTTP to the IoT for the realization of the WoT.

In [19], the authors propose to reuse the open, interoperable and standard principles of the modern Web architecture to integrate physical objects to the Web. The field of application considered in the work is the emerging field of Smart Home. Since in REST/HTTP the discovery is done by following links and consequently there is not mechanism for device discovery, the authors develop an ad hoc device discovery procedure which locates and register information related to embedded devices. In order to achieve interoperability with heterogeneous embedded devices and services, the authors propose to use the Web Application Description Language (WADL), an XML based file format that provides a machine readable description of HTTP based Web applications. WADL is language and platform independent and it models the resources provided by an embedded device and the relationships between them. The authors use the Tmote Sky, a sensor/actuator programmable platform, on which they implemented five different resources: three resources to monitor temperature, humidity and illumination, one resource to activate a LED lamp and one resource to measure the energy consumption of some electrical appliances. The system developed in this work also includes an application framework which supports the creation of mashups and advanced rules in a simple way and in any programming language that supports HTTP.

In [8], the authors apply the REST/HTTP principle to embedded devices. They use the Sun SPOT platform, a WSN platform which includes a Web server which exposes sensors, actuators and internal functionalities as REST resources represented in JSON and accessed using HTTP requests. The actuators are controlled using synchronous

HTTP calls (client pull) while monitoring functionalities are accessed through a syndication mechanism via an external Atom(Pub) server. The authors implement a *physical mashup* mechanism which apply typical Web 2.0 patterns to embedded devices. They show the benefits of the system by developing two applications: an energy aware Web dashboard, which allows people to control and experiment with the energy consumption of their appliances, and the physical mashup editor, which enables users to create physical mashups by means of visual components and without requiring any programming skills.

In [21], the authors apply embedded Web technologies to the building automation domain. They present an approach to integrate tiny wireless sensor/actuator nodes into an IP/Web based network. Sensor and actuator nodes run a small Web server on top of a TCP/IP stack. The Web server exposes the sensing and actuating functionalities in a RESTful way and therefore can be accessed by means of HTTP methods. The data is represented in the JSON format. Service discovery based on multicast DNS messages enables the system to integrate new devices without additional configuration effort. The authors measure the average response time of the embedded Web based devices when responding to periodic HTTP GET requests and show that the system offers an acceptable performance given the limited computing power and memory constraints of the hardware platform.

Although the feasibility of adopting REST technologies in the WoT has been proven and its advantages have been discussed as shown in the aforementioned research work, there are still performance problems in the realization of a REST based WoT, especially for the verbosity of HTTP which makes it not appropriate to constrained devices. The authors in [8] evaluate the performance of the proposed REST based WoT system and argue that the verbosity of HTTP does not prevent highly efficient applications to be implemented. However, when battery life time and latency is critical, more optimized protocols which minimize the latency have a better performance. When a sub-second latency can be tolerated and battery life time is not critical, the advantages of HTTP outweigh the loss of performance. In addition to the battery life time and latency, when applying the web service principle to network of constrained devices the difference between IoT applications and traditional Web applications need to be taken into account. IoT applications are short-lived and web services reside in battery operated devices which most of the time sleep and wake up only when there is data traffic to be exchanged. In addition, such applications require a multicast and asynchronous communication compared to the unicast and synchronous approach used in standard Web applications [6]. These characteristics of IoT applications suggest that HTTP is not appropriate for the WoT and that a more optimized REST based application protocol needs to be adopted.

4 Constrained Application Protocol

The previous section has described how the REST/HTTP principle has been applied to smart objects to realize the WoT and how HTTP might not be optimal for constrained devices as a consequence of its verbosity and of its design principles which

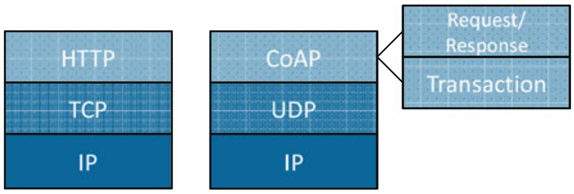

Fig. 2 Difference between the HTTP and CoAP stacks

are not suited for the IoT. The unsuitability of HTTP for the IoT has motivated the effort of the IETF CoRE Working Group in defining a new protocol called CoAP. This section is dedicated to a description of CoAP with main accent on the major difference with HTTP and on the interoperability effort being done during the standardization process. We will also review the main CoAP related literature work.

