A Distributed Time Server for the Real-Time Extension of CoAP

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

***Abstract*—In the recent past, the development of applications and protocols for the Internet of Things (IoT) made a big leap forward. New approaches have emerged 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. It is predicted for the number of smart interconnected devices participating in automation systems to grow significantly in the future. However, the industrial domain introduces new requirements for IoT technologies regarding the timeliness of interactions. Current IoT protocols, like the Constrained Application Protocol (CoAP), do not yet provide real-time behavior for the inter-device communication. In our previous work, we have already proposed a real-time extension for CoAP that enables deterministic network behavior through a TDMA-based approach. We have shown that the proposed mechanisms for time synchronization, time slot management, and access control can be realized purely software-based. However, a central instance is needed as a time server. This introduces a Single Point of Failure (SPoF) to the system, limiting the robustness and scalability of the approach. In this paper, we introduce a concept for a distributed time server for CoAP. The proposed concept includes a refined time synchronization mechanism as well as strategies to select multiple time servers and share information between them. Furthermore, the described amendments to the real-time extension are integrated into the lightweight platform-independent jCoAP communication stack and evaluated in a multi-device real-world test bed.**

I. Introduction

In the past few years, the development of technologies for the Internet of Things (IoT) has taken a big leap forward. A great variety of communication protocols have emerged, that enable different devices to interact with each other in a vendor-independent manner. A very promising candidate among these protocols is the Constrained Application Protocol (CoAP). CoAP is a RESTful (Representational State Transfer) web transfer protocol that combines high interoperability with a low communication overhead and machine-to-machine (M2M) communication features. These M2M features include, among others, discovery and publish/subscribe mechanisms and group communication capabilities. A stateless HTTP mapping allows for an easy integration into already existing networks and information systems. Recently emerged efforts aim to adopt IoT technologies in the realm of industrial automation. The main goal is to allow a flexible composition of automation systems and an easy integration into a company's IT infrastructure. This development is also referred to as Industrial Internet of Things or Industry 4.0. The number of smart interconnected devices that participate in automation systems is predicted to grow significantly in the future [1]. However, industrial applications introduce new requirements to IoT protocols apart from interoperability. Real-time capabilities are an essential part of these requirements. Delayed reactions can cause serious hazards for health or property. The timeliness of transactions is currently controlled on the lower layers of the network stack, e.g., through fieldbus systems. However, fieldbusses are drastically limited in address space and only provide limited interoperability. Therefore, they are not feasible for IIoT applications. To omit these problems, a variety of Ethernet-based interconnection solutions called Industrial Ethernet (IE) have emerged [2]. However, the majority of IE solutions relies on proprietary hardware or non-standard conform protocol adaptations. In consequence, the existing IE systems are neither compatible with each other nor with common Ethernet, which leads to separate incompatible communication domains. Widely used standards like common Ethernet do not guarantee deterministic timing behavior, though. Here, a probabilistic network access scheme, buffering within switches, and the limited bandwidth of switches lead to latency fluctuations and thereby cause non-deterministic timing behavior of distributed applications. In our previous work we have already proposed a real-time extension for the CoAP standard, that allows deterministic communication over common Ethernet through a TDMA-based medium access control. We have shown that the proposed mechanisms for time synchronization, time slot management, and access control can be realized purely software based. The extension was integrated into the platform independent jCoAP communication stack, a lightweight Java implementation of CoAP. However, a central instance was needed to enable time synchronization resulting in a Single Point of Failure (SPoF). In this paper, we introduce a distributed time server concept to increase the robustness and scalability of the proposed real-time extension for CoAP. The described concept was integrated into the jCoAP communication stack and evaluated in a real-world testbed. The main contributions of this paper are:

- Distributed time servers and time slot management
- Refinement of the time synchronization between CoAP nodes
- Evaluation in a real-world test bed with multiple devices

The remainder of this paper is organized as follows. In Section II, an overview of the related work in the field of device and real-time communication is given. Section III gives a basic understanding of the Constrained Application Protocol. In Section IV, the currently proposed real-time extension for CoAP is discussed. Section V describes the proposed concept for a distributed time server for CoAP including the refinement of the time synchronization and balanced time slot management between multiple servers. Section VI deals with the evaluation of the distributed time server concept in a real-world test bed. Section VII draws conclusions from the experimental results and gives an outlook on our future work.

