An Optimized WS-Eventing for Large-Scale Networks

Jan Skodzik, Vlado Altmann, Peter Danielis, Moritz Koal, Dirk Timmermann
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany, Tel./Fax: +49 381 498-7284 / -1187251
Email: jan.skodzik@uni-rostock.de

***Abstract*—Web Services are becoming more and more relevant also in the domain of embedded devices as they are becoming an important aspect of the Internet of Things. Embedded devices, especially in the field of automation, require real-time behavior. The Devices Profile for Web Services defines WS-Standards for embedded systems. The WS-Eventing is one of this standards within DPWS and enables the distributions of events. However, WS-Eventing has a scalability problem as in ad-hoc networks the notification is transmitted sequential by an event source. Under these circumstances, it is not suitable for large-scale networks with a high amount of devices. Therefore, a new approach is presented to solve this issue by acquiring helping devices. The communication is only based on unicast IP communication and thus is easy to integrate into automation infrastructures and could be established over several subnets. This approach has been investigated with an experimental setup and shows the high scalability compared to the original notification mechanism. Additionally, the real-time capability is given due to the chosen platform and real-time operating system. Thus, the new approach enables the idea of Internet of Things in automation scenarios.**

I. Introduction

Web Services are an established way to offer services in the modern connected world in an effective way due to the decoupling of devices. As an enterprise solution or external API, they are already established by many companies like Amazon or Google [1] [2].

With the idea of the Internet of Things (IoT), the number of devices in the network increases enormously. IoT also is an important factor in the industry and therefore faces new requirements like the need for real-time behavior [3]. This also includes devices with limited resources, e.g., embedded systems. With the 2006 initially released Devices Profile for Web Services (DPWS) by Microsoft, these devices are also able to communicate via a common standard and are easy to integrate into the existing networks. DPWS, which became an OASIS standard in 2009, is a compound of different standards to be able to adapt the application to any needs of the costumers [4]. An overview of the standards is given in Figure 1.

Typical WS standards are WS-Discovery and WS-Eventing, which are the essential standards to perform operations [6], [7].

WS-Discorvery is used to find services in an network and is realized in two different modes - the managed-mode and the ad-hoc mode. The ad-hoc mode is preferred as it has no central instance, which represents a single point of failure

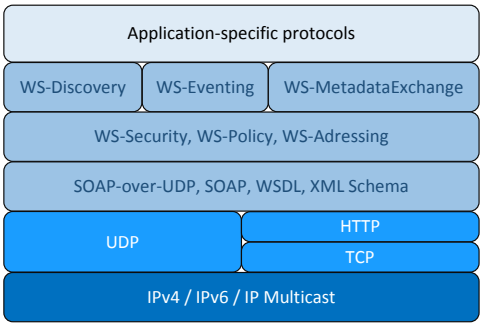

Fig. 1. DPWS stack [5]

(SPoF). Every device acts autonomously while entering the network. This also has a disadvantage related to the scalability of searching for new devices. If a device searches for all services of one type by a multicast request it gets all responses within a specified time. Thus, this could lead to an immense overload of the searching devices and therefore leads to the discarding of responses. An approach is presented in [8] to solve this issue in an ad-hoc network by using a DHT-based discovery algorithm. This algorithm has a very good scalability as it actively controls the responses of a search discovery command and avoids the usage of multicast. The DHT-based algorithm has been realized using Kad (an implementation of Kademlia) as software basis. Furthermore, after finding resources, the idea is to use the new discovery mechanism for services. One of further applications could be an eventing system, which would require the WS-Discovery to find event sources. An eventing system based on a publish/subscribe mechanisms offers a good opportunity to enable asynchronous communication. Additionally, eventing allows a high grade of decoupling of the devices, which could increase the scalability and also enables the independent communication between different devices [9].

A publish/subscribe system is also standardized in the WS-Eventing and has a big disadvantage in terms of scalability as only one device has to send the event notification to all subscribed nodes. This shows a bad scalability and is not suited for real-time scenarios as the event notifications will be sent much later with an increasing number of devices. In this paper, a new approach is presented to solve this issue and to create an

eventing system, which is suitable for large-scale scenarios as it has a better scalability than the original eventing algorithm described in the WS-Eventing standard. Briefly summarized the main contributions of this paper are the following:

- An approach for an optimized eventing system.
- A prototype implementation of the suggested approach.
- Measurement results for experimental setup.
- Comparison with the standard WS-Eventing.