CoAP is a web transfer protocol optimized for resource constrained networks. It consists of a subset of HTTP functionalities which have been re-designed taking into account the low processing power and energy consumption constraints of small embedded devices such as sensor motes. In addition, various mechanisms have been modified and some new functionalities have been added in order to make the protocol suitable to the IoT. The difference between HTTP and CoAP protocol stacks is illustrated in Fig. 2.

The first significant difference between HTTP and CoAP is the transport layer. TCP flow control mechanism is not appropriate for IoT applications and its overhead is considered too high for short-lived transactions. In addition, TCP does not have multicast support. CoAP is built on top of UDP which has lower overhead and provides multicast support.

CoAP is organized in two layers. The *Transaction* layer handles the single message exchange between end points. The *Request/Response* layer is responsible for the requests/responses transmission and for the resource manipulation. The dual layer approach allows CoAP to provide reliability mechanisms even without the use of TCP as transport protocol. In fact, a message can be labeled as *Confirmable* which implies that it is retransmitted using a default timeout and exponential back-off between retransmissions, until the recipient sends the *Acknowledgement* message. In addition, the two layer approach enables asynchronous communication which is a key requirement for the WoT in which embedded web servers are not always able handle requests immediately.

Differently from HTTP, CoAP includes a built in observation mechanism which allows a client to subscribe to a resource hosted on a CoAP server and to receive notifications upon a change of the resource state [22]. The protocol also includes a technique for discovering and advertising resource descriptions. The mechanism is based on two approaches recently defined by IETF for HTTP: the well-known resource path */.well-known/scheme* (RFC 5785), which defines a "well-known location" where to store resources to be easily located, and the *Web Linking* (RFC 5988), which defines a framework for typed links [10]. Since the resource descriptions need to be machine interpreted, the IETF CoRE group has also worked on the standardization of the *CoRE Link Format* which standardizes the resource description format itself [23]. In order to achieve a uniform and interoperable resource discovery, a smart object running a CoAP server can provide a resource description available via the well-known URI */.well-known/core*, as done in the HTTP the well-known resource path. The */.well-known/core* resource accessed by a CoAP server returns a list of links to the resources hosted by the server and other link relations.

One of the major design goals of CoAP has been to keep the message overhead as small as possible and limit the use of fragmentation. HTTP has a significantly large message overhead. This implies packet fragmentation and associated performance degradation in networks of constrained devices. CoAP uses a short fixed-length compact binary header of 4 bytes followed by compact binary options. A typical request has a total header of about 10–20 bytes.

In [24], we have compared a REST/HTTP based WSN with a REST/CoAP WSN. We have demonstrated that when using CoAP as application protocol there is a significant performance improvement in terms of energy consumption and response time. The improvement is consequence of the lower number of bytes transferred in a CoAP client-server transaction compared to an HTTP client-server transaction. Thanks to the header compression, after being encapsulated in the UDP, 6LoWPAN and MAC layer headers, the CoAP packet can be transferred into a single MAC frame which has a size of 127 bytes.

Although CoAP is work in progress, it has already been addressed in some research work, such as in [25] and in [26].

In [25], Maenpaa et al. propose an architecture for peer-to-peer (P2P) federation of geographically distributed WSN islands. The WSN islands are low power and lossy networks which use CoAP as application protocol. The border routers of the WSN islands are equipped with cellular radio interface and are organized in a wide area P2P overlay network. The wide area network provides various services such as a common namespace and rendezvous. It uses Resource Location And Discovery (RELOAD), a P2P signaling protocol being designed and standardized by IETF. The P2P system is proposed as a decentralized alternative to the centralized approaches used in CoAP, such as DNS-SD and Resource Directory. In order to integrate CoAP and RELOAD, the authors introduce a CoAP application usage for RELOAD. The application uses basic functionalities such as registration of CoAP resources in the RELOAD overlay, *rendezvous* using dedicated connections for CoAP and *rendezvous* using tunneling and the use of RELOAD overlay as a cache for sensor data. The use case chosen to simulate and study the performance of the approach is road traffic and road condition monitoring in the Finnish highway network.

