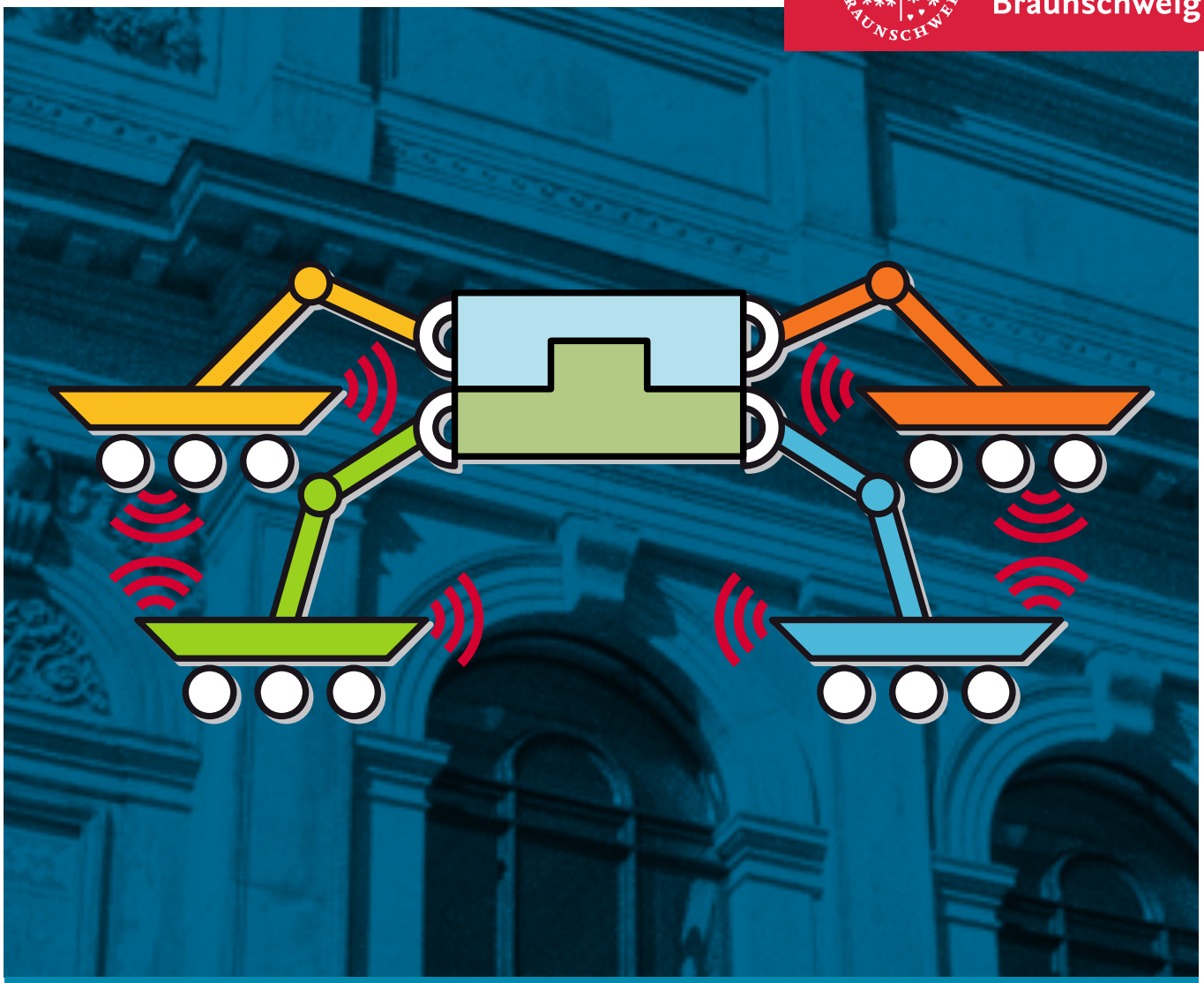
Institut für Beriebssysteme und
Rechnerverbund

Technische
Universität
Braunschweig



Resource Allocation in Mobile Wireless Real Time Networks

Georg Constantin von Zengen

Resource Allocation in Mobile Wireless Real Time Networks

Von der
Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines
Doktoringenieurs (Dr.-Ing.)

