Improving User Interactions with Constrained Devices in the Web of Things

Floris Van den Abeele, Enri Dalipi, Ingrid Moerman, Piet Demeester, Jeroen Hoebeke
Department of Information Technology, Ghent University – iMinds
Technologiepark Zwijnaarde 15, B-9052 Ghent, Belgium
{fvdabeele,edalipi,imoerman,pietdm,jhoebeke}@intec.ugent.be

Abstract—As the Internet of Things continues to grow rapidly in the coming years, the number of devices with limited resources will continue to grow as well. These so-called constrained devices typically implement specialized protocols and data formats for increased efficiency. While this reduces the load on constrained devices, it also limits the usability of such devices for their users. This paper presents a HTTP-CoAP proxy for improving the usability of constrained devices that implement embedded web services. This is accomplished by rendering user interfaces and solving naming and routing issues. As a result, the user experience of highly-optimized embedded web services is similar to that of conventional web services. By means of small-scale experimentation, the presented approach is evaluated functionally and the usability is evaluated in terms of user interface responsiveness.

Keywords-Internet of Things; Constrained Application Protocol; HTTP-CoAP proxy; usability; user interfaces

I. INTRODUCTION

According to numerous market research reports [1][2][3], the number of Internet-connected devices will have risen drastically by the end of this decade. A considerable amount of these newly connected devices, are so-called "constrained devices", i.e. devices with tight limits on power, memory, connectivity, processing power, cost and physical size. Due to their anticipated widespread usage, IoT devices are expected to impact a large number of aspects of our society and our daily lives by enabling a whole new range of services [4]. In order to realize these services, IoT devices have to interact with each other, with their environments and with their users.

On the other hand, users of such services expect an experience similar to the conventional Internet services with which they are familiar: e.g. web browsing, mobile applications, web search, etc. For services that build on constrained devices, this is a challenge as the constraints intrinsic to these devices impose limits on their functionality and as such on their usability. Consider, for example, a battery-powered air monitoring device with a hundred kilo bytes of memory. For such a device it is impossible to offer a web-based user interface that offers a user experience similar to today's popular web platforms.

There exist a number of solutions for overcoming this problem. A subset of these rely on systems external to the constrained device for overcoming the limited usability of such devices. More specifically, this work presents a web proxy based approach for improving interactions between users and constrained devices in a web of things context. We demonstrate how our approach solves a number of usability problems common to low-power and embedded web services, thereby showing the feasibility and effectiveness of our work.

II. PROBLEM STATEMENT AND RESEARCH GOALS

Constrained devices are subject to a number of limitations, most of which are the result of the required low device cost. As these limitations directly impact the usability of constrained devices, they are briefly discussed in this paragraph. Firstly, popular low-power micro controller families such as the ARM Cortex M3, TI MSP430 and AVR ATmega offer many different models where the available volatile memory varies between 8KB and 100KB and the read-only memory between 32K and 1024KB. In every use case a trade-off has to be made between cost vs available memory space: 16KB RAM and 128 or 256KB ROM is a common choice for systems in sensor and mesh networks. For battery-powered devices, power consumption is a second important consideration. Low-cost devices typically have lifetimes equal to their battery lifetimes, as replacing the battery is deemed too expensive. As a result, energy consumption should be kept to a minimum by e.g. limiting computation and communication. Finally, a third important constraint of networked systems is the employed communication technology. For our discussion, it should enable low-power communication while keeping the component cost (e.g. transceiver, amplifiers, antenna) low. A comprehensive overview of the constraints is available in RFC 7228 [5]. As per RFC 7228 terminology, this work focuses on Class 1 constrained devices, with ~10KiB RAM and ~100KiB ROM.