In [26], the authors propose an approach which moves the application logic form the embedded devices to the cloud. The approach enables the reuse of deployed devices for different applications without changing the firmware. On the device side, the approach is based on the implementation of thin servers, embedded web servers which do not host any application logic and which only provide interfaces to elementary functionalities such as sensor/actuator access and configuration of device parameters to interact with the physical world. The thin server's interfaces are RESTful APIs mainly implemented with CoAP, or with HTTP for more powerful devices.

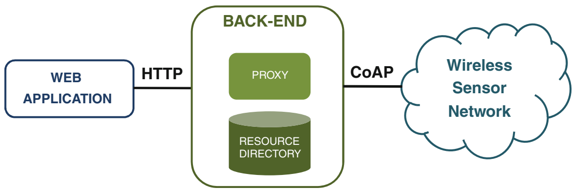

Fig. 3 Building blocks of an end-to-end system integrating a REST/HTTP Web application with a REST/CoAP WSN

The authors argue that in order to simplify the application development, APIs must be self-descriptive and concise, like the ones based on the CoRE Link Format and Microformats. On the back-end side, application logic is based on scripting languages (e.g. JavaScript) which are widely known by application developers. The authors use Californium, a CoAP framework implemented in Java, integrated with JavaScript support for the development of the application logic. In particular, a coapRequest object API provides the same functionality as the well known xmlHttpRequest API.

5 Architecture of an End-to-End System for Integration of a WoT with the Web

The previous sections discussed on the enabling technologies and protocols for porting the REST principle to the IoT and therefore for the realization of the WoT. This section discusses on the integration of the WoT with the Web in order to show how standard embedded web technologies facilitate the integration and simplify the end-to-end architecture. As baseline for the discussion, we use the prototype design and development of a system which integrates a REST/CoAP based WSN with a web application and which facilitates the seamless exploration, access and visualization of WSN data. The platform has been developed for real-world deployment in a greenhouse. However, The core building blocks of the platform can be reused for several application domains which need to seamlessly explore, access and visualize WSN data. An overview of the overall system is given in Fig. 3.

As illustrated in Fig. 3 the end-to-end system consists of three main building blocks: the REST based WSN, the back-end and the application. Sections 5.1 and 5.2 describe the WSN and the back-end, respectively. This section only aims to describe the architecture of an end-to-and REST based system which integrates a WSN with a Web application. Therefore, the description of the monitoring application and its functionalities is out of the scope of this work.

5.1 The REST Based WSN

The WSN is a set of Zolertia Z1 motes. Zolertia is a programmable wireless sensor platform with a radio chip supporting the 2.4 GHz IEEE 802.15.4 standard. The sensor motes run Contiki OS, a widely known operating system for constrained devices such as wireless sensors and actuators. Each mote runs ContikiMAC, which is the Contiki default MAC protocol, the Contiki 6LowPAN stack and RPL routing protocol, and a Contiki low-power CoAP server called Erbium [27]. The motes are equipped with an embedded temperature sensor and have been expanded with additional humidity and light sensors to meet the requirements of greenhouse monitoring. Temperature, humidity and light measurements are exposed as CoAP resources.

For interoperability purpose, the resources are exposed based on the IPSO Application framework, which defines a template used to expose REST resources hosted by CoAP or HTTP servers [28]. The framework groups resource types into *Function Sets*, each one having a recommended root path under which its sub-resources are organized. Each Function Set is assigned a Resource Type parameter which allows the resource to be discovered. The resource semantics defined are compatible with the HTTP Web Linking and with the CoRE Link Format. The data encoding format used is JSON.

5.2 The Back-End

The back-end is the core component of the end-to-end system. It is located on the edge beetween the WSN and the Internet and aims to integrate the WSN with the Web application in a seamless and transparent way. Its main modules are implemented in Java and run on a standard Linux Ubuntu machine physically located at the edge between the WSN and the Internet. It is therefore equipped with two data link layer technologies, being Ethernet and IEEE 802.15.4. The former one is a standard Ethernet card integrated in the machine, while the latter one is realized with a Zolertia Z1 mote running Contiki, connected to the USB port and configured to be a border router.

