Evaluation of Constrained Application Protocol for Wireless Sensor Networks

Walter Colitti[(1)], Kris Steenhaut[(1)(2)], Niccolò De Caro[(1)], Bogdan Buta[(3)], and Virgil Dobrota[(3)]

[(1)]Vrije Universiteit Brussel, Dept. of Electronics and Informatics (ETRO), Pleinlaan 2, 1050, Brussels, Belgium
[(2)]Erasmushogeschool Brussel, IWT Dept., Nijverheidskaai 170, 1070, Brussels, Belgium
[(3)]Technical University of Cluj-Napoca, 26-28 Baritiu Street, 400027 Cluj-Napoca, Romania

Email:{wcolitti, ksteenha, ndecaro}@etro.vub.ac.be Bogdan.Buta@net.utcluj.ro Virgil.Dobrota@com.utcluj.ro

***Abstract*—IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) and IPv6 Routing Protocol for Low-power and Lossy networks (RPL) have accelerated the integration of Wireless Sensor Networks (WSNs) and smart objects with the Internet. At the same time, the development of the Constrained Application Protocol (CoAP) has made it possible to provide resource constrained devices with web service functionalities. CoAP is an HTTP like web transfer protocol able to extend the REpresentational State Transfer (REST) architecture to LoWPANs. The major difference between CoAP and HTTP is the different transport layer protocol (UDP instead of TCP) and the header compression which makes the packet size significantly smaller. This work provides an evaluation of CoAP compared to HTTP. The performance is evaluated in terms of mote's energy consumption and response time. The aim of the paper is to demonstrate, with a quantitative and qualitative analysis, that CoAP is more suited to REST based WSNs compared to HTTP. The results have been obtained by means of simulation as well as tests on real sensor motes.**

***Index Terms*—Web of Things, REST, CoAP, HTTP, energy consumption**

I. 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 environments to the Internet, unleashing exciting possibilities and challenges for a variety of application domains, such as smart metering, e-health, logistics, building and home automation [9].

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) [10]. 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 [12]. 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) [11].

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 from the services and therefore resources can be arbitrarily represented by means of various formats, such as Extensible Markup Language (XML) or Java Script Object Notation (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 [8].

REST architectures allow IoT and Machine-to-Machine (M2M) applications to be developed on top of shareable and reusable web services. 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 services 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 wake up only when there is

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

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 [13].

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 several HTTP functionalities re-designed for small embedded devices such as sensor nodes. In order to make the protocol suitable to IoT and M2M applications, various new functionalities have been added [14].

With 6LoWPAN technology becoming mature, the WoT has started playing a major role in 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 a real world implementation of a RESTful WSN. The network is deployed across various university buildings and it is designed for the development of applications and services for the university community.

The aforementioned research work focuses on RESTful WSNs but does not use CoAP as application protocol. The activity of the CoRE group has only recently started and therefore CoAP has not yet been largely considered. To the best of our knowledge, only the work in [6] and [7] have already used CoAP as application protocol for WSNs. However, the protocol is not extensively evaluated and a clear comparison with HTTP is not provided.

The aim of this work is to show that CoAP is more appropriate for WSNs compared to HTTP. It discusses the major differences between CoAP and HTTP and compares the two protocols. The comparison considers the number of bytes transferred in a client-server transaction, the energy consumption of the server mote and the response time.

The rest of the paper is organized as follows. Section II 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 III discusses the experiments executed to compare CoAP and HTTP and reports the results. Section IV concludes the paper.

II. 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 being 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 sensor motes. 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.

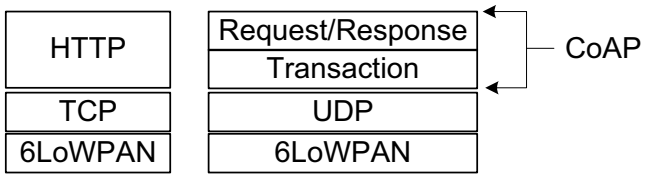

Figure 1. HTTP and CoAP protocol stacks.

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. 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 it 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 [16].

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 an associated performance degradation in the context 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. The next section shows the significant advantage of the compact overhead of CoAP, in terms of energy consumption and response time, with respect to HTTP. The CoAP message format is illustrated in

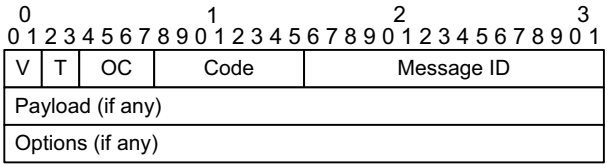

Figure 2. CoAP message format.

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 respective resource. The duration of the observation relationship is negotiated during the registration procedure [15].

