A Kad Prototype for Time Synchronization in Real-Time Automation Scenarios

Jan Skodzik, Vlado Altmann, Peter Danielis, Arne Wall, 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;dirk.timmermann}@uni-rostock.de

Abstract

In this paper, a prototype to synchronize the Peer-to-Peer network Kad is presented. The approach bases on a deterministic algorithm, which is required for hard real-time applications, to synchronize the network in a decentralized manner. Existing industrial Ethernet solutions include and support real-time capable machine-to-machine communication in automation scenarios. However, the coordination of the communication and the synchronization to achieve hard real-time are managed by a central instance, leading to deficient resilience and scalability. Additionally, a centralized or hierarchical synchronization is required to enable the communication per se. The presented decentralized synchronization instead benefits from nodes helping to synchronize the network and bases on the decentralized structured Peer-to-Peer network Kad. However, the higher the number of helping nodes the higher is the time deviation on the nodes of the network, which contrary results in a higher time error. Therefore, a trade-off between synchronization performance and time error has to be determined to meet predefined constraints depending on the application scenario. Moreover, the individual clock drift of every device needs to be considered to define necessary re-synchronization intervals of the network. Theoretical values are compared to the measured values determined in experiments with 15 nodes. As a result of these experiments, the measured synchronization performance, which is the time needed to synchronize the complete network, confirms the theoretical worst case values. The result is a running prototype system, which is able to synchronize the network with a high performance in a decentralized manner.

1 Introduction

The development of real-time (RT) Ethernet is of high economical interest to replace conventional bus systems in manufacturing plants [1]. The usage of common Ethernet technology to interconnect automation devices offers significant benefits compared to established field bus systems. Industrial Ethernet (IE) solutions allow for a total vertical and horizontal integration of a automation system from the field level up to company level [2]. This enables the exchange of both usual office units information and time critical facility data [3]. The communication protocol and corresponding hardware are specified within the IEEE 802.3 standard, which results in low acquisition costs for this equipment. Furthermore, the high performance of Ethernet is beneficial compared to usual field bus systems, which are increasingly limiting the performance factor. However, with IE new problems arise, which have to be solved to be prepared for the future. As proposed by General Electric, the number of devices will increase enormously but also their computational power will increase, which leads to new future challenges like ensuring high scalability [4]. To enable a deterministic behavior of Ethernet, modifications in the protocols and/or hardware are necessary. The results are systems with centralized structures like SERCOS III, Ethernet Powerlink, EthercCAT or Profinet [5]. The mentioned IE solutions use a master/slave or client/server model with central components representing a single point of failure (SPoF) and bottleneck. If the central instance fails this leads to the total failing of system functionality and the loss of data. Another disadvantage is that IE solutions change or bypass standard protocols or require special hardware, which classifies them as proprietary systems. Moreover, the interoperability between the different IE solutions is not given as each solution is proprietary.

Contrary, Peer-to-Peer (P2P) networks solve the problem of limited scalability, low robustness, and lacking user friendly installation. Each peer provides server and client functionality, which results in a network with high resilience. Kad (an implementation of Kademlia) has been chosen as network to realize a self-organizing distributed hash table (DHT)-based P2P network. The solution is realized as an application without modifying the underlying protocol or hardware and do not require any central instance for management, maintenance, or observation.

To ensure a deterministic data exchange in environments with hard RT requirements, a time division multiplexed access (TDMA) mechanism has been used. As the main requirement for the functional system, all nodes must act on a common synchronized time base. An approach has been presented in [6] to synchronize a Kademlia network in a fully decentralized manner. By allowing a defined grade of parallelism in the communication, the time needed for synchronization is decreased while the time error increases. Therefore, a trade-off is investigated in this regard. Additionally, the drift of each device forces the system to re-

synchronize the network periodically. As a result, the consumed time for synchronization has to be kept minimal while considering the time error due to the controlled parallel communication.
This paper presents a fully functional prototype, which implements the presented approach and evaluates the measurements in experiments with up to 15 Kad nodes.
The remainder of this paper is organized as follows: Section 2 contains a comparison of the proposed approach with related work. Section 3 shortly presents the necessary steps to realize a Kad-based network synchronization. The implemented prototype to realize the synchronization is presented in Section 4. In Section 5, the measurements from the experimental set-up are presented and evaluated. The achieved results are compared with theoretical values in Section 6. In Section 7, two aspects are discussed to increase the synchronization performance before the paper concludes in Section 8.

