SCoAP: An Integration of CoAP Protocol With Web-based Application

Nam K Giang, Minkeun Ha, Daeyoung Kim
Department of Computer Science
Korea Advanced Institute of Science and Technology
Email: {zang,minkeun.ha,kimd}@kaist.ac.kr

***Abstract*—This paper proposes an interesting yet practical use case of CoAP, an important application protocol in realizing the Internet of Things (IoT) vision. Originally, CoAP protocol is designed to communicate between embedded devices, however its applications can spread well over the Internet by exposing the devices' capabilities to the Web as Web resources. Thus, we claim that CoAP protocol could also be used in web-based application to leverage CoAP-based resources. Unfortunately, the design of CoAP protocol, for example, bind to UDP socket to reduce the package size or bidirectional communication to cope with duty-cycle power scheme, prevents it from being supported on conventional web browsers. Additionally, translation of CoAP into HTTP protocol via HTTP/CoAP proxy as recommended by IETF is not the best solution since many CoAP features are limited. Instead, new bidirectional web protocol such as HTML5 Web Socket would help to preserve the protocol's characteristics. We propose a solution called SCoAP that facilitates true CoAP communication in conventional web browsers by applying HTML5 Web Socket protocol. Experiment results have shown significant advantages of SCoAP solution over standard HTTP/CoAP proxy in term of network traffic and computational demand.**

***Index Terms*—Internet of Things, Web of Things, CoAP, IP-WSN, HTML5 Web Socket**

I. Introduction

In the IoT perspective, physical objects such as home appliances and embedded devices are supposed to be connected to the Internet so that they can be monitored and controlled from the Web. Since the embedded devices are often resource-constrained, a large number of technologies and network protocols have been proposed, ranging from physical link layer to application layer. One of these protocols is the Constrained Application Protocol (CoAP), an important application protocol for machine-to-machine (M2M) communication.

Briefly, CoAP protocol inherits much from HTTP protocol, the protocol that runs the World Wide Web. That is, similar to HTTP, CoAP also follows the RESTful[1] client-server model with GET, POST, PUT, DELETE requests and 2.xx, 4.xx, 5.xx responses. However, since CoAP protocol is designed for resource-constrained physical objects, there are two fundamental differences between these protocols. First, CoAP supports bidirectional communication while HTTP is just a client-initiate protocol. Second, CoAP protocol runs on top of UDP transportation in order to reduce the package size and uses layered approach to provide reliability while HTTP's transportation protocol is TCP.

Although CoAP is mainly used in M2M communication, it can also be used to expose CoAP-based resources to the Web. To date, there are two approaches in accessing a CoAP server from the Web: users either use a CoAP client (CoAP browser) or use a web browser with a HTTP/CoAP proxy. While CoAP client implementations in native language such as C, Java or Python [2] [3] [4] are isolated from the Web and are platform-dependent, the use of HTTP/CoAP proxies raises potential limitation when adapting CoAP protocol's features into HTTP. For example, the bidirectional communication scheme in CoAP allows a CoAP server to push notifications to its subscribed clients; however a HTTP client needs to continuously poll the server in order to receive notifications, which will incur additional network traffic.

In this paper we propose a CoAP solution to enable seamless access to CoAP servers from conventional web browsers called SCoAP. SCoAP is a combination of a CoAP implementation in JavaScript called JSCoAP and a special CoAP proxy that uses HTML5 Web Socket protocol called WSCoAP to preserve end-to-end CoAP communication between CoAP servers and web browsers. JSCoAP is a lightweight, portable JavaScript library that allows web developers with minimum programming skill to create, consume CoAP messages and manage CoAP transactions easily and freely. WSCoAP proxy maintains the integrity of CoAP communication between web browsers and CoAP servers without having to translate packages from HTTP to CoAP and vice versa. Therefore CoAP protocol's capabilities are preserved and CoAP communication in conventional web browsers is enabled.