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 three other open source implementations not specifically designed for WSNs: an implementation in C language called *libcoap*[2], one in Python language called *CoAPy*[3] and one in Java called *jcoap*[4].

III. CoAP Evaluation

This section reports on the performance evaluation of CoAP. In order to evaluate the benefits of CoAP for WSNs, we compare its performance to HTTP. The goal of the experiments is to show, via a quantitative analysis, how the lower number of bytes transferred in a CoAP client-server transaction reduce the energy consumption and the response time.

In order to evaluate the performance of the two protocols, the same experiments have been executed on a CoAP client-server pair and on an HTTP client-server pair.

The CoAP server is implemented by means of a 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.

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*[5], a command line program including HTTP functionalities.

In the CoAP and HTTP client-server scenarios, the client queries the server at regular intervals and requests 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 (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 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 digits 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.

[2] http://libcoap.sourceforge.net/
[3] http://coapy.sourceforge.net/
[4] http://code.google.com/p/jcoap/
[5] http://curl.haxx.se/

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) [8].

The experiments have been executed on two different client-server scenarios. The evaluation of the bytes transferred and of the response time has been done on Zolertia Z1 motes with embedded temperature sensor. The evaluation of the energy consumption has been executed on Tmote Sky motes with embedded temperature and humidity sensors. The Tmote Sky motes have been simulated using Cooja, a simulator for Contiki based motes [19]. The motivation of using also a simulation scenario is that quantifying the energy consumption on a real mote is rather fastidious [18]. Instead, the Cooja simulator provides a module called Energest able to estimate the power consumption of Tmote Sky motes [17]. Energest provides information about the power consumption for transmission, reception, processor and idle state energy consumption.

A. Bytes transferred per client-server transaction

In order to quantify the amount of data transferred when using CoAP and HTTP, we analyzed a client-server transaction and quantified the amount of byte transferred. The transaction starts when the client initiates the resource request and ends when the client gets the response with the resource. The transaction has been analyzed by means of Wireshark[6], a network protocol analyzer able to capture and analyze the traffic running on a computer network. Table 1 reports the number of bytes and packets transferred per transaction for CoAP and HTTP.

Table 1. Bytes transferred per transaction in CoAP and HTTP

Protocol	**CoAP**	**HTTP**
Bytes per transaction	184	1288
Packets per transaction	2	17

As illustrated in Table 1, an HTTP transaction has a number of bytes and packets significantly larger than the CoAP transaction. This is a consequence of the header compression executed in CoAP. In fact, as discussed in Section II, 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 transferred into a single MAC frame which has a size of 127 bytes.

B. Energy consumption

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 in terms of energy consumption, we simulated a series of web service requests; firstly between a CoAP client and server and secondly between an HTTP client and server. The two scenarios have been simulated in Cooja.

In order to test CoAP and HTTP in terms of energy consumption, we generated a series of client requests with a certain interarrival time during 30 minutes long experiments. In a first experiment we fixed the client request interarrival time to 10 seconds and we calculated the energy consumption of the server motes in 4 different operation modes. The first mode is while the server mote receives packets, indicated as RX mode; the second mode is while the server mote transmits packets, indicated as TX mode; the third mode is while the server mote is in idle mode, indicated as Low Power Mode (LPM); the fourth mode is while the server mote's Central Processing Unit (CPU) processes packets, indicated as CPU mode. The energy consumption of the CoAP and HTTP server mote in the different operation modes are shown in Figure 3.

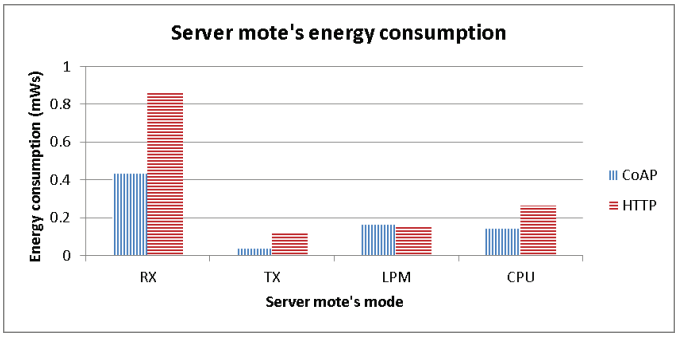

Figure 3. Energy consumption of the CoAP and HTTP server mote in each of the observed operation modes.

