Implementation of CoAP and its Application in Transport Logistics

Koojana Kuladinithi
Communication Networks, TZI
University Bremen
Bremen, Germany
koo@comnets.uni-bremen.de

Olaf Bergmann
Computer Networks, TZI
University Bremen
Bremen, Germany
bergmann@tzi.org

Thomas Pötsch
Communication Networks, TZI
University Bremen
Bremen, Germany
tpoetsch@uni-bremen.de

Markus Becker
Communication Networks, TZI
University Bremen
Bremen, Germany
mab@comnets.uni-bremen.de

Carmelita Görg
Communication Networks, TZI
University Bremen
Bremen, Germany
cg@comnets.uni-bremen.de

ABSTRACT

The use of web services for sensor networking applications is seen as an important part in emerging M2M communications. The Constrained Application Protocol (CoAP) is proposed by the IETF to optimize the use of the RESTful web service architecture in constrained nodes and networks, for example Wireless Sensor Networks.

We present an IP based solution to integrate sensor networks used in a cargo container with existing logistic processes, highlighting the use of the CoAP protocol for the retrieval of sensor data during land or sea transportation. This solution is implemented and evaluated on TinyOS and Contiki, two widely researched embedded system architectures. Our preliminary results show that the performance of CoAP compared to HTTP based resource retrievals performs better in constrained environments such as a cargo container containing several embedded devices.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network architecture and Design—*Network communications*

General Terms

Algorithms, Performance, Standardization, Measurements

Keywords

Constrained Application Protocol, Wireless Sensor Network, Contiki, TinyOS

1. INTRODUCTION

Most Internet applications today use web services, which depend on the basic Representational State Transfer (REST) architecture [1]. The REST architecture allows easy integration with web browsers and web-based service providers.

The main objective of the Constrained RESTful Environments (CoRE) working group at the IETF is to see how the REST architecture can be employed in constrained networks. The CoRE working group has already proposed a standard called Constrained Application Protocol (CoAP) which can easily be translated to HTTP to foster integration of constrained networks with the web.

This paper discusses the use of CoAP for machine to machine (M2M) communication in logistic applications for supervision of the environmental conditions during transport. Most of the devices in the WSN (Wireless Sensor Network) deployed in a cargo container or truck trailer have low bandwidth, scarce memory capacity and limited processing capability. Low power wireless networks mostly consist of nodes that conform to the IEEE 802.15.4 [2] standard proposed by the IEEE. The maximum packet size for IEEE 802.15.4 is only 127 bytes. Therefore, the currently available IP based application protocols (HTTP, FTP, SOAP, etc.) will not be suitable for 802.15.4 based networks. On the other hand, the application protocols that work on the WSN should also be able to integrate with IP based networks which are used for the communications between a container and back-end databases, as the bandwidth on that link is constrained as well.

Unlike HTTP based protocols, CoAP operates over UDP and employs a simple retransmission mechanism instead of using complex congestion control as used in standard TCP. It uses a unique Transaction ID to identify each GET request for retransmission purposes to keep reliability.

The next section details the use of CoAP in supervising a cargo container, which is termed *Intelligent Container*. An

Intelligent Container transmits information such as humidity, temperature of meat, fruits etc. inside a container during land or sea transportation. This information is useful to take logistical process planning decisions to deliver food to markets efficiently and cost effectively. Section 3 discusses implementation details of CoAP, which has been adapted to the Contiki and TinyOS operating systems. Section 4 discusses initial tests that are carried out to compare the performance of HTTP based resource retrievals with CoAP. The last section gives the conclusion.

2. COAP USAGE IN M2M ENABLED LOGISTICS

This section details about deploying CoAP in different components of a transport facility, such as a container. The *Intelligent Container* architecture consists of a WSN, a freight supervision unit (FSU) and telematic devices (see figure 1). The FSU is the main computing device which is the border router for the WSN and manages the different kinds of communication with the back-end software. CoAP is mainly used in the WSN part of the *Intelligent Container* to manipulate resources (e.g. temperature, humidity) by using the following methods:

- The GET method is used to retrieve resources from WSN nodes or the FSU. The resource is identified by the requested Uniform Resource Identifier (URI).
- The PUT method is used to modify an existing resource on a sensor node or the FSU.