genehmigte Dissertation

von
Georg Constantin von Zengen
geboren am 20.02.1987
in Celle

Eingereicht am:	12.11.2019
Disputation am:	18.12.2019
1. Referent:	Prof. Dr.-Ing. Lars Wolf
2. Referent:	Prof. Dr. rer. nat. Nils Aschenbruck

2020

Abstract

The use cases for Cyber-Physical Systems (CPSs) range from industrial automation over automotive to search-and-rescue applications. Nowadays these CPSs work either in static networks, like in production lines, or isolated and mobile, as for example Unmanned Aerial Vehicles (UAVs). The cooperation of mobile CPSs is only possible with very relaxed real-time requirements. For the tight cooperation of mobile CPSs new techniques are needed. The special challenge in such networks is that the communication needs to guarantee hard timing boundaries but also needs to be flexible enough to adapt to changes within the network.

In this work we present an architecture that copes with these networks and their challenges. It consists of four main components: a time synchronization, a real-time networking stack, a scheduling algorithm and a management protocol. As cooperation between mobile CPSs requires feedback loops to be closed via the wireless links, the time synchronization needs to be accurate between several CPSs. To be able to time the execution of tasks as accurate as possible we present a sub-microsecond time synchronization. By utilizing our drift compensation, CPSs can make use of low-cost crystal oscillators without loosing timing accuracy.

To make use of such an accurate time synchronization, we present a real-time network stack that incorporates not only the scheduler for the communication but also the scheduler for the execution of tasks. Thus, the jitter a feedback loop experiences is kept minimal. To support the adaption to changes in the network we designed all operations in a way that they introduce a minimum amount of jitter.

This adaption to changes is one of the key requirements to the scheduling algorithm. As the adaption must happen without harming the timings of running real-time application, a novel scheduling approach is necessary. To fulfill this requirement we introduce a Mixed Integer Linear Programming (MILP)-model to calculate schedules for the presented real-time network stack. As solving MILP-models is computational complex and CPSs often have only limited computational power, we introduce a heuristic to calculate these schedule with far less effort.

To disseminate schedules to CPSs, we evaluate the applicability of Concurrent Transmission (CT) protocols. All previous research on CT was done on similar hardware. We investigate whether the results of this research can be generalized and point out the differences and similarities. Further, we frame the challenges heterogeneous CT networks have to overcome.

Kurzfassung

Cyber-Physical Systems (CPSs) haben vielfältige Anwendungsbereiche von Automatisierungstechnik bis zu Such- und Rettungsanwendungen. Heutzutage arbeiten CPSs entweder in statischen Netzen, wie in Produktionslinien, oder isoliert und mobil, wie z.B. als unbemannte autonome Luftfahrzeuge (UAVs). Die Kooperation mobiler CPSs ist bislang nur mit weichen Echtzeit-Anforderungen möglich. Für die enge Zusammenarbeit mobiler Roboter sind daher neue Techniken notwendig. Die Herausforderungen in solchen Netzen sind, dass die Kommunikation harte zeitliche Grenzen gewährleisten muss, aber auch flexibel genug sein muss, um sich an Veränderungen im Netz anzupassen. In dieser Arbeit präsentieren wir eine Architektur, die mit diesen Herausforderungen erfüllt. Sie besteht aus vier Hauptkomponenten: einer Zeitsynchronisation, einem Echtzeit-Netzwerkstack, einem Managementprotokoll und einem Scheduling-Algorithmus.

Da die Zusammenarbeit zwischen CPSs erfordert, dass Regelkreise über drahtlose Verbindungen geschlossen werden, muss die Zeitsynchronisation zwischen mehreren CPSs sehr genau sein. Um die Ausführung von Aufgaben so genau wie möglich zu terminieren, stellen wir eine Zeitsynchronisation im Submikrosekundenbereich vor. Durch die Verwendung unserer Driftkompensation können die CPSs kostengünstige Oszillatoren nutzen, ohne auf Synchonisationsgenauigkeit zu verzichten. Um die Genauigkeit der Zeitsynchronisation zu nutzen, präsentieren wir einen Echtzeit-Netzwerkstack, der nicht nur den Scheduler für die Kommunikation, sondern auch den Scheduler für die Ausführung von Aufgaben beinhaltet. Dadurch ist der Jitter, der einen Regelkreis beeinflusst, noch geringer. Um die Anpassung an Veränderungen im Netzwerk zu unterstützen, haben wir alle Operationen so konzipiert, dass sie ein Minimum an Jitter aufweisen.

