Connecting the Web with the Web of Things: Lessons Learned From Implementing a CoAP-HTTP Proxy

Christian Lerche, Nico Laum, Frank Golatowski, Dirk Timmermann
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18119 Rostock, Germany
{firstname.lastname}@uni-rostock.de

Christoph Niedermeier
Siemens AG
Corporate Research and Technologies
christoph.niedermeier@siemens.com

***Abstract*—The Constrained Application Protocol (CoAP) is a lightweight application layer protocol for the Internet of Things. CoAP is based on HTTP mechanisms to build RESTful web services. In contrast to HTTP, CoAP was designed for machine-to-machine (M2M) communication and uses a binary representation. This allows efficient transport and processing in resource-constrained networks such as Wireless Sensor Networks (WSN). Due to the analogy to REST, a mapping between CoAP and HTTP is possible.**

In this paper one of the first translating CoAP-HTTP proxies is presented, that provides HTTP clients transparent access to CoAP resources and vice versa. Furthermore, caching relieves CoAP servers, which solves a key challenge for the Internet of Things: to allow a constant/permanent availability of resources from a network of constrained devices, which are required to minimize data transmissions due to their noticeable restrictions in power consumption. We describe, which issues of the translation we found during the time of implementation and testing, and explain how the proxy handles these issues. Finally, an evaluation, using real WSN hardware, is given, and an approximation scheme on how much transmissions can be saved by caching resources is provided.

***Keywords*-CoAP, HTTP, Proxy, Internet of Things, Constrained Networks, 6LoWPAN, Wireless Sensor Networks, Web of Things, M2M**

I. Introduction

One of the main enabler of the Internet of Things (IoT) vision is the *IPv6 over Low power Wireless Personal Area Networks* protocol [1], [2], referred to as 6LoWPAN. 6LoWPAN allows very small devices to become part of the Internet. It specifies an efficient transport of IPv6 network transmissions for low power wireless personal area networks. Initially, the protocol was designed to be used with the low power wireless standard IEEE 802.15.4. IEEE 802.15.4 defines the physical (PHY) and medium access layer (MAC) for wireless communication that is optimized for low power devices like *Wireless Sensor Networks* (WSN). But 6LoWPAN is not limited to IEEE 802.15.4, therefore we follow the notation of Shelby, Z. [3] and generalize these networks as *constrained networks*. Devices being part of these constrained networks, are referred to as *constrained devices*. The communication within constrained networks is characterized by a higher packet loss (5-10 %) and a lower throughput (around $10\,\text{kbit}\,\text{s}^{-1}$) than in IP Ethernet networks [3]. Furthermore, the network structure is more complex. Especially in WSNs, a mesh topology is used. Compared to Star- and Bus topologies, used in Ethernet networks, constrained networks are more complex in terms of packet routing (multi-hop), mobility/roaming and network management. Beyond this, the devices have higher resource-constraints in terms of RAM and ROM, computational power and energy. For example, sensor nodes usually have only a few kB of RAM and ROM, relatively weak MCUs with up to 100 MHz and are battery powered. In order to achieve a high battery lifetime, radio activity and therefore packet sizes must be reduced.

On top of the IP network layer, suitable application layer protocols are on demand. In the Internet, HTTP is one of the prevailing application layer protocols. But HTTP is not suitable for the use in constrained networks due to several reasons. Two of the main reasons are, firstly, the human-readable, but non-efficient, ASCII data representation. Secondly, the TCP binding of HTTP, that requires a three-way-handshake connection establishment to initiate connections and exchange data. To specify a lightweight HTTP alternative for constrained networks, the IETF founded the *Constrained RESTful Environments* (core) Working Group. This Working Group developed the *Constrained Application Protocol* (CoAP) which is currently in IETF draft status [4] and is close to become a proposed standard. CoAP aims to be a lightweight HTTP alternative for the use in constrained networks. (A more detailed description is given in this paper.)

One main advantage of CoAP is, that a general mapping to HTTP is considered by the CoAP specification. This is feasible because both protocols are based on REST principles. Of course an additional intermediate is necessary to translate between both protocols transparently. This intermediate is referred to as *translating proxy*. The proxy builds the connection between the Web and the *Web of Things* by allowing HTTP clients to request resources from CoAP servers and CoAP clients to request resources from

HTTP servers. In this paper we present a CoAP-HTTP proxy implementation. Mainly, the proxy implements the mapping between CoAP and HTTP (and vice versa). Furthermore, the proxy allows caching of resources to avoid unnecessary transmissions of equivalent resources.