The remainder of this paper is organized as follows: Section II contains a comparison of the proposed approach with related work. Section III gives a short overview of the WS-Eventing standard. Different optimizations including a new scalable notification algorithm are presented in Section IV.

In Section V, a prototype is presented with measurements results from a prototype setup realizing the original notification and optimized notification algorithm. The paper concludes in Section VI.

II. Related Work

There are many approaches available for realizing a publish/subscribe system for eventing [9]. We are focusing on the well supported and standardized WS-Eventing and list relevant work, which focuses on improving WS-Eventing especially in terms of scalability and reliability. These aspects are essential for the deployment in large-scale networks with real-time requirements like automation scenarios.

In [10], WS-Eventing and Java Message Service (JMS) are presented as a possibility for a notification system in automation environments. The Java-based JMS introduces a more complex implementation and overhead to the system. Additionally, the hierarchical/centralized structures of the JMS-based publish/subscribe system contradicts the idea of an ad-hoc network as a required message broker or application server results in a SPoF for the system. WS-Eventing as an alternative is criticized due to its low scalabilty as one event source must notify many event sinks in the worst case. Also, it is mentioned that WS-Eventing as part of DPWS allows for an easy integration due to its standardization and offers a good insight due to its lower complexity. Additionally, no indications of device performance in this environment are given to handle the suggested approach. The authors also argue against the usage of shared communication channels but prefer proprietary protocols on the lower device levels. However, Industrial Ethernet uses a shared medium and to use proprietary protocols means to waive the total horizontal and vertical integration of an automation system from the field level up to company level [11].

The issue of a bad scalability of WS-Eventing is also considered in [12]. As a solution, a UDP-based multicast is presented. However, this results in the disadvantage that also router and switches must support multicasts, which could be hard to guarantee over a big IP-based network. Additionally, to the account of reliability of the notification, responses are renounced as the event source will be overstrained if too many event sinks answered. Unfortunately, prototypical performance results are not presented. We are not using any multicast as our approach is based on simple unicast messages. Therefore, the solution can be used unlimited as long as IP and UDP are supported. Additionally, we support a higher reliability as every notification will be confirmed by a response without overstraining the event source.

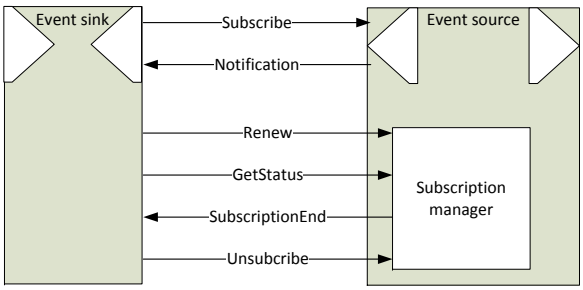

Fig. 2. WS-Eventing operations

In [13] and [14], Huang et al. present WS-Messenger to achieve a better scalability of the notification distribution. In terms of performance, they can beat a original WS-Notification system but their system bases on a centralized infrastructure by using a JMS broker. Performance measurement also only show linear scalability, which is not enough for future challenges with a high amount of devices. Also, there is no statement about the usage in scenarios with embedded devices not to mention environments with real-time requirements.

In [15], optimizations based on the WS-Messenger project are proposed to directly increase the scalability. Event consumers are requested to assume jobs with message queues containing contacts to be notified. A management of the distribution of the jobs is necessary, which results in further overhead. However, the approach of parallelization still relies on a system containing SPoFs like the deployed message broker, which is a disadvantage in ad-hoc networks.

This paper represents a generalized approach, which gets along without any centralized managing or allocation principles. The usage of centralized devices like a message broker or application server is not necessary. In summary, none of the presented solutions, in contrast to our approach, is intended to be deployed in large scale networks with hard real-time constraints as many aspect like the target platform or media access are not resolved. Our developed prototype achieves a high performance and already meets requirements for real-time applications with delivery times in the range of one millisecond.

