Integrating Wireless Sensor Networks with the Web

Walter Colitti
Vrije Universiteit Brussel - ETRO
Pleinlaan 2, 1050 Brussels
+32 2 629 10 27
wcolitti@etro.vub.ac.be

Kris Steenhaut
Vrije Universiteit Brussel - ETRO
Pleinlaan 2, 1050 Brussels
+32 2 629 29 76
ksteenha@etro.vub.ac.be

Niccolò De Caro
Vrije Universiteit Brussel - ETRO
University of Perugia - DIEI
+32 2 629 10 27
ndecaro@etro.vub.ac.be

ABSTRACT
IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) has accelerated the integration of Wireless Sensor Networks (WSNs) and smart objects with the Internet. At the same time, the Constrained Application Protocol (CoAP) has made it possible to provide resource constrained devices with RESTful web service functionalities and consequently to integrate WSNs and smart objects with the Web. The use of Web services on top of IP based WSNs facilitates the software reusability and reduces the complexity of the application development. This work focuses on RESTful WSNs. It describes CoAP, highlights the main differences with HTTP and reports the results of a simple experiment showing the benefits of CoAP in terms of power consumption compared to HTTP. The paper also describes the design and development of an end-to-end IP based architecture integrating a CoAP over 6LowPAN Contiki based WSN with an HTTP over IP based application. The application allows a user to access WSN data directly from a Web browser. The main system's building blocks and functionalities are described.

Categories and Subject Descriptors
C.2.2 [**Computer-Communication Networks**]: Network Protocols – *Applications.*

General Terms
Performance, Design, Standardization.

Keywords
Web applications, Web of Things, REST, CoAP.

1. INTRODUCTION
Recent advances in Wireless Sensor Network (WSN) technology and the use of the Internet Protocol (IP) in resource constrained devices has radically changed the Internet landscape. Trillions of smart objects will be connected to the Internet to form the so called Internet of Things (IoT). The IoT will connect physical (analogic) environments to the (digital) Internet, unleashing exciting possibilities and challenges for a variety of application domains, such as smart metering, e-health logistics, building and home automation [7].

The use of IP technology on embedded devices has been recently promoted by the work of the IP for Smart Objects (IPSO) Alliance[1], a cluster of major IT/telecom players and wireless silicon vendors. At the same time, the Internet Engineering Task Force (IETF) has done substantial standardization activity on IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [8]. This new standard enables the use of IPv6 in Low-power and Lossy Networks (LLNs), such as those based on the IEEE 802.15.4 standard [10]. In addition to 6LowPAN, IETF Routing over Low-power and Lossy networks (ROLL) Working Group has designed and specified a new IP routing protocol for smart object internetworking. The protocol is called IPv6 Routing Protocol for Low-power and Lossy networks (RPL) [9].

One of the major benefits of IP based networking in LLNs is to enable the use of standard web service architectures without using application gateways. As a consequence, smart objects will not only be integrated with the internet but also with the Web. This integration is defined as the Web of Things (WoT). The advantage of the WoT is that smart object applications can be built on top Representational State Transfer (REST) architectures. REST architectures allow applications to rely on loosely coupled services which can be shared and reused. In a REST architecture a resource is an abstraction controlled by the server and identified by a Universal Resource Identifier (URI). The resources are decoupled by the services and therefore resources can be arbitrarily represented by means of various formats, such as XML or JSON. The resources are accessed and manipulated by an application protocol based on client/server request/responses. REST is not tied to a particular application protocol. However, the vast majority of REST architectures nowadays use Hypertext Transfer Protocol (HTTP). HTTP manipulates resources by means of its methods *GET*, *POST*, *PUT*, etc [6].

REST architectures allow IoT and Machine-to-Machine (M2M) applications to be developed on top of web services which can be shared and reused. The sensors become abstract resources identified by URIs, represented with arbitrary formats and manipulated with the same methods as HTTP. As a consequence, RESTful WSNs drastically reduce the application development complexity.