II. Related Work

A. Device Communication

In the IoT domain, IP-based approaches for device communication are preferred, as they simplify integration into already existing networks [3]. A widely used approach is the Devices Profile for Web Services (DPWS) [4]. DPWS aims to bring classical web services technologies based on the WS-* protocols into the domain of resource constrained devices. Web services offer a high degree of interoperability and flexibility. However, like classical web services, DPWS utilizes XML-based SOAP and HTTP messages. This causes a significant communication overhead and requires a sufficient amount of computational power and memory on the device. [5] shows that the communication overhead of DPWS can be significantly reduced by applying compression techniques to the SOAP messages. Furthermore, [6] describes how the memory footprint of DPWS can be reduced. On the other hand, the proposed techniques also lead to a reduced feature set and hard to develop services. Moreover, DPWS still relies on HTTP/TCP which introduces a high message overhead. TCP causes additional overhead due to its handshake procedure and the automatic retransmission of lost messages. However, retransmissions are not necessarily desirable in real-time systems as they cause non-deterministic timing behavior and the information value decreases over time. CoAP is a promising lightweight alternative to DPWS. Since the first specification of CoAP in 2010, many different CoAP implementations have emerged. The most significant one is the Californium stack, which has been developed by Kovatsch et al. and is now the reference implementation of CoAP within the Eclipse Project [7]. In [8] we compared Californium to our own Java implementation called jCoAP in terms of performance and real-time capabilities. The results show that jCoAP can be processed much faster and deterministically on the device level. Besides Californium, Kovatsch et al. developed two other CoAP implementations: Erbium, which is a C implementation and part of the embedded operation system Contiki [9], and Copper, which is a JavaScript based plug-in for the web browser Firefox [10]. Although Copper can only be used as a client, it is often used to test CoAP implementations and applications. Another important C implementation of CoAP is libcoap by Bergmann et. al., which is part of the embedded operating system TinyOS [11]. In [12], a SOAP-over-CoAP binding is presented to replace HTTP as a transport protocol in DPWS. However, the computation overhead is still rather high compared to plain CoAP, as CoAP itself already provides important M2M features. Besides DPWS and CoAP various other protocols like MQTT exist. These protocols have various restrictions, e.g., MQTT is only suitable for purely event-based systems.

B. Real-Time Communication

Traditional fieldbus systems like PROFIBUS or CC-Link have been used in the real-time domain since the 1990s. Yet, these 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. As a result, fieldbus systems are gradually replaced by IE solutions, which allow a much higher number of network participants. Another goal is to increase the interoperability of real-time systems with the remaining IT infrastructure of a company [13]. An overview of currently available IE solutions along with information about their real-time capabilities and hardware/software requirements can be found in [2]. 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 [14]–[16]. Furthermore, many solutions require special hardware to achieve real-time behavior. These hardware components are typically proprietary and very expensive. This leads to strong constraints regarding the plannings and deployment of such networks. For example, PROFINET IO/IRT requires special switches as the network frames are forwarded on a fixed route that is determined by the transmission time [14]. [17] proposes another real-time Ethernet approach that is based on the master-slave principle, where every node can act as a master. In order to achieve deterministic timing behavior, the data link layer is exchanged with a custom protocol. As a result, specialized hardware is needed and hence, compatibility with common Ethernet is lost. [18] also proposes a real-time Ethernet protocol adopting the master-slave pattern. Here, the master represents a SPoF and the slaves require special hardware to participate in the network. In [19] Santos et al. present a new concept called Hard Real-Time Ethernet Switching Architecture (HaRTES). Here, a hierarchical switch architecture is used to enable a dynamic bandwidth reservation for certain message streams. As a proof of concept, the approach was implemented in specialized switches as a hardware-software co-design. [20] extends HaRTES with the ability of multi-hop communication over multiple HaRTES switches. 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 [21], Schmidt et al. present a real-time Ethernet approach called DRTP, that avoids the disadvantage of a central instance. In the described concept, the network access is controlled through two proprietary layers that are introduced directly above the Ethernet layer. The medium access scheme is based on a TDMA approach with defined time slots. Nevertheless, the missing support of TCP/UDP and IP degrades the interoperability and prohibits the seamless

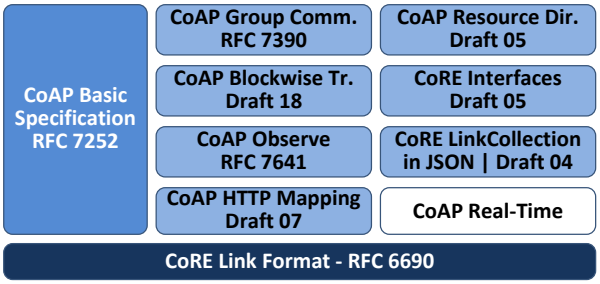

Fig. 1. Development status of CoAP and the proposed real-time extension

integration in already existing infrastructures. Total horizontal and vertical integration of industrial automation systems is a vital goal of the IIoT. In [22], Skodzik et al. present a hard real-time P2P approach called HaRTKad. HaRTKad also employs a TDMA mechanism to control the network access. But in contrast to [21], it uses unchanged versions of Ethernet and IP/UDP. However, HaRTKad still needs protocols on top to provide vital M2M features, like service discovery and publish/subscribe mechanisms. In [23], the use of CoAP on top of HaRTKad to enable the transmission of larger data amounts was evaluated. The resulting system is called CoHaRT. However, CoHaRT only makes use of the blockwise communication feature of CoAP. Hence, no statement about the general feasibility of this approach could be given.