The CoAP functionalities in the back-end box are enabled by Californium, the Java based CoAP framework introduced in [26]. Although being relatively new, there are already several CoAP implementations for several programming language [29]. Our choice for Californium is driven by its modular and flexible architecture and by the fact that it provides HTTP-CoAP proxy functionality (see Sect. 5.2).

The backend integrates the WSN with the Web application via two major functionalities: the transparent WSN resource access and the Web based WSN resource search and discovery. The two building blocks enabling these two functionalities are described in the two following sub-paragraphs.

5.2.1 Cross-Protocol Resource Access

Resources exposed by CoAP based sensor motes can be accessed by requests sent by a CoAP client. However, when integrating the WSN with a Web application, the data need to be accessed via HTTP requests sent by an HTTP client, unless the HTTP client is equipped with CoAP functionalities. The solution of having CoAP functionalities at the Web client side has been proposed in [30]. The author proposes a CoAP plugin for the Mozilla Firefox Web browser. The plugin registers a protocol handler for the scheme *coap* and therefore allows a user to discover and access the WSN resources by directly entering CoAP URIs in the address bar or by followed links on Web pages. Although this is an interesting solution, it cannot be largely adopted while integrating WSNs with the Web. Firstly, it requires users to have the CoAP browser installed which is not a typical scenario. A typical web scenario consists of a user accessing, via an HTTP client, a web application running on a web server, similar to the case shown in Fig. 3. In addition, having a CoAP client running at the user side implies that the communication will go over UDP not only in the WSN network segment but also in the Internet segment. Although CoAP over UDP includes reliability mechanisms based on retransmissions, TCP should remain the preferred transport protocol in the Internet segment for better performance and higher reliability [31].

The problem of accessing CoAP servers from HTTP clients has been addressed by the IETF CoRE group who has defined the guidelines for mapping HTTP requests into CoAP ones and vice versa [31]. The mapping operation is executed in a reverse proxy usually placed at the edge of the constrained network, as illustrated in Fig. 3. The HTTP-CoAP proxy enables cross-protocol resource access by transparently mapping an HTTP request coming from a Web application into a CoAP request to be forwarded to a CoAP server and vice versa.

It is worth underlining that although the proxy translates two application protocols, it does not have the same level of complexity as application gateways. As mentioned in Sect. 2, an application gateway is aware of the application logic, which has major impact on the architectural complexity. The HTTP-CoAP proxy is transparent to the application logic because the two protocols are both based on REST and therefore they use the same set of verbs to manipulate the resources. The proxy only maps the Request/Response model of CoAP into HTTP (the REST requests/responses). The underlying messages exchanged on the Transaction layer have no effect on the proxy functions [31].

In our system shown in Fig. 3, the application server runs in an IPv4 network. When the application wants to query a CoAP server to retrieve its data, the Web application server generates a GET request with the URI http://sensor.domain.com/data, where *sensor* is the name of the the CoAP server, *domain.com* is the network domain of the WSN and of the proxy, *resource* is the requested resource, which in our case can be temperature (*temp*), humidity (*hum*) or light (*light*). Once received the GET request, the proxy generates the URI coap://sensor_ipv6_address.domain.com/resource where sensor_ipv6_address is the IPv6 address of the CoAP server. Once the CoAP server sensor receives the URI, it generates a response with the data embedded into a JSON file. The proxy maps the CoAP response into an HTTP response and sends it back to the Web application server.

In order to decrease the dependency between the Web application and the proxy and between the proxy and the CoAP servers, the system relies on the Domain Name System (DNS) which guarantees that the client does not need to know the IPv4 address of the proxy and the proxy does not need to know the IPv6 addresses of the wireless sensor motes. This implies that the DNS keeps an A record resolving the IPv4 address of the proxy and an AAAA record resolving the IPv6 address of the sensor mote.

The description of the main mapping operation shows that the proxy is protocol agnostic, meaning hat the Web application accesses WSN resources as if it were using HTTP and without having knowledge that the sensor data are accessed via CoAP. The transparency is guaranteed by a homogeneous URI mapping. This implies that during the mapping operation, the authority and the path of the resource URI do not change. The only translation undergone by the URI is in the scheme (*http:* translated into *coap:*).