2 Related Work

A synchronization approach called PariSync for P2P networks is presented in [7]. PariSync bases on Java and achieves a synchronization error of a few hundred milliseconds over the Internet. However, PariSync is suggested to be used in large networks with high churn. As the information exchange and media access is not controlled, this could lead to further errors during synchronizations due to parallel cross talk and switch buffering. The resulting error is not investigated. Also, it takes many seconds to establish a stable network, which is not suitable for an automation scenario with hard RT constraints and deterministic behavior as it also lacks in performance.
An alternative concept is presented in [8], where a system using a gossip-based approach is introduced. The focus is on assessing the impact of corrupted processes on the effectiveness of clock synchronization. However, there is no formal proof of the correctness of clock synchronization convergence. Additionally, several parameters have to be determined to achieve a high probability of information exchange. Therefore, using a gossip-based approach like in [8] is always a trade-off between scalability and reliability [9].
Consequently, we renounce a gossip-based approach as it is probabilistic and not suited for hard RT scenarios requiring a deterministic behavior of the network.
Some IE solutions use the Network Time Protocol (NTP) or the Precision Time Protocol (PTP) as two established protocols to synchronize devices in a network. They achieve an accuracy of several milliseconds in case of NTP [10] and about 100 microseconds for a single link in case of PTP [11]. Furthermore, PTP can achieve better performance than NTP as it uses special hardware to generate accurate time stamps, which increases the time resolution and decreases the time error.
Contrary, this paper presents a software-based solution, without any special hardware requirements. Furthermore, NTP and PTP base on a hierarchical approach, which contrasts with the idea of P2P-based networks such as Kad [12, 13].
The goal is to achieve a similar time resolution and time error like NTP and software-based PTP protocol. We present an effective trade-off between exclusive media access and time error increase by speeding up the synchronization process enabled by parallel communication of nodes. The presented approach does not require any central instance and offers a high performance synchronization of several thousand devices with a low time error. Measurements from the experimental set-up consisting of an own developed prototype show that the presented approach is competitive with existing established mechanisms.

3 Basics

Kad has been choosen as it offers the best performance in terms of search and store operations with $log_2(N)$ where N is the number of nodes in the decentralized structured Kad network [14]. A deterministic approach to synchronize the Kad network is presented in [6], which consists of six stages. For reasons of better comprehensibility, the six stages are shortly described here.

First stage: In the initial stage, the Kad network is created by activating the first Kad nodes. This includes the bootstrapping of new nodes and the maintenance of the network thereby filling the routing tables with contacts.
Second stage: In the second stage, the Kad network is configured in such a way that for each hash value at least one node is responsible. This can be realized by modifying the search tolerance at run-time, which is part of the concept.
Third stage: After the Kad network is configured, the first initial synchronization can be performed. Therefore, the first triggered (FT) node, which receives a self-defined external signal sends a IP broadcast through the network and has to wait for at least the time a packet travels the critical path in the network. The waiting time must be chosen in such a way that every node receives the broadcast message. Now, the FT node exclusively accesses the Ethernet medium. The synchronization is carried out by the Kademlia Discovery and Synchronization (KaDisSy) algorithm, which consists of two steps [6].
In the first step of the KaDisSy algorithm, the FT node tries to acquire helping triggered (HT) nodes to speed up the synchronization of the network. The number of nodes, which are acquired, can be adjusted dynamically by the parameter J. By incrementing J, the number of nodes synchronizing the network is doubled as every new HT node will also acquire a new HT nodes as long as J is not reached. Therefore, the total number of active nodes synchronizing the network is 2^J. The first step is similar to building up a binary tree with the time complexity of $O(log_2(\#HT)$. The second group synchronization step

