Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 19 (2013) 256 – 264

Procedia Computer Science

The 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013)

Energy-Efficient Optimization Layer for Event-Based Communications on Wi-Fi Things

Gérôme Bovet[a,b], Jean Hennebert[b]

[a]*LTCI, Telecom ParisTech ENST*
46 Rue Barrault, 75013 Paris, France
[b]*ICT Institute, University of Applied Sciences of Western Switzerland*
Boulevard de Pérolles 80, 1705 Fribourg, Switzerland

Abstract

The *Web-of-Things* or WoT offers a way to standardize the access to services embedded on everyday objects, leveraging on well accepted standards of the Web such as HTTP and REST services. The WoT offers new ways to build mashups of object services, notably in smart buildings composed of sensors and actuators. Many *things* are now taking advantage of the progresses of embedded systems relying on the ubiquity of Wi-Fi networks following the 802.11 standards. Such things are often battery powered and the question of energy efficiency is therefore critical. In our research, we believe that several optimizations can be applied in the application layer to optimize the energy consumption of things. More specifically in this paper, we propose an hybrid layer automatically selecting the most appropriate communication protocol between current standards of WoT. Our results show that indeed not all protocols are equivalent in terms of energy consumption, and that some noticeable energy saves can be achieved by using our hybrid layer.

Selection and peer-review under responsibility of Elhadi M. Shakshuki

Keywords: Web-of-Things, RESTful services, WebSockets, CoAP, Energy efficiency, Smart buildings

1. Introduction

In the last decade, in a vision of inter-connected sensors and actuators attached to physical objects, the concept of *Internet-of-Things* (IoT) has been rising [1]. This idiom encloses the concept of Wireless Sensor Networks (WSN) and goes further with all kinds of physical objects able to communicate. As result of this, new kinds of Information Systems (IS) were raising, able to communicate and interact with our surrounding environment in real-time. The field of building automation can be pointed as a good example of such IS, where communicating sensors and actuators are nowadays populating new and renovated buildings [21]. New communicating objects are also arising in such smart-buildings, as for example to provide feedback to the user on the energy consumption [3]. The Internet-of-Things has since then been enhanced with well-known Web patterns leading to the *Web-of-Things* (WoT) [2]. With this new paradigm, developers have now standardized and easily integrable tools for communicating with things at the application level.

The management of the energy consumption of this multitude of communicating nodes is certainly one of the main problem of the IoT. Even if new low-power standards like 6LoWPAN, IEEE802.15.4 and RPL

1877-0509 © 2013 The Authors. Published by Elsevier B.V.
Selection and peer-review under responsibility of Elhadi M. Shakshuki
doi:10.1016/j.procs.2013.06.037

are gaining importance at the network layer, the WoT framework is actually not sensible regarding energy. Furthermore, significant reduction of energy consumption can be achieved by optimizing the protocol and data structure used for communicating with things at the highest layers.

In this paper, we show the feasibility of using a middleware at the application level able to select the most suitable communication method that can reduce energy consumption of things connected to the Internet through Wi-Fi. We base our work on the *Web-of-Things* paradigm proposing to use WebSockets or RESTful APIs, either based on HTTP or CoAP [4], for event-based data exchange. Instead to constrain application developers to choose a communication way, they can rely on an hybrid layer dynamically selecting the less consuming method of communication, depending on how much data should be sent. This represents a significant benefit by letting developers focus on other tasks than thinking about costs.

The next section of this paper refers and summarizes related work. In Sect. 3, we provide a brief overview of event-based communications axioms of the *Web-of-Things* and present an expansion of it in Sect. 4. In Sect. 5, we develop our test environment and our approach for optimizing the energy consumption of things using Wi-Fi. Sect. 6 shows how we implemented our hybrid layer. Energy consumption results attesting the efficiency of our hybrid layer are presented in Sect. 7. Finally, Sect. 8 concludes our paper and provides insights on further research.