Each of these discussed limitations impacts the usability of constrained devices in different ways. In the class 1 systems under consideration, the limited memory commonly has to fit the entire communication stack (i.e. everything above the PHY layer) as well as the necessary logic to realize the intended service. As a result, the remaining amount of memory left to also implement a high quality user interface is typically very low. For example, a simple index page based on the popular bootstrap template for responsive web interfaces [1] requires 89.6KB of memory: Javascript (minimal

---

[1] https://getbootstrap.com/

jQuery and bootstrap: 44.6KB), CSS styling (38.4KB) and HTML (6.5KB). One can increase the available memory to include all necessary files or host the static media files (JS and CSS) externally. Even so, a considerable amount of additional data for the UI (in the order of (tens of) kilo bytes) would have to be transferred between the constrained device and the client. For battery-powered devices, this would drastically hasten the depletion of the energy source and therefor limit the lifetime of the device. In the case of low-throughput networks, transferring the additional UI data could lead to long latency penalties as the networks are not dimensioned to transmit large chunks of data. Consequently, user experience would suffer under these long delays. As a result, class 1 devices are considered to offer 'bare-bone' RESTful resources - via the specialized CoAP protocol - that are cumbersome to use due to the lack of a UI.

Low-power network protocols such as 6LoWPAN and the RPL routing protocol also impact the usability of constrained nodes. In such networks, separate IPv6 networks are typically assigned to the low-power and lossy networks (LLNs). In cases where global IPv6 routing for the LLNs is unfeasible (e.g. private LLNs), the user is expected to reconfigure its network configuration to add a routing rule to the LLN. For most users this is unrealistic. Additionally, the use of IPv6 means that constrained nodes are reachable via 128 bit IPv6 addresses. As these long addresses are impractical for human users, an alternative has to be provided. Also, discovery of devices by a user might be difficult.

The goal of this work is to answer the following research question: given the problems outlined above, how can direct user interactions with constrained devices in low-power and lossy networks be improved? In answering the question, this work looks at the problem from an embedded web services point of view as realized with the IPv6 and CoAP protocols [6]. Although the focus is on these technologies, the core concepts of this work are more broadly applicable.

III. User friendly interactions

A. Requirements

In analyzing the posed research question, the following requirements for suitable solutions that improve user interactions are identified:

1) **Impact on constrained devices should be kept to a minimum.** Consequently the device constraints outlined in the previous section remain unaltered.
2) **Easy to use interfaces for the user.** The user experience should be similar to popular web-based services.
3) **Handle a wide variety of constrained and user devices.** When looking only at embedded web services, a constrained device can serve many purposes. Similarly, there exists a large range of user devices.
4) **Minimal configuration and easy discovery.** Any usable solution should require minimal configuration

from the user. It should also facilitate easy discovery of constrained devices and their services.

5) **Easy to build user interfaces.** While building interfaces will require some technical knowledge, it should be based on open and readily available technology to facilitate designers.

There are a number of approaches that fit the requirements outline above, some of which are presented in the related work section. The approach in this work is discussed in the following section and relies on web-based application proxies combined with naming and discovery services.

B. Approach

Figure 1 outlines the approach of this work and how it differs from what is available today. Today, users interface with devices directly via CoAP or indirectly via HTTP through a gateway. In both cases the user is served the unaltered CoAP response, which is typically encoded in a compact but obscure binary format. Our approach introduces web-based application proxies, whose main task is serving web interfaces to users. The proxies process web requests from the user's browser and translate the requests into RESTful CoAP requests for constrained devices. Responses from constrained devices are processed by proxies and used as input for rendering web interfaces to users.
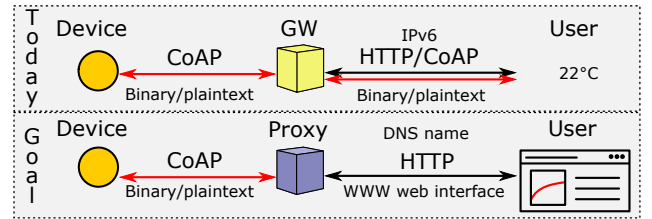

Figure 1. Our approach serves users web interfaces of embedded web services on constrained devices.

