AIRCoN-Stack - Introducing Flexibility to Wireless Industrial Real-Time Applications

von Zengen, Georg and Garlichs, Keno and Wolf, Lars C.

Authors post-print published on 2020-07-22

Originally published in *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies*

Publisher version available at https://dl.acm.org/citation.cfm?id=3229578

DOI: 10.1145/3229574.3229578

Abstract:

Wireless networking is a key technology to enable smart production scenarios. It expands the design space for solutions in factory design tremendously. Currently, applications suitable for wireless connections are limited to monitoring purposes due to lacking reliability. This is especially critical when the network needs to adjust to new application needs, environmental conditions or network topologies. In such cases real-time conditions required for certain tasks might break. To overcome those limitations we present AIRCoN-stack. It was specifically designed to perform real-time operations while parts of the network are changing. It uses TDMA with a combined radio- and CPU-scheduler to keep the task execution jitter between nodes in the network as low as possible. We evaluate all components of our network stack in a real-world setup to prove its capabilities.

AIRCoN-Stack - Introducing Flexibility to Wireless Industrial Real-Time Applications

Georg von Zengen, Keno Garlichs and Lars C. Wolf
Institute of Operating Systems and Computer Networks
Technische Universität Braunschweig
Braunschweig, Germany
{vonzengen,garlichs,wolf}@ibr.cs.tu-bs.de

ABSTRACT

Wireless networking is a key technology to enable smart production scenarios. It expands the design space for solutions in factory design tremendously. Currently, applications suitable for wireless connections are limited to monitoring purposes due to lacking reliability. This is especially critical when the network needs to adjust to new application needs, environmental conditions or network topologies. In such cases real-time conditions required for certain tasks might break. To overcome those limitations we present AIRCoN-stack. It was specifically designed to perform real-time operations while parts of the network are changing. It uses TDMA with a combined radio- and CPU-scheduler to keep the task execution jitter between nodes in the network as low as possible. We evaluate all components of our network stack in a real-world setup to prove its capabilities.

CCS CONCEPTS

• **Networks → Cyber-physical networks**; *Network protocol design*; *Mobile networks*; • **Computer systems organization → *Real-time operating systems***;

ACM Reference Format:
Georg von Zengen, Keno Garlichs and Lars C. Wolf. 2018. AIRCoN-Stack - Introducing Flexibility to Wireless Industrial Real-Time Applications. In *NEAT'18: ACM SIGCOMM 2018 Workshop on Networking for Emerging Applications and Technologies , August 20, 2018, Budapest, Hungary.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3229574.3229578

1 INTRODUCTION

In future factories, machines will be organized in far more flexible compounds than today [9]. An example of this flexibility might be to re-order the tasks assigned to a certain machine or to add machines. As reorganizing the cables in a factory is rather expensive, wireless technologies are promising to lower the costs of more flexible production plants. With wireless technologies new challenges like packetloss and more complex medium access methods arise. These can cause a higher variance in the transmission delay – called jitter – compared to wired networks.

By now, industrial standards allow to have either changing conditions for the network – like WirelessHART [10] – or to close control loops – like Wireless Interface for Sensors and Actuators (WISA) [12]. Changing conditions like interfering networks, adjustments in the network's application or even changes in the network's topology cause a reorganization. As most of industrial wireless standards are based on Time Division Multiple Access (TDMA) in order to guarantee certain data delivery deadlines, such a reorganization in most cases requires new TDMA-schedules [8, 10, 12]. These schedules need to be transferred to all nodes in the network and afterwards applied in the network. Transferring and applying new schedules can cause jitter if the order of tasks in the schedules is changing.

Jitter is particularly a problem in applications realizing closed loop controllers utilizing wireless links. Parameterizing controllers is even more complicated if the exact timing new measurements arrive with is unknown. Therefore the jitter must be kept as low as possible.