III. The Constrained Application Protocol

The Constrained Application Protocol is a specialized web transfer protocol. It was designed by the IETF CoRE working group specifically to meet the requirements of resource constrained environments [24]. CoAP is based on the client-server scheme, where a client sends a request to a server to invoke a service or store or retrieve data. Furthermore, it was designed after the REST principle. Here, every request can be interpreted on its own without further information. The main advantage of the REST principle is that the server does not have to keep track of any session data [25]. As this task is delegated to the client side, the server can be much more lightweight. In CoAP, all data objects or services are represented by resources that are addressed through a Uniform Resource Identifier (URI). The CoAP standard consists of a main specification, that defines the header structure as well as the general communication flow and several extensions that describe useful but not mandatory features. These extensions include important M2M features like a publish/subscribe mechanism (CoAP Observe) or group communication. Vital features, e.g., the resource discovery through the mandatory *"/.well-known/core"* resource that must be provided by all servers, are described in the main specification. As shown in Fig. 1 many parts of the CoAP specification are still under development. The modular structure of CoAP allows for an easy extension without the loss of standard compliance.

CoAP is based on UDP, which is a good prerequisite for the use in real-time systems. UDP omits the expensive handshaking process, non-deterministic rate adaptations, and retransmissions of TCP. Retransmissions are not necessarily desired in real-time systems, as the information value is zero after the deadline.

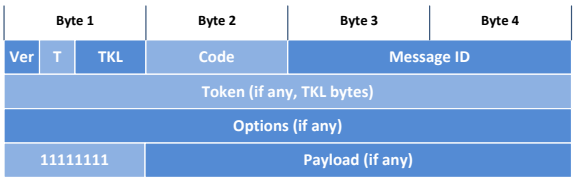

Fig. 2. Header structure of CoAP messages

CoAP itself consists out of two sublayers. The messaging layer defines the four message types: confirmable (CON), non-confirmable (NON), acknowledge (ACK), and reset (RST). CON messages enable a reliable communication between two CoAP nodes. The definition of reliability mechanisms on higher layers becomes necessary as UDP does not provide reliability on the transport layer. CON messages must be acknowledged by the receiver with an ACK. If no ACK is received after a timeout, the message is resent. However, this may lead to non-deterministic behavior. For non-reliable communication NON messages are used, as they do not require an ACK. RST messages are used when erroneous or unwanted CoAP messages are received. The request/response layer is placed on top of the messaging layer and defines the four basic operations GET, PUT, POST, and DELETE that can be performed on any resource. Thereby, GET is used to retrieve the current state of a resource. The PUT operation is either used to update or create a resource from the appended payload, whereas POST is used to process the data contained in a request. A POST can result in an update of a resource or the creation of a new resource. Hereby, the result depends on the resource itself and is implementation-specific. The DELETE operation can be used to delete resources on a server. The header of a CoAP message is binary coded to reduce the message size and the parsing complexity. Fig. 2 shows the basic header structure of a CoAP message.

The first byte of the header contains the protocol version (Ver), the message type (T) and the length of an optional message token (TKL). If the message contains no token, the TKL field is zero. The second byte contains a message code. In a request, the message code indicates the desired operation (GET, PUT, POST, DELETE). In a response, the message code represents a response code, that indicates the result of the requested operation. The bytes three and four contain a message ID that is used to match ACKs to the corresponding CON messages. The message ID is followed by the optional token and the header options. CoAP header options consist of an option number, the length of the option, and the option value. The same option can occur multiple times in a CoAP header and the options are ordered by their option number. Options can be used to trigger special behavior on the server like the subscription to changes of a resource. If the message contains any payload, the options are followed by a 1 byte payload marker. The minimal header size of a CoAP message is 4 byte. However, it is very likely to be larger as the URI path of the target resource is represented by header options.

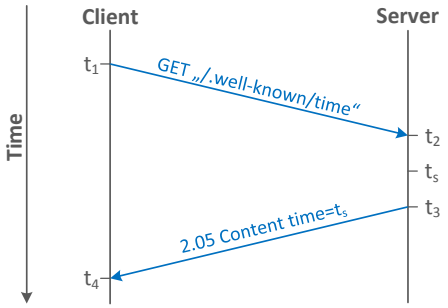

Fig. 3. Flow of the synchronization process according to Cristian's algorithm

IV. A REAL-TIME EXTENSION FOR COAP

In our previous work, we have already proposed a real-time extension for CoAP. That extension enables a deterministic communication between multiple CoAP nodes over common Ethernet by applying a TDMA-based network access control. To achieve this, we added mechanisms for time synchronization and time slot management to the CoAP specification. Two new resources are added to the *"/.well-known/"* path, that is mandatory for all CoAP servers. The *"/.well-known/time"* resource is used for the time synchronization among CoAP nodes. Additionally to this resource, a new SYN CoAP header option was added to the specification. The SYN option contains the desired synchronization mode and a sequence number that is used to map the server responses to the corresponding SYN requests. The synchronization mode can be chosen by the client device with regard to the desired accuracy. In our first prototype, the only implemented mode was Cristian's algorithm as depicted in Fig. 3 [26].