2. Related work

One of the early projects considering people, places and things as Web resources is the *Cooltown* project [12], using HTTP GET and POST requests for manipulating things. Since then, with sensing and actuating devices taking benefit of advances made in embedded systems and having more computing power on smaller devices, it was possible to embed Web servers on things. In the *WebPlug* framework [13], where sensors and actuators are playing a central role, the strengths of the WoT allow to build so-called *mashups*.

Issues relative to performance and energy consumption were rapidly observed with the emergence of sensor networks based on Web services. The main advantage of the SOAP protocol resides on the benefit of standardizing the communication between Web services. However, the large overhead of XML and the protocol itself result in SOAP to be not optimized in terms of energy consumption [9]. A clear answer to this problem is provided by RESTful APIs being much lighter, which are increasingly adopted in many IS, especially in the domain of IoT and WoT [8] [10]. In recent times, persistent TCP connections called WebSockets have been proposed for the communication between things [11]. Preliminary comparisons between HTTP and WebSockets in terms of energy consumption have been reported [6]. This previous research showed differences between these protocols in terms of energy consumption, with complex variations as a function of the payload and frequency of the communication. Motivated by this previous work, we present our research focusing on the analysis of the optimal choice between RESTful APIs (HTTP and CoAP) and persistent TCP connections from an energy consumption point of view. More particularly, we open the question if rules may exist and can be implemented on things for selecting by their own the most economical way of communicating.

3. Common event-based communications

Sensors and actuators data naturally varies in terms of quantity and frequency, depending on the context of use and the type of thing. We can demonstrate it with an example of a power outlet continuously notifying about power consumption, while a presence sensor will only signal a change of state. This kind of behaviour is leading to so-called *event-based communications*. The WoT proposes two fundamentally different approaches for managing event-based communication: RESTful callbacks (HTTP and CoAP) and persistent TCP connections [5]. All three approaches are detailed below.

Registration: The first thing to do for an event-based system is to register the consumer at the producer. Working with *REST*, we can expand the API with services assigned for registration [5]. A thing (consumer) interested in being notified about changes of state of another thing (producer) will announce itself by providing the necessary callback information. For example, a lamp will register at a door contact sensor to

be notified when someone enters or leaves the room. This is realized by the lamp sending a HTTP POST request to `http://door.office.home/register`. The request can be of two types: (1) REST service – The request contains a JSON message indicating a REST service as callback, (2) WebSocket – The request contains the HTTP upgrade header field for switching to WebSocket, the connection is then kept open.

HTTP requests: The *REST* paradigm, is used by the WoT to expose things as resources to the Web [7]. Unlike SOAP, REST uses HTTP as application protocol for interacting with things and not only as transport protocol. In addition, REST being resource oriented, it does naturally fit with physical objects. Things can locate services through self-descriptives URLs and access them by using common HTTP requests, as GET, POST, PUT and DELETE. The GET verb is used to read a sensor, while a POST request will be sent for actuation. Only the POST operation is necessary for event-based communications. A ”consumer” object has to provide a REST service on his side that will be called by the producer. The URL of this service is provided inside the JSON message when registering on the producer. This procedure is a major aspect of our approach as we are now able to link sensors with actuators.

CoAP requests: The CoAP protocol can be considered as an evolution of HTTP for constrained environments like WSN. The main differences consist in using UDP as transport protocol and reducing the header size by encoding fields in bytes instead of ASCII characters. It allows the CoAP header to be only 4 bytes big without optional headers. CoAP is very similar to HTTP and offers the same functionalities, like GET, POST, PUT and DELETE methods, making it an economical alternative to HTTP. The WoT can only benefit from CoAP as it is easy to implement on resource limited things, and operates exactly the same way as HTTP. In addition, it exists gateways translating CoAP to HTTP or vice versa, facilitating its integration into IS.

Persistent TCP connections: At last, the WoT framework is also using persistent TCP connections, also known as WebSockets [14] for managing event-based communications. This way of communicating is often used when consumers are unable to provide a REST service, like PCs with Web browsers. The channel is kept open on both sides as long as possible.