III. Basics

First of all, before we can use WS-Eventing it is necessary to find an event source in the network. This is done via WS-Discovery. We use an optimized and much more scalable version of the WS-Eventing, which is based on a DHT-based discovery algorithm. The enhanced version of the WS-Discovery suggests to use Kad as Peer-to-Peer network to be able to find all services in a network only via UDP unicast communication in ad-hoc networks instead of initially used TCP transmissions. TCP and multicasts are avoided and therefore suitable for automation scenarios with real-time requirements.

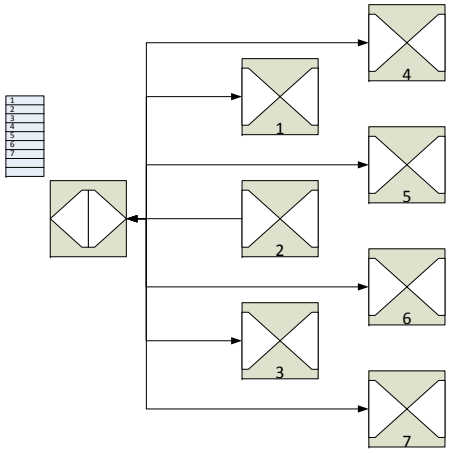

Fig. 3. WS-Eventing: Normal publishing of events

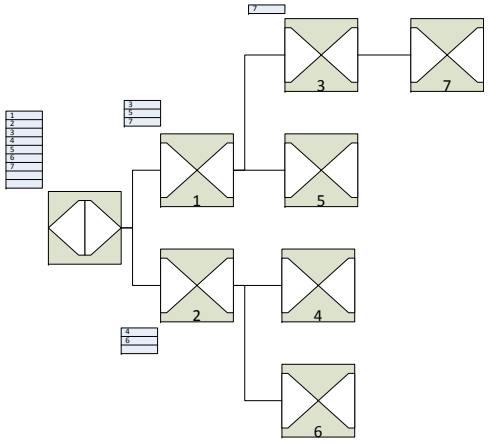

Fig. 4. WS-Eventing: Optimized publishing of events

This framework will be used to realize a highly scalable WS-Eventing after the event source is found. There are several operations to interact with the event source, which are depicted in Figure 2.

To receive notifications from an event source, a consumer must become an event sink by performing a subscription at the event source. The event source comprises a subscription manager object, which now stores the new event sink and manages it. To manage the subscription state, an event sink can renew its subscription, which expires after a time to live value. Furthermore, it can ask the subscription manager for its actual subscription status by performing a GetStatus operation. If the event sink is no longer interested in the notification of the event source it performs an Unsubscribe operation. Alternatively, if a subscription terminates unexpectedly the subscription manager can send a SubscriptionEnd message to affected devices. The mentioned messages are non-timing critical as they are management messages.

If an event must be published to the subscribed event sinks the event sources send the notification sequentially (see Figure 3). As this happens sequentially, it does only scale linearly in the best case. Thus, if there is a high number of subscribed event sinks the event source will be overburdened and the functionality cannot be guaranteed. The notification itself is time critical as the information in a real-time scenario must be delivered within a specified time. However, a time interval can be defined for real-time scenarios as it has linear scalability, but with an increasing number of devices these time intervals would increase significantly. Thus, the normal notification mechanism is not suitable for real-time applications. Therefore, an alternative approach has to be investigated, which shows both high scalability and high reliability.

IV. Optimizations

In this chapter, three optimization aspects are presented. The first is the main aspect of optimizing the forwarding of the notification by applying a parallel distribution with the help of event sinks. The second aspect is the usage of compressing XML with its overhead to achieve a more effective data transmission. Finally, the opportunities of modifying the subscription list at the subscription manager is discussed to adjust it in an optimal way for the new event publishing algorithm.

A. Paralleled Notification Mechanism

To solve the problem of low scalability, a new approach to distribute events is presented. Instead of using only one event source, event sinks will be acquired to help distributing an event. The chosen devices for helping are usually event sinks as well, which are subscribed for the same event. To stay compatible with the WS-Eventing standard, XML is used to describe the payload of the messages. The subscribe and unsubscribe method is the same like in the normal WS-Eventing process. When a device, which is the event source, throws an event it will have a list of event sinks, which are subscribed for it. It will now contact the first node and will send the first half of the list with further contacts. In a second step, it will contact the second event sink in the list and will send the second half of the list. After these two steps have been performed, the event source will go idle after it has received two responses from the first two nodes. The responses are ensuring the reliability as the messages are only sent via UDP. The two contacted nodes will continue this procedure for their list. An example is show in Figure 4. This new structure is denoted as notification tree.