The use of web service in LLNs is not straightforward as a consequence of the differences between Internet applications and IoT or M2M applications. IoT or M2M applications are short-lived and web services reside in battery operated devices which most of the time sleep and wakeup 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 of standard Internet applications [11].

The Internet Engineering Task Force (IETF) Constrained RESTful environments (CoRE) Working Group has done major standardization work for introducing the web service paradigm into networks of smart objects. The CoRE group has defined a REST based web transfer protocol called Constrained Application Protocol (CoAP). CoAP includes the HTTP functionalities which have been re-designed taking into account the low processing power and energy consumption constraints of small embedded

[1] http://ipso-alliance.org/

devices such as sensors. In order to make the protocol suitable to IoT and M2M applications, various new functionalities have been added [12].

With 6LoWPAN technology becoming mature, the WoT has started playing major role among the research community. Various research papers proposing REST/HTTP architectures for WSNs have recently appeared. The work in [1] proposes a RESTful architecture which allows instruments and other producers of physical information to directly publish their data. In [2], the authors propose a REST/HTTP framework for Home Automation. The work in [3] proposes a toolkit which allows the user to create web services provided by a specific device and to automatically expose them via a REST API. The authors in [4] show how different applications can be built on top of RESTful WSNs. The work in [5] illustrates the real world implementation of a RESTful WSN. The network is deployed across various university buildings and it is thought for the development of applications and services for professors and students.

The aforementioned research work focuses on RESTful WSNs but do not use CoAP as application protocol. The activity of the CoRE group has only recently started and therefore CoAP has not yet been considered.

In this work we present a RESTful WSN based on CoAP. It has twofold objective. Firstly, it describes the major differences between CoAP and HTTP and compares the two protocols in terms of power consumption and overhead. In order to demonstrate the benefits of CoAP, we ran two simple experiments with the Contiki Operating System: the first one using CoAP over 6LoWPAN and the second one using HTTP over 6LoWPAN. The results show that the power consumption is drastically lower when using CoAP compared to HTTP.

Secondly, the paper describes the design and development of an end-to-end IP based architecture integrating a CoAP over 6LowPAN Contiki based WSN with an HTTP over IP based application. The application allows a user to access WSN data directly from a Web browser. The system has been designed for Greenhouse monitoring. However, it is work in progress and it has not yet been deployed. Therefore, the aim of the paper is to show how the use of CoAP and 6LoWPAN simplifies the integration of WSNs with Web applications. The paper provides an overview of the basic application building blocks focusing on the gateway which connects HTTP clients to the WSN.

The rest of the paper is organized as follows. Section 2 describes the major functionalities of CoAP highlighting the differences with HTTP. It also gives a brief overview of the existing open source implementation of CoAP. Section 3 reports the results of an experiment illustrating the benefit of CoAP in terms of power consumption compared to HTTP. Section 4 describes the design and development of an end-to-end IP based architecture integrating a CoAP over 6LowPAN Contiki based WSN with an HTTP over IP based application. Section 5 concludes the paper.

2. Constrained Application Protocol

In March 2010, the IETF CoRE Working Group has started the standardization activity on CoAP. CoAP is a web transfer protocol optimized for resource constrained networks typical of IoT and M2M applications. CoAP is based on a REST architecture in which resources are server-controlled abstractions made available by an application process and identified by Universal Resource Identifiers (URIs). The resources can be manipulated by means of the same methods as the ones used by HTTP: GET, PUT, POST and DELETE.

CoAP is not a blind compression of HTTP. 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 sensors. In addition, various mechanisms and have been modified and some new functionalities have been added in order to make the protocol suitable to IoT and M2M applications. The HTTP and CoAP protocol stacks are illustrated in Figure 1.