Combining wireless technologies with flexible machinery compounds that need closed control loops using these technologies, therefore, is a tough challenge. This is especially true when taking into account that a manufacturer does not want to stop the whole production for only minor adjustments. To support such scenarios we present our Adaptive Industrial Real-time Controller Network (AIRCoN)-stack which is capable of seamlessly switching configurations to adapt to changes in the network's environment, topology and application.

After discussing existing work in Section 2, we present the design of our AIRCoN-Stack in Section 3 which provides the required flexibility. We implemented our design on top of Free Real Time Operating System (FreeRTOS) that runs on an STM32 discovery kit with STM32F407VG MCU (STM32). The DecaWave DW1000 UWB transceiver [3] was used as a communication module. With that implementation we performed our real world evaluation, presented in Section 4. Section 5 concludes the paper and gives an outlook to further research on this topic.

2 RELATED WORK

Industrial Standards One approach to bring wireless communication systems into process automation is ISA100.11a [8] which is based on the IEEE 802.15.4 [2] standard but incorporates TDMA to provide real-time abilities. The standard is not yet supposed to be used in factory automation processes because it has a maximum delay up to 100ms [7].

WirelessHART is part of the well-known Highway Addressable Remote Transducer (HART) and provides a self organizing mesh

NEAT'18, August 20, 2018, Budapest, Hungary
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5907-8/18/08…$15.00
https://doi.org/10.1145/3229574.3229578

network that connects field devices wirelessly while still being very robust. Like ISA100.11a, it is based on IEEE 802.15.4 with an additional TDMA protocol but with fixed 10 ms slots to substitute the original token passing mechanism of HART. WirelessHART does at least support mobile handheld devices for diagnostics. Yet, it is not designed to support closed loop controllers while the network topology is changing [10].

WISA is an industrial standard which was designed to close control loops wirelessly using IEEE 802.15.1 [1]. WISA is very well suited for fixed production cells with many nodes, requiring a very low and reliable delay and jitter. A issue is that while changing the TDMA schedule, real-time properties might break.

Other research The Mobility-Aware Real-Time Scheduling for Low-Power Wireless Networks (MARS) system by Dezfouli et al. [4] is one of the first real-time schedulers that is aware of mobility and able to handle it. The presented system is based on the hierarchical network model of WirelessHART [10] with one gateway and several non-mobile infrastructure nodes wirelessly connected to it. The mobile nodes exchange data with these infrastructure nodes, which then forward the packets towards the gateway. To handle mobility, a mobile node gets a slot at each of the infrastructure nodes at the time it joins the network (*on-join reservation*). Therefore, it does not matter to which infrastructure node a mobile node transmits data. This is rather inefficient due to the waste of slots currently not used by the node and this lowers the scalability.

Thaskani et al. [13] published a mobility tolerant TDMA-based Medium Access Control (MAC)-protocol for Wireless Sensor Networks (WSNs) in 2011. Their MAC-protocol clusters the nodes and can handle intra-cluster as well as inter-cluster mobility with the drawback of several seconds delay in case of topology changes. This introduces jitter which is too large by several orders of magnitude for most industrial real-time applications.

In 2012, Ferrari et al. [5] published their Low-power Wireless Bus (LWB) which builds upon "Glossy" [6]. It can handle node mobility and has a very high delivery probability achieved by the use of both TDMA and flooding. The LWB is targeted more towards classical WSN applications, not to industrial automation scenarios [5].

The GINSENG project [11] bases on the IEEE 802.15.4 physical layer and was designed for refinery surveillance. The so called *GinMAC* protocol uses a single-channel TDMA schedule with a predetermined sender and receiver for each slot. Changing the deployed schedule during operation is not supported.

All the presented work does not consider the jitter of packet delivery, whether it is introduced by packet loss, routing decisions, rescheduling, or the processing of the packets itself. This negligence of the jitter in networks that are designed to be used in closed loop controllers leads to serious problems during their implementation and operation.

