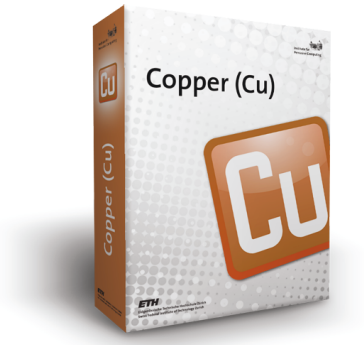
CoAP for the Web of Things: From Tiny Resource-constrained Devices to the Web Browser

Matthias Kovatsch
Institute for Pervasive Computing
ETH Zurich
Zurich, Switzerland
kovatsch@inf.ethz.ch



UbiComp'13 Adjunct, September 8–12, 2013, Zurich, Switzerland.

ACM 978-1-4503-2215-7/13/09...$15.00.

http://dx.doi.org/10.1145/2494091.2497583

Abstract

The Constrained Application Protocol (CoAP) is a new Web protocol standardized by the IETF. It is not a mere compression of HTTP, but a re-design from scratch following the REST architectural style. Thus, its features are tailored for Internet of Things (IoT) applications and machine-to-machine (M2M) scenarios with highly resource-constrained devices. While this makes CoAP very interesting for the Web of Things (WoT) initiative, it is still detached from the Web world of browsers and intuitive user interaction. We present the first attempts to unite these two worlds, so that everyday objects endowed with tiny, low-cost computing devices can become first class citizens of the Web. Our Copper (Cu) project brings CoAP support to the Web browser and has been out in the wild since late 2010. Thus, we were able to conduct a user study among industry and research developers who know both, Web-based CoAP and earlier proprietary protocols for networked embedded systems. The result shows that industry developers and those with longer experience agree even more that Internet protocols and patterns from the Web ease application development for tiny, resource-constrained devices.

Author Keywords

CoAP; WoT; IoT; Web browser; scripting; user study

ACM Classification Keywords

H.5.m [Information interfaces and presentation]: Misc.

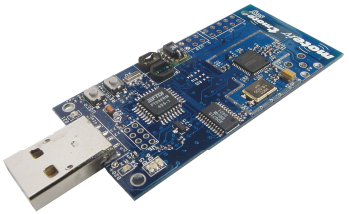


Figure 1: The Tmote Sky is a highly resource-constrained platform: It only has 48 KiB of ROM and 10 KiB of RAM. It is able to implement an RFC-compliant IP stack together with an energy-saving MAC protocol and CoAP at the application layer. The program flash is, however, too small to also include the cryptographic libraries required for a full security handshake. Requiring extreme optimization or external support makes the Tmote Sky a borderline Class 0 device.

Introduction

Web technology is ubiquitous. Using the Web has become part of our everyday lives and, moreover, many tech-savvy people without explicit training are able to create their own Web applications. Thus, the Web of Things initiative aims for applying the well-known and proven patterns from the Web to the demanding IoT domain [3, 20]. As a result, devices can be browsed and bookmarked, Web pages can directly include real-time data from sensors, and users can build physical mashups that augment and control their everyday objects [5]. This is an important step from traditional networked embedded systems to a truly ubiquitous Internet of Things.

In this course, the IETF has been standardizing the Constrained Application Protocol (CoAP) [16]. It is a new Web protocol that was designed to meet the requirements of highly resource-constrained devices and M2M scenarios. CoAP closes the gap between microcontroller-based low-power devices and the Web of Things, as HTTP over TCP is not feasible in these environments. RESTful applications can now talk end-to-end to tiny devices using URIs for addressing and uniform interfaces for interaction.

Yet CoAP targets M2M applications without the direct user interaction known from the Web of Things. Our goal is to push CoAP closer to the Web world of browsers with intuitive front-ends, easy scripting, and the human in the loop. For this, we enabled CoAP support directly in the Web browser and released *Copper (Cu)*[1], a Firefox add-on whose source code is also available on GitHub[2]. In another project, we extended the JavaScript language with an API for direct interaction with tiny IoT devices [9]. Our *CoapRequest* object is similar to AJAX's XmlHttpRequest and allows to create mashups with tiny devices without the need for application-level gateways.