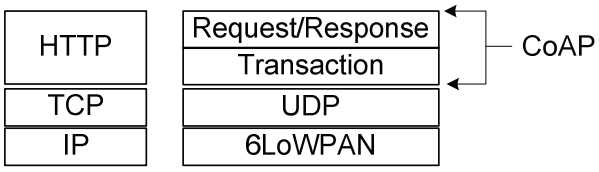

Figure 1. HTTP and CoAP protocol stacks

The first significant difference between HTTP and CoAP is the transport layer. HTTP relies on the Transmission Control Protocol (TCP). TCP's flow control mechanism is not appropriate for LLNs and its overhead is considered too high for short-lived transactions. In addition, TCP does not have multicast support and is rather sensitive to mobility. CoAP is built on top of the User Datagram Protocol (UDP) and therefore has significantly lower overhead and multicast support.

CoAP is organized in two layers. The Transaction layer handles the single message exchange between end points. The messages exchanged on this layer can be of four types: *Confirmable* (it requires an acknowledgment), *Non-confirmable* (it does not need to be acknowledged), *Acknowledgment* (it acknowledges a Confirmable message) and *Reset* (it indicates that a Confirmable message has been receive but context is missing to be processed). The Request/Response layer is responsible for the transmission of requests and responses for the resource manipulation and transmission. This is the layer where the REST based communication occurs. A REST request is piggybacked on a Confirmable or Non-confirmable message, while a REST response is piggybacked on the related Acknowledgment message.

The dual layer approach allows CoAP to provide reliability mechanisms even without the use of TCP as transport protocol. In fact, a Confirmable message is retransmitted using a default timeout and exponential back-off between retransmissions, until the recipient sends the Acknowledgement message. In addition, it enables asynchronous communication which is a key requirement for IoT and M2M applications. When a CoAP server receives a request which is not able to handle immediately, it first acknowledges the reception of the message and sends back the response in an off-line fashion. Tokens are used for request/response matching in asynchronous communication.

The transaction layer also provides support for multicast and congestion control [14].

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 overhead. This implies packet fragmentation and consequent performance degradation of LLNs. 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. Next Section shows the significant advantage of the compact overhead of CoAP in terms of battery lifetime with respect to HTTP.

Since a resource on a CoAP server likely changes over time, the protocol allows a client to constantly observe the resources. This is done by means of observations: the client (the observer) registers itself to the resource (the subject) by means of a modified GET request sent to the server. The server establishes an observation relationship between the client and the resource. Whenever the state of the resource changes, the server notifies each client having an observation relationship with the resource. The duration of the observation relationship is negotiated during the registration procedure [13].

Although CoAP is work in progress, various open source implementations are already available. The two most known operating systems for WSNs, Contiki and Tiny OS, have already released a CoAP implementation. In addition, there are two open source implementations not specifically designed for WSNs: an implementation in C language called *libcoap*[2] and one in Python language called *CoAPy*[3].

3. CoAP power consumption evaluation

The use of UDP as transport protocol and the reduction of the packet header size significantly improve the power consumption and battery lifetime in WSNs. In order to evaluate the performance improvement of CoAP compared to HTTP, we executed a simple experiment. We generated a series of web service requests first between a CoAP client/server system and then between an HTTP client/server system.

The CoAP server is implemented by means of a Tmote Sky sensor mote running Contiki with 6LoWPAN/RPL on the network layer and CoAP on the application layer. The CoAP implementation of Contiki already includes many features of the protocol, such as message syntax and semantics, methods, response codes, option fields, URIs and resource discovery. However, being work in progress there are still important functionalities missing such as asynchronous transactions, observations and congestion control. The CoAP client is implemented by running libcoap on a Linux Ubuntu machine with a USB Contiki-gateway which interfaces with the WSN. This is one of the basic building blocks of the gateway described in Section 4.

The HTTP server is obtained with the same Tmote Sky platform as in the CoAP server and Contiki loaded with the HTTP server instead of the CoAP server. The HTTP client is obtained by replacing *libcoap* with *cURL*[4], a command line program including HTTP functionalities. In both experiments the client polls the server every 10 seconds for 20 minutes by requesting temperature and humidity. When using CoAP the request has the following format: *GET coap:///[<mote_ip_address>]:<port_number>/readings,* where *mote_ip_address* is the mote's IPv6 address, *port_number* is the mote's port number and *readings* is the resource the client is requesting for (in this case temperature and humidity). When using HTTP the request has the following format: *GET http:///[<moteip_address>]:<port_number>/readings* where the parameters have the same meaning as when using CoAP.