3 AIRCON-STACK

As we have shown in Section 2, there is currently no real-time communication protocol that meets the requirements to control factory automation processes and supports flexible network topologies, application needs, and environments at the same time. Thus, we propose AIRCoN-Stack which is capable of this. In this section we give an overview of AIRCoN-Stack's design and how it tackles all the previously explained challenges.

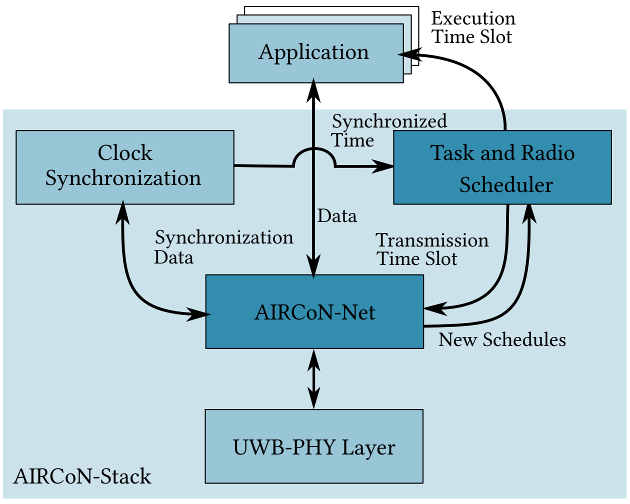


Figure 1: AIRCoN-Stack design overview with the information flows between its components

Figure 1 shows an overview of the architecture of AIRCoN-Stack with its components and the information flow among them. The following subsections describe the individual components.

3.1 Application Layer

The application layer consists of FreeRTOS tasks. They read sensor values, control actuators and exchange data with tasks running on other nodes. The users can freely implement them as they please. The only restriction is related to timing. As stated earlier, AIRCoN-Stack is based on TDMA to ensure real-time data delivery. Thus, the tasks have to finish execution in their assigned time slots in order to send their data in time.

3.2 Scheduler

At what time those application tasks are executed is determined by the scheduler. Although FreeRTOS already comes with its own real-time scheduler, an additional one was implemented on top of it. This had to be done due to the fact that the original scheduler works solely based on different priorities of the tasks. It always schedules the task with the highest priority which is currently in `READY`-state (i.e., it is ready to execute and not waiting, e.g., for I/O or a timer to expire). While that might be suitable for use-cases where the nodes operate in isolation, it is clearly not when forming a real-time network with multiple nodes competing for access to the radio channel. Thus, it was decided to couple the network scheduling and the processor scheduling. This way both schedules are TDMA-based and timely data transmission is guaranteed alongside timely execution of the respective task generating the data prior to transmission. A coupled example schedule can be seen in Figure 2. It shows two rounds of the same schedule which in this case consists of three slots. Note that the CPU is not used during

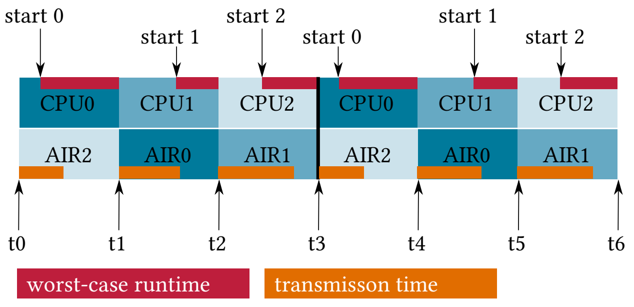


Figure 2: Example AIRCoN-Stack schedule of three tasks and two rounds. The execution times of the tasks are shown in red while the duration of the packet transmission is marked orange.

the transmission of a packet and can execute the succeeding task. Thus, there is no problem if the red and orange bars in Figure 2 are overlapping in time.