The paper is organized in six sections. In section II, we discuss related approaches to CoAP implementation and their shortcomings. SCoAP solution is introduced in section III. Section IV shows our implementation and section V evaluates the advantageous of SCoAP solution. Finally section VI concludes our work.

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the CITRC(Convergence Information Technology Research Center) support program (NIPA-2013-H0401-13-2008) supervised by the NIPA(National IT Industry Promotion Agency) and by the International Research & Development Program(EU FP7 IoT6) of the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT&Future Planning of Korea(2012-0008824) and partly supported by the IT R&D program of MOTIE/KEIT[10041313, UX-oriented Mobile SW Platform"].

978-1-4799-1353-4/13/$31.00 ©2013 IEEE

II. Related works

In this section, we discuss how CoAP can be used to access CoAP-based resources from users' terminal. These include CoAP client implementations and HTTP/CoAP (HC) cross protocol proxies. CoAP clients allow users to interact directly with CoAP servers from users' terminals. Meanwhile HC proxies translate CoAP protocol to HTTP protocol and vice versa so that CoAP-based resources can be accessed from HTTP clients or conventional web browsers.

Popular CoAP client implementations that can be counted are libcoap [2], JCoAP [3], Coapy [4], Californium (Cf) [5], and Copper [6]. While JCoAP, Cf, libcoap and Coapy are developed in native languages such as Java, C and Python, Copper is developed as an add-on for Firefox browser. Native CoAP client implementations do not work across platforms such as PC, Smartphone, or Tables and do not support integration with web applications. Thus, using conventional web browsers, users cannot access CoAP-based resources. Copper is an advanced CoAP implementation which is integrated into Firefox browser so that using Firefox browser, users can access CoAP-based resources. However, since Copper is an add-on for Firefox browser, it cannot be used in other browsers, in mobile browsers, or in web-based applications. To this end, we developed JSCoAP, a portable JavaScript library that turns every web browsers into a CoAP client and every web applications can get the most out of CoAP protocol easily. For JSCoAP, web developers only need to embed JSCoAP library in a script tag of a web page and their web application is ready to create, consume CoAP packets and manage CoAP transactions. While JSCoAP provides fundamental functionalities for a CoAP endpoint, web developers are free to determine CoAP configurations that suit the most to their applications such as message size, caching capability or retransmission time out.

Another solution to communicate with CoAP servers from users' terminals is using conventional web browsers with a HC proxy, which is originally proposed in CoAP design [7]. HC proxies do the translation between HTTP and CoAP so that HTTP clients can access CoAP-based resources. However, the use of HC proxies is not always straightforward since some important features of CoAP protocol such as separate response or observation are not fully preservable while being adapted into HTTP protocol. This is because timeout at HTTP clients which terminates the communication could happen anytime while continuous polling at HTTP clients would incur significant network traffics. Moreover, network delay is likely to occur in standard HC proxies, for example when they have to wait for all blockwise-transfered packets to arrive or have to do cross protocol translation. For the purpose of providing the most seamless communication with CoAP servers, we propose WSCoAP, a special CoAP proxy which features end-to-end CoAP interaction. By utilizing Web Socket protocol, WSCoAP delivers CoAP messages which are created by JSCoAP library directly to CoAP servers and vice versa without any modification. Thus it overcomes all the disadvantages of standard HC proxies.