As illustrated in Figure 3, the higher number of bytes transferred in an HTTP transaction (see Table 1) implies more time spent by the mote in receiving, transmitting and processing packets and consequently higher power consumption in RX, TX and CPU modes. While receiving packets (in RX mode), the energy consumed when using CoAP is approximately half compared to the one consumed when using HTTP (0.43 mWs for CoAP and 0.85 mWs for HTTP). The same happens when processing packets (CPU mode): the energy needed for processing CoAP packets (0.14 mWs) is roughly half of the energy needed to process HTTP packets (0.26 mWs). While transmitting packets (TX mode), the energy required by CoAP is 4 times lower than the energy required by HTTP (0.03 mWs for CoAP and 0.12 mWs for HTTP). While being in idle mode (LPM mode), CoAP and HTTP result in similar value of energy consumption (0.159 mWs for CoAP and 0.155 mWs for HTTP). This is a consequence of the fact that in in both scenarios we use the same MAC protocol (ContikiMAC) and therefore the time spent in idle mode is approximately the same.

In a second experiment, we executed 5 tests each one with a different value of the client request interarrival time (5, 10, 30, 60 and 120 seconds). Each experiment ran for 30 minutes. By choosing a fixed length of the simulation, each experiment has a different total number of generated client requests (360, 180,

[6] http://www.wireshark.org/

60, 30 and 15, respectively). This is motivated by the fact that we wanted to analyse the impact of the number of requests on the energy consumption. In fact, the results demonstrate that the lower the client request interarrival time, the higher the difference of the energy consumption values.

The results have been taken in steady state conditions. In fact, the first 5 minutes of simulation are not taken into consideration because the high quantity of control messages exchanged between client and server might alter the results. Figure 4 illustrates the outcome of this experiment.

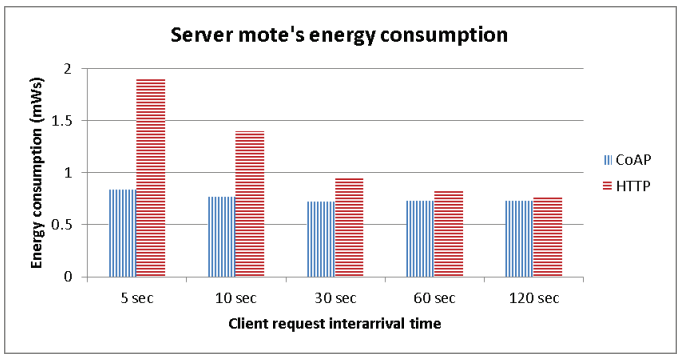

Figure 4. Energy consumption of the CoAP and HTTP server motes for different values of the client request interarrival time.

As illustrated in Figure 4, when the interarrival time of client requests is low (5 seconds) the average energy consumption of CoAP is drastically lower compared to HTTP (0.83 mW for CoAP and 1.9 mW for HTTP). This is a consequence of the high number of client requests initiated during the 30 minutes experiment (360 requests for a client request interarrival time of 5 seconds). The difference is still significant (0.76 mW for CoAP and 1.4 mW for HTTP) when the client request interarrival time is 10 yielding 180 requests in total. With a client request interarrival time of 30 seconds the difference starts being less significant (0.71 mW for CoAP and 0.96 for HTTP) as a consequence of the lower number of client-server transactions (60). The fact that the lower number of client-server transactions entails a lower difference between CoAP and HTTP energy consumption is confirmed by the results related to a client request interarrival time of 60 seconds and 120 seconds. In case of 60 seconds, the number of client-server transaction is 30 and the resulting energy consumption is 0.72 mW for CoAP and 0.81 mW for HTTP. In case of a request interarrival time of 120 seconds the number of transactions is 15 and the resulting energy consumption is 0.72 mW for CoAP and 0.76 mW for HTTP.

The results in Figure 4 demonstrate that the higher the number of client-server transactions, the higher the difference in average energy consumption. This is motivated by the fact that an HTTP transaction implies a more intensive activity of the mote's transceiver and CPU and consequently higher energy consumption (as demonstrated in Figure 3). In addition, Figure 4 shows that the average energy consumption of CoAP remains approximately constant when varying the client request interarrival time. Even when the client request interarrival time is 5 seconds and number of client-server transactions is 360, the average energy consumption of CoAP does not significantly increase as it happens for HTTP. This is a consequence of the fact that, as illustrated in Figure 3, the server mote spends more time in receiving packets and consequently the energy consumption in the RX mote is higher in CoAP than in HTTP. Therefore, from the analysis of the results in Figure 4, we can conclude that the energy consumption in CoAP is not significantly sensitive to the increase in the number of client-server transactions. This characteristic is particularly useful when considering that WSNs are used in various fields of applications with different requirements. There are applications requiring frequent access to the sensor data (for data consistency, integrity, security, etc.). If such applications transferred and manipulated data by means of HTTP the energy consumption would strongly impact the performance of the network. Instead, the use of CoAP guarantees that the energy consumption does not significantly increase when sensor data are accessed more frequently.

