Towards a TDMA-based Real-Time Extension for the Constrained Application Protocol

Björn Konieczek, Martin Kasparick, Michael Rethfeldt, Frank Golatowski, Dirk Timmermann
Institute of Applied Microelectronics and Computer Engineering, University of Rostock
18051 Rostock, Germany, Tel./Fax: +49 381 498-7269
Email: bjoern.konieczek@uni-rostock.de

***Abstract*—Current IoT protocols, like the Constrained Application Protocol (CoAP), do not yet provide real-time behavior for the inter-device communication. In this paper, we propose a real-time extension for the CoAP standard that defines interfaces for the time synchronization among nodes and the time slot management. These interfaces enable a controlled exclusive network access based on a Time Division Multiple Access (TDMA) approach. With this extension, it is possible to realize access control on the application layer without the modification of lower layer protocols. The described interfaces are prototypically implemented within the jCoAP communication stack and evaluated in a multi-device real-world testbed. In our prototype, we used established algorithms for the time synchronization. The results, reveal the weaknesses of the chosen synchronization algorithm. However, the interface definition allows the usage of more accurate algorithms.**

I. Introduction

In the past few years, great effort has been put into the development of technologies for the Internet of Things (IoT), pushing the boundaries of applications and services. A great variety of protocols that enable interoperable device interaction have emerged. A very promising candidate is the Constrained Application Protocol (CoAP) [1]. It combines high interoperability and low communication overhead and offers important features for the machine-to-machine (M2M) communication. Recently emerged efforts aim to adopt IoT technologies in the realm of industrial automation. This development is also referred to as Industrial Internet of Things (IIoT) or Industry 4.0. Industrial applications introduce new requirements to IoT protocols. The real-time capability of the applications is an essential part of these requirements, as delayed reactions can cause serious danger for health and property. Currently, the specifications and implementations of IoT protocols like CoAP do not consider real-time constraints. In our previous work, we have introduced the jCoAP communication stack, a lightweight platform independent Java implementation of CoAP that can be processed in real-time on the device level. However, due to the distributed nature of IoT applications, the influence of the actual network transmission on the overall timing behavior is significant. Hence, it is necessary to enable real-time behavior on the network level. In the past, this problem has been addressed with lower layer approaches like fieldbus systems. Yet, fieldbuses lack scalability as they are drastically limited in address space. Furthermore, the different solutions are not compatible with each other. To overcome these weaknesses, several Ethernet-based solutions called Industrial Ethernet (IE) have emerged [2]. Most IE solutions rely on proprietary hardware or non-standard conform protocol adaptions. Hence, the existing IE systems are neither compatible with each other nor with common Ethernet. Widely used standards like common Ethernet do not guarantee deterministic timing behavior. A probabilistic network access scheme, buffering within switches and the limited bandwidth of switches lead to latency fluctuations. In this paper, we propose a real-time extension for the CoAP standard, that defines CoAP interfaces for time synchronization and access time management. These interfaces were prototypically implemented within the jCoAP communication stack utilizing established algorithms for the time synchronization. It enables exclusive network access based on a Time Division Multiple Access (TDMA) approach without the adaption of lower layer protocols. The main contributions of this paper are:

- Conception of a real-time extension for CoAP.
- Prototypical implementation within jCoAP.
- Evaluation in a real-world testbed.

The remainder of this paper is organized as follows. Section II gives a brief overview of the related work. Section III covers basic information about CoAP. In Section IV, the basic concept of the proposed CoAP extension is described. Section V discusses the performance evaluation in a real-world testbed. Section VI draws conclusions from the experimental results and gives an outlook on our future work.

II. Related Work

The widely used fieldbus systems suffer from a very limited address space and hence lack scalability. However, the number of interconnected devices in real-time systems is steadily increasing. Thus, fieldbus systems are gradually replaced by IE solutions. Another goal is to increase the interoperability of real-time systems with the remaining IT infrastructure [3]. In [2] an overview of currently available IE solutions is given. Many of these IE approaches like PROFINET IO/IRT, EtherCAT or SERCOS III are based on the master-slave or client-server principle, which implies the existence of a central instance [4]–[6]. Furthermore, many solutions require special hardware, which is typically proprietary. This leads to strong constraints regarding the planning and deployment of such networks. E.g., PROFINET IO/IRT requires special switches as the network frames are forwarded on a fixed route that is determined by the