The WoT initiative mainly advocates the ease of application development resulting from the adoption of Web patterns. To evaluate this intuitive feeling, Guinard at al. conducted a study with 69 computer science students [4]. It shows that REST is considered easier to learn and more suitable for programming IoT devices than the RPC-like WS-* Web services.[3]

With Copper (Cu) available since late 2010 and around 400 steady users, we were able to find 48 participants to conduct a related study that, apart from researchers, also includes 16 industry developers. This paper focuses more on the Web integration of devices opposed to traditional networked embedded systems. Those usually run proprietary protocols that are highly optimized for a given application, but increase development costs and cause technological islands. The study supports our hypothesis that Internet and Web protocols ease development in this domain. Furthermore, it also gives good input on how to continue with the seamless integration of CoAP into the existing Web.

Constrained Application Protocol (CoAP)

In 2010, the Internet Engineering Task Force (IETF) charted a new working group for "Constrained RESTful Environments" (CoRE). Its goal is to provide a framework for resource-oriented applications intended to run on constrained IP networks. These networks consist of resource-constrained devices, which the IETF divides into

[1] https://addons.mozilla.org/de/firefox/addon/copper-270430/
[2] https://github.com/mkovatsc/Copper

[3] In a business environments where many WS-* services and moreover policies for QoS and security are already established, WS-* is recommended. Here, the Devices Profile for Web Services (DPWS) is a good alternative that also builds on IP connectivity for resource-constrained devices. [10]

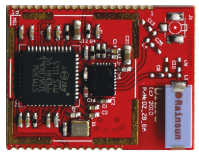


Figure 2: New platforms with an ARM Cortex-M3 SoC fall into the Class 1 category. With 128 KiB of ROM and 16 KiB of RAM they have enough space for a full network stack secured through Datagramm Transport Layer Security (DTLS). Larger Cortex versions with up to 1 MiB of ROM already fall into Class 2.

the following three classes. Note that the given memory sizes are not strict rules, but rather a rule of thumb. The classification is mainly decided through the device capabilities:

Class 0 devices are not capable of running an RFC-compliant IP stack **in a secure manner**. They require application-level gateways to connect to the Internet.

Class 1 devices are the most resource-constrained devices that can directly connect to the Internet with integrated security mechanisms. This requires about 100 KiB of ROM and about 10 KiB of RAM. They cannot employ a full protocol stack using HTTP over TLS, though, and require lightweight and energy-efficient protocols. This class is in the focus of the CoRE working group.

Class 2 devices almost show the characteristics of normal Internet nodes like notebooks or smartphones, which is possible with about 250 KiB of ROM and about 50 KiB of RAM. Yet they can still benefit from lightweight and energy-efficient protocols to free resources for the application or reduce operational costs.

Also other networking aspects are subject to constraints such as low achievable data rates and high packet loss. Class 1 devices have very limited buffers and mostly use low-power communications such as IEEE 802.15.4 through 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) [7]. The verbosity of HTTP is ill-fitted for such environments. Also TCP performs badly when it is only used for short-lived request/response exchanges instead of bulk data transport. The SYN/ACK handshake alone might take a second when a low radio duty cycle is required for battery operation.

Thus, the working group designed a new protocol suite from scratch that goes far beyond a simple compression of HTTP (cf. EBHTTP [19]). Following the REST architectural style [2], CoAP is based on patterns from the Web: a request/response interaction model between application endpoints, uniform interfaces that allow for interoperability, resources that are addressable by URIs, Internet Media Types that represent resource state, and caching and proxying to enable scalability.

CoAP is a binary protocol that runs over UDP. A messaging sub-layer adds a thin control layer that provides duplicate detection and optionally reliable delivery of messages based on a simple stop-and-wait mechanism for retransmissions. On top, the request/response layer enables RESTful interaction through the well-known methods `GET`, `PUT`, `POST`, and `DELETE` as well as response codes that are defined with only a few deltas to the HTTP specification [1]. In addition, CoAP offers features that make the real difference for the IoT.