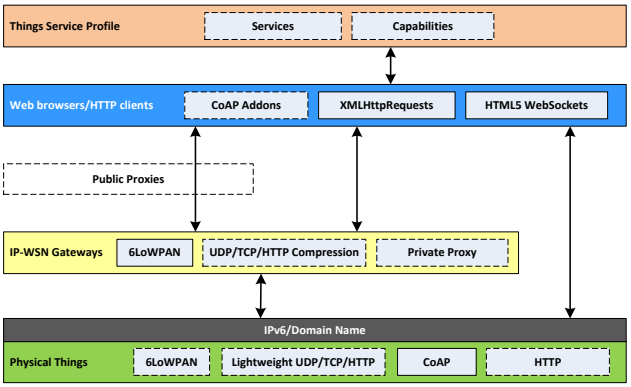


Fig. 1: Assumptions for SCoAP solution

III. SCoAP: JavaScript CoAP implementation and HTML5 Web Socket-based CoAP proxy

This section introduces SCoAP solution which is composed of JSCoAP, a JavaScript implementation of CoAP and WSCoAP, a Web Socket-based special CoAP proxy. SCoAP provides web developers an easy and efficient method to integrate CoAP-based resources into their web applications.

A. Assumptions

Before introducing our SCoAP solution, we give some assumptions regarding components and technologies that are applied to the Internet of Things context. Fig. 1 illustrates these requirements.

First, we assume that physical things in the IoT context will all have IPv6 connection and/or a domain name to interact with. Physical things are expected to run a CoAP protocol stack in the application layer to communicate with other things and with clients. Some smart things are also capable of running a HTTP protocol stack.

Second, physical things can be connected to directly from the Internet or via an IP-based Wireless Sensor Network (IP-WSN) gateway. In case an IP-WSN gateway is required, the gateway and the things can be implemented with a 6LoWPAN [8] adaptation layer for adapting Internet Protocol into constrained environment. For things which operate in constrained environment, a lightweight UDP/TCP/HTTP protocol suit and protocol compression should present as proposed in [9].

Third, conventional web browsers can interact with things via CoAP protocol by a number of communication means namely CoAP addons/plugins (like *Copper*), XMLHttpRequests with a HC proxy and Web Socket with a WSCoAP proxy. We assume that XMLHttpRequest and Web Socket are available as a matter of fact since most web browsers have already supported these protocols while CoAP addons and/or plugins are optional. HC proxies can be either public or private as shown in the picture.

Lastly, there could be a Things Service Profile as proposed in [10] which is maintained by things' vendors to store things' services and their capabilities such as which services they provide or which application protocol they support. Web

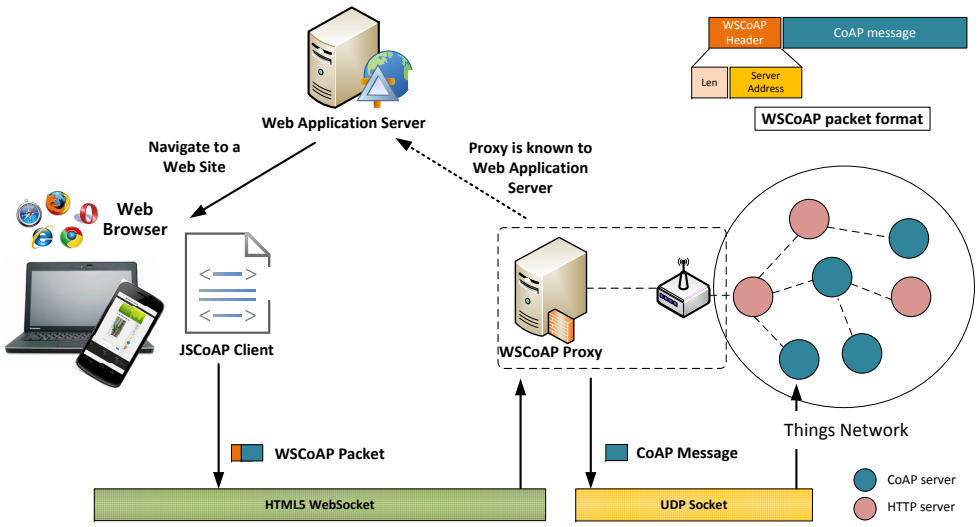


Fig. 2: SCoAP architecture

applications can either refer to this entity to learn about a things' features or conduct service discovery itself.

We assume that all of those components are loosely coupled with each other so that it can work across different vendors.