As mentioned before, CoAP uses the datagram-oriented UDP transport protocol to exchange messages. Different message types are used to establish a message exchange between client and server:

- Confirmable (CON) messages always carry a request or response and require an Acknowledgment.
- Non-Confirmable (NON) messages are used for regularly repeated messages and do not require an Acknowledgment (e.g. subscriptions of reading a sensor).
- Acknowledgment (ACK) messages acknowledge CON messages and must carry a response or be empty.
- Reset (RST) messages are sent in case a CON message is not received properly or some context is missing.

Independent of the message type, a message may carry a request, a response, or be empty. The type of the message is signaled by the type field in the CoAP header together with a response code to indicate the result of an request.

CoAP client and server can run on the FSU as well as the sensor nodes. Table 1 and Table 2 show the resources that can be provided on the sensor nodes and the FSU. For example, for a CoAP CON+GET request issued by the FSU to retrieve humidity from a sensor node, the CoAP server that runs on the sensor node sends the CoAP response with the humidity piggy-backed on an ACK message.

Resource	GET	PUT	Comments
/st	X		Temperature
/sh	X		Humidity
/sv	X		Voltage
/r	X		Temperature, humidity and voltage together
/l	X	X	LEDs
/ck	(X)	X	AES Encryption Key

Table 1: CoAP Resources on Sensor Nodes

Resource	GET	PUT	Comments
/ni	X		Inform about node integration into 6LoWPAN/RPL network
/warntemplow		X	Below Warning Temperature Low
/warntemphi		X	Above Warning Temperature High

Table 2: CoAP Resources on the FSU

There are three possible options that the FSU uses to send data retrieved from the WSN to the back-end software.

- Option 1: The communication is a proprietary protocol as currently used in M2M telematic systems.
- Option 2: The communication between the FSU and the back-end software can also use CoAP. This helps in transmitting a low number of bytes especially via telematic devices with cellular/satellite connections using an open protocol.
- Option 3: The communication between the Decision Support Tool (DST) and the back-end software can use a standard IP protocol such as HTTP. In this case, the FSU can be used as a proxy between CoAP and HTTP based networks.

3. LIBCOAP

Our implementation of CoAP is termed `libcoap`, an open-source C-library that is specifically targeted at embedded systems with constrained resources.[1] While our tests were done, `libcoap` provided support for the current working group drafts of CoAP [3, 4] as well as its optional extensions for block-wise transfer [5], resource observation [6], and additional CoAP options defined in [7].

The library provides functions and data structures for parsing and in-place editing of CoAP protocol data units (PDUs) to minimize memory overhead in embedded systems. An additional application server and a multi-purpose command-line client built upon this library demonstrate the use of the API in stand-alone CoAP-enabled applications. Both have participated in several official plug-fests of the IETF CoRE working group with great success in terms of feature-completeness and interoperability.

The code has been ported to various embedded system architectures. This includes in particular the operating systems Contiki and TinyOS as described in the following sections.

[1]See `http://libcoap.sourceforge.net`.