Observing Resource
CoAP enables native push notifications. Clients can "observe" resources for state changes through a simple publish/subscribe mechanism [6]. The server keeps track of interested clients and pushes the new representation whenever the observed resource changes. This follows a best-effort approach and aims to guarantee eventual consistency. That means not every state change will arrive at the clients, but all clients will eventually receive the latest representation of the resource. This can be compared to the debouncing of a button, where intermediary states can be ignored and filtering is actually desirable. However, Reliable propagation of every event can still be achieved with the right design of the resource and its representations.

Group Communication
Being based on UDP, CoAP is able to use IP multicast to provide RESTful group communication. Clients can use safe and idempotent methods to interact with a group of devices. Group management and best practices are defined in a supplementary document [13].

Resource Discovery
The CoRE working group also defined a format for resource metadata, the CoRE Link Format [15]. It is based on Web Linking [11] and uses link attributes to provide information such as the provided content formats ("ct") or and the maximum expected size ("sz") of a resource. It is also used to annotate light-weight semantic information such as resource type ("rt") or interface usage ("if"). CoAP servers provide a list of all their resources at the well-known URI path `/.well-known/core` [12]. This list can also be filtered using the desired attributes as URI query parameters. On start-up, endpoints can also register at a resource directory [17] to enable look-ups while the node is sleeping or otherwise disconnected from the network.

Alternative Transports
Although UDP is the primary transport for CoAP, the protocol was designed to run over several alternatives as well, including TCP for back-end purposes. Many IoT devices are connected though cellular networks that do not have IP connectivity, for instance to lower implementation costs, due to limited coverage, or to temporarily conserve energy. They can use the Short Message Service (SMS) or Unstructured Supplementary Service Data (USSD) bindings of CoAP, as those transports match the properties of constrained RESTful environments [18].

`coap+sms://+123456789/container/bananas/temperature`

Figure 3: CoAP does not only work over IP. Alternative transports also use URIs for addressing. Being delay tolerant, CoAP can for instance run over SMS, which uses its own reliable and delay-tolerant delivery.

CoAP in the Web Browser

To prototype the full Web experience for tiny IoT devices, we implemented an add-on for Mozilla Firefox [8]. Our Copper (Cu) Coap user-agent allows interaction with embedded Web resources by simply entering a CoAP URI into the addressbar and using the RESTful methods `GET`, `PUT`, `POST`, and `DELETE`. It is comparable to other REST add-ons such as Poster[4] or RESTClient[5], but Cu in addition implements CoAP. It registers a protocol handler for the coap URI scheme, which integrates these URIs seamlessly into the browser. Users can browse devices, bookmark their resources like normal Web pages, and follow links in HTML documents to discover new devices.

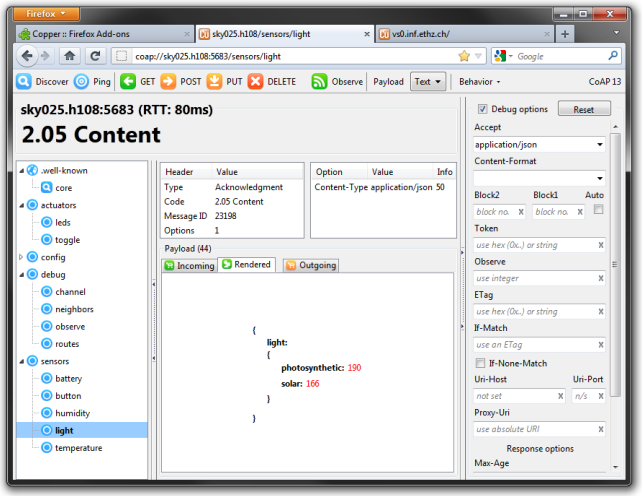


Figure 4: The server resources are discovered through the CoRE Link Format and shown on the left for browsing. The debug options on the right are optional and can be used to set header options to custom values for testing and debugging.