978-1-5090-2339-4/16/31.00 © 2016 *IEEE*

transmission time [4]. Schlesinger et al. present an optimization for PROFINET IRT to omit costly reconfiguration through an automatic frame packing approach [7]. The approach is based on the insertion of subframes with delimiters. However, it still operates on layer 2 and hence, requires specialized hardware. In [8], Schlesinger et al. propose a real-time Ethernet approach that is based on the master-slave principle, where every node can act as a master. Deterministic timing behavior is achieved through a custom protocol on the data link layer. Thus, specialized hardware is needed and compatibility with common Ethernet is lost. In [9] Santos et al. present a concept called Hard Real-Time Ethernet Switching Architecture (HaRTES). Here, a hierarchical switch architecture is used to enable a dynamic bandwidth reservation for message streams. The approach was implemented in specialized switches as a hardware-software co-design. [10] extends HaRTES with the ability of multi-hop communication. To achieve this, a distributed scheduling algorithm called DGS is presented. However, the utilization of specialized switches degrades the interoperability and leads to higher deployment costs. In [11], Skodzik et al. present a hard real-time P2P approach called HaRTKad. It also employs TDMA to control the network access, but uses unchanged versions of Ethernet and IP/UDP. However, HaRTKad still needs protocols on top to provide M2M features.

III. The Constrained Application Protocol

CoAP follows the REST (Representational State Transfer) paradigm, where each request contains all necessary information and can be processed without prior knowledge (stateless) [12]. Furthermore, it applies the client-server principle, where a client sends a request to a server to invoke a service on or retrieve data from this server. All data objects or services are represented as resources that can be addressed through uniform resource identifiers (URI). CoAP comprises two sub-layers. The messaging layer defines the four message types confirmable (CON), non-confirmable (NON), acknowledgement (ACK), and reset (RST). Since CoAP is based on UDP, it can not rely on transport layer reliability. Therefore, it offers optional reliability through CON messages. The receiver of a CON message must respond with an ACK. When no ACK is received within a certain amount of time, the message is resent. However, retransmissions are not desired in real-time systems as the information value of not timely transmitted data is zero. Hence, it is preferable to use NON messages which do not require confirmation. RST messages are used to respond to invalid requests or requests that can not be interpreted. The request/response layer is placed on top of the message layer and performs the four basic methods GET, PUT, POST and DELETE. The CoAP header is binary encoded to reduce its size and parsing complexity. It consists of a 4 Byte mandatory part, an optional token and header options. An option comprises an option number, the option length and the option value. The CoAP standard consists of a basic specification and multiple extensions. The basic specification defines the communication model and the packet structure. Moreover, it describes the resource discovery through the mandatory *"/.well-known/core"* resource. The extensions introduce special features like a publish/subscribe mechanism (CoAP Observe). CoAP extensions may also define new options or response codes. Fig. 1 shows the available CoAP extensions and how our proposed real-time extension would fit into the CoAP specification.

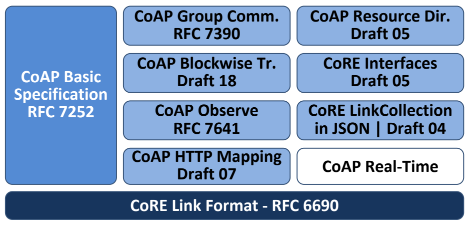

CoAP Basic Specification RFC 7252
CoAP Group Comm. RFC 7390
CoAP Resource Dir. Draft 05
CoAP Blockwise Tr. Draft 18
CoRE Interfaces Draft 05
CoAP Observe RFC 7641
CoRE LinkCollection in JSON | Draft 04
CoAP HTTP Mapping Draft 07
CoAP Real-Time
CoRE Link Format - RFC 6690

Fig. 1. Development status of CoAP and the proposed real-time extension

IV. Interface definition for CoAP Real-Time

The extension introduces CoAP interfaces for time synchronization and time slot management.These interfaces enable the use of a TDMA scheme to control the access to the physical network. In consequence, only one device at a time will access the network and hence, collisions as well as increased buffering times in switches and thereby non-deterministic timing behavior of the network will be avoided. Our approach is divided into two phases. In the initialization phase, all devices synchronize their clocks to establish a common time base and obtain a time slot. Then, every CoAP node is able to calculate the time span in which it can access the network exclusively. In the communication phase, all devices use only their assigned time slots to send data. The following sections describe the proposed time synchronization and time slot management approach.