Die Anpassung an Veränderungen im Netzwerk ist eine der wichtigsten Anforderungen an den Scheduling-Algorithmus. Da diese Anpassungen erfolgen müssen, ohne die Echtzeitanforderungen der laufenden Echtzeitanwendung zu beeinträchtigen, ist ein neuer Scheduling-Ansatz erforderlich. Um dies zu erfüllen, stellen wir ein Mixed Integer Linear Programming (MILP)-Modell zur Berechnung der Schedules für den vorgestellten Echtzeit-Netzwerkstack vor. Da die Lösung von MILP-Modellen rechenintensiv ist und CPSs oft nur über eine begrenzte Rechenleistung verfügen, führen wir eine Heuristik ein, die diese Zeitpläne mit weniger Aufwand berechnet.

Für die Verbreitung der Schedules unter den CPSs, betrachten wir die Anwendbarkeit

von Concurrent Transmission (CT)-Protokollen. Alle bisherigen Forschungen zu diesem Thema wurden auf sehr ähnlicher Hardware durchgeführt. Wir haben untersucht, ob die Ergebnisse dieser Forschung auch auf andere Hardware verallgemeinert werden können und zeigen die Unterschiede und Gemeinsamkeiten. Weiterhin stellen wir die Herausforderungen vor, die heteroge CT-Netzwerke überwinden müssen.

Contents

1 Introduction

Cyber-Physical Systems (CPSs) are widely used in all sorts of use cases. For now, they work either isolated or in static, often wired, networks. Isolated and mobile CPSs exist in various forms, for example, Unmanned Aerial Vehicles (UAVs) that deliver small packages like medicine to remote places. An example for networked but static CPSs are production lines, where several units work in parallel or series to finish a certain product. These production lines are considered as an example for a so called Cyber-Physical Network (CPN). A CPN typically consist of multiple networked CPSs that execute a greater application.

The use cases of CPSs and CPNs range from factory automation to search and rescue missions in disaster recovery. In their use cases CPSs often perform the exhaustive, tedious, or dangerous tasks, to relieve humans. This is already practiced in daily life and in recent years stationary robots were developed, which are able to cooperate with humans to fulfill their tasks.

The use cases of these robots could be even wider, if they were more mobile and still had the ability to cooperate with each other. One example for such a use case are multiple autonomous UAVs lifting and carrying heavy loads together, balancing the weight between all participating devices. Another use case are mobile manufacturing robots, holding work-pieces while another robots is welding them together.

Ideas like these are emerging together with recent advances in areas like real-time wireless networking [14, 9, 34], indoor localization [39, 63, 45] and the broader use of wireless field buses like WirelessHART (WirelessHART) [24].

Wireless communication, in the previously described use cases, is mandatory, as cables would restrict the flexibility of the robots. In disaster recovery scenarios or other outdoor use cases, there might be a lack of external infrastructure to handle central coordination. Nevertheless, wireless CPNs still need to close feedback loops, with tight time constraints, to guarantee safe operation while for example carrying and balancing heavy loads. To close a feedback loop it is important to transfer the feedback with a minimum delay and even more important with a variance in that delay. The less predictable the delay is, the harder it gets to estimate the error that effects the controlled process at the time the correction variable arrives at its destination. Thus, a real-time transmission of measurement data and actuating values is key. These requirements are not yet met by present protocols.

Initiatives like the DFG Priority Programme Cyber-Physical Networking (SPP 1914)[1] underline that the classical, in other research areas often followed, research objective of increasing the data rates is not sufficient for CPNs.