[4] https://addons.mozilla.org/en-US/firefox/addon/poster/
[5] https://addons.mozilla.org/en-US/firefox/addon/restclient/

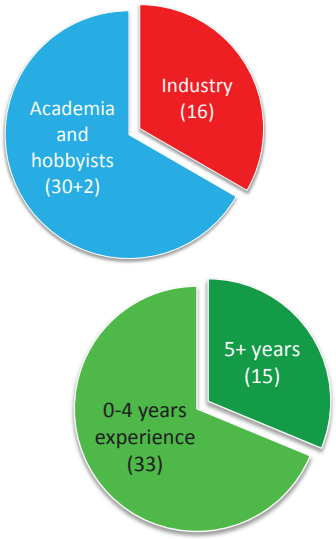


Figure 5: Our study has 48 participants of which 16 are using IoT devices or CoAP as industry developers. Two participants are pure hobbyists. In the evaluation, we counted them as non-industry, i.e., academia.
The average experience is 4.3 years whereas 15 participants have been active in the field for at least five years.

Our add-on can also render different Internet media types typically provided by devices, e.g., JSON as depicted in Figure 4.

Copper (Cu) primarily targets developers that want to explore, test, and debug RESTful Web services based on CoAP—for HTTP-based services, the Web browser is already a popular tool to do so. In addition to the basic GUI elements to browse resources and issue different requests, Cu provides manual override for the full set of CoAP header options and a detailed log in the Firefox console, so developers can intensively test their applications or own CoAP implementations. While experienced users can also use Copper (Cu) to configure their devices or retrieve data from them, it is not made for pure end-users of WoT applications. Our add-on is a generic browser for tiny resource-constrained devices and thus is missing a presentation layer which is usually application-specific.

Although Copper (Cu) is fully written in JavaScript, CoAP requests cannot be issued from other scripts running in the browser. This means, JavaScript from an external Web page cannot include CoAP resources in an AJAX-like manner and user scripting is only available through editing the add-on sources. Our assumption was that WoT mashups would mainly be faceless scripts that augment and automate devices invisibly in the background. For this, we created the CoapRequest object for a standalone RESTful runtime container that executes server-side JavaScript [9]. The following study shows, however, that there is a broad interest in AJAX-like interaction with CoAP-enabled devices on Web pages to provide application-specific front-ends for end-users.

CoAP User Study

Following up the study by Guinard et al. [4], we published a questionnaire to evaluate the WoT vision in the context of highly resource-constrained networked embedded systems. In this domain, the use of IP and Web patterns is relatively new and numerous alternative, often proprietary, protocols exist. To find enough experts who know and worked with both approaches and, hence, can give a qualified feedback, we advertised the study over the following channels: (i) The Contiki[6] and TinyOS[7] mailing lists, which reach mostly researchers in the area of Wireless Sensor Networks, (ii) European Telecommunications Standards Institute (ETSI) M2M associates, who have a good overview of the available technologies for IoT and M2M solutions, and (iii) followers of the IETF standardization, who often have decades of experience in the field. The latter two groups mostly consist of people with an industry background.

We received N=48 responses from people who worked with CoAP or IoT devices as researchers (51%), industry developers (34%), students (28%), hobbyists (9%), and lecturers (4%); multiple roles are possible. The experience in the relevant fields (wireless sensor networks, traditional networked embedded devices, and Web technologies) varies from beginners to experts with up to 20 years of experience (4.3 years on average). Their answers are used to confirm or reject the following hypotheses:

1. Internet protocols and Web patterns ease the development of distributed software for resource-constrained devices.
2. CoAP is a required extension for the Web to integrate resource-constrained devices.