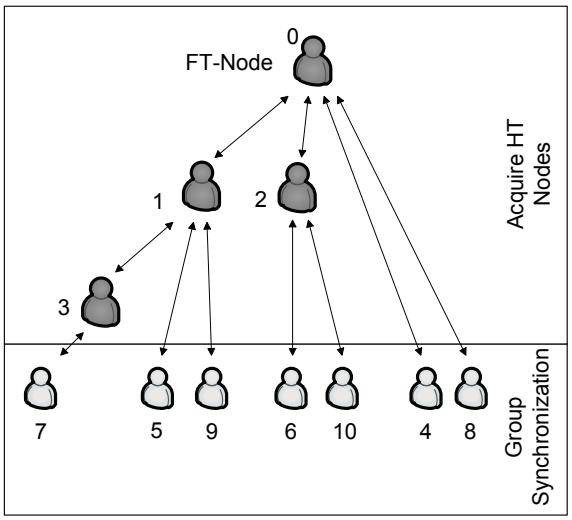

Figure 1: KaDisSy example for $J = 2$

serves to synchronize the rest of the network with the help of the optional acquired HT.
An example is depicted in Figure 1, where J is set to 2 and the number of nodes in the network is 10. The synchronization time $T_{SynComp}$ is given by Formula 1 and expresses the time needed to synchronize the whole network. T_{Syn} is the time needed to synchronize any node by any other node. First, it is necessary to synchronize and acquire the HT nodes (1). After that, the rest of the network is synchronized with all active synchronizing nodes (2).

$$T_{SynComp} = T_{Syn} * (\overbrace{J}^{(1)} + \overbrace{\frac{N}{2^J}}^{(2)} - 1) \quad (1)$$

$T_{SynComp}$ depends on the number of nodes N and J as an indicator for the number of HT nodes.
Fourth stage: In this stage, the application on top of the presented approach is performed. This could be data or information exchange, which require hard RT requirements. Furthermore, the presented approach acts like a middleware and is transparent to any application on top.
Fifth stage: Due to changes in the network, a maintenance procedure has to be performed, which is the fifth stage. Nodes, which have left the network, can be identified and removed from the routing tables to stabilize the network. Also, new nodes are allowed to enter the hard RT network. Additionally, this stage is intended to allow other applications to access the communication channel to perform non-deterministic data exchange. Thus, one combined integrated network can be realized.
Sixth stage: As every Kad node has a unique drift, we need to re-synchronize the network. The re-synchronization therefore must be performed periodically and can be described by the re-synchronization period T_{ReSyn}, which is defined by Formula 2. The maximum allowed time deviation error of any node from the common time base is given with $T_{MaxError}$. D_{Clk} is the drift of the clock of the devices and given as $\pm$ ppm.

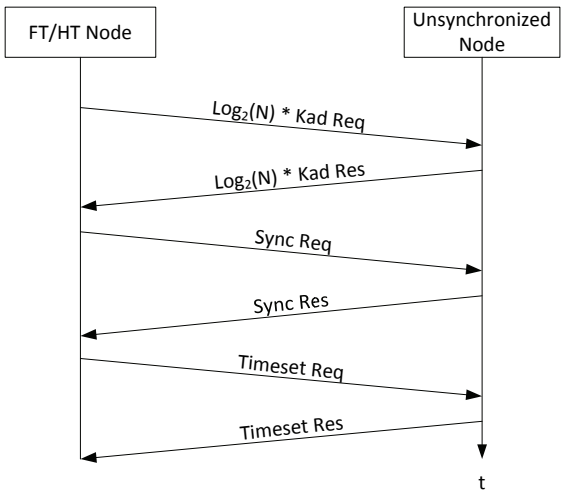

Figure 2: Synchronization between two nodes

T_{ReSyn} directly depends on $T_{SynComp}$ as the first node already drifted away from the correct clock value while the last node is synchronized.

$$T_{ReSyn} < \frac{T_{MaxError} - T_{SynError}}{2 * D_{Clk}} - T_{SynComp} \quad (2)$$