5.2.2 Resource Search and Discovery

As described in Sect. 4, CoAP includes a technique for discovering and advertising resource descriptions. A CoAP server can provide a description of the available resources via the well-known URI */.well-known/core* which returns a list of links to the resources hosted by the server. There are cases in which this approach does not guarantee good performance in terms of search and discovery capability. As an example, we have calculated the time needed to discover all the resources in the WSN with an increasing number of sensor motes. We define the *discovery time* as the time needed by the CoAP client to send a GET request to the */.well-known/core* interface of every sensor mote and to receive the description of all the resources exposed by the motes. For a number of motes equals to 9, 13 and 25 the discovery time is around 8, 13 and 42 s, respectively. This indicates that the resource discovery mechanism might degrade the performance in case of large networks and when trying to explore resources exposed by several sensor motes. Another case in which the CoAP discovery mechanism might hamper the search and discovery functionality is when there are sleepy motes which sleep for long periods of time [23].

The IETF CoRE group has addressed this problem by introducing the Resource Directory (RD), a registry which stores the descriptions of the resources exposed by the CoAP servers. When exploring the WSN resources, the web application can send a query to the RD instead of accessing the */.well-known/core* interface of every sensor mote.

CoAP server motes register their resource descriptions in the RD by sending a POST message which also indicates the period of time during which the registration is valid. Once this period has elapsed, the RD erases the corresponding resource description, which indicates that the CoAP server is not reachable. The RD keeps the registration alive upon receiving refresh messages from the CoAP server.

Resources registered on the RD can be queried by means of CoAP GET requests which can also include filtering options which limit the scope of the lookup to certain resource types or to a limited set of CoAP servers. The resource discovery process is made transparent to the Web application by means of the HTTP-CoAP proxy module described in Sect. 5.2.

6 Discussion

Up to now we have seen how embedded web services allow smart objects to become web servers which expose resources in a REST fashion as any traditional Web server. Although they facilitate the integration of the IoT with the Web, embedded web services might have a cost for constrained devices and therefore might cause performance degradation. Firstly, the CoAP/UDP/6LoWPAN/ stack significantly increases the memory footprint which might not be tolerated by certain severely constrained devices. In addition, the overhead introduced by the IP and web oriented communication protocols might have a significant impact on the power consumption and consequently on the battery lifetime. Battery duration remains a serious bottleneck for the realization of the IoT vision and therefore needs to be taken into consideration when engineering networks of smart objects.

In order to have a general idea of the degradation introduced by embedded web mechanisms in a WSN, we show a simple experiment which measures the average mote's power consumption in two different data collection simulation scenarios. In the first scenario the data is collected via CoAP request/responses, while in the second via a simple application which uses the same request/response approach as CoAP, but does not include the reliability and other more sophisticated features provided by CoAP. The reference application has been developed on top of the same UDP/6LowPAN stack as the one used in the CoAP scenario. In both cases the requests are initiated by the client running in the back-end. The experiments have been executed with Cooja, an emulator of Contiki based sensor motes. Cooja provides a tool called Energest able to estimate the energy spent by a mote while transmitting, receiving, processing and while being in idle state. The aim of this experiment is not to provide an exhaustive and quantitative analysis of the power consumption but to give a qualitative indication of the cost that needs to be paid in terms of power consumption when using CoAP. The experiment has been executed with three mesh networks containing 9, 13 and 25. The results have been calculated as average values of several independent simulation runs. The results are shown in Fig. 4.

Figure 4 clearly shows that when using CoAP the average power consumption of a server mote is significantly higher than the simple application built on top of UDP. This is mainly consequence of CoAP's reliabilty mechanism based on retransmissions. The diffeence between the two protocols is about 20 % for 13 motes, 25 % for 15 motes and 30 % for 25 motes. In fact larger networks are likely more congested

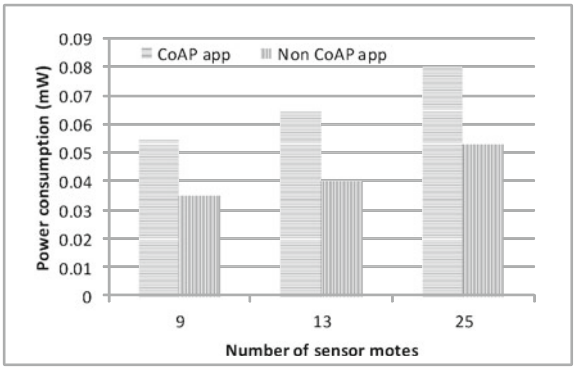

Fig. 4 Power consumption of a CoAP and non-CoAP based application

and consequently CoAP executes more retransmissions. The higher the number of retransmissions, the higher the power consumption.