4. Proposal for energy efficient communications

The main idea of our proposal is to let the producer decide the most energy efficient way for communicating, either through REST or through WebSockets. As it will be shown later in Sect. 5, either mode become optimal as a function of the frequency and payload of the messages exchanged. To enable dynamic switching between modes, we explain here the modifications that are requested. The registration process and the persistent TCP connections concept explained above are mainly concerned by those changes.

In our approach, both modes have to be supported and therefore, the available callbacks have to be included within the JSON message with the request. The best suited communication method will be automatically selected by the producer. Figure 1 illustrates this behaviour with our previous example involving a lamp and a door.

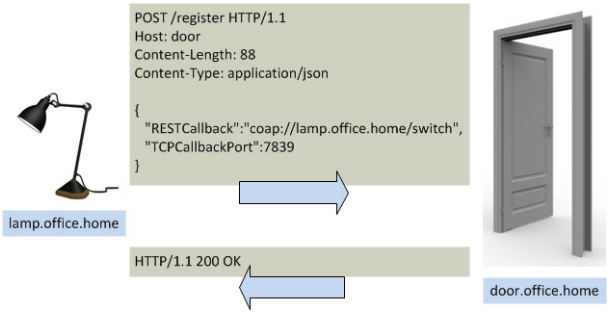

Fig. 1. Example of the registration process

Our approach slightly differs from the WoT one. The WoT assumes that consumers will open the connection, keeping it open all long. In our vision, the producer has to initiate the connection with the consumer,

because the selection between REST or TCP has to be done from its side. If because of networking errors the connection should be lost, a connection on the same port will be reopened, unless the consumer unregisters.

5. Experimental measurements and analysis

Both event-based methods are addressing the same work, periodically notify concerned things about changes of states. Nevertheless, we can assume REST requests and persistent TCP connections having different impacts on energy consumption. This is indeed the case for objects using a Wi-Fi transceiver. In this chapter, we will show how the energy consumption of things is influenced by those protocols.

5.1. Platform and test environment

We used the openPICUS FLYPORT programmable Wi-Fi module. It is Wi-Fi IEEE 802.11 certified, embeds a full TCP/IP stack, and can connect to IEEE 802.11b/g/n networks. It supports 1 or 2 Mbit/s transmission rates, with WEP, WPA-PSK and WPA2-PSK as security. The FLYPORT can be powered either at 5V or at 3.3V and drains 128mA current at 3.3V when connected to Wi-Fi [15]. One can use the provided IDE for developing in C.

We set up an isolated test environment built of a FLYPORT module acting as the producer, a Wi-Fi access point, a PC acting as the consumer, the Hameg HM8115-2 programmable power meter [16] for measuring the energy consumption of the FLYPORT and a PC for recording the measurements. The configuration of the access point is as follows: 802.11g, no encryption and long preamble. It is necessary to set up a dedicated test bench to ensure that no other device will be disturbing the proper running of the experiment as it would be in a public network. All kinds of not used services were deactivated on the participants of the network to guarantee that no other traffic can influence our measurements. The FLYPORT Wi-Fi module will produce data sent to the PC either by TCP, CoAP or by HTTP. Working in the context of smart buildings where the reaction delay of actuators is crucial, the module is not using duty-cycled models limiting the activity by putting it into hibernation for certain amounts of time as developed in [18]. The recorded measurements are then saved in a CSV format file for being further processed with Matlab for example.

5.2. Power consumption measurements

Our measurement campaign consists of tests of 30 seconds each, where the producer sent packets with a fixed payload size at a specific interval, applied to TCP, HTTP and CoAP. The values of payload and interval are chosen to match the behaviour of some specific devices used in smart buildings and therefore to obtain more pragmatic measures. We made the payload size in bytes vary as follows: 1, 10, 50, 100, 200 and 400, and the intervals in milliseconds between the sending of each packet as follows: 50, 100, 200, 400 and 800. The combination of the payload sizes and intervals gives us a campaign of 30 measurements. The 800ms interval limitation is coming from the fact that it is not possible to observe significant differences of power consumption at higher intervals. This limitation is due to the insignificant repercussion on energy consumption at such high intervals. Nevertheless, we were able to see a difference of consumption for the benefit of HTTP and CoAP over TCP at intervals higher than 10 seconds, due to the keep-alive packets sent by TCP.