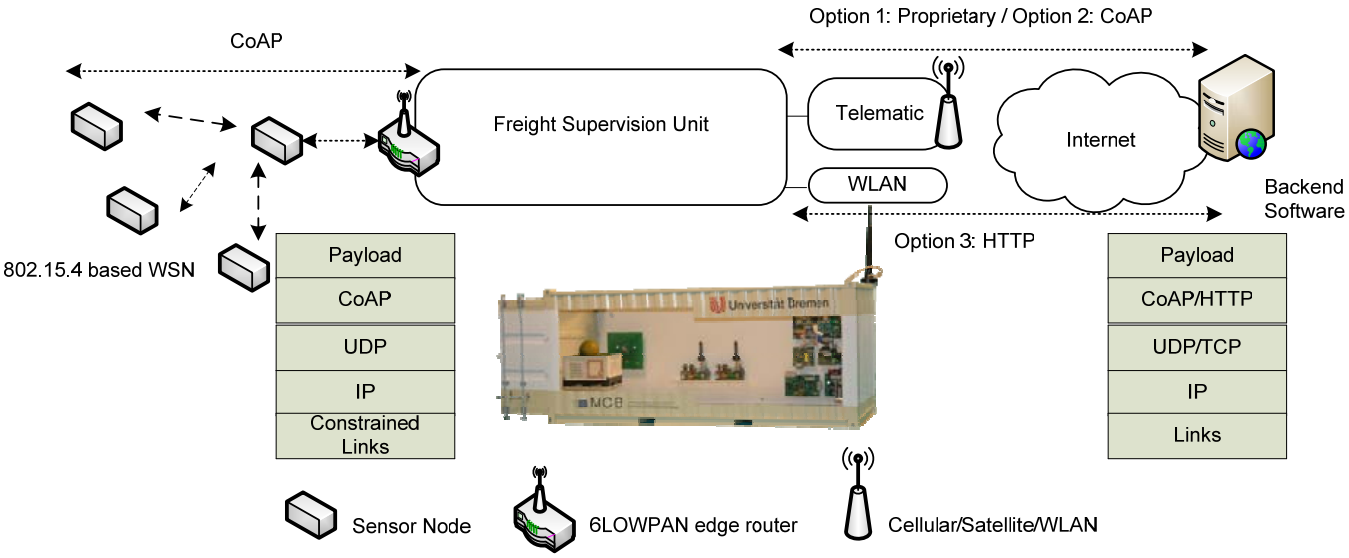

Figure 1: Deployment of CoAP in the Intelligent Container

3.1 Porting libcoap to Contiki

One specific goal of the `libcoap` development was to bring CoAP-support to Sensinode `N740` sensor nodes [8]. The nodes are equipped with various sensors and a TI `CC2431` system on a chip with integrated radio module [9]. As the Contiki operating system [10] in version 2.4 already provided basic support for the `N740` hardware platform (though lacking support for few sensors), it was decided to port `libcoap` to use the intrinsic uIP TCP/IP stack of Contiki [11].

As the N740 platform provides at most 128 KB flash, both, Contiki and `libcoap` have been stripped down to fit into three memory banks of 32 KB each, leaving an entire memory bank for the CoAP-enabled application. To achieve this, most functions have been modified to support the *banked* invocation model. Moreover, TCP-specific parts of uIP have been removed entirely.

The runtime behavior of `libcoap` has been optimized to better fit the limited stack size of only 256 Bytes in the default memory layout. To do so, a number of global variables were introduced to be shared by Contiki protothreads [12].

With this port of `libcoap` to the Contiki operating system, an easy-to-use platform for building CoAP-enabled applications has been demonstrated. The Contiki port of the basic CoAP code takes about 12 KB ROM on a Sensinode N740 while the `rest-coap` application that comes with Contiki version 2.5 adds about 26 KB[2]

3.2 Porting libcoap to TinyOS

For using CoAP on TinyOS nodes, server and client components have been created to demonstrate the usage of `libcoap`. Currently, the server components cover the GET and PUT method for selected resources, while POST and DELETE support is not implemented. Due to the component based architecture in TinyOS, changes in wiring of resources during runtime is not possible and does therefore not allow creation or removal of resources. Since our CoAP application does not have the requirement to transfer payload higher than the maximum of IEEE 802.15.4 messages, the support for block-wise transfer [5] is omitted for TinyOS. To provide IPv6 for CoAP on TinyOS nodes, the (Berkeley Low-Power IP Implementation (blip-1.0 [13]) for TinyOS is used to transmit and receive message over UDP. Installation instructions for CoAP on TinyOS are available at [15].

In order to adapt `libcoap` to TinyOS, the following unavailable parts in TinyOS, which are used in the `libcoap` C code, need to be excluded by `#ifdef IDENT_APPNAME` and are replaced by TinyOS components:

- `socket.h` replaced by TinyOS blip-1.0 UDPSocketC.
- registration of resources with `libcoap` is provided by an interface.
- `time.h` and `string.h` dependencies of `libcoap` were removed (CoAP subscription [6] support would need to be mapped to TinyOS Timers).