Although having an impact on the mote's power consumption, the more sophisticated mechanisms of CoAP are needed to increase the network reliability. Figure 5 shows the success rate measured in the two scenarios described in the previous experiment. The success rate is defined as the ratio between the number of responses successfully received by the client and the number of requests sent by the client. CoAP achieves a very high success rate in all the three simulated networks while the application without CoAP has a success rate of about 95 % for 9 and 13 motes and drastically reduced (86 %) for 25 motes.

These two simple experiments show that a key aspect of IoT communication protocols is to find the right balance between power consumption and reliability. This is also true and rather straightforward for protocols in other wired and wireless communication networks as well. But the constrained nature of smart objects makes the performance optimization of IoT protocols significantly more complex. This has been one of the main reasons why the IoT has largely been dominated by several service specific communication protocols. Communication engineers have mainly tried to design protocols highly optimized for specific tasks and application domains. Highly specialized proprietary protocols were also used by monitoring and control system vendors to force customers to be locked into a long term relationship.

Proprietary and domain specific protocols can no longer be the adopted solution in the present and future IoT. Sensor and actuator networks are no longer a niche technology used in a few specific sectors. The trillion smart objects vision suggests that we will be largely surrounded by sensors and actuators. This is also consequence of the recent emerging open hardware based technologies which allow people to build their own connected objects even without engineering skills. The ubiquity of sensors and actuators will unlock huge opportunities for the creation of new innovative services. But a necessary requirement is that the IoT will be based on

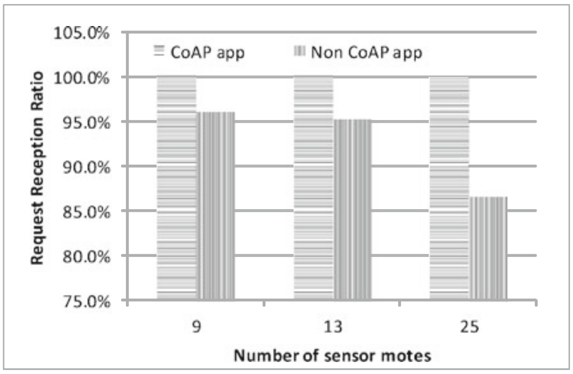

Fig. 5 Success rate of a CoAP and a non-CoAP based application

open standard communication technologies which enable interoperability and plug-and-play capabilities [3].

Interoperability has been the main reason for the success and large scale adoption and deployment of IP and Web technologies. It has been the result of a large open standardization effort which has guaranteed that IP equipment made by different manufacturers are interoperable in a plug-and-play fashion and that Web client and servers can communicate in a platform and programming language agnostic way. The IETF has reproduced the same interoperability driven standardization process for the IoT communication protocols. This is proven by the CoAP plugtest events organized by the IETF CoRE group in collaboration with the European Telecommunications Standards Institute (ETSI). These events aim to test the interoperability of CoAP in a multi-vendor and multi-service environment. The first CoAP plugtest event has seen the participation of several research organizations and industrial players. A total of 3406 tests were made, to test several CoAP functionalities between CoAP client/server implementation of different institutions. More than 90 % of the tests succeeded and demonstrated the robust interoperability nature of CoAP [29].