In the following section (Section II) we give an overview about the use of web services technologies in constrained networks. Subsequently, we give a detailed introduction into the lightweight RESTful protocol CoAP. This is necessary to understand the proxy functionality. At the end of the section we discuss the role of the proxy in the Web of Things architecture. In Section III we give an overview about related work and current CoAP implementations to show that CoAP is a feasible application layer protocol for constrained networks such as wireless sensor networks. In Section IV and V we describe and evaluate the developed proxy. The paper ends with a conclusion in Section VI.

II. Web Services for the Web of Things

Originally, web protocols were designed for Human-Machine (H2M) interaction. With the evolution of the Web also Machine-Machine (M2M) applications became increasingly important. Therefore *web services* are used. Especially in the Web of Things, M2M applications dominate and the choice of the right web service technology is important. In the Internet two different types of web services are the prevailing technologies. These are *SOAP web services* and *RESTful web services*. Both are based on different principles. In the following, each web service technology is discussed briefly, with the focus on the use in constrained networks. Afterwards, CoAP is presented as a lightweight HTTP alternative to make use of RESTful web services in constrained networks. Finally, the role of the presented proxy in the Web of Things is described.

A. SOAP Web Services

SOAP web services were originally intended to invoke *Remote Procedure Calls (RPC)*. Today, they are often used to implement Service-oriented architectures (SOA). A SOA consists of several services. A service again, provides a set of operations. To describe a service the Web Services Description Language (WSDL) is used. As SOAP is only an XML based message format, further specifications are necessary around SOAP to provide functionality. These are e.g., WS-Addressing, WS-Policy, WS-Eventing, etc.[1]. Due to the huge number of specification, SOAP web services are seen as very complex. For the use of SOAP web services on devices, the *Devices Profile for Web Services* (DPWS) [5] was specified to define a minimal set of functionality for device communication, but the intended devices (e.g., Printer, Router, etc.) still have more resources than the constrained devices focused in this paper. For that reason, some of the authors of this paper investigated the use of DPWS in constrained networks [6]. It turned out, that DPWS can only be used under specific conditions and high implementation efforts are necessary.

[1]Commonly referred to as WS-* specifications

B. RESTful Web Services

RESTful web services have become more famous as they are less complex then SOAP web services [7]. The main component in a RESTful architecture is a *resource*. Compared to a service, a resource provides a fixed set of operations to *Create*, *Read*, *Update* and *Delete* (CRUD) a resource. A resource can be addressed by an URI. The Web itself has a RESTful architecture based on the HTTP protocol. HTTP covers the CRUD operations with the HTTP operations PUT, GET, POST and DELETE. For that, HTTP uses a Request-Response message pattern. As transport and network layer TCP/IP is used, which inhibits asynchronous and multicast messaging. To define the type and encoding of the resource (e.g., HTML document, XML document, Picture, etc.), HTTP uses MIME types. According to [3], the advantages of RESTful web services in contrast to SOAP web service are less parsing complexity, statelessness, and tighter integration with HTTP. But although RESTful web services are more simple in concept, HTTP and XML payloads were not designed to spare resources.

C. The Constrained Application Protocol

CoAP has been developed as a lightweight alternative for HTTP. The aim of CoAP according to the CoAP specification [4] is *"not to blindly compress HTTP [RFC2616], but rather to realize a subset of REST common with HTTP but optimized for M2M applications"*.

Because of the huge overhead of TCP, CoAP uses UDP as transport protocol. In contrast to TCP, UDP does not guarantee a successful transmission (reliability) and that packets are transmitted in the right order (ordering), hence CoAP defines an additional Message Layer. This layer defines four packet types, which are Confirmable (CON), Non-Confirmable (NON), Acknowledgement (ACK) and Reset (RST). Confirmable packets (CON), sent by the client, are acknowledged (ACK) by the server. In case a server cannot accept a new packet, the server resets the connection (RST). If the client does not receive any answer after a given time, it retransmits the request. As reliability causes unnecessary overhead in some cases, reliability is optional in CoAP. If reliability is not necessary, a client sends a NON packet instead of a CON and the server answers with a NON packet (only) if an answer is requested. As UDP does not allow any assumption about the ordering and duplication of transmitted packets, every packet has a mandatory 16 bit message ID to allow an assignment of requests and responses.