The CoAP message exchange model of immediate and deferred replies has been mapped to TinyOS `calls` and `signals`. The server and client applications are connected to their corresponding blip UDPSocketC instances (UDPServer, UDPClient) by using the LibCoapAdapter component, which has been created to provide an independent wiring of both components (figure 2), so that it would also be possible to run several CoAP end-points on different UDP ports. The LibCoapAdapter interface is given below:

[2]This estimation is based on the `rest-server-example` with debug code being disabled and resource-specific handlers removed.

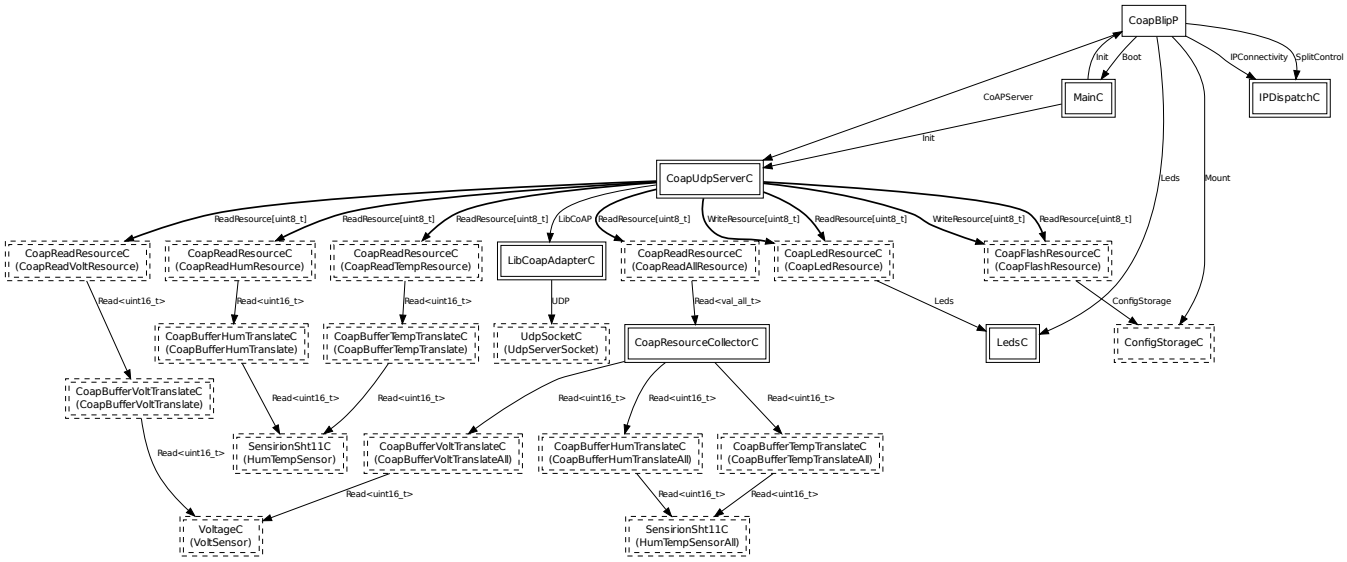

Figure 2: CoAP Server and Client implementation

```
interface LibCoAP {
    command error_t bind(uint16_t port);
    command coap_tid_t send(coap_context_t *ctx,
                            struct sockaddr_in6 *dst,
                            coap_pdu_t *pdu,
                            int free_pdu);
    event void read(struct sockaddr_in6 *from,
                    void *data,
                    uint16_t len,
                    struct ip_metadata *meta);
}
```

In figure 2 the wiring of the CoAP server and client components is depicted. Both components are wired to the above mentioned LibCoapAdapter to connect to the UDP socket for sending and receiving messages. CoAP uses Uniform Resource Identifiers (URIs) to identify resources located on a server and the resources have to be registered in advance. Resources may for example be temperature and humidity sensors, LEDs, or flash memory. In sensor nodes, the registration of the resources has to be done at startup time by allocating memory and setting a function pointer to invoke the corresponding interface at incoming requests.