In this thesis we use a production environment with cooperating, wirelessly connected, mobile robots as the primary use case for our approaches. This use case combines all of the challenges in a way that is very comprehensible. Nevertheless, it is just one use case of many, to which the approaches we introduce in this thesis are applicable to. It needs real-time communication to close feedback loops wirelessly, and enable multiple robots to cooperate in fulfilling a task. This real-time communication needs to guarantee upper bounds of delay and jitter. An application that a group of robots performs jointly typically consists of several smaller tasks that need to be fulfilled. Such a group of robots forms a Task Cluster (TC), Figures 1.1a and 1.1b depict two examples of such TCs. Figure 1.1a shows two TC that coexist and may need to cooperate in the future. A TC is not limited to consist of different robots, it might be one highly modular robot that uses wireless communication between its modules, Figure 1.1c shows how such a TC could look like.

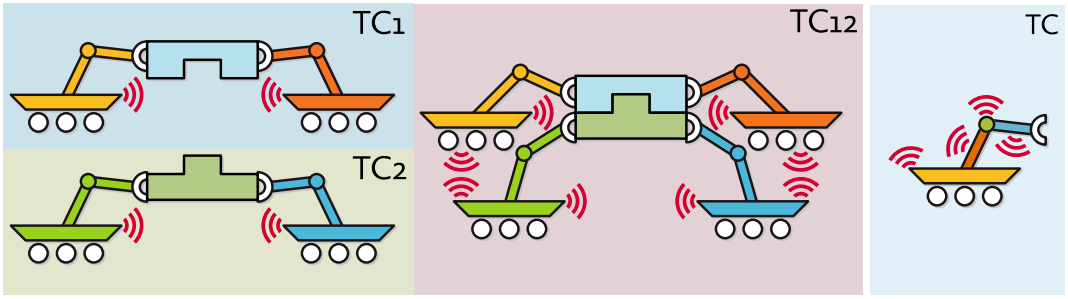


Figure 1.1: Different TC configurations, (a) shows two TCs consisting of two robots each, (b) shows a TC which combines all robots from (a) into one TC, (c) shows a robot that consists of several wirelessly connected parts forming a TC.

The tasks in an application are spread over several units in a TC and need to be executed in a defined sequence. If parameters in the production do change, this sequence might change. Further, such changes could lead to the replacement of tasks or even whole robots. The challenge for the communication is to adapt to these changes, while guaranteeing safe operation. Thus, the network formed by multiple robots has to work in an ad-hoc manner.

[1] https://www.spp1914.de

The demand for an ad-hoc network gets even clearer in an example where robots travel through the production space to transport goods or to get to the next place to do some work. During these travels they will come into proximity of cooperatively working robots. To prevent interference between the different TCs, formed by various robots, the network topology needs to be changed while the real-time communication is ongoing. A combination of both modifications, in the application and changes in the network topology, happens if a new TC is formed from two smaller TCs to fulfill an application. A good example are the welding robots: if the two parts were built in two distanced locations, the carrying robots need to carry the parts to a common location. In that location they need to form one TC and also have to incorporate the robot that welds the two parts. The changes needed, to ensure a smooth operation of the TC, have to be done autonomously, as it is impossible to foresee all variations which can happen to a TC while planning a production.

Such a planned approach would also prevent one of the most important promises seen in such a modular production: the ability to restructure a production space rapidly. This new structure would allow cheaper adjustments to the productions and also allows a speedup in innovation cycles, as the productions could be reassembled without major work in the production space.

The most obvious lack of present wireless networks is the inability to close feedback loops via wireless links. Wireless Interface for Sensors and Actuators (WISA) is the only industrial standard that is specified to close feedback loops but it needs to be manually configured to fit a predefined task in a well known environment. Such a static configuration is incompatible with the goal of self adaptation. The self adaptation is necessary, to guarantee safe operation in changing or unknown environments. To close a feedback loop the main objectives for a network is to guarantee a maximum time that a message needs to reach its destination and to guarantee a certain reliability in the communication.

Research projects that are able to cope with changes in the topology or environment, like the low-power wireless bus [13] by Ferrari et al., do minimize the delay an information needs to travel the network but do not consider the jitter this delay suffers from. In feedback loops a delay can be modeled, the jitter on the other hand in unpredictable and therefore can not be modeled and adds an error into the controlled system.