$T_{SynError}$ is the error, which is generated during the synchronization between two nodes and is described by Formula 3. $T_{SynError}$ consists of two parts. The first part (3) is the error due to the round trip time deviation ΔRTT between the synchronizing node and the node, which have to be synchronized. Also, the error propagation between multiple HT nodes acquiring new HT nodes have to be considered. The HT node acquiring a new HT node forwards its own synchronization time error to the next node. The second part (4) is the error due to allowed parallel communication. As we assume a switched Ethernet network, packets can be queued in the switches as other packets are preferred due to the FIFO principle. The additional time for a packet waiting for another packet in the switch FIFO is given with T_{Pkt}. T_{Pkt} is 610 ns and is defined by the biggest Kad packet possible in the network. Therefore, the error depends on the parameter J, which defines the amount of parallel communication traffic due to other HT nodes.

$$T_{SynError} = \overbrace{\Delta RTT * (J + 1)}^{(3)} + \overbrace{((2^J - 1) * T_{Pkt})}^{(4)} \quad (3)$$

In summary, stage four till six must be repeated periodically to ensure the proper functionality of the system.

4 The Running Prototype

The prototype, which performs the presented approach and therefore the KaDisSy algorithm has been realized as an OSI Layer 7 application and is therefore easy to migrate to new platforms. The chosen platform for the prototype is the Zedboard, which contains an ARM processor running with up to 667 MHz [15]. No additional proprietary hardware in terms of FPGA resources is used and therefore the system

should be easily portable to other platforms like, e.g., Raspberry Pis, which also support FreeRTOS [16]. At the moment, only one ARM Core is used to simplify the profiling step. FreeRTOS is the RT operating system and uses lwIP as TCP/IP stack [17] [18]. The implementation of Kademlia called Kad [19] has been ported for FreeRTOS and all threads have been prioritized. Therefore, it is possible to profile the application and measure times of the KaDisSy algorithm. During the execution of the KaDisSy algorithm, the nodes have to synchronize each other. This is realized directly within the Kad application (see Figure 2).

First, it is necessary to find the node in the Kad network. Therefore, we need $log_2(N)$ search steps in the worst case whereby N is the total number of Kad nodes in the network. The search is performed by Kad Req and Kad Res packets. If the node is found the FT or HT node requests the unsynchronized node to synchronize with it by a Sync Req packet. Additionally, the FT/HT node takes a first time stamp to determine the round trip time (RTT) between the two nodes. This is confirmed by a Sync Res packet and the FT/HT takes the second time stamp. Now, it is possible to calculate the RTT value. The FT/HT node now send the new time value, which is the actual time plus half of the RTT value, within the Timeset Req. The unsynchronized node is synchronized and confirms this with a Timeset Res packet. This process is always performed if a synchronization between two nodes takes place during the execution of the KaDisSy algorithm.

5 Experimental Measurements

The experimental measurements have been realized with a set of 15 Zedboards connected by a 1 GBit/s switched Ethernet network. Every Zedboards runs the same modified Kad client software. Via a host PC, a trigger and a value for J is send to one of the Zedboards and is processed as an external trigger to initiate the synchronization process. The node, which receives the trigger, becomes the FT node, starts the synchronization, and creates a time stamp $Stamp_{Start}$ with a resolution of 1 μs. First, the FT node acquires optional HT nodes depending on J and then synchronizes the rest of the network.

In the experimental set-up, all nodes, which are synchronized, send their Timeset packet to the FT node. Thereby, the FT node is able to identify if all other 14 nodes are synchronized and creates a second time stamp $Stamp_{Finish}$. Therefore, $T_{SynComp}$ is defined as the difference between $Stamp_{Start}$ and $Stamp_{Finish}$.

Synchronization performance: The time $T_{SynComp}$ for a different number of nodes N and different J is depicted in Figure 3.

As apparent, for $J = 0$ only one node synchronizes the network. The time needed to synchronize is linearly dependent on the number of nodes, which have to be synchronized. With increasing J, $T_{SynComp}$ can be decreased significantly.