B. SCoAP solution for web-based applications

As discussed in the assumptions, web applications in web browsers can interact with CoAP servers via several communication means. We have come up with the idea of using HTML5 Web Socket for transporting CoAP messages for several reasons. HTML5 Web Socket is a new web protocol that aims at providing two-way information exchange between two terminals, hence overcomes the synchronous and client-initiate natures of HTTP protocol. This feature of HTML5 Web Socket is what exactly CoAP protocol provides. HTML5 Web Socket brings not only benefit to client side web applications but also to server-side applications since HTML5 Web Socket can be deployed in many server-side platform such as JavaEE, NodeJS or PhP. In addition, HTML5 Web Socket does not suffer from protocol overhead so that it is as lightweight as CoAP itself. Lastly, HTML5 Web Socket runs on top of TCP transportation layer so that it ensures reliability.

Thus, by using HTML5 Web Socket for transportation of CoAP packets and using JavaScript to create and parse such packets, we can provide seamless CoAP transactions between web browsers and CoAP servers.

Detail SCoAP solution is depicted in Fig. 2. The figure shows a typical web application in which JSCoAP and WS-CoAP work closely coupled with each other to provide CoAP-based resources to web users. In this use case, a user navigates to a web application that leverages physical things' resources. Within the web application, JSCoAP JavaScript library is downloaded to the user's web browser. After the web page is fully loaded, client side JavaScript code opens a Web Socket connection to a WSCoAP proxy. Then JSCoAP library is used to create and consume CoAP messages that are transported back and forth over the Web Socket connection.

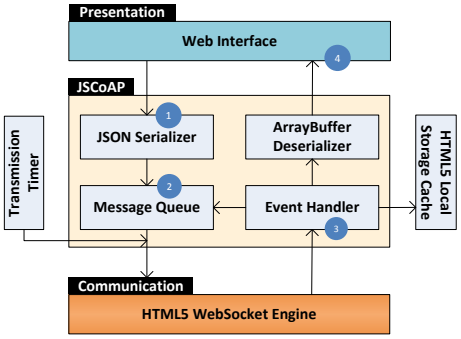


Fig. 3: JSCoAP block diagram

Since the CoAP message itself does not carry any information regarding the destination CoAP servers, a small header is added to the CoAP packet that tells the WSCoAP proxy where to forward the packet to. This header is comprised of a 1-byte-length field which indicates the length of the destined CoAP server's host name, and the host name. That is, the header covers up to 255 characters for the host name according to [11]. We call the header along with the CoAP packet WSCoAP packet henceforth.

By applying SCoAP solution, CoAP-based resources from physical things are accessible from conventional web browsers just like other web resources, bring about the ability of integrating CoAP-based services to the web.

C. JSCoAP: JavaScript implementation of CoAP

We have implemented CoAP in JavaScript language, leveraging advantage of JavaScript in web applications. JavaScript is the most popular scripting language on the Web that runs client-side applications. Applications written in JavaScript run on most popular web browsers, thus making them platform-independent. The use of JavaScript offloads complex tasks of web applications to client side so that increases performance of web servers. Moreover, CoAP implementation in JavaScript

```
var coapRequest = new CoapMessage();
coapRequest.version = 1;
coapRequest.type = coapMessageType.CON;
coapRequest.code = coapMethod.GET;
coapRequest.id = Math.floor(Math.random() * (65536));

var requestOption = new CoapOption();
requestOption.option = coapOptionType.URI_PATH;
requestOption.value = "temp";
requestOption.length = 4;
coapRequest.options.push(requestOption);

coapRequest.optionCount = coapRequest.options.length;

var coapHost = "mything.iot.kr";
var sendBuffer = serialize(coapRequest,coapHost);
var connection =
   new WebSocket('ws://wscoap.iot.kr:8888/coap');
connection.send(sendBuffer);
```

