An Energy Efficient Layer for Event-Based Communications in Web-of-Things Frameworks

Gérôme Bovet, Hennebert Jean

An Energy Efficient Layer for Event-Based Communications in Web-of-Things Frameworks

Gérôme Bovet[1] and Jean Hennebert[2]

[1]LTCI, Telecom ParisTech, Paris, France
[2]ICT Institute, University of Applied Sciences of Western Switzerland, Fribourg, Switzerland
gerome.bove@telecom-paristech.fr, jean.hennebert@hefr.ch

Abstract. Leveraging on the Web-of-Things (WoT) allows standardizing the access of things from an application level point of view. The protocols of the Web and especially HTTP are offering new ways to build mashups of things consisting of sensors and actuators. Two communication protocols are now emerging in the WoT domain for event-based data exchange, namely WebSockets and RESTful APIs. In this work, we motivate and demonstrate the use of a hybrid layer able to choose dynamically the most energy efficient protocol.

Keywords. Web-of-Things, RESTful services, WebSockets

1 Introduction

In the last few years, a vision of inter-connected sensors and actuators attached to physical objects has emerged, leading to the concept of Internet-of-Things (IoT) [1]. This idiom includes the concept of Wireless Sensor Networks (WSN) and goes beyond with all kind of physical objects able to communicate. The field of building automation is a potential target for IoT approaches where numerous communicating sensors and actuators are in use [2]. In such smart-buildings, new communicating objects are also appearing, for example to provide the user with feedback on the energy consumption [3]. The IoT has since then been extended from the IP usage towards the inclusion of well-known Web patterns to ease the the integration and communication with things at the application level, leading to the concept of Web-of-Things (WoT) [4]. One of the main problems of the IoT is certainly in the management of the energy consumption of this multitude of communicating nodes. Although new low-power standards like 6LoWPAN, IEEE802.15.4 and RPL are being established at the network layer, the WoT framework is actually not energy aware at the application level. We believe that the protocol and data structure used for communicating with things at the highest layers could contribute to a significant reduction of the energy consumption.

In this paper, we show the feasibility of using an additional layer at the application level able to select the most suitable communication method in order to reduce the energy consumption of things connected to the Internet through Wi-Fi. We rely on the *Web-of-Things* paradigm proposing to use WebSockets or RESTful APIs for event-based data exchange. Instead of forcing application developers choosing a communication method, they can rely on an hybrid layer dynamically selecting which method is less energy consuming depending on how much and how frequently data should be sent. This represents a meaningful advantage letting developers focus on other tasks than thinking about costs. Sections 2 and 3 summarize related work and the principles of event-based WoT communications. In Section 4, we present our proposal for improving the event-based communication. In Section 5, we present the experimental measurements and their analysis. Section 6 provides details on the implementation of our hybrid layer and energy consumption measurements. Section 7 concludes our paper and provides insights on further research.

2 Related work

The Cooltown project [5] is one of the early projects considering people, places and things as Web resources, using HTTP GET and POST requests for manipulating things. The recent progresses in embedded devices are now enabling the integration of Web servers on things. The tendency is clearly shown with, for example, the WebPlug WoT framework where sensors and actuators used to build so-called mashups [6]. An important step towards a standardization of the communication at the application level for web services was the introduction of the SOAP protocol. However, SOAP is not optimized in terms of energy consumption due to the large overhead of XML and of the protocol itself [11]. Much lighter, RESTful APIs provided a clear answer to this problem, with an increased adoption for many IS, especially in the domain of IoT and WoT [10], [12]. Recently, persistent TCP connections called WebSockets have been proposed for the communication between things [13]. Preliminary comparisons between HTTP and WebSockets in terms of energy consumption have been reported in [8]. This previous research shown 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, the research presented in this paper focuses on the analysis of the optimal choice between RESTful APIs and persistent TCP connections targeting energy efficiency. More specifically, we open the question if rules may be implemented on things for choosing automatically the most efficient way of communicating.