A. Time Synchronization

Time synchronization can either be done in a distributed or centralized way. Centralized approaches have the advantage of being more straightforward to implement and typically require less resources on the client device. Nevertheless, a centralized time server poses a bottleneck and SPoF. Hence, centralized mechanisms lack scalability and robustness. Distributed synchronization techniques provide much better scalability and avoid a SPoF. However, they are much more complex in terms of implementation, maintenance and execution. The disadvantages of centralized mechanisms can be compensated when multiple synchronized time servers are used. Therefore, we propose a centralized approach, where any CoAP server can act as time server. For the time synchronization, we introduce a new *"/.well-known/time"* resource that every real-time capable CoAP server must provide. Additionally, we add a new CoAP SYN header option. The SYN option value contains a sequence number and the requested synchronization mode. The sequence number is used to match synchronization requests and responses. The synchronization mode can be chosen by the client with regard to the required synchronization accuracy. If a server does not offer the requested synchronization mode, it must respond with a RST message. For our practical evaluation and to show the feasibility of the interface mechanisms, only a single synchronization mode was implemented. We have chosen Cristian's algorithm [13] because of its simplicity. However, more synchronization modes can be defined. The client sends a GET request for the server's *"/.well-known/time"* resource and measures the round trip time (RTT) of this request.The request must include a SYN option. If no SYN option is present, the server must respond with an appropriate error message. Otherwise the server sends a response with the response code *"2.05 Content"*. It must contain a time stamp t_s and may contain additional information dependent on the synchronization mode. The new clock value t_{now} is calculated through Eq. (1).

$$t_{now} = t_s + RTT/2 \tag{1}$$

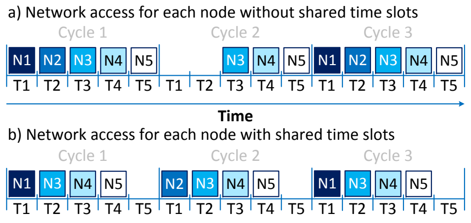
Cristian's algorithm only provides limited synchronization accuracy, as it assumes that the request and response latency are equal. Furthermore, it assumes that the computation time between the reception of a request and the creation of the time stamp t_s equals the time span between t_s and the actual sending of the response. These assumptions seldom prove to be true. However, Cristian's algorithm is easy to implement and our experimental results show that the accuracy is sufficient for a proof of concept implementation. Additional and more accurate synchronization modes will be added in the future. A node must repeat the time synchronization periodically to counteract clock drift. All subsequent SYN requests are send in the node's time slot to avoid interference with other nodes. The period of the resynchronization is chosen by the client.

B. Time Slot Management

To enable the time slot assignment by the time server, we added a *"/.well-known/timeslot"* resource. Typically, in TDMA systems a unique time slot is assigned to every device. This does not take their respective communication requirements into account. Automation systems consist of heterogeneous devices with different communication behavior. Some devices may need to communicate every 50 ms whereas others have communication periods of up to 250 ms. Such significant differences in the communication requirements are very typical, e.g., in the medical domain. The cycle time (the time span in which every device is allowed to communicate at least once) is chosen with regard to the highest requirement. This may result in a low network utilization as time slots may remain unused for multiple cycles. Our approach allows multiple nodes to share a single slot to increase the network utilization and capacity. n network participants can share a time slot, when the condition of Eq. (2) is fulfilled. Here, t_{cycle} is the cycle time and t_{period} is the communication period of the nodes.

$$t_{period} \geq n * t_{cycle} \tag{2}$$

Fig. 2 illustrates the benefits in a scenario with five nodes, a cycle time of 50 ms and five time slots T1-T5. The nodes N1 and N2 want to access the network every 100 ms whereas the nodes N3, N4 and N5 need to communicate every 50 ms. Without shared time slots, the highest possible network utilization is 80% as the time slots T1 and T2 remain unused every two cycles. Furthermore, all available time slots are assigned. Thus, the network capacity is exhausted. The network utilization seems to remain unchanged when N1 and N2 share a time slot. However, a sixth device could join the network as the time slot T5 is unassigned. In the proposed CoAP extension, a client sends a POST request to the *"/.well-known/timeslot"* resource of the time server to obtain a time slot. This request must include its desired communication period as payload. In our proof of concept implementation, the period was transmitted as plain 32 bit signed integer value that indicates the communication period in milliseconds. In the future, a JSON format will be described for all messages. The server must check whether a time slot with the same communication period already exists. In this case, the server can assign the same time slot to the requesting node if the capacity K of this slot is not yet reached.