One of the benefits of this approach is that all user interactions with the constrained devices make use of standard web technology (i.e. TCP/HTTP/WWW). As a result, the web interfaces are available to a wide range of user devices (only a web browser is needed). The proxies implement a template lookup interface that returns the web interface template to be used for rendering a response to the user. This lookup interface takes into account the resource type of the RESTful resource and the device type of the user device. Combined with leveraging the CoAP standard, different web interfaces can be rendered for different types of resources thereby supporting a wide range of constrained devices. Additionally, the device detection of the proxies combined with well-known web technology for designing templates enable user friendly interfaces that are adapted to the device of the user.

As CoAP is a highly optimized, binary application protocol specifically designed for constrained environments, the impact of our approach on constrained devices in terms of
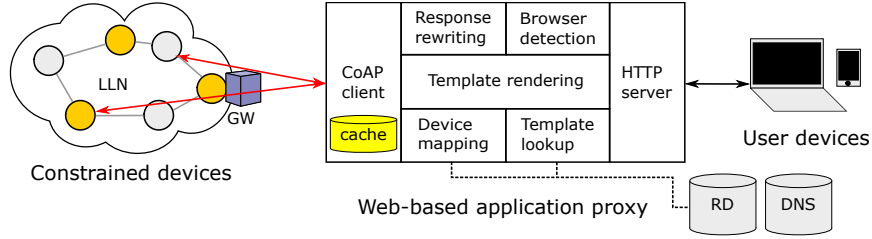
Figure 2. System overview

memory usage, communication overhead and battery life is kept to a minimum. Also, this approach does not require any changes to the software running on the constrained devices or the user's web browser. Finally, by interfacing with directory and naming services the proxies enable easy discovery and naming of devices respectively.

C. Design

An overview of the resulting system is shown in figure 2. The devices through which a user interacts with the constrained devices are shown on the right, the constrained devices are shown on the left. The middle shows the seven submodules which make up the design of the web-based application proxy. In order to offer all necessary functionality the proxies also interface with a Resource Directory (RD) [7] and a DNS name server, these are shown in the bottom right.

When a user directs its browser to a constrained device via the proxy, the proxy starts processing the HTTP request by extracting the target CoAP URI of the constrained device from the hosting HTTP URI. Depending on whether the proxy is processing the request as a forward or reverse HTTP-CoAP cross protocol proxy (as per "Guidelines for HTTP-to-CoAP Mapping Implementations" terminology [8]), the extraction method will differ. In the forward proxy case, the target CoAP URI can readily be extracted from the hosting HTTP URI; e.g.: http://proxy.example.com/hc/coap://s.example.com/light. In the reverse proxy case, where the user surfs directly to the device (e.g. http://s.example.com/light), the device mapping submodule maps the HTTP URI to the target CoAP URI.

Once the target CoAP URI is known, the proxy looks up the template for rendering the CoAP resource response to the user. The template lookup module maintains a database of templates for different CoAP resource types and different web browsers (mobile, desktop and miscellaneous). In case the resource type of the CoAP resource is unknown, the template lookup module contacts the Resource Directory to retrieve all meta information related to the target CoAP URI. The browser of the user is identified by the browser detection module. This module processes the HTTP header fields (mainly the User-Agent header) and determines whether the user is surfing from a mobile or desktop device. Once the user's browser type and CoAP resource type are

known, the template lookup module searches for a matching web template and returns the result to the template renderer. In case no matching template is found, a default template may be used depending on the user browser.

Apart from the web page contents, a template also specifies whether the proxy should wait for the CoAP response before rendering the template and returning the HTTP response to the user. As CoAP response times might be long and unpredictable (order of seconds), the user could experience long delays if the proxy were to wait on the CoAP response for rendering the web interface. Therefor, templates that anticipate long response times can indicate to the proxy that they should be rendered immediately. These templates then retrieve the CoAP response (via the proxy) once they have been rendered by the browser of the user.