3.2.1 Resource Access

The server component in addition is wired over the ReadResource interface to the generic components of the respective read only resources on the TelosB mote, i.e. Humidity/Temperature sensor. As an example, the ReadResource interface is given:

```
interface ReadResource {
    command error_t get(coap_tid_t id);
    event void getDone(error_t result,
                       coap_tid_t id,
                       uint8_t asyn_message,
                       uint8_t* val,
                       uint8_t buflen);
    event void getPreAck(coap_tid_t id);
}
```

To read values of the sensors, the parameterized ReadResource interface has been introduced to provide multiple independent instances of the same interface for all available resources. This design saves code space, eliminates fan-outs and allows portability to different nodes by simply changing the wiring. The ReadResource interface currently provides the `get()` command and the two events `getDone()` and `getPreAck()`, which are discussed in section 3.2.2 in more detail. Example providers for the ReadResource have been implemented for the typical TelosB sensors, which transform values of the generic Read<val_t> interface to a uint8_t* buffer, cf. 3.

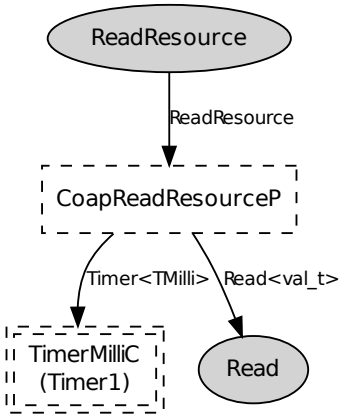

Figure 3: CoAP ReadResource implementation

While ReadResource provides the `get()` command, the WriteResource provides the `put()` command. A readable and writable resource such as the LED provides the ReadResource as well as the WriteResource interface, an example is part of the TinyOS `libcoap` source code. Since the LED resource

provides both, Read and WriteResource interfaces, another component is created to provide full access to the LedsC module. This CoapLedResource component differs from the CoapReadResource component in its `put()` method (which is part of the WriteResource interface) which is needed to set the state of the LED's. The same holds for the CoapFlashResource, where in addition the access of the storage support is implemented to read from and write to the flash memory.

Default handlers for `get()` and `put()` commands deal with resources that do not support these commands and send appropriate CoAP response codes back.

In logistics, sensor data representation needs to be independent of the actual sensor hardware (e.g. SHT11), a fixed-point SI unit representation was chosen. Since most sensor nodes do not have a hardware floating-point unit and software floating-point calculations are very expensive in terms of code size and computational time, all sensor values are calculated (according to [14]) by using a fixed-point representation with a precision of 1/100 per bit, e.g. 3454 to represent a relative humidity of 35.54 %. The calculations are performed in the various TranslateC components shown in figure 4 using and providing the Read interface. Therefore, the retrieved data at the client side has to be divided by 100 to obtain the appropriate floating-point value.

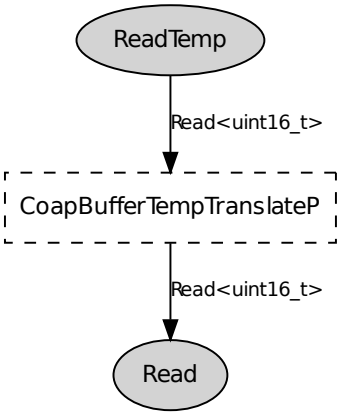

Figure 4: CoAP Data Representation Translation

3.2.2 Deferred Message Response

As described in [3], CoAP specifies two response methods, immediate and deferred. In most cases the server response is carried piggy-backed in the Acknowledgment message that acknowledges the request. Retrieving sensor data from a resource may take longer in some cases, e.g. the temperature sensor on TelosB nodes takes ~280 ms. To prevent retransmissions after a timeout (RESPONSE_TIMEOUT = 1s in coap-03, = 2s in coap-04/-05) of the requesting client, an Acknowledgment should be sent out by the server to signal a deferred message exchange. After the sensor data retrieval, the server sends the requested information in a new Confirmable message, which has to be acknowledged in return. The correlation between the request and the deferred response is done by the Token option of CoAP.