3 WoT event-based communications

Sensor and actuator data can vary in quantity and frequency according to the context of use. For example a power outlet will continuously notify about the electricity consumption when a device is plugged in, while on the other side a presence sensor will only signal a change of state. This kind of behavior is leading to so-called event-based communications. The WoT proposes two fundamentally different approaches for managing event-based communication: HTTP callbacks and persistent TCP connections [7]. Both approaches are detailed below.

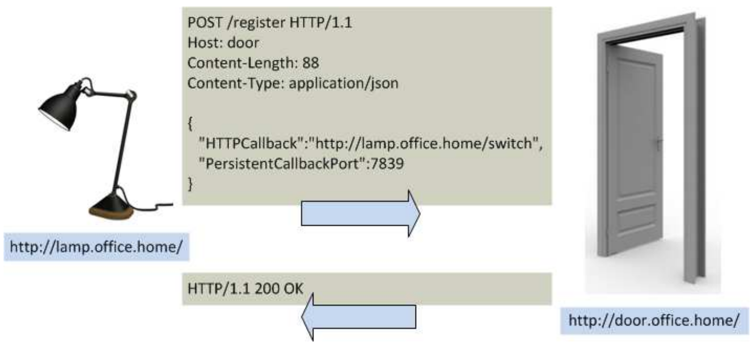
Registration. The first step for event-based system is the registration of the consumer at the producer. Using things with REST, we can simply expand the API with a service dedicated to registration [7]. A thing interested in being notified by change of states of another object will announce itself by providing the required callback information. For example, a lamp actuator will register a door contact sensor to be notified when someone enters or leaves the room. The lamp sends a HTTP POST request to http://door.office.home/register. This request can be of two types: (1) REST service - containing a JSON message indicating a REST service as callback, (2) WebSocket - containing the HTTP upgrade header field for switching to WebSocket, keeping open the connection.

HTTP requests. The WoT relies on REST for exposing things as resources to the Web [9]. Unlike SOAP, REST uses HTTP as application protocol for interacting with things and not only as transport protocol. The advantages of REST over SOAP are in having less overhead, and being resource oriented, which fits naturally with physical objects. With WoT, every object is embedding a built-in Web server exposing an API for interacting with its sensing, actuating and configuration capabilities. Self-descriptives URLs are used through common HTTP requests, like GET, PUT, POST and DELETE. For example, reading a sensor value is done using the GET verb and actuating using POST. For event-based communications, POST is actually the only necessary operation. A "consumer" object typically provides a REST service to be notified of changes in another object. The service URL is provided as callback at the registration on the producer. This is a significant aspect of our approach as we can link sensors with actuators.

Persistent TCP connections. The second way of managing event-based communications proposed in the Web-of-Things framework is using persistent TCP connections also known as WebSockets [14]. This kind of communication is mostly used in push scenarios where data has to be sent from a server to a client not running a Web server, as for example 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 to communicate, either through REST HTTP or through WeSockets. As it will be shown later in Section 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. It concerns mainly the registration process and persistent TCP connections concept explained above. In our vision, both modes are supported and therefore, the registration JSON message has to include the available callbacks for REST and persistent TCP connection. The producer will further select automatically which method is best suited for exchanging data from an energy efficiency point of view. This is illustrated in Figure 1 with an example involving a lamp and a door.



```
POST /register HTTP/1.1
Host: door
Content-Length: 88
Content-Type: application/json

{
  "HTTPCallback":"http://lamp.office.home/switch",
  "PersistentCallbackPort":7839
}
```

```
HTTP/1.1 200 OK
```

Fig. 1. Example of the registration process

For persistent TCP connections, our proposal slightly differs from what is currently done with WoT. Indeed, WoT approaches suppose that the consumer initiates the persistent connection, keeping the channel open while the producer sends its data. In our approach, the producer has to select between HTTP or TCP and therefore initiates the connection. If the connection is lost due to network faults, the producer will retry to open a connection on the same port, unless the consumer registers with another one.

5 Experimental measurements and analysis

HTTP requests and persistent TCP connections have different impact on energy consumption. This is especially true for objects connected to the Internet with a Wi-Fi transceiver. We show here how each method can influence the energy consumption of things.