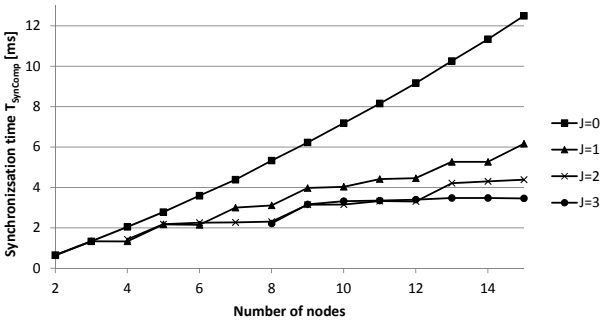

Figure 3: Synchronization performance for different J values and number of nodes

It is also visible that with the highest possible J = 3 $T_{SynComp}$ is less than 3.5 ms for 15 nodes, which is 72.30 % better than synchronizing with one FT node.

In Figure 4, the parameter J has been varied and the number of nodes N is constantly set to 15. Consequently, it can be seen that $T_{SynComp}$ is proportional to $1/log_2(2^J)$. These measured values prove the values from the theoretical examination of [6] and are even slightly better. It is also visible that $T_{SynComp}$ goes into saturation and therefore it is often not useful to chose the highest possible value for J. A high value for J also leads to a higher error propagation when acquiring HT nodes as with increasing J the error will be forwarded to the next HT node in the worst case. We can also reduce $T_{MaxError}$ to several 100 μs.

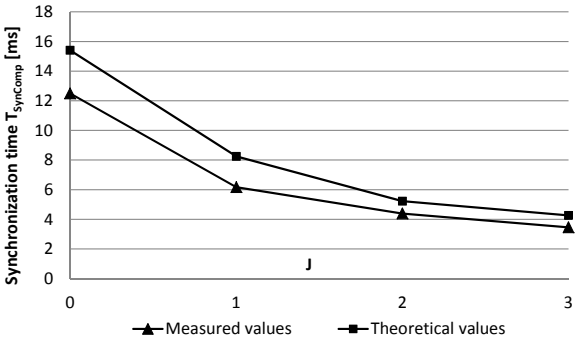

Figure 4: Synchronization performance for 15 nodes and different J values

Determination of RTT deviation: The re-synchronization also depends on the error during the synchronization of two nodes. As apparent from Formula 3, it depends directly on ΔRTT. Therefore, we have measured the RTT values in our experimental set-up under different circumstances. First, we measured the RTT value with only one 1 GBit/s switch. In a second set-up, a second switch was used. In both cases, for different nodes the RTT value was below 152 μs. The maximum difference between the fast and slowest RTT was 14 μs, which is denoted as ΔRTT. Furthermore, one additional switch only results in a difference ΔRTT of 1 μs.

Determination of the Clockdrift: The reason for the need of re-synchronization is the unique drift of each node caused by the oscillator. Therefore, it is necessary to determine the

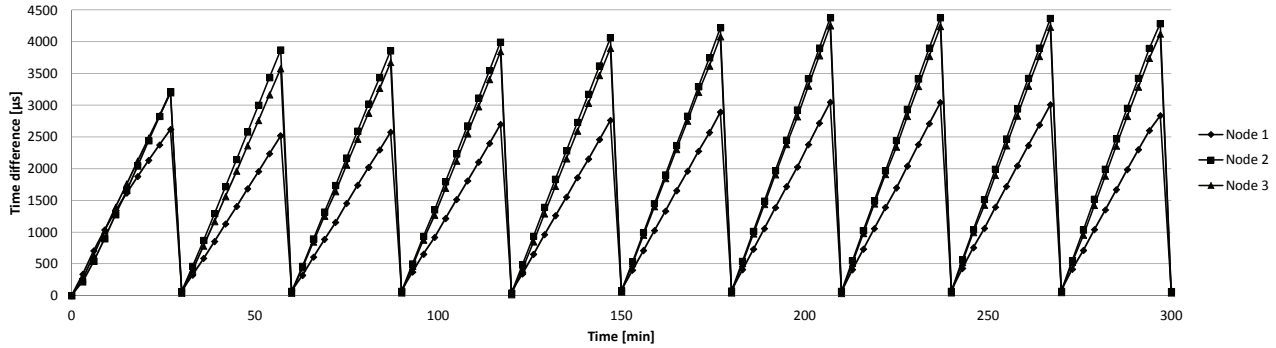

Figure 5: Four nodes performing the KaDisSy algorithm with J=0

drift of the nodes and to compare it with the given parameters by the manufactures. The Zedboard, which is the target platform, has a drift of 50 ppm given by manufacturer.