For retrieving CoAP responses, a template may employ Asynchronous JavaScript And XML (AJAX) techniques to send an AJAX request to the hosting HTTP URI. The proxy detects that the request is an AJAX request (via the HTTP XMLHttpRequest header) and skips the web template lookup procedure (the web browser is detected as a miscellaneous device in this case). Instead the proxy sends a CoAP request to the target CoAP URI and returns the CoAP response in the AJAX response (which might take a long time). The template is then free to process the AJAX response: e.g. update a text area, a graph, an HTML form, etc. Additionally, these AJAX techniques can be used to drive an actuator (via PUT or POST requests), to poll a resource periodically (e.g. while updating a graph), ... Note that services building on top of the proxy for data access, will not be served a web interface as their user agent is not recognized as a web browser: e.g. the user-agent of the urllib HTTP client in Python 3.4 is "Python-urllib/3.4'. In this case the proxy operates as a standard HTTP-CoAP proxy.

The CoAP client module in the proxy sends requests to the constrained devices for retrieving CoAP responses. It incorporates a cache in order to speed up response retrieval.

The final module in the design is the response rewriting block. For certain Content-Types, this block rewrites the CoAP response in order to display it in the web interface. At the moment, the block only rewrites CoRE link format responses [9] by replacing web links with links that are handled by the proxy. This is necessary when discovering

CoAP devices and resources via the proxy, as explained next.

D. Device mapping, discovery and naming

In the reverse HTTP-CoAP configuration, one might wonder how the device mapping module builds the mapping from HTTP to CoAP URIs. As minimal configuration is an important requirement, the user cannot be expected to maintain this mapping. Instead, the proxy retrieves a list of known constrained devices from the Resource Directory and assigns reverse IPv6 LAN endpoints for each of these devices. In order to make these new endpoints discoverable, the proxy registers the reverse endpoints in the RD (with the same resources as the constrained endpoint). Thus the RD contains both the known constrained devices (in a non-default domain, which is used by the proxy) and the corresponding reverse endpoints (in the default domain, which is used by users for discovery). As a result, when users surf to a reverse endpoint (as discovered in the RD), the proxy is readily able to determine the target CoAP URI. An example of this mapping and discovery procedure is presented in the evaluation section.

In the forward HTTP-CoAP configuration, the device mapping is not needed as the URI mapping is explicit. In this configuration, the user discovers the proxy by means of a HTTP resource with resource type "core.hc" (as per [8]) in the proxy's .well-known/core.

As mentioned earlier, the use of IPv6 and 6LoWPAN can lead to long IPv6 literals in hosting HTTP URIs. To remedy this, the proxy offers a /dns resource for each constrained device that renders a form where users can set a DNS hostname for the constrained device. Afterwards, users can use the host name instead of the IPv6 literal for surfing to the device. Alternatively, the host name could also be retrieved from a CoAP resource on the constrained device itself (e.g. in case the device was preprogrammed with a host name).

IV. EVALUATION

A. Evaluation setup

For evaluating the web-based application proxy, extensive tests are performed using the setup depicted in figure 3. The setup consists of two types of constrained devices: Zolertia Z1s sensor nodes and nodeMCU ESP8266 nodes. There are eight Z1s that form a 6LoWPAN LLN where one Z1 is connected via SLIP to the Raspberry Pi as the border router. Note that the 6LoWPAN network is a private network as the IPv6 prefix (fd00::/64) is a unique local prefix. The Z1s are equipped with a msp430f2617 micro controller (8KB RAM and 92KB flash memory), an IEEE 802.15.4 CC2420 transceiver and run the Contiki OS. The nodeMCUs are IPv4 only devices and are connected to the Raspberry Pi via the Wi-Fi access point. NodeMCUs are based on the low-power ESP8266 ESP-12E Wi-Fi SoC and have 32KB RAM and 4MB flash memory. Both the Z1s and the nodeMCUs are running CoAP servers. All

constrained devices are configured to register themselves with the Resource Directory at start-up.
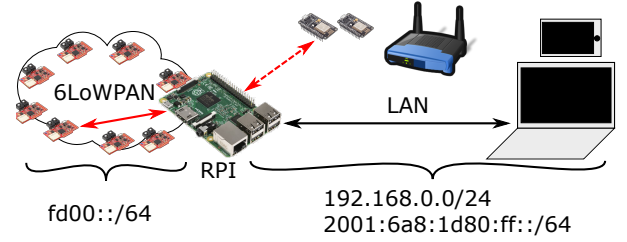

Figure 3. Evaluation setup: 6LoWPAN Zolertia Z1's (red) and 802.11 WLAN nodeMCUs (gray) as constrained devices. Raspberry Pi as application proxy. Red arrows indicate CoAP exchanges, black arrows HTTP exchanges. Solid lines indicate IPv6 datagrams, dashed lines IPv4.