The client sends a GET request for the *"/.well-known/time"* resource to the server. Hereby, the request must contain the SYN option. The server responds with the time stamp t_s. During the transaction the client measures the round-trip time (RTT) of the request and then estimates its new clock value through Eq. (1). The RTT is the time span between t_1 and t_4. The error of the time synchronization t_e is described by Eq. (2), where T_{min} is the minimal transmission time for a CoAP message.

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

$$t_e = \pm \left(\frac{RTT}{2} - T_{min} \right) \tag{2}$$

It can be seen, that the synchronization error increases with the RTT. Hence, the time server always responds immediately to synchronization requests to keep the error low. However, this corrupts the TDMA-based network access and emphasizes the necessity of a refined time synchronization mechanism.

The *"/.well-known/timeslot"* resource was added to allow a client to obtain a time slot from the server in which it is allowed to access the network. Our approach allows n clients to share a time slot, when their desired communication period t_{comm} fulfills the condition of Eq. (3), where t_{cycle} is the length of the time cycle. In this way, the maximum number of network participants is increased and individual requirements of the different devices are taken into account.

$$t_{comm} \geq n * t_{cycle} \tag{3}$$

To obtain a time slot, a client sends a POST request with its desired communication period to the server's *"/.well-known/timeslot"* resource. The server adds the client to a time slot that is already in use if the communication period t_{period} of this time slot is smaller than the desired period of the new client and the slot capacity K is not yet reached. The slot capacity determines the maximum number of nodes that can share this time slot. If a client can not be added to an already used time slot, it is added to a new time slot. The server responds to the client request with the time slot number n_{slot}, the time slot length t_{slot}, the cycle length t_{cycle}, the communication period of the time slot t_{period}, and the cycle offset within the time slot O_{cycle}. Each client can then calculate the beginning of its next time slot t_{start} via Eq. (4), (5), (6) and (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 current cycle, t_{now} the current time, O_c the current offset within the communication period and C_{dist} the number of cycles until the time slot belongs to the CoAP node the next time. If a CoAP node could not be added to a time slot and no unused time slots are available, the server must respond with a *"5.03 Service Unavailable"* error message.

V. BASIC CONCEPT

A main drawback of the approach described in Section IV is the centralized time server that represents a SPoF. Additionally, in our previous work, the time server always responded immediately to synchronization requests instead of in its own time slot. This section describes the basic concept behind our approach to a distributed time server for CoAP. The main improvements of our approach compared to a centralized time server are the distribution of the synchronization load among multiple servers, the balanced assignment of clients to the time servers, and higher robustness due to the elimination of the SPoF. Furthermore, a refined time synchronization is presented that enables a slotted communication of the time servers.

A. Refined Time Synchronization

The time synchronization between network participants is of vital importance to establish a TDMA-based medium access control. The devices need to have the same time base in order to determine their network access time. As described in Section IV, our previous work used Cristian's algorithm. A major disadvantage of this algorithm is, that the time span between t_1 and t_s is assumed to be equal to the time span between

t_s and t_4 (compare Fig. 3). This is not necessarily the case, especially when the server only responds in its own time slot. Therefore, we added a second synchronization mode, which is a modified NTP-like version of Cristian's algorithm [27]. In the new synchronization mode, the server measures the receive time of the request t_2. Furthermore, the actual send time of the response t_3 is estimated. In our prototype implementation, we assume that the response will be sent within the next time slot of the server. However, the actual send time may depend on the number of messages in the server's send queue. Besides the time stamp, the server response also contains the overall processing time t_p on the server and the time span t_{delay} between the time stamp is taken and the response is actually sent. The client can then calculate its new clock value through Eq. 10.

$$t_p = t_3 - t_2 \tag{8}$$

$$t_{delay} = t_3 - t_s \tag{9}$$

$$t_{now} = t_s + \frac{t_4 - t_1 - t_p}{2} + t_{delay} \tag{10}$$

The additional knowledge about t_p and t_{delay} enables a more accurate estimation of the actual network latency as the processing delay can be taken into account. The synchronization error changes from Eq. 2 to Eq. 11.

$$t_e = \pm \left(\frac{RTT - t_p}{2} - T_{min} \right) \tag{11}$$

In consequence, the time synchronization is much more accurate, even when the time server only responds in its own time slot.