In both CoAP and HTTP cases, the server responds by sending the sensor readings embedded into a Java Script Object Notation (JSON) file. JSON is a lightweight text based open standard for data client/server data exchange. An example of the response's payload is the following: *{"sensor":" 0212:7400:0002:0202 ","readings":{"hum":31,"temp":23.1}}*, where the sensor is recognized by the last four groups of its IPv6 address, *hum* is the humidity resource and *temp* is the temperature resource.

CoAP also supports other payload encoding standards such as the widely used Extensible Markup Language (XML). However, the verbosity and parsing complexity of XML makes this language not appropriate for constrained devices. Although the compact data representation in JSON is more suitable for WSNs, JSON does not have the flexibility of XML. As a consequence, there has been significant effort to develop binary XML based representations such as the Extensible XML Interchange (EXI) [6].

Table 1 illustrates the results of the comparison between CoAP and HTTP in terms of byte transferred per transaction, power consumption and battery lifetime.

The power consumption has been calculated by means of Energest, a tool able to estimate the power consumption of Tmote Sky motes [15]. The results have been taken in steady state conditions.

Table 1. Comparison between CoAP and HTTP

	Bytes per-transaction	Power	Lifetime
CoAP	154	0.744 mW	151 days
HTTP	1451	1.333 mW	84 days

As illustrated in Table 1, an HTTP transaction has a number of bytes 10 times larger than the CoAP transaction. This is a consequence of the significant header compression executed in CoAP. In fact, as discussed in Section 2, CoAP uses a short fixed-length compact binary header of 4 bytes and a typical request has a total header of about 10-20 bytes. After being encapsulated in the UDP, 6LoWPAN and MAC layer headers, the CoAP packet can be transfer into a single MAC frame which has a size of 127 bytes.

It is straightforward that the higher number of bytes transferred in an HTTP transaction implies a more intensive activity of the mote's transceiver and CPU and consequently higher power consumption (1.33 mW in HTTP against 0.74 mW in CoAP). In both experiments, the server mote was powered with 2 AA Zinc-carbon. The figures of the power consumption lead to an estimation of the battery lifetime of 84 days in HTTP and 151 in CoAP. Note that the battery lifetime in both cases is unrealistically short as a consequence of the high number of client requests generated during the experiment.

It is worth underlining that the results presented in this paragraph do not exhaustively compare the two protocols. The simple experiment presented is only intended to illustrate how the UDP binding and the header compression introduced in CoAP improve the power consumption of WSNs.

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

[3] http://coapy.sourceforge.net/

[4] http://curl.haxx.se/

4. Integration of a CoAP based WSN with a Web application

The use of an IP based communication and a REST based Web architecture in LLNs facilitates the integration of WSNs with Internet based Web applications. This Section describes the design and development of an end-to-end IP based architecture integrating a CoAP over 6LowPAN Contiki based WSN with an HTTP over IP based application. The aim of the application is to allow a user to access WSN data directly from a Web browser, as illustrated in Figure 2.

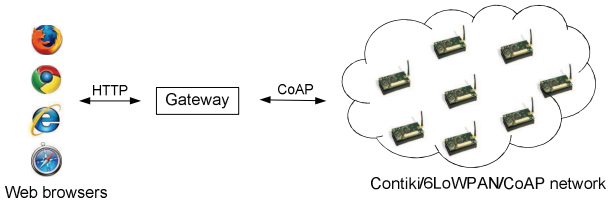

Figure 2. Integration between WSNs and the Web

The system has been designed for experimental greenhouse monitoring. However, it is work in progress and it has not yet been deployed. The aim of this Section is to illustrate the building blocks and functionalities of the first basic prototype. A key component of the system is the gateway described in paragraph 4.1.