The Raspberry Pi is a dual-stack device and is part of both the 6LoWPAN network and the LAN network. The RPI runs the application proxy, as well as the resource directory and the DNS server for the LAN network. The application proxy is implemented as part of our CoAP++ framework, which is built on top of Click Router. The proxy is configured as a reverse proxy for each of the constrained devices. As such, the proxy listens for new device registrations in the RD, allocates reverse endpoints in the IPv6 LAN network, stores the resulting device mapping and registers the allocated endpoint in the resource directory. As a result, the user can access constrained devices (6LoWPAN or Wi-Fi) through the reverse endpoints via the proxy. Finally, the notebook is running Ubuntu 14.04 and the smart phone is a Google Nexus 5. The round trip time between the Z1's and the notebook was measured via ping6 and is on average (μ) 163.1 ms with standard deviation (σ) 69.4 ms.


Figure 4. Device discovery via the RD endpoint lookup interface

B. Functional evaluation

In order to discover the constrained devices, the user surfs to the resource directory in its browser: http://rd.test/ which redirects the user to the rd-lookup/ep web interface. This

HTTP request is handled by the proxy and is translated into a CoAP request for the local resource directory. The CoRE link format discovery response is rewritten to a HTML table and this table is rendered in the template for the core.rd-lookup resource type. Figure 4 shows the discovery response in the mobile Google Chrome browser.

Next, the user taps on the HTTP link of the device of interest which takes the user to the .well-known/core resource. Here all the resources offered by the device as well as HTTP links are listed, as shown in figure 5.

z1rc2055.test/.well-known/core

Easy to use IoT
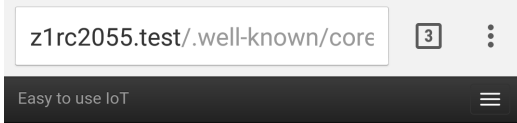
z1rc2055.test/.well-known/core

Web link	Title	Resource type	Content type	HTTP link
/.well-known/core			40	http://z1rc2055.test/.well-known/core
/deviceName		ibcn.dev.name		http://z1rc2055.test/deviceName
/owner		ibcn.owner		http://z1rc2055.test/owner
/d		ibcn.dev		http://z1rc2055.test/d
/sensors/temp		ibcn.temp		http://z1rc2055.test/sensors/temp
/actuators/fan		ibcn.fan		http://z1rc2055.test/actuators/fan
/location		ibcn.location	0	http://z1rc2055.test/location
/image		ibcn.image	1001	http://z1rc2055.test/image
/dns		ibcn.dns	0	http://z1rc2055.test/dns

Figure 5. Rendering .well-known/core of a constrained device

Finally, the user taps on a resource of interest to interact with. Depending on the resource type, the interface will be different. Figure 6(a) shows a template that periodically updates a graph (using chartjs.org) of a temperature resource. The underlying '/sensors/temp' CoAP resource returns plain-text temperature readings. Figure 6(b) shows a template that renders a button for controlling an actuator (in this case a LED is turned ON and OFF). Tapping the button sends an HTTP POST request to the resource on the proxy, which is sent to the underlying CoAP resource by the proxy. Note that the temperature resource is hosted on a 6LoWPAN device, whereas the actuator resource is hosted on a Wi-Fi device.