B. Distributed Time Server

Our approach for a distributed time server is based on the assumption that all services and data objects can be represented as resources. In the first step an initial time server is chosen. It can be defined either by the user at deployment time or through a competition strategy at runtime. The initial server starts a synchronization client as a resource, to enable the sending of requests. The URI of the synchronization client is *"/.well-known/synclient"*. Through this client, it starts a discovery process. The discovery is done with a GET request for the *"/.well-known/core"* resource to the ALL_COAP_NODES multicast address. To ensure that only potential time servers are found, the request contains the query string *"title=/.well-known/time"*. After n server responses were received or a timeout has passed, the initial server starts and configures the synchronization clients on the additionally acquired servers. This is done via a POST request for the *"/.well-known/synclient"* resource. It contains time parameters like the cycle time and slot length as well as a list of all time servers. Upon reception of this request, the additional servers start their synchronization clients and send a GET request for the *"/.well-known/timeslot"* resource of all other time servers. This request contains the OBSERVE option, indicating that they want to be notified if the resource value changes. The notifications include the client's IP address, the time slot number, the time slot's communication

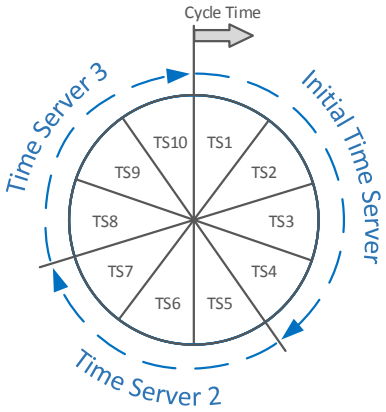

Fig. 4. Area of responsibility partitioning for three time servers in a cycle with ten time slots

period, and the offset of the client within the time slot. In consequence, each time server always has a complete overview of the network participants, their assigned time slots, and their communication periods. Afterwards, the additional servers obtain a time slot from the initial server through a POST request for *"/.well-known/timeslot"*. The initial time server always occupies the first time slot in the cycle. When a time slot was assigned, each additional time server synchronizes its clock with the initial server as described in the previous subsection. When a time server has obtained a time slot and synchronized its clock with the initial server, it initializes the slotted communication. All subsequent requests or responses are only sent within the server's own time slot. When the initialization phase is over, each of the n time servers is responsible for all clients in the nth part of the time cycle as depicted in Fig. 4 for three time servers in a cycle with ten time slots TS1 - TS10.

If a client contacts a time server to obtain a time slot, the server will assign a time slot based on its knowledge about the time cycle and the time requirements of the client. The time server will inform all other time servers about the new client and they will update their view of the time cycle accordingly. The server response to the client contains the address of the responsible time server additionally to the time slot information described in Section IV. The client will send all synchronization requests to its assigned time server. Fig. 5 illustrates the described communication flow for two time servers and a single client that joins the network.

The following subsection describes how the time slots are assigned so that an uniform distribution of the synchronization load is achieved.

C. Balanced Time Slot Management

As described in Section IV, a client sends a POST request containing its communication requirements t_{comm} to the *"/.well-known/timeslot"* resource of a time server to obtain a time slot. The assignment of a suitable time slot is done in a two-staged approach. In the first stage, the server checks whether a time slot with a communication period t_{period} lower or equal to t_{comm} exists. Furthermore, the number of nodes

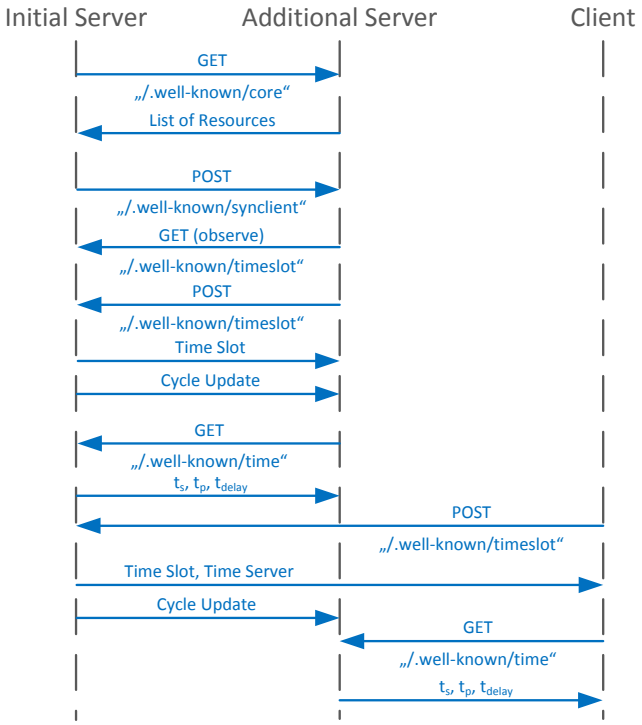

Fig. 5. General communication flow for the distributed time server approach

assigned to this time slot must be lower than the slot capacity. If such a time slot is available, the client is assigned to it. The first stage ensures that the time cycle is used efficiently as two clients with a high t_{comm} will share a single time slot instead of occupying two time slots. The second stage is invoked, when no suitable time slot can be found in the first stage. Here, the server selects an unused time slot within the area of responsibility of the time server with the least number of used time slots. In this way a balanced distribution of the synchronization load is achieved. If there are no time slots available, the server must answer with an appropriate error message. Henceforth, the client will not participate in the network to avoid a corruption of the TDMA scheme.