5.1 Test environment

We used the openPICUS FLYPORT programmable Wi-Fi module. This tiny module (35 x 48 x 11 mm), is Wi-Fi IEEE802.11 certified and embeds a full TCP/IP stack, able to connect to IEEE 802.11b/g/n networks. It supports 1 or 2 Mbit/s rates as well as security protocols such as WEP, WPA-PSK and WPA2-PSK. The FLYPORT can be powered either at 5V or at 3.3V and drains 128mA current at 3.3V when connected to Wi-Fi. An IDE is available for developing application in C [15]. We set up an isolated test environment composed of a FLYPORT module acting as the producer, an access point and a PC acting as the consumer. The wireless network set up is 802.11g, no encryption and long preamble. We also used a Hameg HM8115-2 for measuring the energy consumption of the FLYPORT [16]. Having a dedicated test bench ensures that no other device will be disturbing the proper running of the experiment as it would be in a public network.

5.2 Power consumption measurements

We describe here our measurement campaign for both TCP and HTTP. During each test of 30 seconds, the producer sent packets with a fixed payload size at a specific interval. For measuring precisely the consequence of each method on the power consumption, the FLYPORT was only running a minimal program sending events. The values of payload and interval are chosen to match the behavior of some specific devices and therefore to perform more realistic measurements. We made the payload size in bytes vary from 1 to 400 and the intervals between packets in milliseconds from 50 to 800, which correspond to certain devices one can find in smart buildings. The combination of the payload sizes and intervals gives us a campaign of 30 measurements as illustrated in Figure 2.

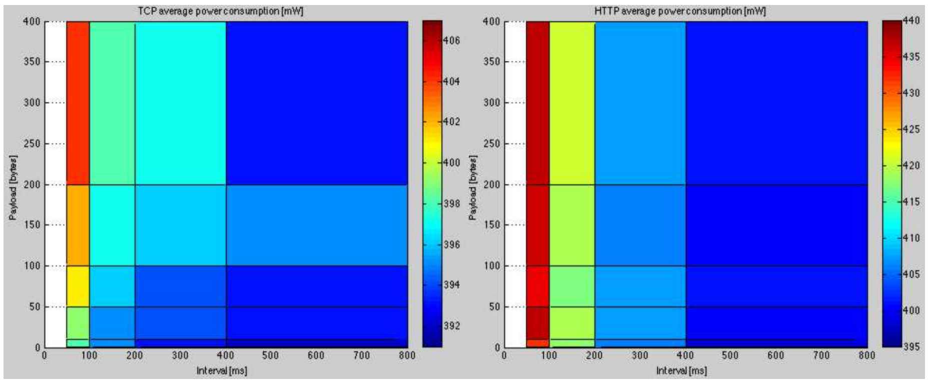


Fig. 2. Results of the average power consumption measurements for TCP and HTTP

From the Figure 2, we see that TCP is overall less energy consuming than HTTP. TCP appears to be on average 4% less consuming than HTTP with a maximal gain of 9.5%. The quantity of transmitted packets is indeed lower for TCP than HTTP. With TCP, once the connection is established, only one packet is necessary to send the JSON message. HTTP is more complex as a connection has to be established every time a JSON message must be sent. An HTTP connection includes the potential TCP window negotiation, the HTTP header, the HTTP response, and finally the connection closing. All this overhead causes an increase in consumption. The measurements also show that the amount of payload data plays a less important role in the power consumption. This is especially true for HTTP consumption. On the other hand, a factor influencing the consumption is clearly the sending interval. The main observation is that both modes are overlapping in terms of efficiency, with TCP is becoming less optimal than HTTP in some conditions.

5.3 Consumption approximation for TCP and HTTP

HTTP. With TCP, the variable is the necessary time needed to send data, including all underlying protocols. With Wi-Fi (802.11g) frame composition is taken from [17].

The energy consumption for one packet of data can be computed with the following function: E(payload) = {PLCP preamble + (MAC header + IP header + TCP header + payload) * ByteRate } * TransmitPower with IPheader, TCPheader and ByteRate known from [18-19] and TransmitPower previously measured. When comparing the theoretical values of the approximation to the measurements of the FLYPORT in Figure 2, it comes out this function is accurate enough with an average error of 0.86%.