The deferred message exchange in TinyOS is handled by the events `getDone()` and `getPreAck()` implemented in ReadResource. After calling the read() command of the requested sensor in ReadResource, a timer is started with a predefined time (PREACK_TIMEOUT). In case the retrieval of the sensor value is shorter than the timeout, ReadResource stops the timer and signals `getDone()` back to the CoAP server component, which then transmits the retrieved value piggy-backed with an Acknowledgment message. In case of a deferred sensor retrieval, the `getPreAck()` event is signalled on timeout which triggers the sending of an Acknowledgment without data in the server component. After the retrieval of the values of the sensor has finished, `getDone()` is signalled which then sends a Confirmable message to the client.

3.3 Evaluation of libcoap Implementations

This section details some of the initial measurement results taken to evaluate `libcoap` implementation on TinyOS. 2 TelosB nodes are used, one as a CoAP server (based on blip-1.0), another one as border router (with blip's IPBaseStation), connected wirelessly via IEEE 802.15.4.

Table 3 shows a few results taken to evaluate the performance when retrieving different resources on the CoAP server. The retrieval time is measured at the CoAP client on the host machine to show the time taken to retrieve a given resource over one hop. The total number of bytes transmitted shows the size of all messages (CON, ACK, PreACK, etc.) exchanged during a retrieval of a resource. The resources /st and /r were implemented as deferred responses (because the RESPONSE_TIMEOUT in coap-03 was rather low), while the others were implemented as immediate responses. The resources implemented as deferred responses obviously show the highest number of retrieval time and bytes transmitted.

Resource	Type	Retrieval Time	Num. of Bytes Transmitted
/st	GET	297.04 ms	223 bytes
/sh	GET	143.57 ms	119 bytes
/sv	GET	92.69 ms	119 bytes
/r	GET	369.99 ms	229 bytes
/l	GET	69.55 ms	117 bytes
/l	PUT	71.12 ms	116 bytes
/ck	PUT	101.51 ms	142 bytes

Table 3: CoAP Implemented Resources on Sensor Nodes

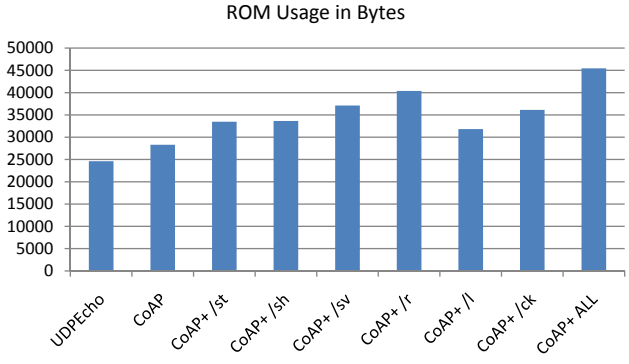

Figure 5: Memory Usage (bytes)

Figure 5 shows the memory usage when running a CoAP server. This is in comparison to a sensor node running with blip-1.0 and its simple application UDPEcho (to provide a UDP echo service on port 7 and a shell service on port 2000)[3]. The same compilation setup was used for compilation of the CoAP server while enabling different individual resources to measure the usage of ROM. 'CoAP+ALL' in Figure 5 shows the used ROM when running the CoAP server with all the resources as given in Table 1.

4. COAP VS HTTP

This section details the initial tests that are carried out to evaluate the performance of CoAP. The comparison is done with several HTTP based retrievals of data as follows.