The results of the average power consumption for each combination of payload and interval are showed in Figure 2. To perform accurate measurements and to observe the influence on the power consumption of each method, only a minimal program sending events was running on the FLYPORT.

5.3. Analysis

From the Figure 2, we can retrieve that HTTP is overall more energy consuming than TCP and CoAP. When examining the obtained measurements, TCP reveals to be on average 3.98% less consuming than HTTP, but only 0.69% less than CoAP. The maximal gain of TCP compared to HTTP is of 9.52% while the minimal one is 0.76%. CoAP performs almost as well as TCP, being in average 3.76% better than HTTP, with boundaries values of 8.70% and 0.76%. It can be explained by the number of packets exchanged, so the total number of transmitted data. In the case of TCP, once the connection established, only one

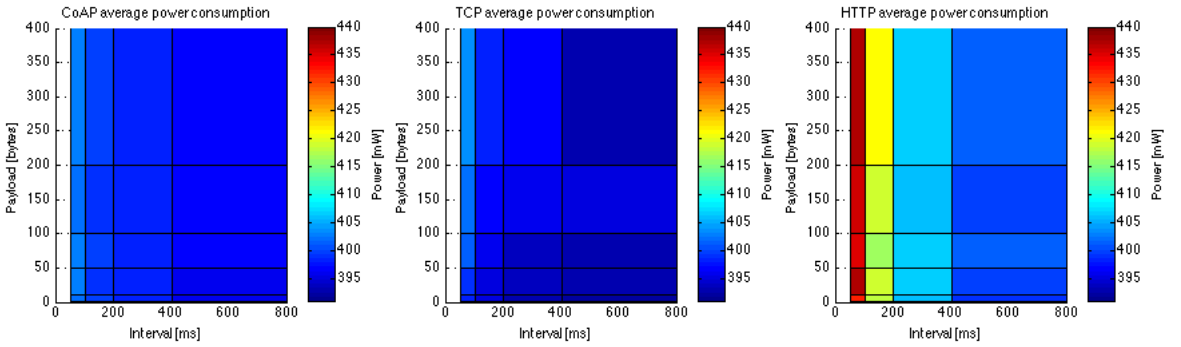

Fig. 2. Measurements of the average power consumption for TCP, HTTP and CoAP on the FLYPORT module

packet is necessary to send the JSON payload. It is exactly the same for CoAP except that no connection is required. Meanwhile, it is more complex for HTTP. Every time a payload has to be sent, a connection must be established first. This includes the inherent TCP window negotiation, the HTTP header, the HTTP response, and finally the connection closing, as illustrated in Figure 3. An increase of consumption is caused by this overhead. The amount of payload data does not play a major role in the power consumption. This is especially true when observing the results for HTTP and CoAP. The influencing factor of the consumption is definitely the sending interval.

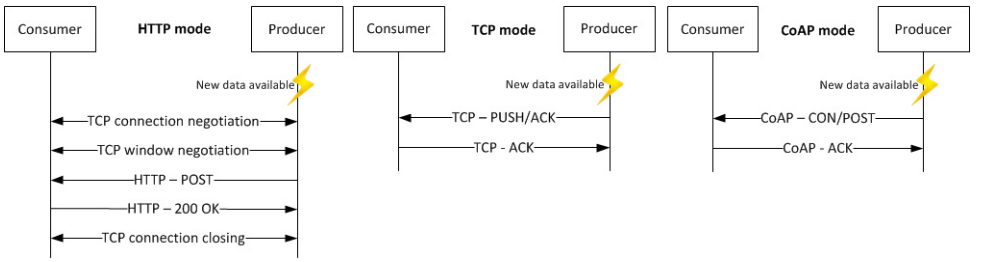

Fig. 3. Packets exchanged in HTTP, CoAP and TCP modes

5.4. Consumption approximation for TCP

For determining which method is most efficient, we propose for this to build a mathematical model to compute the power consumption. As previously pointed out, the consumption depends on the payload and the sending interval. To achieve this, we have to find a function expressing the consumption, with the payload and the sending interval as parameters. We can calculate the time needed to send data, including all underlying protocols. The global composition of a 802.11g frame is taken from [17].