a) Network access for each node without shared time slots
Cycle 1
Cycle 2
Cycle 3
N1 N2 N3 N4 N5 N3 N4 N5 N1 N2 N3 N4 N5
T1 T2 T3 T4 T5 T1 T2 T3 T4 T5 T1 T2 T3 T4 T5
Time
b) Network access for each node with shared time slots
Cycle 1
Cycle 2
Cycle 3
N1 N3 N4 N5 N2 N3 N4 N5 N1 N3 N4 N5
T1 T2 T3 T4 T5 T1 T2 T3 T4 T5 T1 T2 T3 T4 T5
Fig. 2. Network access patterns for five nodes with different communication requirements with (b) and without (a) shared time slots

$$K = t_{period}/t_{cycle} \tag{3}$$

If the node can be added to the time slot, the server responds with a message containing the time slot number n_{slot}, the time slot length t_{slot}, the cycle length t_{cycle} as well as a cycle offset O_{cycle}, and the slot capacity K. The cycle offset determines for which cycle number within the communication period the slot assignment is valid. If the slot capacity is reached or there is no time slot with the same communication period, the client is added to an unused slot. Here, the cycle offset is zero. Afterwards, the client can calculate the beginning t_{start} of its next time slot via Eq. (4)-(7).

$$C_c = t_{now}/t_{cycle} \tag{4}$$

$$O_c = C_c \mod K \tag{5}$$

$$C_{dist} = \begin{cases} O_{cycle} + K - O_c, & \text{if } O_c \geq O_{cycle} \\ O_{cycle} - O_c, & \text{else} \end{cases} \tag{6}$$

$$t_{start} = (C_c + C_{dist}) * t_{cycle} + t_{slot} * n_{slot} \tag{7}$$

Here, C_c is the number of the current cycle, t_{now} is the current time, O_c is the current offset within the communication period of the time slot, and C_{dist} is the number of cycles until the time slot belongs to the CoAP node again. During its time slot, a node may send multiple messages as long as it can finish the transmission before the next time slot starts. If a node could not be added to a time slot and no free slots are available, the server must respond with a *"5.03 Service Unavailable"* error message. The cycle time in the system is still chosen with regard to the highest communication requirement but the low network utilization is avoided through the shared time slots. If the desired communication period of a device is lower than the cycle time, it should not be added to the cycle, as the desired communication behaviour can not be guaranteed. The time slot

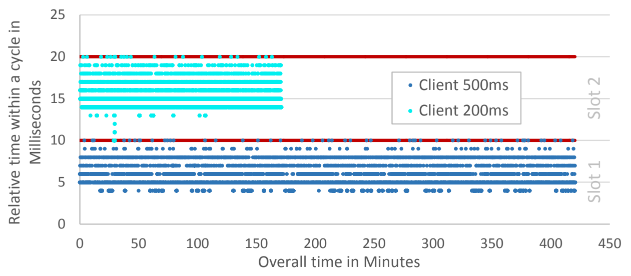


Fig. 3. Receive times measured on the server in a dual client scenario

assignment of a node persists for its life time. The node's life time ends, when it signals its disappearance or crashes. It can indicate that it leaves the network through a POST request to the *"./well-known/timeslot"* resource of the time server with a negative value for the communication period. The server may notice a crashed node through the absence of synchronization requests from that node. If a client is re-introduced to the network, it must obtain a new time slot.