As the scope of CoAP are constrained networks, very small payloads are assumed. However, larger payloads are possible in some cases. Because UDP is not a streaming protocol like TCP and, furthermore, to avoid fragmentation on 6LoWPAN layer, CoAP defines blockwise transfers [8], where the payload is transferred in several blocks and each block is acknowledged separately.

The CoAP Request/Response layer on top of the Message Layer, provides the four HTTP operations PUT, GET, POST and DELETE. Similar to HTTP, the server responds with a status code. In contrast to HTTP, CoAP uses binary header representations, hence, the mandatory CoAP header requires only 4 Byte. Additionally to that, CoAP defines further optional header options (similar to HTTP), like e.g., the type of the content (Content-Type), how long a resource is valid (Max-Age), etc. A header option consists of a constant option number and a value. If the option number is even, the option elective. If the option number is odd, the option is critical. If a server or a client receives a CoAP message, containing an unknown critical option, the message must be rejected. A list of the main header options is given in [4].

To address CoAP endpoints, URIs are used. To reduce packet size as much as possible, the used URIs should be as short as possible. Every part of an URI (host, path, query) is included in a separate header option to reduce parsing complexity.

Because CoAP is intended for the use in M2M applications, CoAP provides some additional features, that are not provided by HTTP. Firstly, CoAP allows unreliable multicast messaging. Secondly, it allows asynchronous messaging to provide an observer message pattern [9]. Thirdly, every CoAP server provides a list of all hosted resources, at a well-known address. This allows discovery of resources. The format of the list is defined by the CoRE Link Format [10].

D. The Proxy in the Web of Things

In Figure 1 the Web of Things architecture is shown as referred to in this paper.

Due to a common IP network layer, CoAP servers can be accessed by CoAP clients directly, no matter if they are located inside the constrained network (Fig. 1, a)) or outside in the Internet (Fig. 1, b)). The same is true for HTTP (Fig. 1, c)). To allow HTTP clients to access CoAP endpoints and to allow CoAP endpoints to access HTTP servers, a translating proxy is necessary (Fig. 1, d) and e)). In Figure 2 typical protocol stacks for normal and constrained networks are shown. A router[2] connects IPv6 networks with 6LoWPAN networks. The proxy translates on application layer and can therefore be located anywhere in the Internet. This is in contrast to common WSN *gateways* (Fig. 1, f)). As proprietary networks mostly have their own network layer, the gateway connects the proprietary network on all layers and must therefore be located at the edge of the WSN. As stated in [7]: *"The gateway must be tailored to the specific protocols used inside the sensor network."* In contrast to that, the proxy is independent from the application data and the application logic and, consequently, does not need to be changed when the application changes.

[2]Routers that connect an IPv6 network with a 6LoWPAN network are called 6LoWPAN Edge Router or 6LoWPAN Border Router