1.1 Outline

In this thesis we propose an architecture capable of uniting the flexibility of an approach like the low-power wireless bus and the ability to close feedback loops like WISA. In Chapter 2 we detail on the problem statement and the architecture to cope with the presented challenges.

The following chapters describe the details of the four main components of this architecture. Chapter 3 discusses how to synchronize the time on all nodes in a TC to an accuracy to be able to close a feedback loop between these nodes. We propose a submicrosecond time synchronization protocol that incorporates a drift compensation to allow the use a low-cost crystal oscillators as a time base for the CPSs.

The real-time network stack to close these loops and to transfer the information needed for the time synchronization is introduced in Chapter 4. Its adaptability is a key ability to allow TCs to be formed and to be broke up while performing real-time tasks. The network stack is based on a Time Division Multiple Access (TDMA) protocol and in contrast to other network stacks able to switch its schedules without introducing unnecessary delay or jitter.

To generate schedules for this network stack, a Mixed Integer Linear Programming (MILP) model is proposed. This model is able to reschedule a TC if its application changed or even to merge two TCs. It takes care that the network can switch from the old to the new schedule without harming any real-time requirements during the transition. Together with a heuristic, following the same constraints, this model is discussed in Chapter 5.

In Chapter 6 we investigate the applicability of protocols based on Concurrent Transmission (CT) to the communication between different TCs. This communication might not need to meet hard real-time boundaries but needs to transfer information, like the new schedule, to all nodes in the current or future TC. As the schedule needs to be known to all nodes prior to joining the TC, it can not be transferred using the real-time communication protocol. CT-based protocols have shown extraordinary performance in disseminating information in a network in a short time with a high reliability. So far all evaluations of CT are based on one specific transceiver. We investigated whether these results can be generalized to other transceivers and which challenges a heterogeneous network has to face.

Chapter 7 summarizes the contributions made in this thesis and gives an outlook on research questions left open and on new ones that emerge from the results we present.

2 Architectural Overview

This chapter gives an overview about the challenges an architecture for networked mobile cooperating CPSs has to face. Further, we propose a system that copes with these challenges. The chapter details on work we presented in [64]. These challenges are discussed in Section 2.1, afterwards we discuss the ability of selected approaches from the literature to solve these challenges in Section 2.2. To handle the discussed challenges, we identify three core operations a network must support, these are introduced in Section 2.3. In Section 2.4 we present the components of our system and how they interact.

2.1 Problem Statement

The communication architecture needed for cooperating mobile or modular robots has to be flexible enough to adapt to the changes of communication needs that happen in such systems but still has to guarantee the strict real-time requirements tight control loops set. For the following network node or node is used as a synonym for one unit in a TC, for example a robot or a part of a robot, that is connected to a network. In real-time communication TDMA-based protocols are the state of the art. This approach of providing guaranteed time-slots for each transmission is used not only in wireless, but also in wired networks. To guarantee that a time-slot is not used by any other transmission, a schedule is needed that allocates the time-slots to transmissions. Since a transmission is only permitted in a time-slot which has been allocated for it, this guarantees that no transmission finds its time-slot occupied which would lead to a missed deadline. To combine both, the adaptability and the high real-time capabilities which are necessary for the considered CPSs, a scheduling algorithm is needed that generates schedules and resource assignments to allow smooth transitions between old and new schedules and allocations when the topology changes. Resources in this case might mean everything from different frequencies or orthogonal codes over different time-slots in a TDMA schedule to different PHYs. Thus, a schedule in our architecture is multidimensional unlike as in pure TDMA systems where it is just the division of time in time-slots.

Another challenge is that the communication capabilities of the nodes in a network are expected to be heterogeneous. It might happen that a certain part of the robot has not all different PHYs the robot's main module has. Therefore, the scheduling has to respect such limitations and schedule the communication in a way that all nodes are able communicate as they need to. To schedule a communication the intersection of the available resources to all involved nodes has to be considered.