The distribution of a notification can be speed up non-linearly with an logarithmic speed of $RTT * log(N)$, whereby RTT is the round trip time between two participants and N is the number of event sinks. As a theoretical result, with every doubling of the event sinks only one further step is needed due to the notification tree structure.

B. Usage of Efficient XML Interchange

As XML is human-readable, it is easy to handle for the user but it has the disadvantage of high overhead. The presented solution needs a reduction of the data amount to be highly effective in the data exchange. Therefore, we decided to use

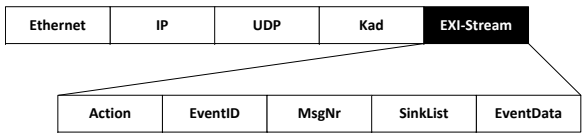

Fig. 5. Notification packet

Efficient XML Interchange (EXI) to keep the transmitted data at a minimum [16]. EXI is a binary coded XML format and thus a high compression rate of the data can be reached. A previous work shows that by using EXI, the enormous data overhead can be reduced and the effectiveness of the data transmission can be increased [17].

As we use Kad as background system to realize the DHT-based discovery [8], the WS-Eventing EXI coded data are integrated directly into the Kad packets. All Kad packets are UDP packets and therefore suitable for real-time applications. The composition of an example notification frame is depicted in Figure 5.

The EXI stream as payload of the Kad packet consists of different elements to realize WS-Eventing, which are described in Table I.

ELEMENT	DESCRIPTION
Action	Contains the message type
EventID	Enables unique identification of events
ReplyTo	IP address of un/subscription confirmations
MsgNr	Number of the message
RelatesTo	Number of the message related to
Sink-IP	IP address of the even sink in subscription message
Subscription-IP	Instance of the subriction manager
Subscription-ID	ID of the subscription
IP-List	Contains Event addresses for distribution
EventData	Payload data like sensor values

TABLE I
EXI ELEMENTS

This composition enables to perform WS-Eventing in highly performance constrained environments, e.g., embedded devices in automation scenarios. We used an event ID to separate between different events offered in the system. Additionally, the question of processing/parsing the EXI-data must be evaluated, to determine the dependency when varying the transmitted data. This has been evaluated in [18] for hard real-time conditions as a software or hardware/software solution.

C. Sorting the Event Sinks

By using the internal list of the subscription manager, it is also possible to have an influence on the distribution of the notifications. The first strategy could be to sort the list in such a way that timing critical devices, which require the notification as fast as possible, receive the notifications first. Alternatively, a more reliable focused second strategy would be to sort the list in a way that depend on the reliability of devices, which will forward the notifications. If an event sink fails the previous device must ask the next node in the event sink list of the failed event sink. The newly asked event sink is now responsible for the notification of the other event sinks of the list. In this case, devices with a high reliability and therefore high robustness should receive the notifications first. This is important as the probability of a device failing is very low at the start of the notification process and a possible delay because of timeouts due to missing responses can be reduced. If a device fails later in the notification chain, less nodes are effected by a worse notification performance. Therefore, the probability that the event source is strained again is low. Therefore, the second strategy should be chosen to relieve the event source.

V. IMPLEMENTATION AND EVALUATION

As target platform, the ZedBoard was used and acts as an exemplary embedded device for our investigations. ZedBoard bases on a Zynq Z-7020 chip, which comprises a Field Programmable Gate Array (FPGA) and an ARM Cortex A9 CPU. The ARM CPU runs at 667 MHz [19]. FreeRTOS is used as the real-time OS and therefore allows the creation of real-time applications [20]. The application comprises the Kad Client, DHT-based discovery algorithm, and the WS-Eventing protocol with its optimizations. The prototype setup consists of 19 devices connected via a 1 GBits connection over a switch. First, the event source boots on the first board and waits for subscription messages. After an event sink has booted, it subscribes automatically to the event source. These steps are not time-critical. Finally, all 18 event sinks have subscribed to the events source and wait for notifications. Messages via a serial COM port are only displayed after measurements have been performed as printing messages would adulterate the result tremendously. Via a debugging PC, it is possible to send triggers to the event source, which now starts to send the notification to the nodes stored in the subscription manager. This is the moment, in which the first time stamp $t_{trigger}$ will be created. The time resolution is set to 1 μs, which is the highest supported resolution of FreeRTOS on the ZedBoard. Now, it depends on the publishing algorithm, which is chosen by the triggering PC, how the publishing process is performed.