Fig. 4: SCoAP Usage

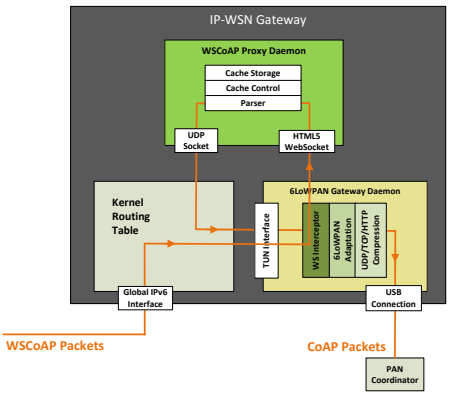


Fig. 5: Private WSCoAP proxy

introduces a potential solution in integrating CoAP-based resources to the Web.

Fig. 3 shows the block diagram of JSCoAP library. JSCoAP design aims at providing an easy-to-use and open CoAP implementation so that web developers are freely to control CoAP configurations according to their applications' need. It has four main components, a JSON Serializer serializes JSON objects into JavaScript typed arrays, an ArrayBuffer Deserializer does the opposite, a Message Queue for transmission of serialized CoAP buffer and an Event Handler for receiving CoAP packets. The implementation of transmission timer is up to web developers to decide according to specific applications. JSCoAP library provides interfaces for working with HTML5 Local Storage as a mean for local caching of CoAP resources to reduce network traffic.

Fig. 4 demonstrates how JSCoAP is used in web applications. In this code snippet, first a CoAP request is created using JSCoAP library with the application's input parameters. Then, the request which is a JSON object is serialized into a JavaScript typed array (i.e., a byte array) and is sent to a WSCoAP proxy via an HTML5 Web Socket connection. The CoAP message is maintained intact from the web browser to the CoAP server so that CoAP communication is preserved. Moreover, since JSCoAP is a JavaScript library, it can be embedded into any web application, thus, JSCoAP and WSCoAP proxy open a new possibility to integrate CoAP-based resources into the Web.

D. WSCoAP: HTML5 Web Socket-based special CoAP proxy

This section provides details on WSCoAP proxy which helps to preserve end-to-end CoAP communication between web browsers and CoAP servers. WSCoAP proxy is suggested to reside on or close to edge routers to minimize unreliability of UDP traffic on the Internet (recall that CoAP is UDP-binding), and is called private proxies in this paper. However, there are situations that the proxy can be located freely on the Internet and its address is known to everybody, this proxy is called public proxy. Unlike standard HC proxies, both of public and private WSCoAP proxies do not require HC cross protocol translation so that they minimize processing time delay and network traffic significantly.

Fig. 5 shows a private WSCoAP proxy implementation which resides in an IP-WSN gateway. Implementation of WSCoAP public proxy should be straightforward since no HC mapping is needed and it is a component of WSCoAP private proxy as seen in the green box. The WSCoAP proxy includes a WSCoAP parser to strip off WSCoAP headers, Cache modules for caching purpose and two communication sockets,a Web Socket/TCP socket and a UDP socket.