TCP. As explained earlier, the HTTP case is more complicated as for TCP. Instead of using a theoretical model, we opted for a parametric model where the parameters are fit to the observations. We converged to an exponential function, approximated as in P(interval) = a*exp(b*interval) + c*exp(d*interval). Through a numerical fitting algorithm, we computed the parameters a, b, c and d for every case of payload (1, 10, 50, 100, 200 and 400), ending up with 6 functions for the different payload sizes. The computed parameters allow the functions to be quite precise with an average error of 0.05%.

6 Implementation and evaluation of the hybrid layer

We had first to develop a **REST server** library in C for the FLYPORT. The services are registered by indicating a URL scheme corresponding to the Web service, and providing a pointer to a callback function that will be called when the server receives a request for this particular service. We then implemented the hybrid layer in charge of dynamically choosing the appropriate method between TCP and HTTP when sending events to registered consumers. We first implemented a history structure for recording the past events sent to consumers. An instance of this structure is created for every consumer registered. This allows computing the energy consumed to send the previous events. According to this result, the layer then switches to the most efficient method and this every time a new event must be sent. For computing the TCP mode energy consumption, we implemented the function described in Sect. 5.3. The implementation includes a rule for intervals higher than 10 seconds to consider the keep-alive packets (specific to the FLYPORT as it may differ on other modules) . 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 as in Sect. 5.4. Using the history, we know the interval and the average payload. Those values are then used as parameters for our approximation function. Linear interpolations are used in the case of payload different as our reference values (1, 10, 50, 100, 200 or 400). The obtained power value is then converted in energy by knowing the time duration of the history.

Table 1 shows the energy measurements of our hybrid layer where some relevant saves were achieved. For comparison purposes, we had to rerun the campaign for each TCP and HTTP modes as our REST server running on the module is also consuming some energy. The column *Gain* shows the percentage of energy saved relative to the highest value between TCP and HTTP. The column *Loss* shows the percentage of energy lost relative to the lowest value between TCP and HTTP. The negative values in the *Gain* can be explained by the consumption due to the hybrid layer. Nevertheless, our hybrid layer clearly shows its usefulness allowing saving 6.2% of energy in the best case and 2.1% on average. The hybrid layer also chooses the best method for higher intervals above 10 seconds as it selects HTTP, which is theoretically the best one for higher intervals.

Payload [bytes]	Interval [ms]	TCP [mW]	HTTP [mW]	Hybrid [mW]	Gain [%]	Loss [%]
1	50	406	429	407	5.41	0.25
1	100	404	420	406	3.45	0.49
1	200	402	409	403	1.49	0.25
10	50	405	430	405	6.17	0.00
10	100	405	422	405	4.20	0.00
10	200	403	411	404	1.73	0.25
50	50	408	432	409	5.62	0.24
50	100	404	422	404	4.46	0.00
50	200	402	409	403	1.49	0.25
100	50	407	430	408	5.39	0.25
100	100	403	422	404	4.46	0.25
100	200	402	411	402	2.24	0.00
200	50	411	431	414	4.11	0.72
200	100	406	423	406	4.19	0.00
400	50	415	429	418	2.63	0.72
400	100	410	423	412	2.67	0.49

Table 1. Power consumption comparison between TCP, HTTP and the hybrid layer

7 Discussion and conclusion

Our measurements showed that TCP and HTTP are not equivalent in terms of energy, even if their purpose is the same. By offering a hybrid layer, we expect to globally reduce the energy consumption and lengthen battery life of Web-of-Things. Although our hybrid layer allows energy savings for sensors sending at a fixed interval, the behavior remains open for varying intervals. The number of records saved in the history will play a role on how the layer will respond to changes of interval. Another unresolved issue concerns the rate of symbols sent over Wi-Fi. The approximation function for TCP requires knowing at which rate the module sends its data. Due to changes in the surrounding environment, traffic congestions and other reasons, this rate may be changing. In our case, we forced a rate of 2Mb/s in our test infrastructure.