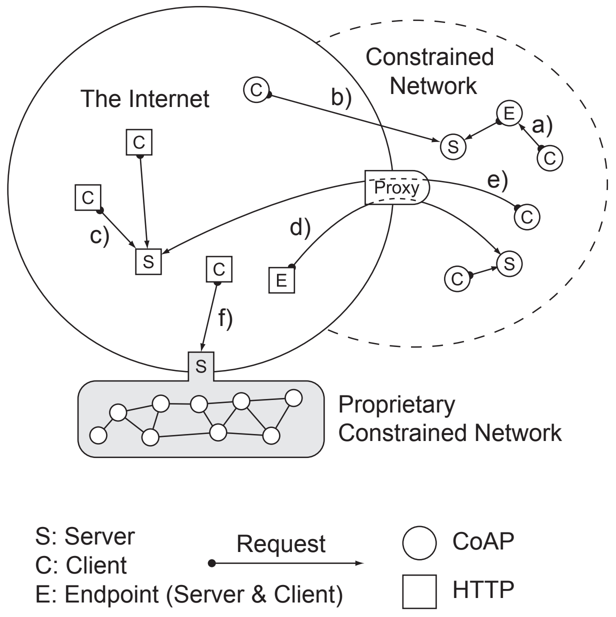

Figure 1. Web of Things Architecture

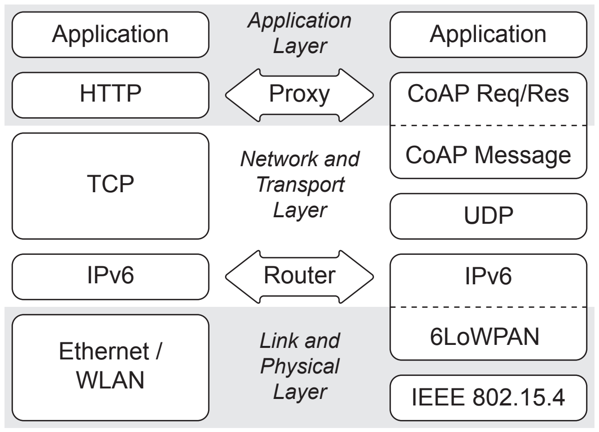

Figure 2. Communication Stacks for RESTful Web Services in Normal and Constrained Networks

Although the proxy should act transparently as possible, it is not feasible to design a fully transparent proxy. At least one partner (server or client) must have knowledge of the existence of the proxy[3]. Consequently, there are two different types of proxies:

[3]For the sake of completeness: There are Interception Proxies which do not require neither the server nor the client to be configured, but Interception Proxies require a specific network configuration and are therefore not considered.

a) Forward Proxy: Instead of making a request directly to the server, the client sends the request to the proxy. The proxy then forwards the request to the server. For the server, the proxy behaves as if it would be a client. Consequently, the client needs to be configured to make use of the proxy, no configuration is necessary for the server (Zero Server Configuration (ZSC)).

b) Reverse Proxy: A reverse proxy is not known by the client. The client expects a normal server at the location of the proxy. The proxy then forwards the request to the hidden server. Therefore the server must be associated with the proxy. Consequently the client does not need to know that the proxy exits and no configuration on client side is necessary (Zero Client Configuration (ZCC)).

Mostly forward proxies are used in the Internet and unless otherwise stated in this paper, the term proxy refers to a forward proxy. The presented proxy is also a forward proxy, which means that the client needs to be configured, the server not.

According to [11] translating proxies can be further classified as follows:

a) Protocol-aware access: The client knows about the cross-protocol translation and specifies (e.g., by the URI scheme) if a translation should be performed.

b) Protocol-agnostic access: The client does not know anything about the cross-protocol translation. A mapping is silently done by the proxy.

The presented proxy implements a CoAP to HTTP protocol-aware mapping, but a HTTP to CoAP protocol-agnostic mapping. This has several reasons which are described in more detail in Section IV.

Furthermore, as known from common HTTP proxies, the proxy can cache resources and, consequently, reduce network traffic inside the constrained network. This is also useful for CoAP to CoAP communication inside the constrained network, especially, when many clients request resources from a single server. A more detailed discussion is given in the following Sections IV and V.

To sum up, the proxy has an important role in the architecture of the Web of Things, since the proxy allows a direct web service based communication between Internet end-points and constrained devices.

III. Related Work & Implementations

Because CoAP is a very new protocol, which is not yet a standard, only a few implementations and evaluations are published. According to what is known, the only published CoAP-HTTP proxy implementation is given in [12]. In the paper a one-way HTTP to CoAP proxy implementation is presented, which allows HTTP clients to request resources from a CoAP endpoint. The implementation is a server-side proxy located at the edge of a WSN. Caching was not considered. As proof of concept a web browser requesting a CoAP resource is presented, further evaluations are not given.

The following publications give an overview about current CoAP implementations and their use in constrained networks, to show that CoAP is a promising web service technology for constrained networks. This is important, as the proxy only adds a value, if CoAP is a feasible web service technology for constrained networks.

An early stage implementation of CoAP (draft version 3), called *libcoap*, for the use on resource-constrained devices is described in [13]. Libcoap is available for the operating systems TinyOS and Contiki. The evaluation shows, that a complete request can be performed in between 70-400 ms (single hop) while the number of transmitted bytes ranges from 120 B to 230 B. The application scenario is a WSN in a cargo transport container. The authors conclude that CoAP is a feasible web service protocol for the use in constrained networks.

In [14] a newer (draft version 7) implementation of CoAP for Contiki is presented. In contrast to [13], the authors evaluated the use in multi-hop WSNs with a duty cycling MAC layer[4]. For the evaluation the same hardware was used. The response time strongly depends on the number of hops and the duty cycling parameters.