Two scenarios will be evaluated for the normal notification and the optimized notification algorithm. The difference is the grade of reliability. In the first case, no responses from the event sinks are sent to event source. In fact, if this scenario is chosen a better performance is expected as the event source does not need to process response packets to the account of the reliability. In the second scenario, responses from the event sinks are expected after receiving a notification. This is more suitable for the intended automation scenarios, which require a high grade of reliability. For completeness, both scenarios will be investigated and compared.

A. Scenario one: No Responses

After the trigger is received by the event source, it starts sending its notifications to the subscribed event sinks. However, this is done sequentially. Only the last event sink will send a response message to event source. The second time stamp t_{last} will be created at the moment the event source indicates the response message of the last event sink.

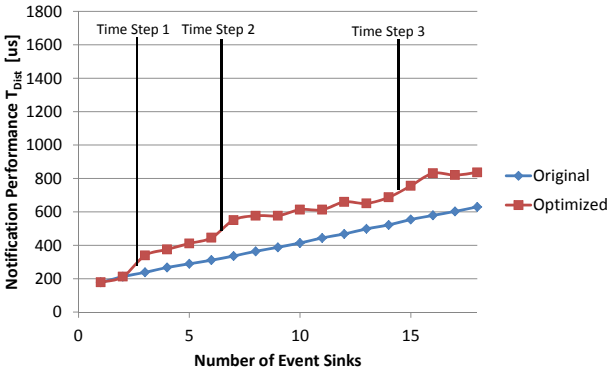

Fig. 6. Prototype results without responses

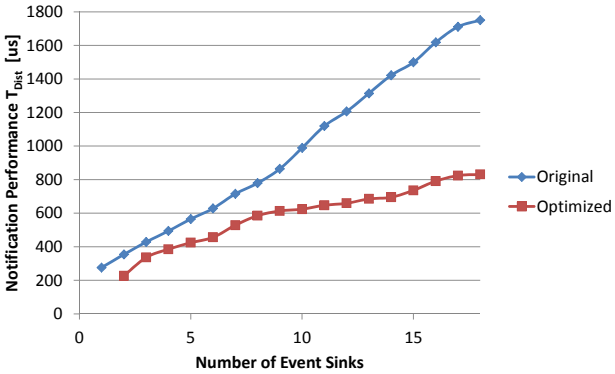

Fig. 8. Prototype results with responses

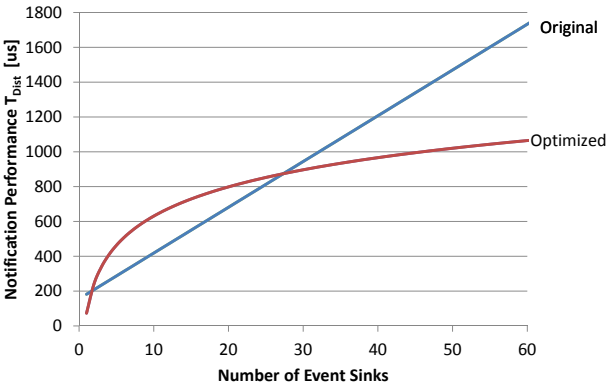

Fig. 7. Trend lines for notification algorithm without responses

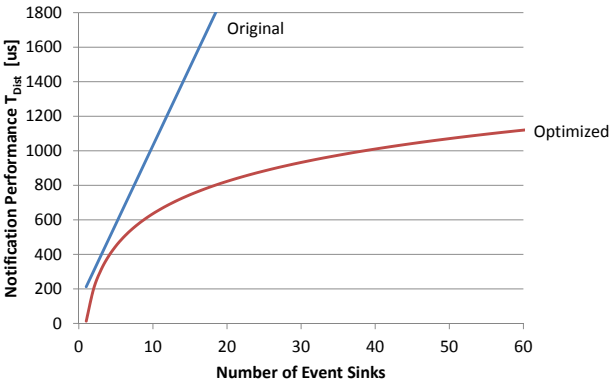

Fig. 9. Trend lines for notification algorithm with responses

The optimized notification performs similarly but it uses the helping event sinks to speed up the notification process. Also, the last node will send a response to the event source to create t_{last}. The last event sink is the sink in the deepest level of the notification tree at the most right side as this is the last notified event sink. The difference between $t_{trigger}$ and t_{last} gives the distribution performance of the notification T_{Dist}. The results of the prototype setup are depicted in Figure 6. In Figure 7 the trend lines of both are compared.

As apparent, a single node can be still better than using helping nodes. The coefficient of determination for a linear trend line of $R^2 = 0.9544$ compared to that of the logarithmic trend line equating to $R^2 = 0.9399$ is nearly identical for the optimized notification algorithm. However, the tendency is to be logarithmic when looking at the optimized notification algorithm. It is clearly visible that with every doubling of the nodes, a significant time step is visible. After the first two event sinks received the notification, there is a significant increased time need until the next event sink will answer. The next significant time step is visible after the next four event sinks have been notified, which will be the event sinks of the next tree level. The last significant time step is visible after the next eight event sinks have been notified.

Therefore with every level of the notification tree a big time step is visible, which happens after every doubling of the subscribed event sinks. The increased consumed time between two event sinks notified on the same level is much less. The time steps haven been marked in Figure 6 for clarification reasons. Contrary, the sequentially original notification is strictly linear ($R^2 = 0.9995$). Therefore, the optimized notification algorithm will be better at a higher number of nodes.