4.1 Gateway design and development

When the sensor's resources are exposed by the device itself with an application protocol like CoAP, the gateway's complexity is significantly reduced with compared to the case in which the sensor's resources are exposed by an application gateway. In fact, an application gateway needs to have full knowledge of the functionalities of each connected device. This reduces the architecture flexibility and hampers the system scalability. This is one of the major problems of non IP based WSN communication standards such as ZigBee. ZigBee does not have a standard IP networking layer which implies that a standard web service architecture cannot be implemented on top of ZigBee. Besides hampering the interoperability, the lack of a web service architecture requires the use of application gateways when interconnecting ZigBee WSNs to the Internet.

In this work, the gateway integrating the CoAP based WSN with the HTTP based Web application consists of a Linux Ubuntu machine with a Contiki-gateway attached via USB port. The main building blocks of the software running on the gateway are illustrated in Figure 3.

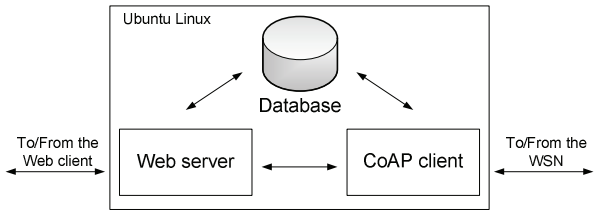

Figure 3. Gateway's main building blocks

As illustrated in Figure 3, the main gateway's building blocks are the web server, the database and the CoAP client. For simplicity, the first gateway prototype includes all the building blocks in the same box. In a second phase the application will be deployed on an application server. Therefore the application logic and the data collection functionality will reside into two different machines. This clearly reduces the complexity of the gateway and improves the system scalability.

The web server includes a set of services which are used to retrieve data either from the database or directly from the CoAP client. When the Web server sends the Web client historical data already available in the database, the web server directly accesses the database without communicating with the CoAP client and sends the data back to the Web client. When the Web Server needs to send fresh data coming from the WSN (upon client request or upon changes of the WSN resources), the web server bypasses the database and directly communicates with the CoAP client. Since the Web application has been developed with Google Web Toolkit (GWT), the web server at the moment is the built-in GWT's server called Jetty. When the application will be deployed on an application server a different web server will be chosen.

The database stores data coming from the CoAP client and makes them available to the web server. The database chosen is Apache CouchDB[5]. Apache CaouchDB is a document-oriented database which can be queried and indexed with MapReduce technique using JavaScript. It stores JSON documents and provides a RESTful API. Since the CoAP client receives WSN data already in JSON documents, the storage operation is rather simple and does not require any intermediate data manipulation. However, the system has not yet been tested under high frequency measurement conditions and therefore the database scalability has not yet been evaluated. If a high number of stored documents results in slow database access, an extra data processing layer might be needed in order to reduce the data accesse latency.

The *libcoap* CoAP client module is responsible for communicating with the WSN. In the current prototype the gateway-WSN data exchanges are always initiated by the CoAP client. This is a consequence of the fact that Contiki does not yet support observations. We are currently adding this functionality so that the WSN can spontaneously send the CoAP client data upon resource status change. Once retrieved the JSON data from the WSN, the CoAP client add a time stamp and stores them into the database. The time stamp is needed when providing the web server with historical data.

For simplicity, the current gateway implementation does not include proxy functionality between HTTP and CoAP. Therefore there is not translation between the HTTP request and the CoAP request and vice versa. Upon receiving the HTTP request, the web server invokes the CoAP client with the parameters included in the HTTP requests (IPv6 address and port of the mote and the resource of interest). This implies that the gateway is not completely transparent to the application and to the WSN. A proxy module able to do the HTTP-CoAP translation and vice versa needs to be implemented in order to increase the transparency of the gateway. This will also facilitate the gateway in handling more complicated operations such as observations. In this case for example, a mechanism that translates an HTTP subscription (e.g. long-polling) needs to be translated in a CoAP observation relationship. There is at the moment an ongoing discussion in the CoRE group to decide on issues related to HTTP-CoAP mapping [16].