D. Time Server Failure

In case of a failure of one of the time servers, two things are of major importance. Firstly, the node failure must be recognized by at least one of the other time servers. If the initial server fails, this will happen automatically as the additional servers will get no response to their SYN requests. The first time server to notice the failure will become the new initial server and communicate its new role to the other time servers. If one of the additional time servers fails, the initial server may notice it through the absence of SYN requests from this server. Secondly, node failures must be handled by the remaining time servers. Thereby, two different strategies can be pursued. On the one hand, a new time server can be acquired and assigned to the same area of responsibility as the failed time server. Hereby, all nodes that are directly affected by the node failure need to be informed about the new time server. On the other hand, the areas of responsibility can be rescaled to fit the new

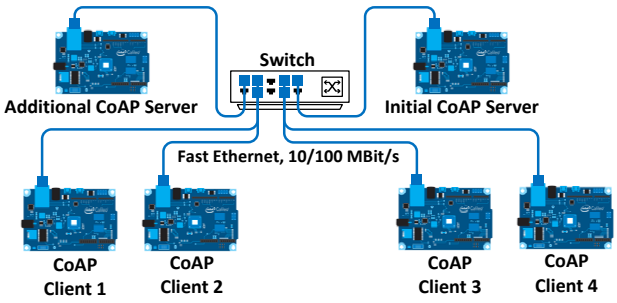

Fig. 6. Test setup for the performance evaluation

TABLE I
TIME SLOT ASSIGNMENT IN BOTH OF THE TEST SCENARIOS

	Scenario 1	Scenario 2
Initial Server	1	1
Additional Server	-	6
Client 1	2	2
Client 2	3	7
Client 3	4	3
Client 4	5	8

number of time servers. This approach will cause potentially high network overhead as a higher number of devices needs to be informed about their new time servers. Hence, it should only be used as fallback strategy when no new time server can be acquired. However, the preferred strategy may be chosen application-specific.

VI. PROTOTYPE EVALUATION

To evaluate the feasibility of our approach for a distributed CoAP time server, we integrated the mechanisms described in Section V into the platform-independent Java-based jCoAP communication stack. Furthermore, we built up a real-world test bed that comprised up to six devices. As hardware platform we used the Intel Galileo Board Gen. 1. It is equipped with a Quark X1000 SoC that has an x86 architecture und runs with a clock frequency of 400 MHz. It additionally provides 256 MB of RAM and a 100/10 Mbit/s Fast Ethernet adapter. As software platform a fully preemptable Linux Kernel ver. 3.8 in combination with the real-time Java Virtual Machine (JVM) Aicas JamaicaVM was used. All devices were interconnected through Fast Ethernet and a switch as depicted in Fig. 6.

For our experiments we evaluated two different scenarios. In the first scenario four clients and a single time server were used. For the second scenario an additional time server was added to the ensemble. First, the servers were started and initialized. Afterwards, the clients were added to the network, obtained a time slot from the initial server, and started to synchronize their clocks with their assigned time server. For all devices a resynchronization period of 30 s was chosen to maintain the clock synchronicity. The cycle time was 100 ms and a time cycle consisted out of ten time slots with a length of 10 ms. Table I shows the time slot assignment for both scenarios.

During the experiments the number of SYN requests, that were sent by the clients and received by the servers, were measured. Fig. 7 shows the obtained results for the first (a) and the second (b) scenario. The y-axis shows the number of

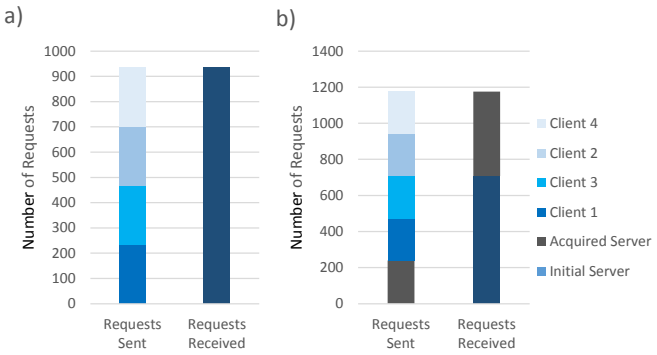

Fig. 7. Synchronization load within the network

TABLE II
TIME DIFFERENCE BETWEEN TRANSMISSION AND RECEPTION OF SYN REQUESTS AND OBSERVED LATENCY WITH STANDARD DEVIATION

	Time Difference	Latency
Average	2.2256 ms	2.1 ms
Median	2 ms	2 ms
Standard Deviation	0.3038 ms	0.2577 ms
Min	1 ms	2 ms
Max	4 ms	3 ms

sent and received requests over a duration of 120 minutes.

It can be seen that in the first scenario, all synchronization requests were received and processed by the only time server in the setup. In the second scenario, with two time servers, the overall synchronization traffic was ca. 25.5 % higher. The main reason for this increase is the additional traffic that is caused by the synchronization between the two servers. However, the maximum synchronization load per server is reduced by 25 %, as client 2 and 4 send their SYN requests to the second time server.