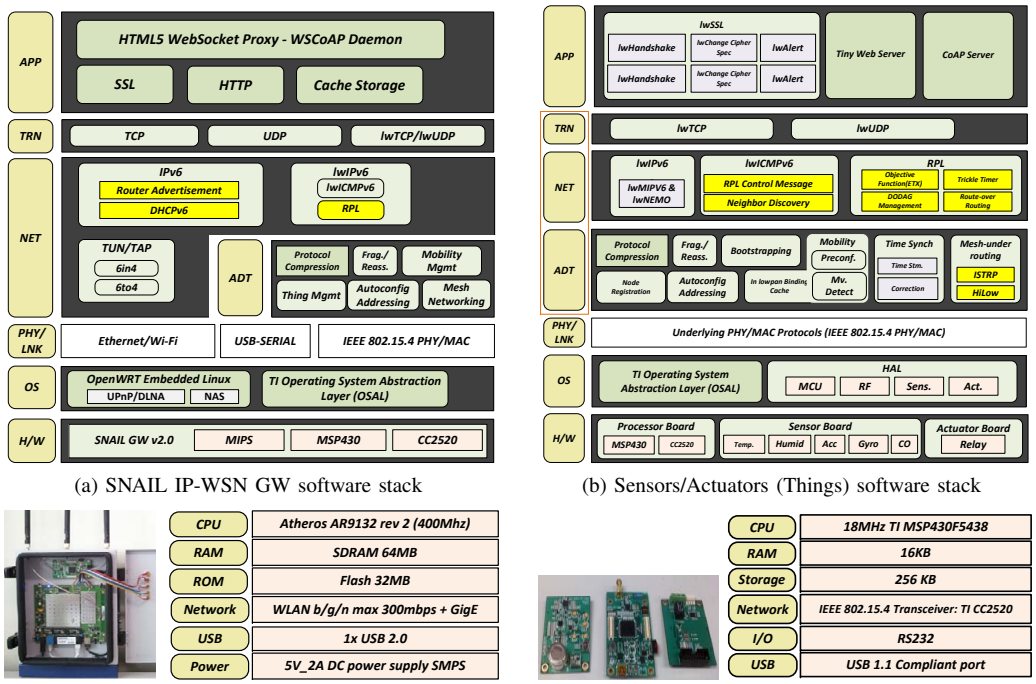
The IP-WSN gateway receives a WSCoAP packet in a Web Socket channel, it is routed via a tunneling interface which is managed by the 6LoWPAN gateway daemon. Through the Web Socket interceptor in the 6LoWPAN gateway, the packet is routed to the WSCoAP proxy daemon where WSCoAP header is stripped off and CoAP packet is parsed to get the resource URI for caching purpose. Next, the real CoAP packet is routed back to the 6LoWPAN gateway daemon via the tunneling interface and from here, it is forwarded to the destined CoAP server that runs on a physical thing over a UDP channel. The returning response should follow the reverse path.

IV. Implementation

Our implementation is based on SNAIL, our IP-based Wireless Sensor Network (IP-WSN) platform [12]. SNAIL platform includes an IP-WSN gateway and sensors/actuators (S/A) which represent a constrained network and physical things. The gateway and S/A hardware specifications are depicted in Fig. 6c and Fig. 6d respectively.

As seen in Fig. 6b, The S/A run both HTTP and CoAP servers so that provide S/A resources over both CoAP and HTTP protocol. The gateway runs an implementation of WSCoAP proxy like shown in Fig. 6a, thus minimizes the transportation of UDP packets over the Internet.

In this implementation, JSCoAP library is used in a web application in order to access S/A resources via CoAP. The web application and JSCoAP is hosted in an application server which runs Apache web server on top of Ubuntu 12.04 Linux distribution. Users' terminals such as PCs, Smartphones and Tables are used to access the application on their conventional



(a) SNAIL IP-WSN GW software stack

(b) Sensors/Actuators (Things) software stack

(c) SNAIL IP-WSN GW hardware

(d) Sensors/Actuators (Things) hardware

Fig. 6: SNAIL IP-WSN Platform

web browsers. From this web application, users are able to access CoAP-based resources from the S/A via the WSCoAP proxy in the gateway. Fig. 7 shows a sample web application that provides a web-based CoAP browser. This web application has been tested to run on most common web browsers which support HTML5 Web Socket protocol such as Chrome or Firefox, and provide full features set of CoAP protocol such as CoAP Observe, Separate response or Blockwise transfer. WSCoAP proxy is implemented as a public proxy in Java using Jetty web socket library[1]. The proxy is deployed in a standalone application server, which runs on a Windows 7 box and located on the same network with the SNAIL gateway. All of these components are connected to the Internet over IPv6 connection.

V. Evaluation

We analyze the advantage of SCoAP solution in communicating with things running CoAP using a conventional web browser. The measurement is carried out in term of computational performance comparison between HC and WSCoAP proxy and how CoAP features are preserved when using WSCoAP proxies.