Although the complexity of communication protocols increases when designed for heterogeneous devices with different functionalities and constraints, open standard communication protocols based on Internet and Web technologies are a way to deliver long term value. They eliminate the vendor lock-in problem, decrease the cost of network maintenance and create opportunities for new and innovative business models. This has been widely accepted by the industrial world and demonstrated by the fact that some industry driven communication protocols which were born without Internet and Web functionalities are being transformed into Internet/Web oriented protocols. An example is ZigBee. ZigBee is a proprietary specification for wireless communication between sensors and actuators maintained and published by the ZigBee Alliance which include several big industrial players. ZigBee is based on the IEEE 802.15.4 radio link layer. The network and transport layer are proprietary and not based on the TCP/IP protocol stack. The application layer in ZigBee is organized in profiles, each profile providing standard interfaces and device definitions for product interoperability in a vertical application domain, such as home automation, building automations and telecom services [9]. In particular, the application profile related to energy services, called Smart Energy Profile (SEP), has gained momentum especially in the US market. In its 1.0 version, the Smart Energy profile does not support IP/Web communications as a consequence of the fact that the ZigBee stack is not IP based. However, when the smart grid market has identified the need for interoperability, the ZigBee Alliance has started a significant re-design of the ZigBee stack and of the SEP. The network/routing layer of the new ZigBee IP stack is now based on the IETF 6LowPAN and RPL, while the transport layer on TCP and UDP. AS for the SEP, its 2.0 version is based on the REST architecture and requires the use of HTTP for interoperability. The use of CoAP as application protocol and UDP as transport protocol is also considered in the case of more constrained devices. The encoding format adopted is EXI.

The migration of a mature and industry driven protocol like ZigBee towards IP/Web technologies proves the fact that the IoT community has accepted that open standards will significantly boost the IoT. There will surely be cases in which more specialized and non IP based protocols will continue to exist. But this will likely happen only in certain vertical domains which need specific functionalities. The trillion smart objects vision will necessarily require a migration from the siloed approach, with service specific sensor and actuator networks, to the open and interoperable IoT and WoT which will pave the way for the realization of an exciting Internet of Services.

7 Conclusion

This chapter gave an overview on the introduction of standard IP/Web technologies in the IoT. We discussed on the benefits of 6LowPAN and showed how it improves interoperability and reduces the overall architectural complexity. We described how embedded Web technologies based on REST/HTTP have been adopted in the IoT for the realization of the WoT. We introduced CoAP, an HTTP like protocol which is more appropriate for constraint devices. We presented the prototype design and development of a system which integrates a REST/CoAP based WSN with a web application. We described the architecture of the back-end system which provides Web based WSN resource search and discovery functionality, enabled by the CoAP based Resource Directory, and the transparent WSN resource access, enabled by the HTTP-CoAP proxy. We showed that the proxy is protocol agnostic and transparent to the application logic. We showed that IP/Web technologies for heterogeneous devices makes it more complicated to find a balance between power consumption and reliability. However, open standard communication protocols based on Internet/Web technologies are a way to deliver long term value and therefore need to be a key ingredient of the future IoT.