C. Interface responsiveness: load times

The previous section illustrated the user interfaces that can be expected from our approach. An important performance metric of such user interfaces is the responsiveness: e.g. a sluggish interface can ruin the user experience. To qualify the responsiveness, the load times for two different types of templates are compared: a simple template, which blocks on the CoAP response before it is rendered, and an AJAX template, which is rendered immediately and fetches the

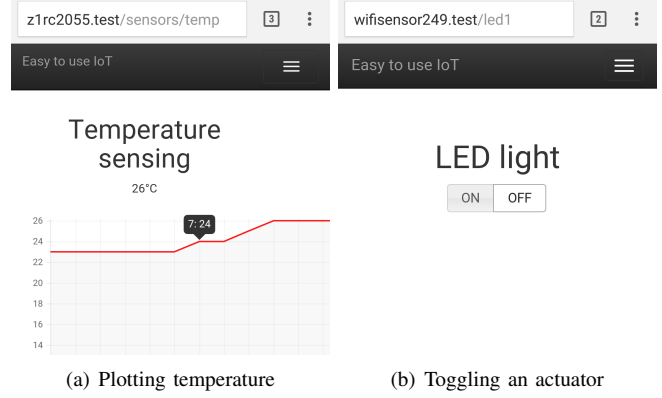(a) Plotting temperature (b) Toggling an actuator

Figure 6. Different templates are rendered depending on the resource type of the target CoAP resource: e.g. ibcn.temp and ibcn.light are shown here.

response afterwards. In order to measure worst case responsiveness, caching in the CoAP client has been disabled.
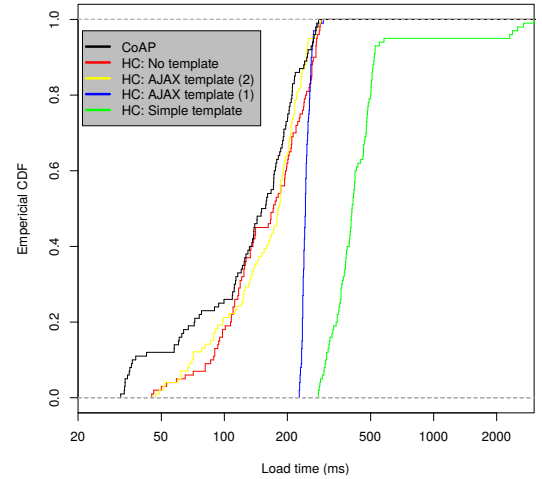

Figure 7. CDFs of load times for different proxy configurations

The cumulative distribution functions of the load times are plotted in figure 7 (for 100 measurements per function). Notice that the load time[2] of the AJAX template (blue line) is always smaller and more consistent when compared to the simple template (green line). While the AJAX template has to perform a second request to fetch the CoAP response (yellow line), it is already rendered in the browser before this second request is issued. Also note that when the round trip time to the constrained devices would increase, the load time for the simple template would increase (green line would shift to the right) whereas the load time for the AJAX template remains constant as it does not depend on constrained device communication. As such, AJAX templates are clearly superior to simple templates in

[2]As browsers typically render a considerable fraction of the UI before the load time has expired, the load time is considered an upper limit for the UI responsiveness. However, larger average load times do indicate a decrease in UI responsiveness.

terms of responsiveness. Finally, the difference between the CoAP (black) and no template (red) lines shows that the delay introduced by our web interface rendering approach is around 14 ms (their minimums differ 13.9 ms).

V. RELATED WORK

There are many options for improving user interactions with constrained devices. In the HTTP-CoAP protocol category, the work of Ludovici et al. [10] presents a design of a forward cross protocol proxy that supports event-like notifications through WebSockets as an alternative to HTTP long polling. In contrast to our work, the proxy of Ludovici et al. does not provide a user interface for web browser-based users. Additionally, the proxy only operates in a forward configuration which requires the user to support the URI format implemented by the proxy. In [11] Colitti et al. describe both a HTTP-CoAP proxy and a HTTP web application for visualizing sensor measurements from a wireless sensor network. While the proposed proxy does implement a reverse proxy model, the web application is written as a stand-alone application on top of the HTTP-CoAP proxy. As such, the approach differs from ours where the template rendering is an integral part of the cross protocol proxy. In [12] Jin et al. present a CoAP service gateway for automatically creating service mash-ups based on semantic similarity between related CoAP servers. While CoAP SG includes a HTTP-CoAP proxy that can return plain-text or JSON HTTP responses, the focus is on aggregating data from multiple CoAP servers rather than on generating user interfaces. Nevertheless, the work does prove that proxies on gateways are valuable for implementing extra functionality.