In order to measure computational performance gain of WSCoAP proxy over HC proxy, we measure the time it takes from proxies accepting clients' requests to returning all responses. To enhance the experiment's accuracy, we deploy resourceful CoAP servers on the same network as the proxy to minimize network delays. To make a fair evaluation, we use the open source CoAP implementation jCoAP [3] from

[1]http://www.eclipse.org/jetty/

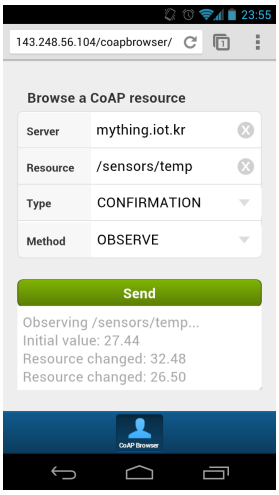


Fig. 7: A Web-based CoAP Browser

Ws4d-jcoap project and modify it so that it can handle the functionalities of a WSCoAP proxy. Then we compare the modified version of jCoAP and the original version to show the advantage of WSCoAP proxy over HC proxy. We try to keep the modification at minimum so as to ensure that computational and network resources are provided equally for the two implementation. The time is recorded according to number of concurrent requests from clients and is reported in Fig. 8. It is clearly seen that the WSCoAP proxy introduces a significant performance gain over the standard HC proxy. That is, for the least burden experiment when only 5 concurrent requests arrive at the same time, it takes the HC proxy 50 ms to

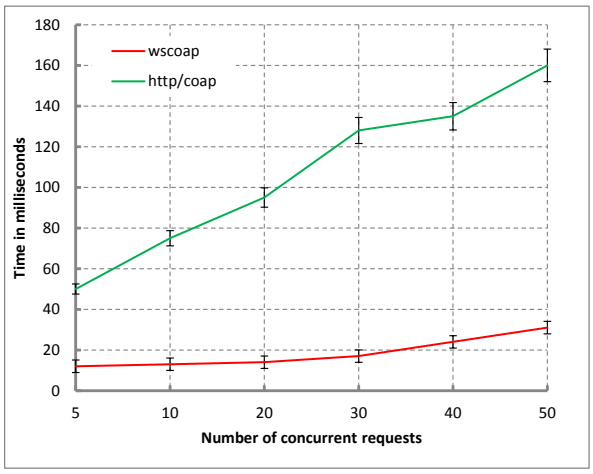


Fig. 8: Computational performance gain

		Http-CoAP proxy	WSCoAP proxy
Packet size (Bytes)	request	5 x 440	1*45
	response	5*170	5*20
Occupied Bandwidth (Bytes/Second)		$\frac{5 * 610 * 6}{60} = 305$	$\frac{(5*45+5*20*6)}{60} = 13.75$

Fig. 9: Advantageous of WSCoAP proxy over HC proxy in term of network traffic

finish all the request while WSCoAP proxy only needs 12 ms. The difference between the two is sky-rocketed when there are more and more concurrent requests arrive. As seen in the most burden case, 31 ms is the time WSCoAP needs to finish the job while HC proxy spends 160 ms. This performance gain is absolutely reasonable since HC proxy must take care of HTTP and CoAP mapping not only once but twice, from HTTP to CoAP and vice versa. Moreover, it has to maintain state of each HTTP/CoAP transaction to pair each HTTP request with the correct CoAP response. On the other hand, WSCoAP proxy works like a pipe without having to cope with HTTP and CoAP translation as shown in Fig. 5. It only needs to parse the exchanging CoAP packets to get the message id and resources representation for caching purpose. This experiment has clearly show the efficiency advantage of WSCoAP proxy over standard HC proxy.