For the feasibility of our approach, the accurate time synchronization between the time servers is vital. Large time differences would lead to concurrent network access and hence, lead to the same problems that can be observed for uncontrolled communication over common Ethernet. To show the accuracy of the refined time synchronization mechanism described in Section V, we measured the send times of the SYN requests on the additional server and the receive times on the initial server. Furthermore, the observed latencies for the synchronization requests were measured by the additional server. The measurement was performed for an experiment duration of 20 hours and 2,400 synchronization periods, respectively. Fig. 8 shows the obtained results. Here, the x-axis shows the overall time in hours and the y-axis shows a relative time in milliseconds. The light blue dots show the observed latency, while the dark blue dots represent the time difference between the transmission on the additional server side and the reception on the initial server side. Table II additionally summarizes the average time difference and latency along with their standard deviation and median.

It can be seen, that the time difference is nearly equal to the observed latency. Furthermore, the standard deviation for both, time difference and latency, are very small. That implies that they only experience small fluctuations. It can be concluded,

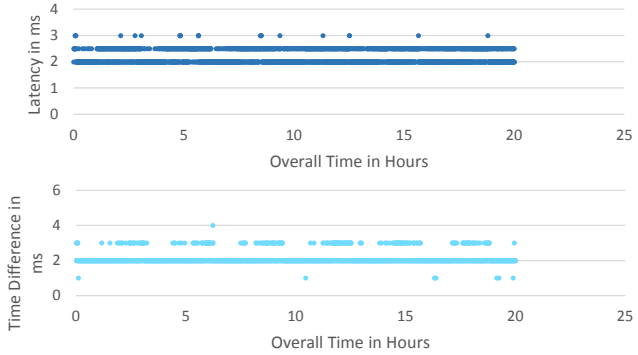

Fig. 8. Time difference between transmission and receipt of SYN requests and observed latency

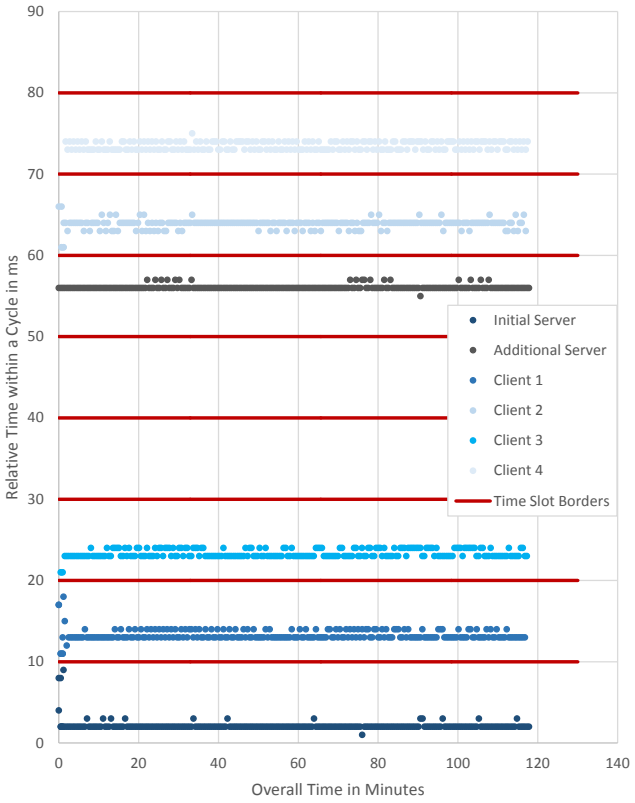

Fig. 9. Relative receive time of all messages within a cycle measured by the two time servers

that the time difference between transmission and reception is almost entirely caused by the transition through the network instead of clock drift between the two servers. Hence, the refined time synchronization provides a sufficient accuracy.

In our final experiment, we measured the receive times of all messages on both of the servers. In this way, the overall performance of the system regarding the time slot compliance of the communication can be assessed. Fig. 9 shows the experimental results. Here, the x-axis shows the overall time in minutes and the y-axis represents the relative time within a cycle in milliseconds. The red lines mark the time slot borders within the cycle.

It can be seen that all devices only communicate in their assigned time slots. Furthermore, it can be seen that the clients are equally distributed over the areas of responsibility of

both of the time servers. The experimental results emphasize the practical feasibility, synchronization performance, and the distribution benefits in terms of synchronization overhead reduction and robustness of our approach.