If a task is supposed to transmit in slot AIR1 (i.e. at time t_2), the AIRCoN-Scheduler schedules that task for execution in the slot CPU1. This way the transmitted data is always as recent as possible while keeping the age of the data constant.
The scheduling in AIRCoN makes use of the vanilla FreeRTOS scheduler and lets it handle context switches, Inter Process Communication (IPC) etc. All tasks wait blocking for a notification. The scheduler notifies only the task that is to be executed. After having completed its execution round, the task waits blocking again.

The AIRCoN-Scheduler works the same way and is unblocked by a notification sent by an Interrupt Service Routine (ISR) invoked by a timer which was set to the beginning of the respective slot. Upon being unblocked, the scheduler determines the task that is ought to be executed and its planned transmission timestamp for that slot according to the schedule. Figure 3 shows the timing of the scheduling procedure. When implementing an application task, it is necessary to estimate the absolute worst-case runtime. The task will always be scheduled (t_{II}) at least that time before t_{IV} – the transmission time. This ensures timely execution and avoids jitter because the tasks are always executed with the same frequency – independent of the possibly varying execution time of the scheduler.

This may in turn result in different t_{III} if the task's execution times differ. In order to still ensure the exact transmission timestamp, the "delayed transmission" feature of the radio transceiver is utilized. The scheduler provides the task with its scheduled transmission timestamp which will pass this on to the transceiver when transmitting data. The radio will then ensure the transmission to be precisely timed at the given timestamp (c.f. Section 3.4).

The goal for AIRCoN-Stack is to support changes of every property of the network configuration. Hence, it is crucial to be able to switch the schedules during runtime with the least additional jitter possible. This schedule switching needs to happen at the same time on all nodes of the network because otherwise the transmissions of different nodes might overlap and therefore interfere with each other. This has to be avoided by all means. AIRCoN-Stack uses a message in its management protocol to specify at what time all

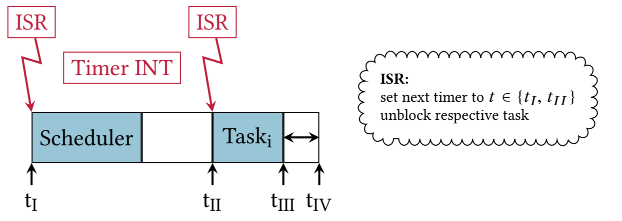


Figure 3: Procedure of $Task_i$ in its slot. At t_I the scheduler calculates the following slots, t_{II} is the defined transmission time t_{IV} minus the worst-case execution time of $Task_i$.

nodes switch the schedule. This message specifies the transition time and the schedule to switch to. This cannot be done during an ongoing schedule cycle but only at the end of a round and thus, in the example from Figure 2, t_0, t_3 and t_6 are valid transition times.

3.3 Time Synchronization

In order to make network-wide TDMA work, a precise time synchronization is crucial. Thus, in an earlier work a precise time synchronization protocol was developed [14]. It uses one node as the clock master. It frequently broadcasts synchronization beacons which the slaves use to adjust their clocks. For more details, please kindly refer to the cited paper.

3.4 UWB-PHY Layer

Not only the time synchronization beacons, but all the packets in our network are transmitted and received using DecaWave's DW1000 [3] radio transceiver which implements the IEEE 802.15.4 Ultra Wide Band (UWB) physical layer. UWB was chosen because of its robustness against narrow-band interference which is very common in factory automation scenarios due to electro magnetic emmissions originating from welding machines, generators etc.

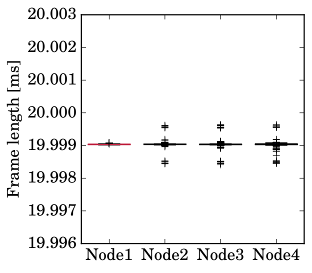
The drivers to use the DW1000 with FreeRTOS on our hardware had already been developed for one of our previous works [15].