The new parallelized approach is not better than the original approach if only a few event sinks are present. Therefore, we need to compare the trend lines to determine the number of event sinks where the new approach is faster than the original. For the normal algorithm, the derived linear, and for the optimized a derived logarithmic trend line is chosen and the trend is depicted in Figure 7.

The intersection between the trend lines indicates the point where the performance of the new approach becomes better than the original and is determined for a number of event sinks equating to 28. Thus, if the number of devices (event sinks) is increasing also for real-time application in the automation scenarios the new approach could improve the notification but with less reliability. Both notification realizations (original and optimized) only need less than 1 ms to notify the event sinks in the prototype setup, which makes the system suitable for automation scenarios. Especially, the real-time capability is a significant feature of the presented realization.

B. Scenario two: With Responses

As we are only using UDP messages for our approach to ensure a deterministic behavior for real-time scenarios, the reliability has to be assured.

Therefore, we have investigated the results if every event sink has to send a response to the event source. In the original version, the event source serialized sends out the notifications and receives and processes the responses also serialized. If it receives the last response, the second time stamp t_{last} will be generated to determine T_{Dist} for completely distributing the notification, which will be displayed via serial communication interface on the debugger PC after the distribution is completely done.

For the optimized algorithm, only the last two nodes send their response directly to the event source so that we are able to determine T_{Dist}. The other nodes send their responses directly to their direct predecessor, which sends the notification to them. The results for T_{Dist} can be seen in Figure 8. The coefficient of determination for the original notification is $R^2 = 0.9929$ for a linear trend. Contrary, the optimized approach has a high coefficient of determination of $R^2 = 0.9785$ for the logarithmic trend (linear trend is $R^2 = 0.9557$).

For comprehension reasons, in Figure 9 the trend lines are depicted. Thus, the optimized approach scales much better than the original version, which was already indicated in the version without responses but only for higher number of event sinks. If responses from the event sinks are required, it is clearly visible that the optimized algorithm beats the original sequential notification mechanism from the start. Therefore, for reliability reasons the new approach could be a good alternative.

Compared to the result with no responses it is clearly visible that the additional responses from the events sinks nearly have no impact on the optimized notification mechanism. The needed time is still below 1 ms. Contrary, the responses have significant negative influence on the original notification mechanism as the event source has to process every response. The optimized approach is 52.51% better than the original notification and has a logarithmic tendency, which enables the usage of this approach for large scale networks.