Following function can be used to compute the energy consumption used to send one packet of data:

$$E(payload) = \{PLCPpreamble+(MACheader+IPheader+TCPheader+payload)*ByteRate\}*TransmitPower$$

with *IPheader*, *TCPheader* and *ByteRate* known from [19] and [20]. The transmission power was previously measured. If we compare the measurements in Figure 2 with the theoretical values that can be computed with the approximation function, it proves that the function is accurate enough to compute the energy consumption of the FLYPORT over TCP, with an average error of 0.86%.

5.5. Consumption approximation for HTTP

Computing the energy consumption for HTTP is more complicated. Due to the connection and window negotiation, due to several acknowledgements, and due to the connection closing, there are a lot more packets exchanged with HTTP than TCP. The model is even more complex to establish considering that the

number of packets may vary from one connection to the other. Instead of using a theoretical model, we opted for a parametric model where the parameters are fit to the observation. We converged to an exponential function which mathematical properties match at best with our measurements. The resulting function is as follows:
$P(interval) = a * \exp(b * interval) + c * \exp(d * interval)$

The parameters a, b, c and d were computed through a numerical fitting algorithm for every case of payload (1, 10, 50, 100, 200 and 400), ending up with 6 functions relative to the payload size. The resulting parameters allow the functions to be highly precise with an average error of 0.05%.

5.6. Consumption approximation for CoAP

As for TCP, it is possible to approximate the consumption with a function taking as parameter the payload. Based on UDP, the header size is smaller than TCP [19]. Being optimized for constrained environments, the CoAP header consists of only a few bytes [4]. The function is as follows:
$E(payload) = \{PLCPpreamble + (MACheader + IPheader + UDPheader + CoAPheader + payload) * ByteRate\} * TransmitPower$

This function is also accurate enough to approximate the energy consumption of CoAP. If we confront the theoretical values we can compute to the measurements, we obtain an average error of 1.44%.

6. Implementation

We now outline how we transposed the previously presented functions into the FLYPORT module introduced in Sect. 5 for saving energy. As visible on Figure 4, our implementation is subdivided in two main layers that are the HTTP-REST server and Event server.

6.1. REST server

For allowing very easily to add REST Web services to the FLYPORT, we decided to develop a REST server library in C, relative similar to more complex libraries that can be found for Java like Jersey, Restlet and others. Our implementation is declarative-oriented where developers indicate the type of service they want to offer. The aim of our library is to ease the integration of REST APIs on constrained devices by requiring only few steps in order to have fully functional REST services consumable by other things. Here we outline the steps necessary in order to offer REST services on the FLYPORT:

1. Include the source files (.h and .c) in the FLYPORT project.
2. Initialize the REST server by giving the listening port.
3. Declare the available REST services by indicating the HTTP verb, the URL scheme including parameters, and giving a pointer to the callback function.
4. Implement the callback function with specific behaviour, fill the return structure with response code, content-type, and payload data if needed.
5. Listen for incoming requests in the main loop.

In this work, the REST server is used for allowing consumers to register at the producers side. It could be used for many other purposes as reading a sensor's value, for actuation or configuration. Its use is in no way restricted to functionalities of a device.