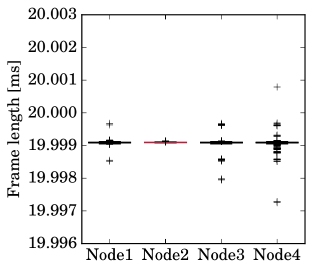
The transceiver was originally developed for indoor localization purposes which require extremely accurate clocks to calculate the time of flight of packets. Therefore, the transceiver comes with an internal clock providing a 15.65 ps resolution to timestamp received packages. Moreover, it has the feature to send a packet exactly at a previously configured time. Both these features are heavily made use of for the time synchronization. The latter is also used for the transmission of packets exactly at the beginning of a slot as indicated in Section 3.2. The details of that "delayed transmission" feature have been explained in [14].

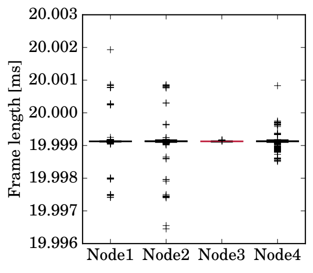
3.5 AIRCoN-Net

As Figure 1 shows, there is one central unit in AIRCoN-Stack organizing all the communication done by the system: AIRCoN-Net. It connects all the previously described units and is implemented as a FreeRTOS task as well. It controls all transmissions and receptions in the real-time network. This section shows how packets received by the transceiver are passed to the applications and vice versa.

Whenever a slot is scheduled to use the Central Processing Unit (CPU), it is allowed to transmit a packet to the network. To


(a) Node1 as Master


(b) Node2 as Master


(c) Node3 as Master

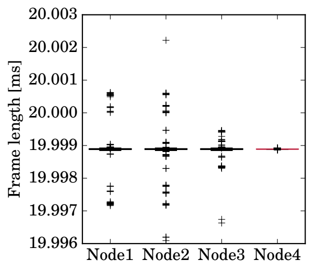

(d) Node4 as Master

Figure 4: Results of execution interval accuracy with different master nodes.

do so, the packet is passed to AIRCoN-Net. Besides the payload and the destination's MAC address, some additional information needs to be provided so AIRCoN-Net can construct an AIRCoN-MAC frame according to the format defined in Table 1. That newly constructed frame is then encapsulated in an IEEE 802.15.4 frame to stay compliant to the standard. This is important to be able to leverage hardware features like frame filtering which every common IEEE 802.15.4 transceiver provides. After the encapsulation, the frame is passed to the radio driver which instructs the transceiver to transmit it at the provided time (c.f. Section 3.4).

The protocol field in Table 1 defines whether the packet is for an application using AIRCoN-Stack or a packet used for purposes like clock synchronization. The message type is mostly used in management packages. Specified message types are:

- **Schedule-change-request:** Requests other nodes to change the current schedule to the one defined in this message.
- **Sync-beacon:** Periodically sent by the master of the clock synchronization.
- **Sync-delay-request:** Request to the master to start propagation delay measurement.
- **Sync-delay-response:** Response of the master to the sync-delay-request.

The addressing of different applications on the same node was inspired by protocols like TCP or UDP and like those, AIRCoN-Stack uses port numbers to address applications. The source port is a unique number for every application, the destination port is not unique. This part is inspired by publish-subscribe systems. Even if this seems uncommon first, it enables AIRCoN-Stack to deliver the same data to different applications without any additional overhead in form of multiple transmissions. This can be very useful if different control tasks on one node need the same data. However,

Table 1: The message format of AIRCoN-MAC

Bits	Content
0-3	Protocol
4-7	Message Type
8-15	Source Port
16-23	Destination Port
24-End	Payload