V. Prototype Evaluation

To evaluate the feasibility of the proposed standard extension, we implemented the presented mechanisms within jCoAP and set up a real-world testbed with three devices. The used hardware platform was the Intel Galileo Board Gen. 1. It is equipped with an Intel Quark X1000 SoC with a clock frequency of 400 MHz, 256 MB of RAM, and a 10/100 MBit/s Fast Ethernet adapter. As operating system a customized Linux with a fully preemptable Kernel version 3.8 was used. All devices used the Aicas JamaicaVM 6.3 as Real-Time Java Virtual Machine. The boards were interconnected through 10/100 MBit/s Fast Ethernet and a switch. For our experiment, we used a single server and two clients with a desired communication period of 500 ms and 200 ms, respectively. A cycle time of 100 ms split into ten time slots with a length of 10 ms and a resynchronization period of 30 s were used. At first, the clients performed an initial time synchronization and obtained a time slot. Then, the slotted sending mode was initiated and the client application was started. The clients retrieved a 16 byte payload from an *"/echo"* resource on the server via a GET request. This request was performed 50,000 times. The server measured the receive time of all incoming messages. Fig. 3 shows the obtained results. The y-axis shows the relative time within a time slot and the x-axis shows the overall time of the test run. The red lines represent the time slot borders within a cycle, while the points show the measured time stamps. As expected, the clients do not share a time slot, as they have different communication requirements. The second client has finished its application 2.5 times faster than the first because of the fixed number of requests. Moreover, it can be seen that no obvious time slot violations occur. In a single synchronization period, the messages of the second client arrived very early within the time slot. This indicates that the medium has been already accessed during the first time slot. This erroneous behavior originates from the time synchronization, as Cristian's algorithm has very limited accuracy. However, the results show the overall feasibility of the proposed extensions.

VI. Conclusion

In this paper, we have presented a basic concept for a real-time extension for the Constrained Application Protocol. The proposed extension is based on a TDMA scheme and includes mechanisms for time synchronization among CoAP nodes and the management of time slots. These mechanisms are purely software based and, in contrast to other approaches, require neither special hardware nor modifications of lower layer protocols. Furthermore, the proposed TDMA approach allows the sharing of time slots between multiple nodes and hence, enables higher network utilization and a higher number of network participants. An evaluation of the prototype implementation has shown the general feasibility of the proposed extension. We have shown that a purely software based TDMA mechanism can be effectively applied to achieve deterministic network access over common Ethernet. However, the performance evaluation has revealed the time synchronization as a potential source of erroneous behavior. In our future work, we will evaluate a refined synchronization approach, where the server response also includes the receive time of the SYN request and the approximate send time of the response. Thus, the client can take the processing time on the server into account. Additionally, we will develop strategies to share information about time slot assignments among multiple servers to increase the scalability of the system. Furthermore, we will evaluate the described extension in a larger testbed and a simulation environment.

References

[1] Z. Shelby, K. Hartke, and C. Bormann, *RFC 7252: The Constrained Application Protocol*, online, IETF Std.

[2] P. Danielis et al., "Survey on Real-Time Communication Via Ethernet in Industrial Automation Environments," in *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Barcelona, Spain, September 2014, pp. 1–8.

[3] T. Sauter, "Integration Aspects in Automation - A Technology Survey," in *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, vol. 2. IEEE, 2005, pp. 9–pp.

[4] R. Pigan and M. Metter, *Automating with PROFINET: Industrial Communication Based on Industrial Ethernet*. Publicis Publishing, 2008.

[5] G. Cena et al., "Evaluation of EtherCAT distributed clock performance," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 1, pp. 20–29, 2012.

[6] F. Klasen, V. Oestreich, and M. Volz, *Industrial Communication with Fieldbus and Ethernet*. VDE-Verlag, 2011.

[7] R. Schlesinger, A. Springer, and T. Sauter, "Improving profinet irt frame packing using ethernet control characters," in *Factory Communication Systems (WFCS), 2015 IEEE World Conference on*. IEEE, 2015, pp. 1–4.

[8] R. Schlesinger and A. Springer, "VABS - A new approach for Real Time Ethernet," in *Industrial Electronics Society, IECON - 39th Annual Conference of the IEEE*. IEEE, 2013, pp. 4506–4511.

[9] R. Santos et al., "Multi-level hierarchical scheduling in ethernet switches," in *Proceedings of the ninth ACM international conference on Embedded software*, 2011, pp. 185–194.

[10] M. Ashjaei et al., "Response time analysis of multi-hop HaRTES ethernet switch networks," in *10th IEEE Workshop on Factory Communication Systems (WFCS)*, 2014, pp. 1–10.

[11] J. Skodzik, P. Danielis, V. Altmann, and D. Timmermann, "Hartkad: A hard real-time kademlia approach," in *11th IEEE Consumer Communications & Networking Conference (CCNC)*, 2014, pp. 566–571.

[12] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, 2000.

[13] F. Cristian, "Probabilistic clock synchronization," *Distributed computing*, vol. 3, no. 3, pp. 146–158, 1989.