[6] http://www.contiki-os.org/
[7] http://www.tinyos.net/

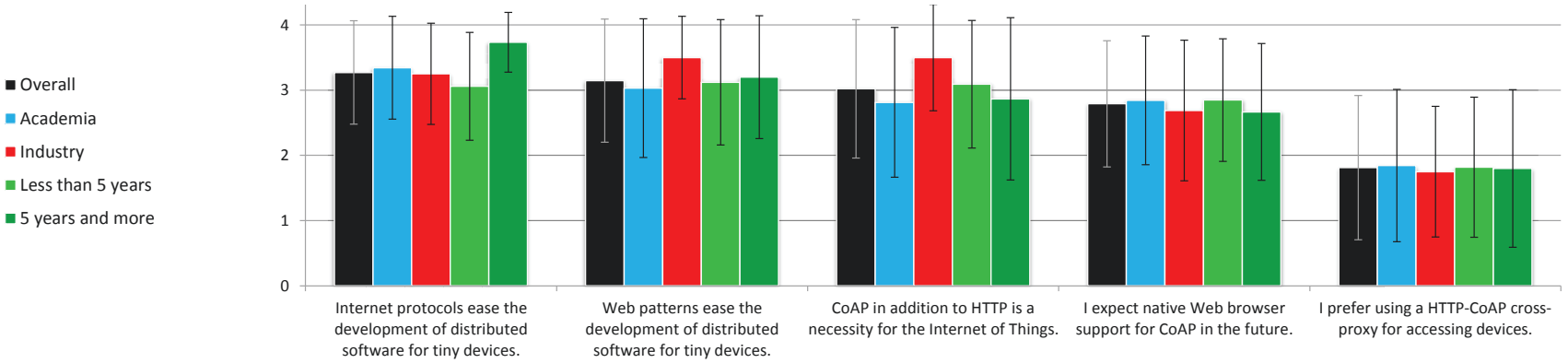


Figure 6: Likert scale responses by our 48 participants (0 = strongly disagree, 4 = strongly agree, error bars: +/- 1 std. dev.)

3. The Web browser is a preferred tool to interact with these devices when there is no physical interface meant for direct interaction.
4. Other tools are only preferred when a task is to be automated through a script or program.

IP, Web Patterns, and CoAP

For the first part, we used a five-level Likert scale from *strongly disagree* (0) to *strongly agree* (4), i.e., resulting values of more than 2 mean agreement with our statement. Based on the given background information (see Figure 5), we separated the responses into different groups: (i) academia, (ii) industry, (iii) less than five years of experience, and (iv) five or more years of experience. For each comparing statements (e.g., academia vs industry), we perform the *Wilcoxon rank-sum test* to see if the two sets significantly different from each other. For each statement, we give the corresponding p-value together with the sample sizes.

Figure 6 shows general agreement on our first hypothesis. Overall, Internet protocols are slightly more accepted to ease development (3.3) than Web patterns (3.1). They are also more appreciated by participants with longer experience (3.7 vs 3.1 with less then five years).[8] Interestingly, participants with an industry background agree more with the advantages of Web patterns (3.5) than academia (3.0).[9]

Our second hypothesis about the necessity of CoAP can also be confirmed, although the overall agreement is slightly lower (3.0) and has a slightly higher standard deviation (1.06). CoAP as additional Web protocol finds high acceptance among participants with an industry background (3.5 vs 2.8 in academia), though.[9] (similar value)

The participants are a little less confident about native CoAP support in Web browsers (2.8), although they

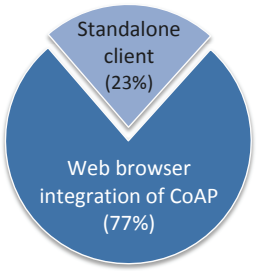


Figure 7: Most participants preferred the Web browser as client for device interaction.

[8] 0-4 vs 5+ years: $p - value < 0.01, N_{0-4} = 33, N_{5+} = 15$

[9] Industry vs academia: $p - value < 0.001, N_i = 16, N_a = 32$

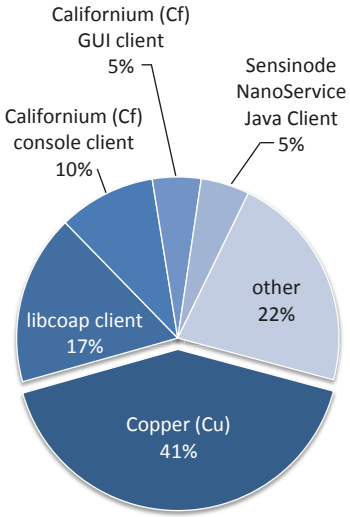


Figure 9: Market share of different CoAP clients on workstations and laptops. Clients mentioned only once or without specific name are consolidated in "other."

rather disagree with the usage of an HTTP-CoAP cross-proxy (1.8). Conversely, the latter means that direct communication with devices is preferred.