- HTTP on TCP based protocols ("Apache2 HTTP server - access from Firefox browser","Apache2 HTTP server - access from Epiphany browser", "Apache2 HTTP server - access from wget" and "BareHTTP server - access from bareHTTPclient using TCP")
- HTTP on UDP based protocols ("BareHTTP server - access from bareHTTPclient using UDP")

The initial tests are carried out using two laptops that are configured as a client and a server, connected via a GPRS network. This setup was chosen in order to have a constrained link (the RTT of GPRS is ∼800 ms which is of the same order of magnitude as is expected for a multi-hop WSN) for which implementations of HTTP and CoAP on top of TCP and UDP were available at that time. Additionally, CoAP should also be applicable for the GPRS link in figure 1. Both laptops are configured to run with HTTP based protocols and also with CoAP. Both, the CoAP request and the HTTP GET are used to retrieve the value of temperature from the server laptop. For example, the Firefox browser sends GET http://134.102.188.208/temp/1.

When using TCP based protocols, one transaction of HTTP GET and HTTP OK consists of TCP connection setup time (TCP-SYN, TCP-SYN-ACK and TCP-ACK messages), TCP-ACK transmission for each HTTP packet and finally the time to close the TCP connection (2 TCP FIN-ACK and 2 TCP-ACK messages). In contrast to TCP based protocols, the UDP based protocols (HTTP server and client on UDP and CoAP) use only 2 messages to retrieve data.

The following results are taken to evaluate the performance of different types of retrieval of data.

- Response time: Time taken from sending the HTTP GET respectively the CoAP Request from the client until the connection is closed (cf. table 4).
- Total number of bytes transmitted: Total number of bytes transmitted within the above mentioned response time (figure 6).
- Overhead of the Header: This shows a separation of bytes in each layer.

Access Method	Time (sec)
Apache2-Firefox	38.774
Apache2-Epiphany	31.972
Apache2-wget	2.660
Apache2-bareHTTP client	3.032
bareHTTP(TCP)	3.076
bareHTTP(UDP)	1.104
CoAP	1.029

Table 4: Response Time (Seconds)

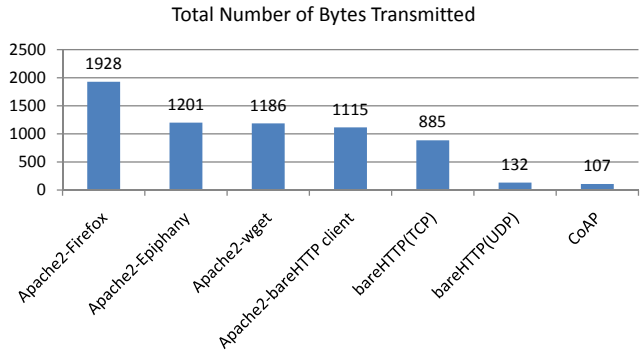

Figure 6: Total Number of Bytes Transmitted

When using a web browser, it tries to download the favicon additionally to the simple HTTP GET command. Therefore, the highest number of response time and the number of bytes transmitted are shown, when accessing an Apache2 server with Firefox or Epiphany browsers. As explained above, TCP based HTTP access methods consist of large number of other messages, both response time and the number of bytes transmitted are much higher compared to UDP based access methods (figure 6).

Table 5 shows the total number of bytes transmitted at each layer. TCP based HTTP has the highest number of bytes transmitted at each layer since it sends several other TCP messages in addition to the 2 HTTP messages. The difference in the size of data is caused by the termination of the data by '\r\n' in the case of HTTP, while the data is not terminated in the case of CoAP.

Header	HTTP/TCP	HTTP/UDP	CoAP/UDP
Link Layer	160	32	32
IP	200	40	40
TCP/UDP	340	16	16
HTTP/CoAP	181	41	17
Data	4	4	2

Table 5: Separation of Bytes at each Layer

These results show that UDP based protocols perform better for constrained networks due to using lower number of messages when retrieving resources. But, compared to UDP based HTTP, CoAP is a reliable protocol since it has its own simple retransmission capability.

[3]The executable was compiled using the gcc-4.4.5 without support for printfUART, but including support for string IP address handling

Previous work has done research on optimizing HTTP protocols to be used in WSNs [16]. Compared to this work, the use of CoAP is promising as a protocol currently being discussed at the IETF.