6.2. Hybrid layer

Additionally to the REST server, a second library acting as event server was developed. It offers an API for registering consumers to resources, implements the energy consumptions models, and handles all the notifications mechanisms for TCP, CoAP and HTTP. When sending events to registered consumers, it is the hybrid layer's (event server) responsibility of choosing the appropriate protocol between TCP, HTTP or CoAP. As we need to record past events sent to customers, we first implemented an history structure. Every

registered customer owns its own instance of the structure. Using this history allows to compute the energy consumed by the previous communications. Every time before a new event should be sent, the layer will compute the method that would have less consumed by using the history. This is true in our case because of working with non varying intervals. However the history plays an important role in our approach as it adapts automatically to the interval and avoids developers to parameter the hybrid layer by giving the interval.

The function described in Sect. 5.4 was implemented to compute the energy consumption for TCP, including a special case for intervals higher than 10 seconds (value for the FLYPORT, it may be different on other types of modules) because of the keep-alive packets. The final value is computed as follows: *energy of each packet sent in history + energy at idle between the shipments + energy of keep-alive packets*.

For HTTP, we implemented the function outlined in Sect. 5.5. With the history, the layer knows the interval and average payload. Our approximation function then takes those values as parameters of it. In the case of the payload being different as our reference values (1, 10, 50, 100, 200 or 400), we do a linear interpolation to find the parameters a, b, c and d. By knowing the time duration of the history, the resulting power value is then converted in energy.

CoAP's approximation function described in Sect. 5.6 is the simplest one. It is very similar to TCP but without the special case for keep-alives. So, the energy is computed as follows: *energy of each packet sent in history + energy at idle between the shipments*.

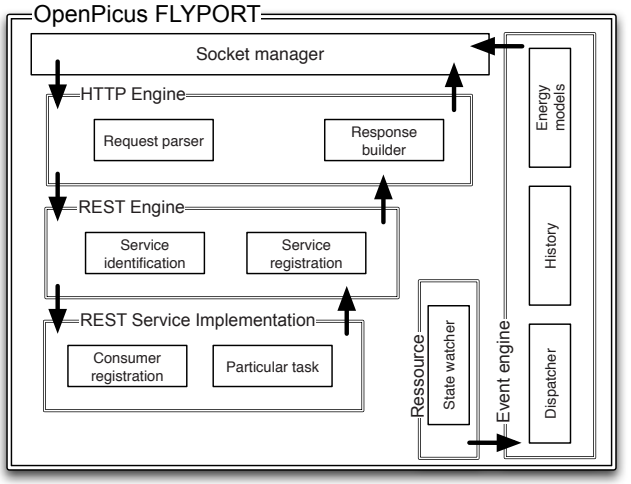

Fig. 4. Architecture deployed on the FLYPORT

7. Evaluation

After having implemented our hybrid layer, we reran the measurement campaign. This chapter points out the results we obtained and opens a discussion about it.

7.1. Quantitative

The Table 1 shows the results we obtained after having performed the measurement campaign a second time. Because having now a REST server running on the module, that consumes some energy, it was mandatory to measure again the consumption for TCP, HTTP and CoAP. We here remind that in Web-of-Things architectures, HTTP and CoAP are often considered as mutually exclusive, so that a thing will prefer using CoAP than HTTP. This is the reason why in our research we do not compare CoAP and HTTP together, but only with TCP.

The column *Gain* shows the percentage of energy saved relative to the highest value between TCP and HTTP or TCP and CoAP. At opposite, the column *Loss* shows the percentage of energy lost relative to the