VI. CONCLUSION

This paper presented a number of methods for improving user interactions with constrained devices. Essential in implementing these methods is the presented HTTP-CoAP proxy, which renders user interfaces for RESTful resources of constrained devices based on web templates. The paper demonstrated this concept by means of two templates for constrained device resources and one template for discovery of devices and resources. Additionally, the interface responsiveness and the delay of our approach was also quantified.

In the future, the authors plan to extend the concept to include transport layer security (i.e. HTTPS-CoAPs proxy) and to support other functionality than user interface generation (e.g. data aggregation, data format rewriting).

ACKNOWLEDGMENT

REFERENCES

[1] D. Evans, "The internet of things: how the next evolution of the internet is changing everything," 2011. [Online]. Available: https://www.cisco.com/c/dam/en{_}us/about/ac79/docs/innov/IoT{_}IBSG{_}0411FINAL.pdf

[2] Ericsson Inc, "More than 50 Billion Connected Devices - Taking connected devices to mass market and profitability," 2011. [Online]. Available: http://vdna.be/publications/Wp-50-Billions.Pdf

[3] P. Middleton, T. Tully, J. F. Hines, T. Koslowski, B. Tratz-Ryan, K. F. Brant, E. Goodness, A. McIntyre, and A. Gupta, "Forecast: Internet of Things - Endpoints and Associated Services, Worldwide, 2015," p. 57, 2015. [Online]. Available: https://www.gartner.com/doc/3159717/forecast-internet-things--endpoints

[4] A. Asín and D. Gascón, "50 Sensor Applications for a Smarter World: Libelium white paper," *Libelium*, 2012. [Online]. Available: http://www.libelium.com/top{_}50{_}iot{_}sensor{_}applications{_}ranking/

[5] C. Bormann, M. Ersue, and A. Keranen, "RFC 7228: Terminology for Constrained-Node Networks," Tech. Rep., 2014. [Online]. Available: http://tools.ietf.org/html/rfc7228

[6] I. Ishaq, D. Carels, G. K. Teklemariam, J. Hoebeke, F. Van den Abeele, E. De Poorter, I. Moerman, and P. Demeester, "IETF standardization in the field of the Internet of Things (IoT): a survey," *Journal of Sensor and Actuator Networks*, vol. 2, no. 2, pp. 235–287, 2013.

[7] Z. Shelby, M. Koster, C. Bormann, and P. van der Stok, "CoRE Resource Directory," 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-core-resource-directory-07

[8] A. P. Castellani, S. Loreto, A. Rahman, T. Fossati, and E. Dijk, "Guidelines for HTTP-to-CoAP Mapping Implementations," 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-core-http-mapping-10

[9] Z. Shelby, "RFC 6690: Constrained RESTful Environments (CoRE) Link Format," 2012. [Online]. Available: https://tools.ietf.org/html/rfc6690

[10] A. Ludovici and A. Calveras, "A Proxy Design to Leverage the Interconnection of CoAP Wireless Sensor Networks with Web Applications," *Sensors*, vol. 15, no. 1, pp. 1217–1244, jan 2015. [Online]. Available: http://www.mdpi.com/1424-8220/15/1/1217

[11] W. Colitti, K. Steenhaut, N. D. Caro, B. Buta, and V. Dobrota, "REST Enabled Wireless Sensor Networks for Seamless Integration with Web Applications," in *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, 2011, pp. 867–872.

[12] X. Jin, K. Hur, S. Chun, M. Kim, and K. H. Lee, "Automated mashup of CoAP services on the Internet of Things," *IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings*, pp. 262–267, 2016.