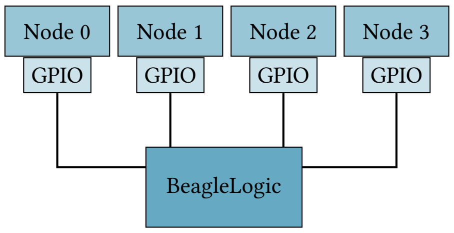


Figure 5: Evaluation setup with 4 nodes connected to the BeagleLogic to measure task executions timings

it comes at the price of a more complicated, centralized buffer management because received data has to be buffered until all addressed applications processed it. This can lead to unpredictable processing times which are to be avoided because they increase the jitter in the communication and might break the real-time requirements. To overcome this, AIRCoN-Stack utilizes ring buffers with a fixed length for each destination port. In this buffer it is always the oldest packet that is overwritten by a new one. This way the system designer can implicitly control the amount of time given to the applications to process received data by changing the amount of buffers in the ring. The system cannot fail due to a lack of memory caused by applications not freeing their buffers.
In order to subscribe to data sent to a specific port, the applications register for that destination port and are getting notified by AIRCoN-Net about newly received data which they can then read from the central ring buffer.

4 EVALUATION

In our evaluation we used a network of four nodes: one master with three slaves. The example application is to simultaneously set a pin high at all three slaves 5 ms before the master does the same. This mimics an application where sensors must be read at the same time on different devices and the master uses the collected data to control a process. To ease the measurement of the timing accuracy we used a pin and let all clients act in the same slot. To measure the timings we use BeagleLogic[1] which was connected to all nodes, as shown in Figure 5 and measured at a sample rate of 100 MSps.

[1]https://github.com/abhishek-kakkar/BeagleLogic

In all evaluations we used schedules consisting of four slots of 5 ms, so a frame is 20 ms long.

4.1 Single Node Timing Accuracy

In the first evaluation we measured the time between two executions of the same task on one node. This shows how accurate the temporal distance between two executions of the same task is. That is important to assess the suitability of our system for control tasks. To mitigate hardware influences we performed this evaluation within four different master-slave configurations, so every node was used as the master for one evaluation. The results of these evaluations are displayed in Figure 4. Most obviously the master has the best timing accuracy in all master-slave combinations. Further, the whole network had the best accuracy with Node1 as the master with about 1 µs and worst with Node4 as master with 6 µs Jitter. These differences are due to manufacturing tolerances of the crystals used to clock the nodes. As the results are represented as boxplots, the vast majority of execution intervals is represented as the bars close to 19.999 ms. This is due to the few outliers that compress the whole box with whiskers into a single bar. In this evaluation the outliers are more important, as they help to estimate the maximum jitter, that is 6 µs. But still, the majority of execution intervals have a jitter below 1 µs. Taking into consideration that our CPU runs at 168 MHz the maximum jitter is about 1000 cycles and the majority is below 168 cycles, the jitter results most likely from clock-synchronization-packet loss, floating point accuracy errors in time-base conversion and context switches. The difference between the ideal frame-length of 20 ms and the measured 19.999 ms is a static offset that might be removed by adding a static amount of cycles to every slot. As we want to give a realistic baseline, we decided against applying such optimizations.

4.2 Network Timing Accuracy

As a second evaluation we measured the time between the execution of our task on the master and the slaves. This was motivated by the need to quantify the jitter between different nodes. Taking the controller example, it shows which jitter is to be expected between the measurement and the actual control task. For this evaluation we chose Node1 as the master as it had the best results in the previous evaluation. All other parameters are the same as in the evaluation in Section 4.1.

In our schedule – schedule 1 in Table 2 – the master has the evaluation task in slot 2 and the slave in slot 1 because it is easier to trigger the measurement that way. For better readability we subtracted the slot length of 5 ms from the delay in the following figures, therefore the displayed delay gets negative in some cases. Figure 6 shows the jitter with a static schedule, that means, there are no changes in the transceiver configuration or the schedule. As expected, the jitter is almost the same as in Figure 4, with up to 2 µs. The fact that network-wide jitter is a little below most of the measurements in Figure 4, is because we measured all nodes in the same frame, therefore jitter introduced by synchronization is mitigated. The evaluation shows that we are able to keep the maximum jitter and delay within 336 CPU cycles in the whole network for a static setup.