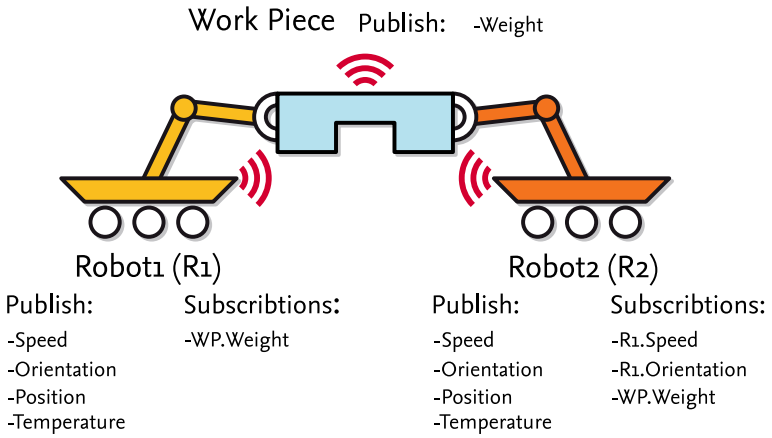


Figure 2.1: Example of a TC of two robots and a work piece which publish several values and have subscriptions to values of each other

We use a publish/subscribe architecture for the communication in a TC. Figure 2.1 depicts a TC that consists of two robots, R1 and R2 and one work piece. In this example R2 needs to follow R1, therefore it has subscribed to the speed and the orientation of R1. R1 and R2 are subscribed to the weight of the work pieces, both robots are carrying, which is published by the work piece or some other device as proxy for the work piece. This example does by no means claim to be complete or functional in the real world, it is used to explain the advantages of a publish/subscribe architecture.

All subscriptions are communicated to the scheduler, which, by using this information, is able to decide which information needs to be transferred to which node. In our example the scheduler would know that the speed and orientation of R2 have no subscriber and, therefore, do not need to be transmitted. The same is true for the position and temperature of both robots. As the scheduler would not reserve any resources for the is information, the publish/subscribe architecture helps to conserve resources.

Another advantage is that the scheduler is able to schedule communication to resources to which not involved nodes do not have access to. For example, the work piece might have a communication interface with less features than the robots. With these features the scheduler is able to realize multicast communication over a broadcast medium. This is achieved by scheduling all involved nodes to listen to the same resource at the time the data is transmitted. As other nodes can use the same time-slots for other communications on different resources or shut down their communications to save power, we call it multicast communication. In our example the transmission of the work piece's weight would be realized as a multicast transmission to both robots. The work piece could go into some sort of sleep state to save power while R1 transmits its speed and orientation.

Such a multicast communication has several advantages, the most important ones are: first, it saves time, as data only needs to be transmitted once. Second, being able to send the same data to different nodes at the same time is beneficial for control loops, several control loops can get their input with the same delay.

In order to guarantee an autonomous operation of a TC there must not be a central management unit. Instead, the communication management needs to be handled decentralized or by one of the nodes inside the TC that is chosen in an ad-hoc-manner. This node is responsible for schedule generation, clock synchronization and communication to other TCs to resolve potential communication conflicts. The mechanism to choose this node is highly application dependent as there might be a node with far more computation power or a better power supply. In other applications it might be beneficial that this node is in a certain geographic position in the network. Therefore, this issue is not in the scope of this work.

One of the key requirements in real-time communication is that a given upper bounds of delay and jitter for communication must not be violated. We define the jitter as the variance of the delay of transmissions. To relax the scheduling we introduce quality of service levels that give different delay and jitter bounds to a transmission. Choosing the most relaxed possible level for each transmission an application can ease the problem the scheduler has to solve.

To guarantee smooth transitions between schedules, great care must be taken not to violate the jitter and delay boundaries between an old and a new schedule.

Merging two TCs can be seen in three abstraction layers, shown in Figure 2.2. On the highest layer two existing TCs are depicted, both consist of two robotic devices. These two TCs need to cooperate to fulfill their application, therefore their schedules need to be merged in order to prevent interference and enable communication between the TCs. In the next abstraction layer the robots are represented by nodes and their application is represented by its communication needs between nodes. Even though, in our example each robot becomes one node, a robot could be a TC on its own and would be represented as several nodes with the communication needs between these nodes. In the third and most abstract layer both TCs are represented as schedules, each slot is reserved for one data transfer between nodes. As TCs might be merged to fulfill an application in cooperation, the communication needs of the application might change as well as the topology of the network. Thus, nodes might need data from nodes that were unknown before. This could lead to the case where multiple nodes need the same data from the same node, e.g. the red and green node need data from the yellow one. By utilizing a publish/subscribe based communication architecture, we are able to identify such cases and join the transmission into one with both nodes listening. Another benefit is that data without any subscribers of other nodes does not need to be scheduled for transmission, which save resources on the communication medium.