In Figure 5, a time trace of five hours of the KaDisSy algorithm has been measured. It comprises a bundle of four nodes as experimental set-up. Node 0 is the reference node and every 5 minutes the other three nodes send their actual time values to the reference node. As apparent, the other nodes drift away constantly due to their unique clock drift. The drifts are given as the absolute difference values. Each 30 minutes, the other nodes are synchronized again, which is the sixth stage from the approach. It turns out that the prototype is working properly and it is also possible to determine the unique drift of the nodes. That is why the re-synchronization period has been set to a high value to ensure convincing values for the measured drift.

Furthermore, we have measured a significant better ppm value of about 3 ppm compared to the specification of the Zedboard, which mention a ppm of 50. The measured drifts of the three nodes over five hours compared to reference Node 0 are given in Figure 6. The drift was always measured below a value of 3 ppm in our tests. Additionally, we can see a slightly changing drift during the five hours, which is caused by changing temperature oscillators.

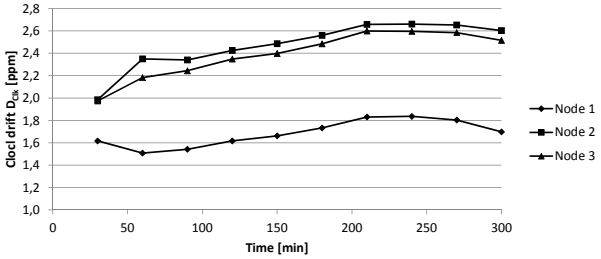

Figure 6: Measured drift D_{Clk} of three nodes

Finally our analysis shows that we can achieved a minimum $T_{MaxError}$ of 100 μs, which competitive with NTP and PTP as software based solution, which enable a accuracy of several hundred microseconds.

6 Comparison to Theoretical Values

The results of the prototype scenario are compared to the simulation results in this section.

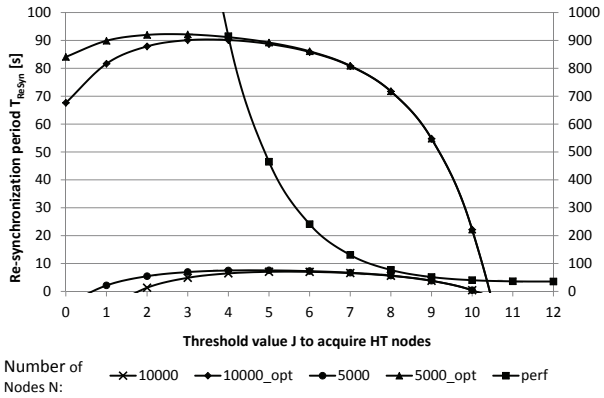

Figure 7: Comparison between theoretical and practical values

If the measured ΔRTT and clock drift are considered, in Figure 7 it is visible that they have a significant influence on T_{ReSync}. The maximum allowed time error is given by $T_{MaxError}$. It is set to 1 ms for all further considerations. The results slopes containing the name "_opt" considered the new values for ΔRTT and the clock drift, while the others are the theoretical results from the simulation in [6]. For 10,000 nodes, we must re-synchronize at least every 10 seconds. With the new measured values we can see under slope of 10,000_opt that we can re-synchronize every 90 seconds and thus we have 800 % more time for the application operations on top.

The highest value for T_{ReSync} can be interpreted at the optimum in terms of the highest time between two synchronizations. Certainly, it it also necessary to consider $T_{SynComp}$ and it can bee seen that with a higher J, $T_{SynComp}$ decreases exponentially. Thus, a trade-off between T_{ReSync} and $T_{SynComp}$ must be defined for the application and is therefore depends on the application.