CoAP support in Web Browsers
In a second part, we directly asked what way of user interaction with tiny devices is preferred. 77% voted for the Web browser over a standalone CoAP client. The most agreed-on reasons are that this way, no additional software is required (3.1, stdev. 0.9) and that the Web integration feels natural (3.0, stdev. 1.0). The only agreed-on reason for using a standalone CoAP client was that it is better suited for scripting and integration into a larger system (2.9, stdev. 0.7). Other reasons had more or less neutral outcome.

Figure 8 (on the next page) shows the client usage profiles of the participants. Being a contributor to the CoRE working group, our CoAP implementations have always been early reference implementations for others. Thus, one of the main use cases is debugging own implementations. The other applications are in line with the role of a Web browser in traditional RESTful Web services and reflect the preference for Web browser integration of CoAP.

CoAP Client Market Share
Copper (Cu) is currently the only CoAP client available that integrates into the Web browser. Figure 9 shows that 41% of the participants who use CoAP on their workstation or laptop (N=41) use our browser add-on as primary client, followed by the command line *libcoap client* with 17%. The main reason for using a different CoAP client was by far the need for automation and scripting with 55% of the answers and the "I do not use Firefox" runner-up with mere 8%. Although the JavaScript source code of our add-on is available, only 2 participants (6%) have adapted it to their needs. 47% stated that they are "not familiar enough with Firefox add-ons" to do so, while the remaining 47% did not consider this option.

Conclusion

Developers from industry and academia that deal with networked embedded systems are convinced that Internet protocols and patterns from the Web facilitate their job. Also the Web-like interaction with tiny devices is preferable, as 77% favor the Web browser integration of CoAP over standalone clients. From the expectations but even more from the individual comments, we conclude that users would prefer full CoAP support in Web browsers. A primary concern is the creation of intuitive Web front-ends that directly include device data without the need for a cross-proxy. We also take away that CoAP scripting support within the browser would be highly appreciated and that our CoapRequest object API is the right approach. This would, for instance, satisfy the feature requests for logging and visualization of historic values, since well-known libraries such as jQuery Flot[10] can be used for this.

In the future, we will continue with the Web browser integration and provide AJAX-like CoAP support by porting Actinium's CoapRequest object to the browser. Another concern is the support for CoAP's security modes. Due to the efforts of the IETF "Real-Time Communication in WEB-browsers" working group, browser vendors already started to integrate DTLS, e.g., for direct video chat or gaming [14]. CoAP**s** support, however, requires additional patches to make DTLS available through the scripting API. This could become the next big step towards native CoAP support in the Web browser and a prosper Web of Things.