For the Human-to-Machine interaction (H2M) with resource-constrained CoAP endpoints, the Firefox CoAP Plugin *Copper* is presented in [14]. It allows users to perform a CoAP request. (Note: Although Firefox is a HTTP client, Copper implements pure CoAP, no HTTP translation/mapping is included).

In summary, it can be stated, that although CoAP is a very new web service protocol, first implementations and evaluations have shown that CoAP can be implemented on constrained devices and the overhead of the CoAP headers is small enough to be transported in constrained networks.

IV. CoAP / HTTP Proxy Implementation

In this chapter the proxy implementation is described. The proxy acts as server and as client. The developed translating proxy therefore has two client modules and two server modules as shown in Figure 3. This implies four different message flows. These are CoAP to HTTP (1), CoAP to CoAP (2), CoAP to HTTP (3) and finally HTTP to HTTP (not shown). The first three cases are covered by the implemented proxy. The forth case equals the behavior of a common HTTP proxy and is therefore not implemented.

The implemented proxy is part of the jCoAP[5] library. It is implemented in Java and runs on any system that runs a

[4]Duty cycling means, that the radio is turned off most of the time. The radio is only turned on in fixed intervals to check if some data is available or when there is some data to be send. The used MAC implementation in this paper was ContikiMAC

[5]http://www.ws4d.org/ws4d-jcoap

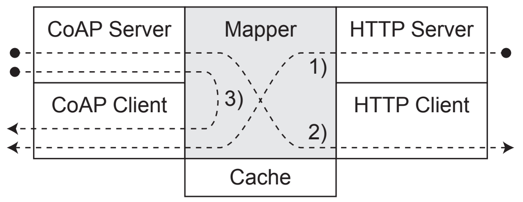

Figure 3. Modularised Structure of the Proxy

Java VM. The core part of the proxy is the mapper module. The mapper module performs the described translations.

In the following sections, several aspects of the proxy implementation are described in more detail.

A. *How to determine if a mapping should be performed.*

On an incoming request, regardless of the protocol, the proxy needs to decide if a mapping needs to be performed or not. As the CoAP protocol was designed with the mapping in mind, CoAP defines the Proxy-Uri header option. If present, this option indicates that the client is aware of an intermediate proxy. To indicate that a proxy should perform a mapping, the Proxy-Uri must have an "http" or "https" scheme. As secure connections are currently not supported by the proxy, the Proxy-Uri scheme must be either "coap" (no mapping) or "http" (a mapping is performed).

The HTTP specifications does not consider any mapping and has therefore no intended mechanisms. According to [4], a "coap" or "coaps" scheme of a request URI indicates that a mapping should be performed. However, HTTP does not require an absolute Request-Uri, and many clients (like Firefox that was used for testing) omit the protocol scheme and the URI host. If no scheme is giving, the proxy can not determine if mapping to CoAP should be performed or if the client wishes the proxy to behave as a non-mapping HTTP proxy. However, as the proxy does not implement the HTTP to HTTP message flow, a mapping is performed in any case, even if no protocol scheme is given in the Request URI or the scheme is "http". This means that the use of the proxy implies a mapping wish.

B. *Header Options*

The CoAP core specification defines 14 different header options. The Uri-Host, Uri-Path and Uri-Query options are mapped to the HTTP request URI and vice versa. The Proxy-Uri option is handled by the proxy itself as described in the previous section. The Token option[6] is only handled when the proxy acts as CoAP server (CoAP to HTTP mapping). In case of a Token option, the token is saved inside the proxy and added to the final response. How the Max-Age option is handled is described in Section IV-E about caching. All remaining CoAP header options are directly translated to the corresponding HTTP header options and vice versa. These are: Content-Type, ETag, If-Match, If-None-Match, Accept, Location. The HTTP Content-Length option is automatically generated. The size is determined by using the UDP packet length.

[6]The Token option can be used by clients to multiplex parallel requests. A server only needs to copy the token from the request to the response.

C. *Message Layer*

As previously described, CoAP provides either reliable or unreliable messaging. In case of an incoming CoAP request (CoAP to HTTP mapping), the CoAP client determines if reliable or unreliable messaging is used. In case of a HTTP request (HTTP to CoAP mapping), the proxy acts as client and therefore predetermines if reliable or unreliable messaging is used. By default, the proxy uses reliable messaging, however, the proxy can be easily configured to use unreliable messaging.