[5] http://couchdb.apache.org/

5. Conclusions

This paper discussed the integration of WSNs with the Web. This is being facilitated by the development of CoAP, an IETF protocol providing LLNs with a RESTful architecture. CoAP offers the same methods for the resource manipulation as HTTP. In addition, CoAP supports additional functionalities typical of IoT and M2M applications, such as multicast, asynchronous communication and subscriptions. Unlike HTTP, CoAP is built on top of UDP and has a compact packet overhead. The paper illustrated how the introduction of UDP and the packet overhead compression drastically reduce the mote's power consumption and consequently increase the battery lifetime. The paper also described the design and development of an end-to-end IP based architecture integrating a CoAP over 6LowPAN Contiki based WSN with an HTTP over IP based application. The application allows a user to access WSN data directly from a Web browser. The paper described the main building blocks of the gateway connecting the Web client with the WSN. The gateway is still in prototype phase and it requires the development of proxy and observation functionalities. The database performance needs to be tested for scalability purpose. The application will be deployed and tested in a greenhouse monitoring testbed.

6. ACKNOWLEDGMENTS

This work has been done in the scope of the ITEA project Interoperable Sensor Networks (ISN) in collaboration with the company Freemind[6]. The authors acknowledge IWOIB for their financial support.

7. REFERENCES

[1] Dawson-Haggerty, S., et al. sMAP – a Simple Measurement and Actuation Profile for Physical Information. In *Proceedings of 8th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2010.

[2] Kovatsch, M., et al. Embedding Internet Technology for Home Automation. In *Proceedings of IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2010.

[3] Mayer, S., et al. Facilitating the Integration and Interaction of Real-World Services for the Web of Things. In *Proceedings of Urban Internet of Things – Towards Programmable Real-time Cities (UrbanIOT)*, 2010.

[4] Guinard, D., et al. A Resource Oriented Architecture for the Web of Things. In *Proceedings of Internet of Things 2010 International Conference (IoT)*, 2010.

[5] Castellani, A. P., et al. Architecture and Protocols for the Internet of Things: A Case Study. In Proceedings of First International Workshop on the Web of Things (WoT), 2010.

[6] Shelby, Z. Embedded Web Services. *IEEE Wireless Communications*, pp. 52-57, December 2010.

[7] Atzori, L., et al. The Internet of Things: A survey. *Computer Networks*, pp. 2787-2805, October 2010.

[8] Kushalnagar, N, IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919.

[9] Vasseur, J. P., and Dunkels, A., Interconnecting Smart Objects with IP: The Next Internet. Morgan Kaufmann, 2010.

[10] Shelby, Z., and Bormann, C., 6LoWPAN: The Wireless Embedded Internet, Wiley, 2009.

[11] Trifa, V., et al. Design and Implementation of a Gateway for Web-based Interaction and Management of Embedded Devices. In *Proceedings of the 2nd International Workshop on Sensor Network Engineering (IWSNE)*, 2009.

[12] Shelby, Z., et al. Constrained Application Protocol (CoAP). *Internet-Draft*. draft-ietf-core-coap-04.

[13] Hartke, K., et al. Observing Resources in CoAP. *Internet-Draft*. draft-ietf-core-observe-01.

[14] Eggert, L., Congestion Control for the Constrained Application Protocol (CoAP). *Internet-Draft.* draft-eggert-core-congestion-control-01.

[15] Dunkels, A., et al. Demo abstract: Software-based sensor node energy estimation. In *Proceedings of the Fifth ACM Conference on Networked Embedded Sensor Systems (SenSys)*, 2007.

[16] Castellani, A., et al. Best Practice to map HTTP to COAP and viceversa. *Internet-Draft.* draft-castellani-core-http-coap-mapping-00.txt.

[6] http://www.freemind-group.com/fmc2/