[10] http://www.flotcharts.org/

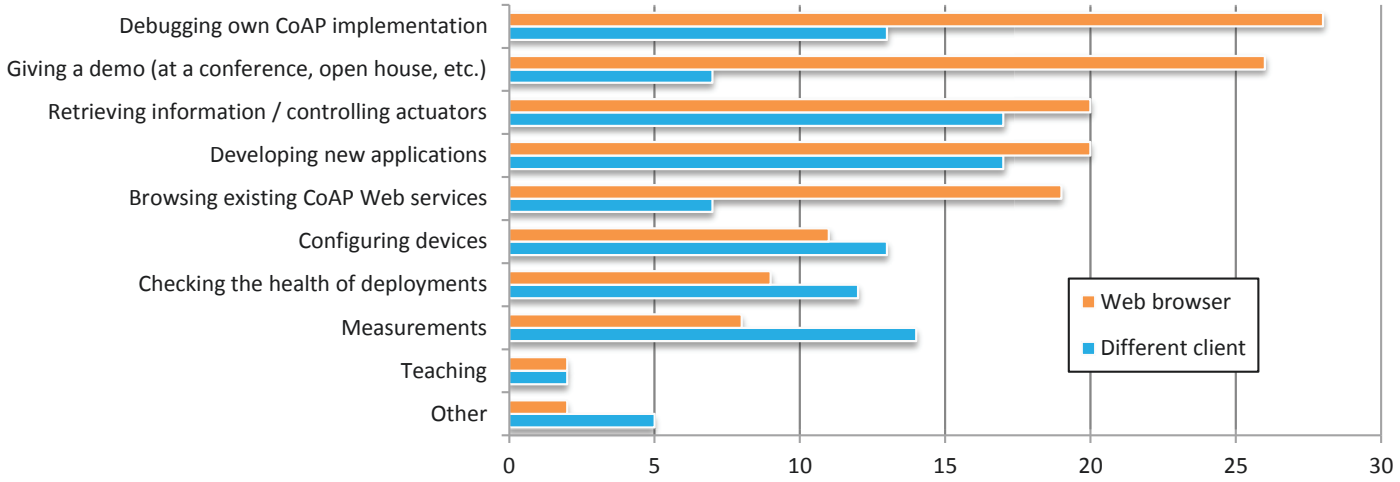


Figure 8: Copper (Cu) has been available since CoAP draft 03. Thus, one of the main applications of it is the debugging of other CoAP implementations. Still, there have been other clients around and usability must have been the reason for choosing this reference implementation in particular.

Acknowledgements

We want to thank all study participants for their time. Special thanks to those, who provided us with detailed feedback in the individual comment.

References

[1] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, 1999.

[2] Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[3] Guinard, D. *A Web of Things Application Architecture - Integrating the Real-World into the Web*. PhD thesis, ETH Zurich, 2011.

[4] Guinard, D., Ion, I., and Mayer, S. In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective. In *Proc. MobiQuitous* (Copenhagen, Denmark, 2011).

[5] Guinard, D., Trifa, V., Pham, T., and Liechti, O. Towards Physical Mashups in the Web of Things. In *Proc. INSS* (Pittsburgh, PA, USA, 2009).

[6] Hartke, K. Observing Resources in CoAP. I-D: draft-ietf-core-observe-08, 2013.

[7] Hui, J., and Thubert, P. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282, 2011.

[8] Kovatsch, M. Demo Abstract: Human–CoAP Interaction with Copper. In *Proc. DCOSS* (Barcelona, Spain, 2011).

[9] Kovatsch, M., Lanter, M., and Duquennoy, S. Actinium: A RESTful Runtime Container for Scriptable Internet of Things Applications. In *Proc. IoT* (Wuxi, China, 2012).

[10] Lerche, C., Laum, N., Moritz, G., Zeeb, E., Golatowski, F., and Timmermann, D. Implementing Powerful Web Services for Highly Resource-Constrained Devices. In *PERCOM Workshops* (Seattle, WA, USA, 2011).

[11] Nottingham, M. Web Linking. RFC 5988, 2010.

[12] Nottingham, M., and Hammer-Lahav, E. Defining Well-Known Uniform Resource Identifiers (URIs). RFC 5785, 2010.

[13] Rahman, A., and Dijk, E. Group Communication for CoAP. I-D: draft-ietf-core-groupcomm-09, 2013.

[14] Rescorla, E. RTCWEB Security Architecture. I-D: draft-ietf-rtcweb-security-arch-06, 2013.

[15] Shelby, Z. Constrained RESTful Environments (CoRE) Link Format. RFC 6690, 2012.

[16] Shelby, Z., Hartke, K., and Bormann, C. Constrained Application Protocol (CoAP). I-D: draft-ietf-core-coap-17, 2013.

[17] Shelby, Z., Krco, S., and Borman, C. CoRE Resource Directory. I-D: draft-ietf-core-resource-directory-00, 2013.

[18] Silverajan, B., and Savolainen, T. CoAP Communication with Alternative Transports. I-D: draft-silverajan-core-coap-alternative-transports-01, 2013.

[19] Tolle, G. Embedded Binary HTTP (EBHTTP). I-D: draft-tolle-core-ebhttp-00, 2010.

[20] Wilde, E. Putting Things to REST. Tech. Rep. 2007-015, School of Information, UC Berkeley, Berkeley, CA, USA, 2007.