5. CONCLUSIONS AND OUTLOOK

The authors highlighted the use of CoAP in M2M enabled logistic applications. CoAP is used on the FSU and the sensor nodes in the container to retrieve resources in both directions. The CoAP implementation, `libcoap`, is made available for the two major embedded operating systems Contiki and TinyOS. For TinyOS, the `libcoap` library has been adapted to match the typical TinyOS split-phase operation and programming style (e.g. wiring resources to the server).

The initial evaluation of CoAP compared to HTTP based resource retrieval has shown the feasibility of using CoAP in an environment with constrained nodes and networks.

A further evaluation of the `libcoap` implementation will be done in a real environment using a container filled with food (e.g. bananas) and WSNs. The simulation of CoAP is also planned in order to investigate its performance in networks with higher number of nodes with the simulation tools OMNeT++ [17] and TOSSIM [18].

6. ACKNOWLEDGMENTS

This research project ('The Intelligent Container') is supported by the Federal Ministry of Education and Research, Germany, under reference number 01IA10001.

7. REFERENCES

[1] R. Fielding. Representational State Transfer (REST). *Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine*, 2000.

[2] IEEE Computer Society. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). In *IEEE Standard for Information technology- Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements.* Sept. 2006. IEEE Std 802.15.4-2006.

[3] Z. Shelby, B. Frank, and D. Sturek. Constrained Application Protocol (CoAP). Internet-Draft (work in progress), Available at: `http://tools.ietf.org/html/draft-ietf-core-coap-03`, Oct. 2010.

[4] Z. Shelby. Constrained Application Protocol (CoAP). Internet-Draft (work in progress), Available at: `http://tools.ietf.org/html/draft-ietf-core-link-format-01`, Oct. 2010.

[5] Z. Shelby (Ed.) and C. Bormann. Blockwise transfers in CoAP. Internet-Draft (work in progress), Available at: `http://tools.ietf.org/html/draft-ietf-core-block-00`, Oct. 2010.

[6] K. Hartke (Ed.) and Z. Shelby. Observing Resources in CoAP. Internet-Draft (work in progress), Available at: `http://tools.ietf.org/html/draft-ietf-core-observe-00`, Oct. 2010.

[7] C. Bormann and K. Hartke. Miscellaneous additions to CoAP. Internet-Draft (work in progress), Available at: `http://tools.ietf.org/html/draft-bormann-coap-misc-06`, Aug. 2010.

[8] Sensinode, Ltd. NanoSensor 2.4 GHz. White paper. Available at: `http://www.sensinode.com/media/flyers/sensinode-n740-flyer-web.pdf`. Last accessed 2011-02-09.

[9] Texas Instruments. System-on-Chip for 2.4 GHz ZigBee/IEEE 802.15.4 with Location Engine. White paper. Available at: `http://www.ti.com/lit/gpn/cc2431`. Last accessed 2011-02-09.

[10] A. Dunkels, B. Grönvall, and T. Voigt. Contiki – A Lightweight and Flexible Operating System for Tiny Networked Sensors. *Local Computer Networks, Annual IEEE Conference on*, 0:455–462, 2004.

[11] A. Dunkels. Full TCP/IP for 8-bit architectures. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, MobiSys '03, pages 85–98, New York, NY, USA, 2003. ACM.

[12] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Protothreads: simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 29–42, New York, NY, USA, 2006. ACM.

[13] S. Dawson-Haggerty. Design, implementation, and evaluation of an embedded IPv6 stack. Master's Thesis, UC Berkeley, 2010.

[14] Sensirion. Datasheet SHT1x. Datasheet. Available at: `http://www.sensirion.com/en/pdf/product_information/Datasheet-humidity-sensor-SHT1x.pdf`. Last accessed 2011-03-30.

[15] Th. Pötsch and M. Becker. TinyOS CoAP installation instructions. Available at: `http://docs.tinyos.net/index.php/CoAP`. Last accessed 2011-02-11.

[16] 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*, BuildSys '09, pages 43–48, New York, NY, USA, 2009. ACM.

[17] A. Varga and R. Hornig. An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, Simutools '08, pages 60:1–60:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[18] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137. ACM, 2003.