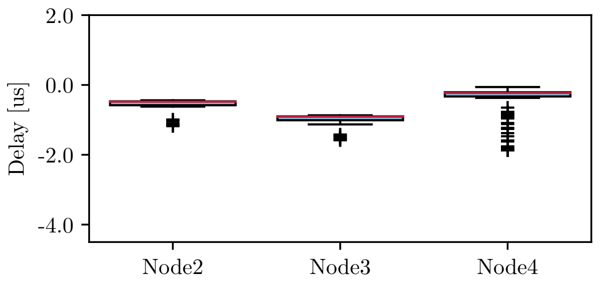


Figure 6: Network timing accuracy measurement without schedule or transceiver configuration changes.

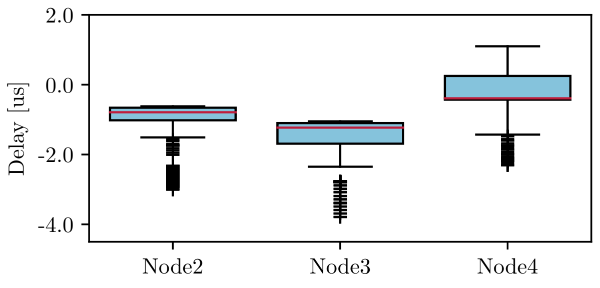


Figure 7: Network timing accuracy measurement with transceiver configuration changes.

Table 2: The two alternated schedules for each master and slave, showing the position of the evaluation tasks

	Slots	1	2	3	4
Schedule 1	Master	—	Eval	Sync	—
	Slave	Eval	Sync	—	—
Schedule 2	Master	Sync	—	—	Eval
	Slave	—	—	Eval	Sync

Our network stack was designed for networks with changing applications and time variant topologies. Therefore we performed two more evaluations: one where the transceiver's radio channel is changed periodically to emulate an adaption to changing environment. In the second one we periodically changed the order of the tasks in the schedule to emulate changing topologies or applications. While changing the order of the tasks in the schedule, the evaluation task on the master and slave always stayed in the same slot relative to each other. Only the absolute position of the tasks was changed as shown in the two schedules in Table 2. Figure 7 shows the delay between the master and the slaves with changing transceiver configurations. As jitter is up to 5 µs, there is a significant difference to Figure 6. This jitter is introduced by blocking Serial Peripheral Interface (SPI)-transfers to reconfigure the transceiver chip and reading/writing the *Schedule-change-request*-packet. As on our prototype hardware the SPI-Bus is only able to use 10 MHz, communicating with the transceiver takes the majority

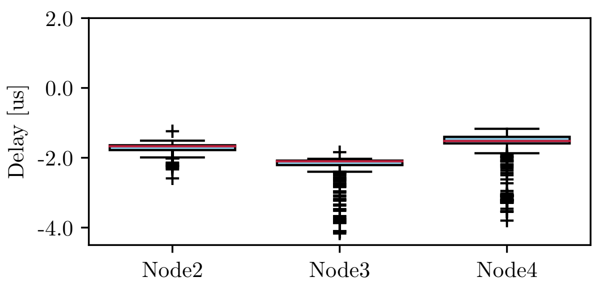


Figure 8: Network timing accuracy measurement with two alternating schedules switched every 200 ms**.**

of the 5 ms slot. We expect lower jitter on a PCB designed for higher frequencies. To evaluate the capability to handle changing topologies and applications we switched the schedule every 10 frames (i.e. 200 ms). We used the two schedules shown in Table 2. They were alternated. As the evaluation tasks at master and slave are in consecutive slots in both schedules, the jitter should not increase significantly. In Figure 8 the results of this evaluation are shown. Compared to the other evaluations, there is no significant difference. This shows, that AIRCoN-Stack is able to change schedules without introducing additional jitter.

Concluding the evaluations, we have shown that our system is able to keep the jitter within a range of 6 µs even if changing environments demand a reconfiguration of the radio transceiver – e.g. changing the radio channel – or if the application or the network topology changes and a new schedule needs to be applied.