Payload [bytes]	Interval [ms]	TCP–HTTP Gain[%]	TCP–HTTP Loss[%]	TCP–CoAP Gain[%]	TCP–CoAP Loss[%]
1	50	5.41	0.25	0.00	0.00
1	100	3.45	0.49	0.00	0.00
1	200	1.49	0.25	0.00	0.00
1	400	0.00	0.25	0.00	0.00
1	800	0.00	0.50	0.00	0.00
10	50	6.17	0.00	0.00	0.00
10	100	4.20	0.00	0.25	0.00
10	200	1.73	0.25	0.25	0.00
10	400	0.50	0.25	0.50	0.00
10	800	-0.25	0.25	0.25	0.00
50	50	5.62	0.24	0.49	0.00
50	100	4.46	0.00	0.25	0.00
50	200	1.49	0.25	0.25	0.25
50	400	0.25	0.25	0.50	0.00
50	800	-0.25	0.50	0.00	0.00
100	50	5.39	0.25	0.00	0.00
100	100	4.46	0.25	0.50	0.00
100	200	2.24	0.00	0.00	0.25
100	400	0.50	0.25	0.25	0.00
100	800	-0.25	0.25	0.00	0.00
200	50	4.11	0.72	0.49	0.00
200	100	4.19	0.00	0.25	0.00
200	200	1.73	0.00	0.00	0.00
200	400	0.00	0.25	0.00	0.00
200	800	-0.25	0.25	0.25	0.00
400	50	2.63	0.72	0.48	0.00
400	100	2.67	0.49	0.00	0.00
400	200	0.98	0.00	0.25	0.00
400	400	0.25	0.00	0.00	0.00
400	800	0.00	0.50	0.00	0.00

Table 1. Power consumption efficiency of the hybrid layer

lowest value between TCP and HTTP or TCP and CoAP. A negative value in column *Gain* means our hybrid layer to consume more than TCP, HTTP or CoAP alone, and can be explained by the consumption due to the hybrid layer. Nevertheless, our hybrid layer has proven its usefulness allowing to save 6.17% of energy in the best case and 2.10% on average when using TCP and HTTP. When using CoAP instead of HTTP, the results are much less impressive, because TCP and CoAP being very close in terms of consumption, leaving very little room for optimization. The hybrid layer also chooses the best method for higher intervals above 10 seconds as it selects HTTP or CoAP, which are theoretically the best ones for higher intervals. Our energy savings being quite low for single consumer, it becomes much more interesting with multiple customers. Indeed in the case of multiple consumers registered (limited to 3 in our case due to memory limitation of the FLYPORT), the positive gains were almost multiplied by the number of consumers, up to 15% in the best case.

7.2. Discussion

Consequences on energy are seldom taken into account by developers implementing callbacks. With our idea to provide an hybrid layer doing the job at their place, we hope to contribute to a reduction of energy consumption on things and extend battery life of WoT based WSN without using any duty-cycle or synchronisation technique. By relying on the impact of TCP, HTTP and CoAP on the Wi-Fi, we are able to reduce the energy consumption of event-based systems. Even if the purpose of those protocols is the same, carrying data, our measurements showed that they are indeed not equivalent in terms of energy management. However, this study pointed out that TCP and CoAP are almost identical.

Although our hybrid layer allowing energy saves for sensors sending at a fixed interval, its behaviour remains open for varying intervals. The reaction time of the layer will be significantly influenced by the number of records stored in the history. The rate of symbols sent over Wi-Fi is another issue, as it is part of the approximation functions for TCP and CoAP. This rate is continually adapting itself to the surrounding environment. In our test infrastructure, it was forced to 2Mb/s. As no device can actually communicate about the actual rate it is sending, this remains a significant obstacle to spread our layer.

At last, we would like to insist on the fact that, our research being still preliminary, there is still room for improvements. One could save even more energy by caching and grouping events, or using multicast capabilities to notify more than one consumer. Another way, instead of being only reactive, could be by

predicting the behaviour of devices with self learning algorithms, and to choose the method in a proactive manner.

8. Conclusion

In this paper, we explored a new applicative way to reduce the energy consumption of things leveraging on the WoT framework. We introduced an hybrid layer doing the developers work, instead to give them the responsibility of choosing the application protocol for event notifications. Our results show that a significant saving of energy can be achieved by selecting the most convenient protocol, especially for multi-consumers on one provider. Further to this, we believe that our callback approach is more consistent for "things-to-things" registration and communication procedures. Future work includes addressing the varying interval of events and finding the best history size to conciliate reaction time and filtering of outlier intervals. In addition, we will investigate a caching method of events by considering time penalties to limit the radio's use. A merge of our approach with duty-cycle methods will also be explored.

Acknowledgement

This work was supported by the research grants HES-SO RCSO project ee-WoT and by the Hasler Foundation project Green-Mod.