VII. Conclusion

Real-time communication is an important issue for future IoT and industrial applications. In our previous work, we have already presented a real-time extension for the CoAP standard, that allows TDMA-based deterministic communication over common Ethernet. The presented mechanisms are purely software-based and omit the need of non-standard conform protocol adaptations on the lower layers. In this paper, we have presented a new standard-compliant approach for a distributed CoAP time server as amendment to the previously proposed real-time extension for CoAP. Our approach enables an accurate time synchronization among CoAP nodes, a fair distribution of the synchronization load among multiple time servers, and a higher robustness towards node failures. The proposed mechanisms were integrated into the platform-independent Java-based jCoAP communication stack and evaluated in a real-world test bed with up to six devices. The experimental results have clearly shown the feasibility and benefits of our approach. The synchronization load of a single server could be significantly reduced. Furthermore, the refined time synchronization was proven to provide a sufficient accuracy. In a final experiment, the overall performance of the system was demonstrated. It has been shown, that our approach enables a purely software-based network access control that enables deterministic communication over common Ethernet without a central instance. In our future work, we will compare our approach to the P2P-based CoHaRT [23]. Both systems will be evaluated in a real-world test bed with up to forty nodes and in a larger simulation environment.

Acknowledgment

This work was partially financed by the German Research Foundation (DFG) within the graduate school Multimodal Smart Appliance Ensembles for Mobile Applications (MuSAMA, GRK 1424).

References

[1] P. C. Evans and M. Annunziata, "Industrial Internet: Pushing the Boundaries of Minds and Machines," in *General Electric Tech. Report*, November 2012.

[2] P. Danielis, J. Skodzik, V. Altmann, E. B. Schweissguth, F. Golatowsk, D. Timmermann, and J. Schacht, "Survey on real-time communication via ethernet in industrial automation environments," in *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'2014)*, Barcelona, Spain, September 2014, pp. 1–8.

[3] L. Atzoria, A. Ierab, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54 Issue 15, pp. 2787–2805, October 2010.

[4] T. Nixon, A. Regnier, D. Driscoll, and A. Mensch, *Devices Profile for Web Services Version 1.1*, Online, OASIS Std.

[5] R. Kyusakov, P. P. Pereira, J. Eliasson, and J. Delsing, "Exip: A framework for embedded web development," *ACM Transactions on the Web (TWEB)*, vol. Volume 8 Issue 4, no. 23, October 2014.

[6] C. Lerche, N. Laum, G. Moritz, Z. Elmar, F. Golatowski, and D. Timmermann, "Implementing powerful web services for highly resource-constrained devices," in *Pervasive Computing and Communication Workshops (PerCom Workshops), IEEE International Conference*, Seattle, WA, USA, March 2011.

[7] M. Kovatsch, M. Lanter, and Z. Shelby, "Californium: Scalable cloud services for the internet of things with coap," in *Proceedings of the 4th International Conference on the Internet of Things (IoT 2014)*, 2014.

[8] B. Konieczek, M. Rethfeldt, F. Golatowski, and D. Timmermann, "Real-time communication for the internet of things using jcoap," in *18th IEEE Symposium on Real-Time Computing (ISORC)*, Auckland, New Zealand, April 2015.

[9] M. Kovatsch, "A low-power coap for contiki," in *8th IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)*, Valencia, Italy, October 2011, pp. 855–860.

[10] ——, "Coap for the web of things: from tiny resource-constrained devices to the web browser," in *UbiComp '13 Adjunct, Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, Zurich, Switzerland, September 2013, pp. 1495–1504.

[11] O. Bergmann. libcoap: C-implementation of coap. [Online]. Available: http:// libcoap.sourceforge.net

[12] G. Moritz, F. Golatowski, and D. Timmermann, "A lightweight soap over coap transport binding for resource constraint networks," in *2011 IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS)*. IEEE, 2011, pp. 861–866.

[13] 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.

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

[15] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "Evaluation of EtherCAT distributed clock performance," *Industrial Informatics, IEEE Transactions on*, vol. 8, no. 1, pp. 20–29, 2012.

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

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

[18] T. Hu, P. Li, C. Zhang, and R. Liu, "Design and application of a real-time industrial Ethernet protocol under Linux using RTAI," *International Journal of Computer Integrated Manufacturing*, vol. 26, no. 5, pp. 429–439, 2013.

[19] R. Santos, M. Behnam, T. Nolte, P. Pedreiras, and L. Almeida, "Multi-level hierarchical scheduling in ethernet switches," in *Proceedings of the ninth ACM international conference on Embedded software*. ACM, 2011, pp. 185–194.

[20] M. Ashjaei, P. Pedreiras, M. Behnam, R. J. Bril, L. Almeida, and T. Nolte, "Response time analysis of multi-hop HaRTES ethernet switch networks," in *Factory Communication Systems (WFCS), 2014 10th IEEE Workshop on*. IEEE, 2014, pp. 1–10.

[21] K. W. Schmidt and E. G. Schmidt, "Distributed real-time protocols for industrial control systems: Framework and examples," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 10, pp. 1856–1866, 2012.

[22] 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.

[23] J. Skodzik et al. , "CoHaRT: A P2P-based deterministic transmission of large data amounts using CoAP," in *IEEE International Conference on Industrial Technology 2015 (IEEE ICIT)*, Sevilla, Spain, March 2015.

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

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

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

[27] D. Mills et al., *RFC 5905: Network Time Protocol Version 4: Protocol and Algorithms Specification*, online, IETF Std.