5 CONCLUSION

Research and industry started to come up with wireless field buses to cope with requirements of more and more flexible assembly lines. While being an improvement, they are still lacking the adaptability to change communication configuration (e.g., TDMA-schedules or radio channels) while maintaining real-time communication. Hence, we presented AIRCoN-Stack: a network stack for industrial applications that is able to perform seamless configuration changes. Due to this ability, changing application needs, environmental conditions, or network topologies can be handled without interrupting real-time tasks using the changing network facilities. We evaluated these abilities in a real-world testbed and measured a total maximum jitter of 6 µs on a single node. For task execution jitter on different nodes our network stayed within 5 µs even while changing schedules and transmitter configurations.

For future work, we are planning to automatically identify situations requiring network configuration changes. All nodes involved then send airtime requests to a central scheduler which then computes a new schedule incorporating as many requests as possible and distributes it to the nodes. The network can then switch to the new schedule using the means we developed in this work.

REFERENCES

[1] IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements. - Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs). *IEEE Std 802.15.1-2005 (Revision of IEEE Std 802.15.1-2002)*, 2005.

[2] IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, 2011.

[3] DecaWave Ltd. DW1000 User Manual - How to use, configure and program the DW1000 UWB transceiver. https://www.decawave.com/support/download/file/nojs/948, 2018. last accessed: 03.04.2018.

[4] B. Dezfouli, M. Radi, and O. Chipara. Mobility-aware real-time scheduling for low-power wireless networks. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.

[5] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 1–14. ACM, November 2012.

[6] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with Glossy. In *10th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 73–84. IEEE, April 2011.

[7] International Society of Automation. ANSI/ISA-100.11a-2011 Wireless systems for industrial automation: Process control and related applications. *https://www.isa.org/store/products/product-detail/?productId=118261 last accessed 03.04.2018.*

[8] ISA100 Standards Committee. ANSI/ISA-100.11a-2011 Wireless systems for industrial automation: Process control and related applications, 2011.

[9] H. Kagermann, W. Wahlster, and J. Helbig. Recommendations for implementing the strategic initiative INDUSTRIE 4.0. *acatech – National Academy of Science and Engineering*, April 2013.

[10] A. Kim, F. Hekland, S. Petersen, and P. Doyle. When HART goes wireless: Understanding and implementing the WirelessHART standard. In *IEEE International Conference on Emerging Technologies and Factory Automation*, pages 899–907, September 2008.

[11] T. O'donovan, J. Brown, F. Büsching, A. Cardoso, J. Cecílio, J. D. Ó, P. Furtado, P. Gil, A. Jugel, W.-B. Pöttner, U. Roedig, J. S. Silva, R. Silva, C. Sreenan, V. Vassiliou, T. Voigt, L. Wolf, and Z. Zinonos. The GINSENG System for Wireless Monitoring and Control: Design and Deployment Experiences. *ACM Transactions on Sensor Networks*, 10(1):4:1–4:40, Dec. 2013.

[12] R. Steigmann and J. Endresen. *Introduction to WISA*. ABB, July 2006.

[13] Thaskani, S. and Kumar, K.V. and Murthy, G.R. Mobility tolerant TDMA based MAC protocol for WSN. In *2011 IEEE Symposium on Computers Informatics (ISCI)*, pages 515–519, March 2011.

[14] G. von Zengen, K. Garlichs, Y. Schröder, and L. C. Wolf. A sub-microsecond clock synchronization protocol for wireless industrial monitoring and control networks. In *2017 IEEE International Conference on Industrial Technology (ICIT) Special Sessions*, Toronto, Canada, Mar. 2017.

[15] G. von Zengen, C. Wulf, Y. Schröder, and L. C. Wolf. Real time capable uwb wireless network sniffer: Poster. In *Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '16, pages 389–390, New York, NY, USA, July 2016. ACM.