References

1. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Net. **54**,(15) 2787–2805 (2010)
2. Miorandi, D., Sicari, S., De Pellegrini, F., Chlamtac, I.: Internet of things: vision, applications and research challenges. Comput. Net. **10**(7), 1497–1516 (2012)
3. Ko, J., Eriksson, J., Tsiftes, N., Dawson-Haggerty, S., Vasseur, J. P., Durvy, M., Terzis, A., Dunkels, A., Culler, D.: Beyond interoperability pushing the performance of sensor network IP stacks. In: 9th ACM Conference on Embedded Networked Sensor Systems, November 2011
4. Zorzi, M., Gluhak, A., Lange, S., Bassi, A.: From today's INTRAnet of things to a future INTERnet of things: a wireless- and mobility-related view. IEEE Wireless Commun. **17**(6), 44–51 (2010)
5. Dunkels, A., Eriksson, J., Finne, N., Österlind, F.: Low-Power IPv6 for the internet of things. In: 9th International Conference on Networked Sensing Systems, 1–6, June 2012
6. Shelby, Z.: Embedded web services. IEEE Wireless Commun. **17**(6), 52–57 (2010)
7. Hui, J.W., Culler, D.E.: IPv6 in low-power wireless networks. Proc. IEEE. **98**(11), 1865–1878 (2010)
8. Guinard, D., Trifa, V., Wilde, E.: A Resource oriented architecture for the web of things. In: Internet of Things 2010 International Conference (IoT 2010), November 2010
9. Vasseur, J.P., Dunkels, A.: Interconnecting Smart Objects with IP: The Next Internet. 1st ed. Morgan Kaufmann. Burlington, Massachusetts (2010)
10. Bormann, C., Castellani, A.P., Shelby, Z.: CoAP: an application protocol for billions of tiny internet nodes. IEEE Int. Comput. **16**(2), 62–67 (2012)
11. Shelby, Z., Hartke, K., Frank, B.: Constrained Application Protocol (CoAP). Internet-Draft, draft-ietf-core-coap-13, 2012 (Work in progress).
12. Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., Alexander, R.: In: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks, RFC 6550, 2012
13. Gaddour, O., Koubâa, A.: RPL in a nutshell: a survey. Comput. Net. **56**(14), 3163–3178 (2012)
14. Zeng, D., Guo, S., Cheng, Z.: The web of things: a survey. J. Commun. **6**(6), 424–438 (2011)
15. Guinard, D., Trifa, V., Pham, T., Liechti, O.: Towards physical mashups in the web of things. In: 6th IEEE International Conference on Networked Sensing Systems, June 2009
16. Priyantha, N.B., Kansal, A., Goraczko, M., Zhao, F.: Tiny web ser- vices: design and implementation of interoperable and evolvable sensor networks. In: The 6th ACM conference on Embedded Network Sensor Systems (SenSys '08) 253–266, 2008
17. Jammes, F., Smit, H.: Service-oriented paradigms in industrial automation. IEEE Trans. Industr. Inf. **1**(1), 62–70 (2005)
18. Castellani, A. P., Gheda, M., Bui, N., Rossi, M., Zorzi, M.: Web services for the internet of things through CoAP and EXI. In: IEEE International Conference on Communications (ICC), 2011
19. Kamilaris, A., Trifa, V., Pitsillides, A.: The smart home meets the web of things. Int. J. Ad Hoc Ubiquitous Comput. **7**(3), 145–154 (2011)
20. Yazar, D., Dunkels, A.: Efficient application integration in IP-based sensor networks. 1st ACM Workshop on Embedded Sensing Systems for Energy Effciency in, Buildings, 43–48 November 2009
21. Schor, L., Sommer, P., Wattenhofer, R.: Towards a zero-configuration wireless sensor network architecture for smart buildings. In; Proceedings of 1st ACM Workshop On Embedded Systems For Energy-Efficient Buildings (BuildSys: pp. 31–36. USA, Berkeley) 2009
22. Hartke, K.: Observing Resources in CoAP, Internet-Draft, draft-ietf-core-observe-07, 2012 (Work in progress)
23. Shelby, Z.: Constrained RESTful Environments (CoRE) Link Format. RFC 6690, 2013 (Work in progress)

24. Colitti, W., Steenhaut, K., De Caro, N., Buta, B., Dobrota, V.: Evaluation of constrained Application Protocol for Wireless Sensor Networks. In: 18th IEEE International Workshop of Local and Metropolitan Area Networks (LanMan), 1–6 November 2011
25. Maenpaa, J., JIMENEZ Bolonio, J., Loreto, S.: Using RELOAD and CoAP for wide area sensor and actuator networking. EURASIP J. Wireless Commun. Net. **2012**, 121. Springer (2012)
26. Kovatsch, M., Mayer, S., Ostermaier, B.: Moving application logic from the firmware to the cloud: towards the thin server architecture for the internet of things. In: 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012), July 2012
27. Kovatsch, M., Duquennoy, S., Dunkels, A.: A low-power CoAP for contiki. In: 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems, October 2011
28. Shelby Z., Chauvenet, C.: The IPSO Application Framework, draft-ipso-app-framework-04, 2012, (Work in progress)
29. Lerche, C., Hartke, K., Kovatsch, M.: Industry adoption of the internet of things: a constrained application protocol survey. In: 7th International Workshop on Service Oriented Architectures in Converging Networked Environments, September 2012
30. Kovatsch, M.: Demo abstract: humanCoAP interaction with copper. In: 7th IEEE International Conference on Distributed Computing in Sensor Systems, June 2011
31. Castellani, A., Loreto, S., Rahman, A., Fossati, T., Dijk, E.: Best Practices for HTTP-CoAP Mapping Implementation, Internet-Draft, draft-ietf-core-http-mapping-02, 2013 (Work in progress)