References

[1] F. Mattern and C. Floerkemeier, *From the Internet of Computers to the Internet of Things*, volume 6462 of *LNCS*, pages 242-259. Springer, 2010.
[2] D. Guinard, V. Trifa, F. Mattern and E. Wilde, *From the Internet of Things to the Web of Things : Resource Oriented Architecture and Best Practices*, *Architecting the Internet of Things*, pages 1-34. FlorianEditors, 2011.
[3] C. Gisler, G. Barchi, G. Bovet, A. Mugellini and J. Hennebert, *Demonstration Of A Monitoring Lamp To Visualize The Energy Consumption In Houses*, *Proc. of the 10th International Conference on Pervasive Computing (Pervasive2012)*, Newcastle, UK, 2012.
[4] Z. Shelby, K. Hartke, C. Bormann and B. Frank, *Constrained Application Protocol (CoAP)*, IETF, Draft 12, Oct 2012.
[5] D. Guinard, *A Web of Things Application Architecture – Integrating the Real-World into the Web*, PhD Thesis, 2011.
[6] G. Bovet and J. Hennebert, *Communicating With Things - An Energy Consumption Analysis*, *Proc. of the 10th International Conference on Pervasive Computing (Pervasive2012)*, Newcastle, UK, 2012.
[7] R. Fielding and R. Taylor, *Principled design of the modern Web architecture*, volume 2 of *ACM Trans. Internet Technol.*, pages 115-150. ACM, 2002.
[8] F. Aijaz, M. Chaudhary and B. Walke, *Performance Comparison of a SOAP and REST Mobile Web Server*, *Context*, 2009.
[9] C. Groba and S. Clarke, *Web services on embedded systems – a performance study*, volume 3 of *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops PERCOM Workshops*, pages 726-731. IEEE, 2011.
[10] H. Hamad, M. Saad and R. Abed, *Performance Evaluation of RESTful Web Services*, volume 1 of *Computer Engineering*, pages 72-78. Computer Engineering Department, 2010.
[11] N. Priyantha, A. Kansal, m. Goraczko and al., *Tiny Web Services: Design and Implementation of Interoperable and Evolvable Sensor Networks*, *Computer*, pages 253-266. ACM, 2008.
[12] T. Kindberg and al., *People, Places, Things: Web Presence for the Real World*, volume 7 of*Mobile Networks and Applications*, pages 365376. Building, 2002.
[13] B. Ostermaier, F. Schlup, and K. Römer, *WebPlug: A Framework for the Web of Things*, *Proc. of the First IEEE International Workshop on the Web of Things (WOT2010)*, Mannheim, Germany, 2010.
[14] I. Fette and A. Melnikov, *The WebSocket Protocol*, IETF, RFC 6455, Dec 2011.
[15] OpenPicus FLYPORT Datasheet, http://space.openpicus.com/u/ftp/datasheet/flyport_wifi_datasheet_rev8.pdf
[16] Hameg HM8115-2 power meter description, http://www.hameg.com/0.147.0.html
[17] D. Vassis, A. Rouskas and I. Maglogiannis *The IEEE 802.11g standard for high data rate WLANs*, volume 19 of *IEEE Network journal*, pages 21-26. IEEE, May-june 2005.
[18] B. Ostermaier, M. Kovatsch and S. Santini *Connecting things to the web using programmable low-power WiFi modules*, *Proc. of the Second International Workshop on Web of Things (WoT2011)*, San Francisco, USA, 2011.
[19] R. Stevens *TCP/IP illustrated (vol. 1): the protocols*, Addison-Wesley Longman Publishing Co., Boston, USA, 1993.
[20] M. Gast *802.11 Wireless Networks: The Definitive Guide, Second Edition*, O'Reilly Media, 2005.
[21] G. Bovet and J. Hennebert *The Web-of-Things conquering Smart Buildings*, volume 10s/2012 of *Bulletin*, pages 15-19. Electro-Suisse, 2012.