In the second experiment, we evaluate how the HC and the WSCoAP proxies maintain CoAP functionalities in the Web environment. Since the biggest difference between CoAP and HTTP protocol lies on message exchange model as discussed in Section I, we conduct the test with CoAP Observe feature. Since HTTP is a client-initiate protocol, a HTTP server cannot send push notification to a HTTP client when needed. Therefore, the only solution for having CoAP Observe feature with a standard HC proxy is to continuously poll the server for resource changes. In contrast, HTML5 Web Socket protocol provides a bidirectional communication channel so that WSCoAP supports CoAP Observe seamlessly.

Fig. 9 summarizes the advantages of WSCoAP proxy over standard HC proxy in serving CoAP Observe. This is the result of network bandwidth occupancy calculation for observing a resource on a CoAP server with the rate of 1 observation per 10 seconds and with a total of 5 clients concurrently observe the resource. In this experiment, the number of notification messages is virtually set to be equal to the number of HTTP polling request in the standard HC proxy case. In practice, this number of CoAP notification messages is much smaller than the number of HTTP polling messages since CoAP notifications are only sent whenever changes occur on the observing resource. Despite of the equality, the result still shows that standard HC proxy occupied a bandwidth of 305Mbps while it is only 13.75Mbps for WSCoAP proxy, it is approximately 22 times traffic reduction.

VI. Conclusion

In this paper, we addressed an interesting use case of CoAP protocol where CoAP is used in web-based applications to leverage CoAP-based resources from physical things. We proposed SCoAP architecture, which helps integrating CoAP protocol with the Web. SCoAP consists of JSCoAP, a JavaScript CoAP implementation and WSCoAP, an advanced CoAP proxy that exploits HTML5 Web Socket protocol. By applying SCoAP solution, conventional web browsers can interact directly and seamlessly with physical things over CoAP protocol, bypasses HTTP/CoAP translation and reduces network traffic. Finally, the evaluation has shown the advantages of SCoAP solution in term of network traffic reduction, computational demand reduction over standard HTTP/CoAP proxy in integrating CoAP-based resources to the Web.

References

[1] Fielding and R. Thomas, *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine, 2000.
[2] O. Bergmann, "libcoap: C-implementation of coap," 2012, http://libcoap.sourceforge.net/.
[3] U. of Rostock, "Ws4d-jcoap: an implementation of the constrained application protocol (coap) for java," 2012, http://ws4d.e-technik.uni-rostock.de/ws4d-jcoap/.
[4] P. P. Co., "Coapy: Constrained application protocol in python," 2010, http://coapy.sourceforge.net/.
[5] M. Kovatsch, S. Mayer, and B. Ostermaier, "Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things," in *Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012)*, Palermo, Italy, Jul. 2012.
[6] M. Kovatsch, "Demo abstract: Human-coap interaction with copper," in *Proceedings of the 7th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2011)*, 2011.
[7] C. Bormann, A. Castellani, and Z. Shelby, "Coap: An application protocol for billions of tiny internet nodes," in *Internet Computing, IEEE*, 2012, pp. 62–67.
[8] IETF, "Rfc 4944: Transmission of ipv6 packets over ieee 802.15.4 networks," 2007.
[9] S. Bae, D. Kim, M. Ha, and S. H. Kim, "Browsing architecture with presentation metadata for the internet of things," in *IEEE International Conference on Parallel and Distributed Systems*, 2011.
[10] S. Jeong, S. H. Kim, M. Ha, T. Kim, J. Yang, N. Giang, and D. Kim, "Enabling transparent communication with global id for the internet of things," in *International Workshop on Extending Seamlessly to the Internet of Things (esIoT)*, 2012.
[11] IETF, "Rfc1034: Domain names - concepts and facilities," http://www.ietf.org/rfc/rfc1034.txt.
[12] S. Hong, D. Kim, M. Ha, S. Bae, S. J. Park, W. Jung, and J.-E. Kim, "Snail: An ip-based wireless sensor network approach toward the internet of things," in *IEEE Wireless Communications*, 2011.