C. Response time

When accessing sensor resources via web services it is important to keep the response time at low values. A high response time might have an impact on the performance of certain applications which have strict requirements in terms of latency. In addition, the response time impacts the user experience (i.e. a user trying to access quasi real time sensor data from a web browser).

The aim of the experiment presented in this paragraph is to compare the response time of two client-server systems using CoAP and HTTP and show how the use of UDP as transport protocol and the reduction of the packet header size significantly reduce the response time.

The response time is defined as the time elapsing from the moment the client initiates the request till the moment the client receives the response with the payload. The experiment has been executed on a client-server pair consisting of Zolertia Z1 motes. In order to evaluate the impact of the number of hops on the response time, a 1-hop and 2-hop scenario have been set up. In the 1-hop scenario, there is a direct connection between client and server mote (similar to the test-bed used for the experiment described in paragraph A). In the 2-hop scenario, the server mote is not in range of the client and the data has to be routed via an intermediate mote. Both scenarios have been tested with 55 CoAP and HTTP requests. The results are illustrated Figure 5.

As illustrated in Figure 5, there is significant difference in the response time for CoAP compared to HTTP. With one hop, the average response time calculated for the 55 client requests is 184 milliseconds for CoAP and 1.8 seconds for HTTP. With two hops, CoAP experiences an average response time of 336 milliseconds while HTTP 3.2 seconds. Likewise the energy consumption, the the drastic improvement in the response time experienced by CoAP is a consequence of the use of UDP as transport protocol and of the reduction of the packet header size.

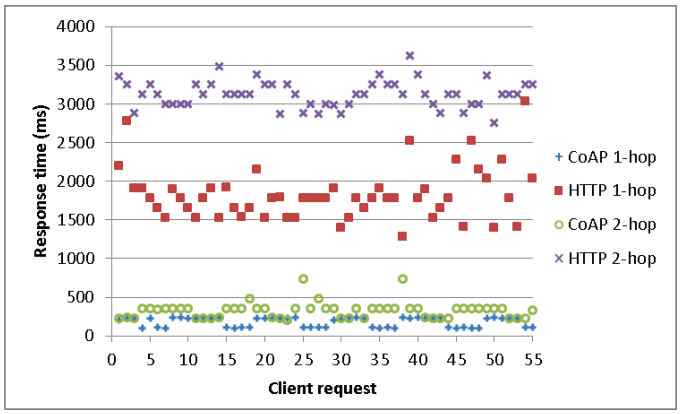

Figure 5. Response time of CoAP and HTTP in the 1-hop and 2-hop scenario.

If we consider the ratio between the values of the response time experienced with two hops and one hop, HTTP does not undergo performance degradation. In fact, the value of the ratio is 1.8 for CoAP and 1.7 for HTTP and therefore adding one hop in the path between client and server does not impact the response time. However, the values of the response time resulting when using HTTP in the 2-hop scenario are significantly high and unacceptable for many applications.

IV. Conclusion

This work 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. This has a considerable impact on the mote's energy consumption and response time. The paper quantified, by means of simulations as well as tests on real motes, the benefits of CoAP in terms of energy consumption and response time compared to HTTP. We showed how the compact header and small packet size allow a CoAP server to spend less time in receiving, transmitting and processing packets and therefore to consume a significantly lower amount of energy compared to HTTP. In addition, we demonstrated that CoAP is not particularly sensitive to the increase of client request intererrival time. This is an important advantage because there are certain applications which require frequent access to WSN data. We also showed that CoAP has a significantly lower response time compared to HTTP. We also evaluated the impact of adding one hop to the communication path and showed that the response time values are still low for CoAP while they become unacceptable for HTTP.

Acknowledgment

This work has been done in the scope of the ITEA project Interoperable Sensor Networks (ISN) in collaboration with the company Freemind[7]. The authors acknowledge INNOVIRIS Brussels[8] for their financial support.

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] Duquennoy, S., et al., Leveraging IP for Sensor Network Deployment, In *Proceedings of Extending the Internet to Low Power and Lossy Networks (IP+SN), 2011.*

[7] Kuladinithi, k., Implementation of CoAP and its Application in Transport Logistics, In *Proceedings of Extending the Internet to Low Power and Lossy Networks (IP+SN), 2011.*

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

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

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

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

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

[13] 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.

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

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

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

[17] 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.

[18] Hurni, P., et al. On the Accuracy of Software-Based Energy Estimation Techniques. In *Proceedings of the 8th European Conference on Wireless Sensor Networks (EWSN),* 2011.

[19] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with COOJA," in *Proc. 31th Conference on Local Computer Networks*, Tampa, Florida (USA), 2006, pp. 641-648.

[7] http://www.freemind-group.com/fmc2/
[8] http://www.irsib.irisnet.be/