D. *Operations*

HTTP defines more options than CoAP. PUT, GET, POST and DELETE are known by both protocols and can therefore be translated directly. The HTTP HEAD operation allows a HTTP client to only request the HTTP header. When translated to CoAP, the CoAP GET operation is used. After receiving the response, the payload of the response is removed during translation. All other operations (TRACE, OPTIONS, CONNECT, PATCH) cannot be mapped and a 501 Error ("Not Implemented") is returned by the proxy.

E. *Caching*

Caching is a core feature of HTTP and CoAP to reduce network traffic by avoiding the transmission of equivalent resources. During the implementation of the proxy, it turned out, that only the Max-Age option of CoAP does not allow to determine exactly how long a resource is valid due to the transmission time.

Because CoAP does not provide a Date header option like HTTP, it is not possible to determine exactly when the resource was created. Based on the assumption, that the creation time is at some point between the time when the request was sent and the respond is received, three strategies are conceivable of how to determine how long a resource is valid. In Figure 4 all three cases are shown. In case a) the creation time is assumed to be at the time when the request is sent by the client. This leads to a False Positive (resource is withdrawn although it is still valid). In case b) the creation time is assumed to be at the time when the response is received. This leads to a False Negative (resource is assumed as valid but already expired). In case c) a synchronous transmission is assumed (the request transmission time equals the response transmission time) and the creation time is approximated by the half of the round trip time. This can reduce the error but the error classification can not be determined. The proxy implements

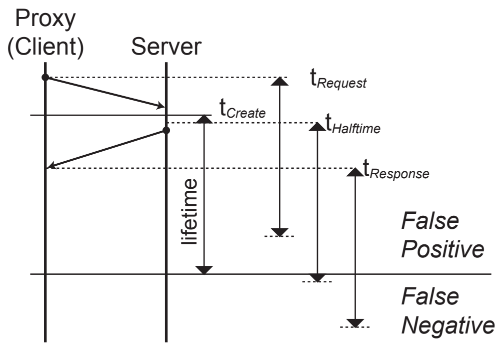

Figure 4. Caching Faults

the third case by default, however, the proxy can be easily changed to use any other strategy.

The described error is also relevant for normal CoAP clients but in case of a proxy the error occurs at least twice (server to proxy and proxy to client). CoAP does not define a Date option like HTTP with respect to constrained-devices that often does not have a global clock. However, a Date option could be useful in case of CoAP devices that have a global clock. Therefore this paper proposes a CoAP Date header option as follows: The option number should be elective, that means, the option number could be any not used even number. The value should be an 4 B unsigned integer containing the UNIX Date time[7].

F. IPv4 and IPv6

Because TCP (transport layer of HTTP) and UDP (transport layer of CoAP) can be used with both IPv4 and IPv6, the proxy can also be used for IPv4-IPv6 protocol translation. Currently IPv4 is the prevailing protocol version in the Internet, whereas constrained-devices with 6LoWPAN require IPv6 based communication. The intermediate proxy solves this issue. The proxy accepts both protocol versions on both interfaces.

V. Evaluation & Results

To evaluate the correct behavior of the proxy, mapping and caching functionalities were tested. In Figure 5 the experiment setup is shown. As hardware reference platform Crossbows TelosB is used. The TelosB is a widely known wireless sensor development board. It is based on an MSP430 microcontroller and equipped with an IEEE 802.15.4 radio chip, light and temperature sensors, and an USB interface for programming and debugging. The MSP430 operates at 8 MHz and has 10 kB of internal RAM and 48 kB Flash memory.

[7]seconds since 00:00:00 UTC on 1 January 1970

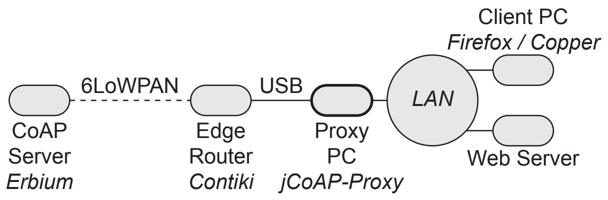

Figure 5. Experiment Setup

The CoAP server implementation for TelosB is based on Erbium[8]. Erbium is a CoAP implementation for Contiki that implements the 6LoWPAN and IPv6 communication layer.