7 Further Improvements

Some steps could be performed to gain a better performance. It is possible to exclude the determination of the RTT value during the exchange of the Kad Req and Kad Res packet. As a result, 33 % of the traffic could be saved and also the synchronization performance can be increased. For a better comprehension, this has not been applied to the prototype but is intended for the final version. Another very promising optimization step would be to use information from the underlay. To mitigate the so-called mismatch between the overlay (Kad) and underlay (physical Ethernet infrastructure). If there is a knowledge about the infrastructure parallel non-concurrent paths can simultaneously carry packet traffic without having a negative influence due to buffering switches. In automation scenarios, which are not highly dynamically in terms of the churn of devices in their infrastructure, there is a high optimization potential.

8 Conclusion and Future Work

A system to synchronize a network in a decentralized manner has been presented. Besides the concept, real measurements from a system consisting of 15 implemented prototypes have been presented. The influence and importance of several parameters have been determined and evaluated. The performance is competitive with existing centralized hierarchical concepts and is directly integratable in the application layer within the Kad implementation. Therefore, it is suited for easy migration to other platforms. Additionally, a comparison between theoretical and practical values show that supposed theoretical values can be achieved by a prototype and even exceed.

References

[1] Panel Building & System Integration, "Ethernet adoption in process automation to double by 2016," 2013. [Online]. Available: http://www.pbsionthenet.net/article/58823/Ethernet-adoption-in-process-automation-to-double-by-2016.aspx

[2] 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.

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

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

[5] M. Felser, "Real Time Ethernet: Standardization and implementations," in *Industrial Electronics (ISIE), 2010 IEEE International Symposium on*, 2010, pp. 3766–3771.

[6] J. Skodzik, P. Danielis, V. Altmann, and D. Timmermann, "Time Synchronization in the DHT-based P2P Network Kad for Real-Time Automation Scenarios," in *The Second IEEE WoWMoM IOT-SoS Workshop, IoT-SoS 2013*, 2013.

[7] P. Bertasi, M. Bonazza, N. Moretti, and E. Peserico, "Parisync: Clock synchronization in p2p networks," in *Precision Clock Synchronization for Measurement, Control and Communication, 2009. ISPCS 2009. International Symposium on*, 2009, pp. 1–6.

[8] R. Baldoni, M. Platania, L. Querzoni, and S. Scipioni, "A peer-to-peer filter-based algorithm for internal clock synchronization in presence of corrupted processes," in *Dependable Computing, 2008. PRDC '08. 14th IEEE Pacific Rim International Symposium on*, 2008, pp. 64–72.

[9] P. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulie, "Epidemic information dissemination in distributed systems," *Computer*, vol. 37, no. 5, pp. 60–67, 2004.

[10] U. Windl, D. Dalton, Hewlett-Packard, M. Martinec, J. S. Institute, and D. R. Worley. (2006) The ntp faq and howto. [Online]. Available: http://www.ntp.org/ntpfaq/NTP-s-algo.htm

[11] D. Seely, "AN-1728 IEEE 1588 Precision Time Protocol Time Synchronization Performance," Texas Instruments, Tech. Rep., 2007. [Online]. Available: http://www.ti.com/lit/an/snla098/snla098.pdf

[12] D. Mills, "Network Time Protocol (NTP)," RFC 1305, Internet Engineering Task Force, March 1992. [Online]. Available: http://www.ietf.org/rfc/rfc1305.txt

[13] K. Lee and J. Eidson, "IEEE-1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," in *In 34 th Annual Precise Time and Time Interval (PTTI) Meeting*, 2002, pp. 98–105.

[14] R. Steinmetz and K. Wehrle, Eds., *Peer-to-Peer Systems and Applications*, ser. Lecture Notes in Computer Science, vol. 3485. Springer, 2005.

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

[16] J. Walmsley, "A port of freertos to the raspberry pi." 2013. [Online]. Available: https://github.com/jameswalmsley/RaspberryPi-FreeRTOS

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

[18] Free Software Foundation, Inc. [Online]. Available: http://savannah.nongnu.org/projects/lwip/

[19] R. Brunner and E. Biersack, "A performance evaluation of the Kad-protocol," *Institut Eurecom, France*, 2006.