VI. Conclusion and Future Work

In this paper, an optimized algorithm for distributing notification in WS-Eventing is presented. As the original sequential notification algorithm in ad-hoc networks would overburden the event source, the new algorithm uses event sinks to distribute the notification in parallel with a logarithmic speed in the network and therefore relieves the event source. Consequently, the new approach has high scalability and enables the system to inform more nodes within a specified time.

The presented prototype shows high performance and indicates that notification period in the range of some milliseconds could be achieved. Together with the mentioned DHT-based discovery algorithm, which has real-time capabilities as well, the system enables web services in automation scenarios. For future work, we want to investigate the usage of more event sources, which leads to the requirement of a controlled media access as concurrent traffic could lead to buffer overload of switch buffers and therefore to packet loss. A solution could be using the work presented in [21] as middle ware.

References

[1] Amazon. [Online]. Available: http://aws.amazon.com/

[2] Google. [Online]. Available: https://developers.google.com/maps/documentation/business/webservices/

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

[4] OASIS, "Devices profile for web services version 1.1." [Online]. Available: http://docs.oasis-open.org/ws-dd/dpws/wsdd-dpws-1.1-spec.html

[5] E. Zeeb, G. Moritz, D. Timmermann, and F. Golatowski, "Ws4d: Toolkits for networked embedded systems based on the devices profile for web services," in *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, Sept 2010, pp. 1–8.

[6] OASIS, "Web services dynamic discovery (ws-discovery) version 1.1," 2009. [Online]. Available: http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.html

[7] D. Davis, A. Malhotra, K. Warr, and W. Chou, "Web services eventing (ws-eventing)," W3C Recommendation, 2011. [Online]. Available: http://www.w3.org/TR/ws-eventing/

[8] V. Altmann, J. Skodzik, P. Danielis, J. Mueller, F. Golatowski, and D. Timmermann, "A dht-based scalable approach for device and service discovery (accepted)," in *The 12th IEEE International Conference on Embedded and Ubiquitous Computing*, 2014.

[9] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, Jun. 2003. [Online]. Available: http://doi.acm.org/10.1145/857076.857078

[10] V. Trifa, D. Guinard, and M. Koehler, "Messaging methods in a service-oriented architecture for industrial automation systems," in *Networked Sensing Systems, 2008. INSS 2008. 5th International Conference on*, June 2008, pp. 35–38.

[11] T. Sauter, "Integration aspects in automation - a technology survey," in *10th IEEE Conference on Emerging Technologies and Factory Automation, 2005. ETFA 2005.*, vol. 2, 2005, pp. 255–263.

[12] D. Gregorczyk, "Ws-eventing soap-over-udp multicast extension," in *Web Services (ICWS), 2011 IEEE International Conference on*, July 2011, pp. 660–665.

[13] Y. Huang and D. Gannon, "A comparative study of web services-based event notification specifications," in *Parallel Processing Workshops, 2006. ICPP 2006 Workshops. 2006 International Conference on*, 2006, pp. 8 pp.–14.

[14] Y. Huang, A. Slominski, C. Herath, and D. Gannon, "Ws-messenger: a web services-based messaging system for service-oriented grid computing," in *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, vol. 1, May 2006, pp. 8 pp.–173.

[15] R. Jayasinghe, D. Gamage, and S. Perera, "Towards improved data dissemination of publish-subscribe systems," in *Web Services (ICWS), 2010 IEEE International Conference on*, July 2010, pp. 520–525.

[16] J. Schneider, T. Kamiya, D. Peintner, and R. Kyusakov, "Efficient xml interchange (exi) format 1.0 (second edition)," W3C Recommendation, 2014. [Online]. Available: http://www.w3.org/TR/2014/REC-exi-20140211/

[17] V. Altmann, J. Skodzik, F. Golatowski, and D. Timmermann, "Investigation of the use of embedded web services in smart metering applications," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, Oct 2012, pp. 6172–6177.

[18] V. Altmann, J. Skodzik, P. Danielis, F. Golatowski, and D. Timmermann, "Real-time capable hardware-based parser for efficient xml interchange," in *9th IEEE/IET International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP14)*, July 2014.

[19] Avnet. [Online]. Available: http://www.zedboard.org/

[20] Real Time Engineers Ltd. [Online]. Available: http://www.freertos.org/

[21] 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.