The Firefox plugin Copper is used as CoAP client. A common Firefox 11.0 browser (without Copper) serves as HTTP client. The HTTP server is a common Apache web server.

All components, except the wireless sensor, are located within the same Ethernet LAN. The wireless sensor is connected to the LAN via a 6LoWPAN edge router.

A. Mappings

All message flows described in Section IV were tested. In the following each test configuration is described.

1) HTTP to CoAP: In the first case, a HTTP client requests a CoAP resource on a CoAP resource server. Therefore the Firefox proxy settings must be changed. (remember that the implemented proxy is a forward proxy and therefore the client is aware of the intermediate). The proxy address of Firefox is set to the address of the proxy. As Firefox does not know anything about CoAP, the Firefox browser assumes a common HTTP proxy (protocol-agnostic access). In Firefox then a GET request to the CoAP Erbium server on the TelosB can be sent directly to the address of the CoAP server (`http://[constrained device address]:[port]/[resource path]`).

2) CoAP to HTTP: In the second case, a CoAP request is sent to a HTTP server. Copper is used as CoAP client. To enable proxying, the Proxy-Uri option must contain the request URI (`http://[web server address]:[port]/[resource path]`). Then the request is sent to the proxy directly.

3) CoAP to CoAP: In the third case, a CoAP resource is requested by a CoAP client. Again, the Copper client is used but with a different Proxy-Uri, containing a coap scheme and the address of the constrained device (`coap://[constrained device address]:[port]/[resource path]`. No mapping is performed in this case, the proxy acts as a normal proxy.

B. Caching

Caching can reduce network traffic by avoiding unnecessary transmissions of equivalent resources. Especially in

[8]http://people.inf.ethz.ch/mkovatsc/erbium.php

Table I
CACHING RATE FOR A DIFFERENT NUMBER OF CLIENTS (x) WITH A FIXED CACHING PERIOD OF $t_C = 20s$ AND A FIXED CLIENT REQUEST INTERVAL $t_R = 10s$

	$r_{\text{theor.}}$	r_{real}
$x = 1$	50 %	66.0 %
$x = 2$	75 %	79.1 %
$x = 5$	90 %	90.4 %
$x = 10$	95 %	94.8 %

the case when a lot of clients request a single resource frequently.

For evaluation purposes, the following generic scenario is assumed: x clients request a resource frequently with an interval $t_r = 1/f_r$ through the proxy, the resource is cached for the time t_c by the proxy. This means that in the worst case the proxy makes a request to the constrained device with a frequency of $f_c = 1/t_c$. In a given time t the proxy would make $t \cdot f_c$ requests to the constrained device. The number of requests to the proxy is given by $x \cdot f_r \cdot t$. Consequently the number of requests that are served from the cache can be approximated by the difference of these two values, which is $x \cdot f_r \cdot t - t \cdot f_c$ The amount of requests that are served from the cache in contrary to the theoretical number of requests without a proxy is referred to as caching rate r and defined as follows:

$$r = \frac{\text{number of requests served from cache}}{\text{number of total requests}} \qquad (1)$$

Hence, the theoretical caching rate r is (Note that t can be canceled):

$$r = \frac{x \cdot f_r - f_c}{x \cdot f_r} \qquad (2)$$

The caching rate r is an indicator of how many requests to the server could be avoided. A caching rate of 0 % means no caching. A caching rate of 100 % means all requests are served from the cache (not possible, at least one request is necessary). For a fixed time, the caching rate increases when, firstly, the number of clients x increases, secondly, the caching time t_c increases and thirdly, the frequency of requests f_r increases.

In Table I the results of an experiment are shown, that proofs the quality of the presented caching rate approximation. In the first three cases the caching rate is slightly better. The difference can be explained among others with the effect described in Section IV-E. As described, the proxy caches resources too long by default (False negative), therefore the caching rate can become better than theoretically possible. Furthermore, the approximation does not cover the case, that two requests from two different clients occur shortly one after the other. The second request will also be forwarded (wrongly) to the constrained device because the response from the first request was not yet received by the proxy and is therefore not in the cache. This case leads to a worse caching rate like in the forth case.

As shown in Table I the approximated caching rate allows a quantitative statement about how much the proxy can reduce the number of requests within the constrained network.