In this paper, we explored a new way on how to reduce the energy consumption of things working inside the WoT framework. Instead of giving the responsibility of choice between TCP and HTTP for event notifications to developers, we introduce an hybrid layer doing the job for them. Our results show that energy savings can be achieved by selecting the most appropriate transport protocol. Further to this, we believe that our approach simplifies callbacks between things. Future work includes addressing the varying interval of events and finding the best history size to conciliate reaction time and filtering of outlier intervals. While the measured energy savings are relatively limited, we believe our hybrid layer has further potentials, for example if used as caching method of events by considering time penalties to limit the radio's use.

References

1. Mattern F and Floerkemeier C (2010) From the Internet of Computers to the Internet of Things. In: Sachs K, Petrov I, Guerrero P (eds) From Active Data Management to Event-Based Systems and More. Springer, Heidelberg, p 242
2. Bovet G and Hennebert H (2012) The Web-of-Things conquering Smart Buildings. Bulletin 10s/2012:15-19
3. Gisler C, Barchi G, Bovet G, Mugellini H and Hennebert J (2012) Demonstration Of A Monitoring Lamp To Visualize The Energy Consumption In Houses. In: Proc. of the 10th International Conference on Pervasive Computing, Newcastle, UK, June 2012
4. Guinard D, Trifa V, Mattern F and Wilde E (2011) From the Internet of Things to the Web of Things: Resource Oriented Architecture and Best Practices In: Uckelmann D, Harrison M, Michahelles F (eds) Architecting the Internet of Things. Springer, Heidelberg, p 97
5. Kindberg T et al (2002) People, Places, Things: Web Presence for the Real World. Mobile Networks and Applications 7:365-376
6. Ostermaier B, Schlup F and Römer K (2010) WebPlug: A Framework for the Web of Things. In: Proc. of the First IEEE International Workshop on the Web of Things (WOT2010), Mannheim, Germany, April 2010
7. Guinard D (2011) A Web of Things Application Architecture - Integrating the Real-World into the Web. ETHZ, 2011
8. Bovet G and Hennebert H (2012) Communicating with Things – An Energy Consumption Analysis. In: Proc. of the 10th International Conference on Pervasive Computing, Newcastle, UK, June 2012
9. Fielding R and Taylor R (2002) Principled design of the modern Web architecture. ACM Transactions on Internet Technology 2:115-150
10. Aijaz F, Chaudhary M and Walke B (2009) Performance Comparison of a SOAP and REST Mobile Web Server. In: Proc. of the 3rd International Conference on Open-Source Systems and Technologies, Lahore, Pakistan, December 2009
11. Groba C and Clarke S (2010) Web services on embedded systems – a performance study. In: Proc. of the 8th IEEE International Conference on Pervasive Computing and Communications, Mannheim, Germany April 2010
12. Hamad H, Saad M and Abed R (2010) Performance Evaluation of RESTful Web Services. Computer Engineering 2:72-78
13. Priyantha N, Kansal A, Goraczko M et al (2008) Tiny Web Services: Design and Implementation of Interoperable and Evolvable Sensor Networks. In: Proc. of the 6th ACM conference on Embedded network sensor systems, Raleigh, USA, November 2008
14. Fette I and Melnikov A (2011) The WebSocket Protocol. RFC, December 2011
15. OpenPicus (2012) FLYPORT Datasheet. http://space.openpicus.com/u/ftp/datasheet/flyport_wifi_datasheet_rev8.pdf
16. Hameg (2012) HM8115-2 power meter description. http://www.hameg.com/0.147.0.html
17. Vassis D, Rouskas A and Maglogiannis I (2005) The IEEE 802.11g standard for high data rate WLANs. IEEE Network journal 9:21-26
18. Stevens R (1993) TCP/IP Illustrated: the protocols. Addison-Wesley Longman Publishing Co., Boston, USA
19. Gast M (2005) 802.11 Wireless Networks: The Definitive Guide, Second Edition. O'Reilly Media