VI. CONCLUSION AND OUTLOOK

We have presented one of the first public available translating CoAP-HTTP proxies, that provides HTTP clients transparent access to CoAP resources and vice versa. It can be seen as a proof of concept of the CoAP-HTTP mapping defined by the CoAP protocol specification. The proxy allows a high flexibility in creating Internet of Things applications, due to its independence from the application logic in contrast to common application gateways.

First tests have shown, that the implementation works according to the specification and is compatible with other existing CoAP implementations and common HTTP servers and clients. As the proxy is open source licensed, it is likely that more evaluations and enhancements will be made by the IoT community in the near future.

For future work, a great potential in the area of security can be seen. The additional intermediate has the advantage that basic security aspects can be taken over by the proxy, that have usually more resources than the constrained devices. If the proxy is located at the edge of the constrained network (e.g., the constrained network is protected by a firewall), the proxy could detect traffic overloads and avoid them. Furthermore the proxy could provide a HTTPS (HTTP + TLS) interface, so that the communication outside of the constrained network (e.g., the Internet) is encrypted. In addition, the proxy could allow only authenticated clients access to the constrained network.

ACKNOWLEDGMENT

The authors would like to thank the Corporate Research and Technologies Division of Siemens AG, Germany for their inspiration and continued support.

REFERENCES

[1] N. Kushalnagar, G. Montenegro, and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals," RFC 4919 (Informational), Internet Engineering Task Force, Aug. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4919.txt

[2] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944 (Proposed Standard), Internet Engineering Task Force, Sep. 2007, updated by RFC 6282. [Online]. Available: http://www.ietf.org/rfc/rfc4944.txt

[3] Z. Shelby, "Embedded web services," *Wireless Communications, IEEE*, vol. 17, no. 6, pp. 52 –57, December 2010.

[4] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "Constrained Application Protocol (CoAP)," IETF, Tech. Rep., March 2012, draft. [Online]. Available: http://www.ietf.org/id/draft-ietf-core-coap-09.txt

[5] D. Driscoll and A. Mensch, "Devices Profile for Web Services Version 1.1," OASIS, Tech. Rep., July 2009. [Online]. Available: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf

[6] C. Lerche, N. Laum, G. Moritz, E. Zeeb, F. Golatowski, and D. Timmermann, "Implementing powerful Web Services for highly resource-constrained devices," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*, March 2011, pp. 332 –335.

[7] D. Yazar and A. Dunkels, "Efficient application integration in IP-based sensor networks," in *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, ser. BuildSys '09. New York, NY, USA: ACM, 2009, pp. 43–48. [Online]. Available: http://doi.acm.org/10.1145/1810279.1810289

[8] C. Bormann and Z. Shelby, "Blockwise transfers in CoAP," IETF, Tech. Rep., February 2012, draft. [Online]. Available: http://www.ietf.org/id/draft-ietf-core-block-08.txt

[9] K. Hartke, "Observing Resources in CoAP," IETF, Tech. Rep., March 2012, draft. [Online]. Available: http://www.ietf.org/id/draft-ietf-core-observe-05.txt

[10] Z. Shelby, "CoRE Link Format," IETF, Tech. Rep., January 2012, draft. [Online]. Available: http://www.ietf.org/id/draft-ietf-core-link-format-11.txt

[11] A. Castellani, S. Loreto, A. Rahman, T. Fossati, and E. Dijk, "Best practices for HTTP-CoAP mapping implementation," IETF, Tech. Rep., March 2012, draft. [Online]. Available: http://www.ietf.org/id/draft-castellani-core-http-mapping-03.txt

[12] W. Colitti, K. Steenhaut, N. De Caro, B. Buta, and V. Dobrota, "REST Enabled Wireless Sensor Networks for Seamless Integration with Web Applications," in *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, October 2011, pp. 867 –872.

[13] K. Kuladinithi, O. Bergmann, T. Poetsch, M. Becker, and C. Goerg, "Implementation of CoAP and its Application in Transport Logistics," in *"Extending the Internet to Low power and Lossy Networks (IP+SN 2011)"*, Chicago, USA, 2011.

[14] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A Low-power CoAP for Contiki," in *Proceedings of the IEEE Workshop on Internet of Things Technology and Architectures*, Valencia, Spain, October 2011. [Online]. Available: http://www.sics.se/~adam/kovatsch11low-power.pdf