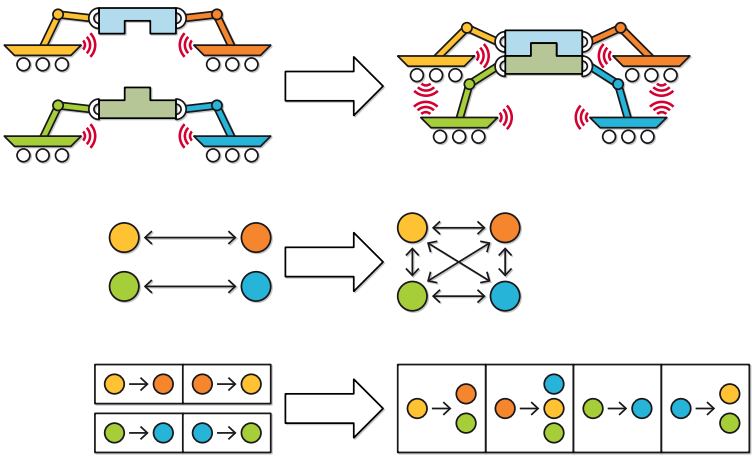


Figure 2.2: Merging of TCs on different abstraction levels (left: before merge, right: after merge). Top: Two groups of robots merge into one. Middle: Network clusters merge with new available communication paths. Bottom: Schedules are merged.

In order to reduce the end-to-end delay, the schedule for the network should be incorporated into the process schedule of a node's operating system. In our case end-to-end delay means the time between the data is produced/measured on one node until it is used by the another node. Such a combined scheduler introduces the benefit, that the sampling of sensor data can be done right before its transmission. Another benefit is that the subscribed task can be scheduled right after the reception of the data. A system utilizing two separated schedulers could easily schedule the sampling of data right after the last transmission of the same data. Such a scheduling would lead to unnecessary delay which could be avoided by integrating the operation system's scheduler with the network communication scheduler.

The system should feature three distinct traffic classes:

1. **Real-time:** Delay and jitter sensitive data to close feedback loops is sent in this class. The scheduling algorithm schedules transmissions for this data and makes sure requirements for delay and jitter are fulfilled by the network nodes.

2. **Contention:** In industrial applications, monitoring processes is an important use case for the communication infrastructure. Such traffic is not as critical in delay and jitter as the previous class. Therefore, such traffic must not be scheduled and can be handled by a contention-based communication infrastructure. In this class all nodes can employ reserved network resources in this traffic class as needed. However, no guarantees for delay and jitter can be satisfied for this traffic class.

3. **Management:** The scheduling algorithm will further reserve network resources for management traffic. Depending on available network resources, this can be a fixed PHY that only signals management data. Management signalling is used to coordinate TCs. It performs merge and split operations and transmits new schedules in the network. This is a special form of contention traffic as management operations are typically not as time sensitive as control data.

Such a communication architecture requires novel concepts for management operations. These new concepts can benefit from the utilization of hardware features of modern radio chips, in terms of maintaining tight real-time control applications even during changes in network topology.

2.2 Related Architectures and Approaches

The field of related areas to networked mobile cooperative robotics is vast, and the communications architectures are diverse. We focus on cooperative & mobile robotics and mobile real-time networking.

2.2.1 Cooperative Robotics

In recent years, major advances were made in the research fields of cooperative and networked robotics. However, the research does not focus on real-time wireless communication. On the one hand there is research focused on wireless communication like the work of Giordano et al. [17]. Research with this focus does not consider real-time communication. On the other hand there is research in the need for real-time communication for cooperative robots like the work of Oung et al. [37, 38]. To realize the communication they go back to wired connections. Disregarding the wireless communication aspect leads to cooperative robots that rely on wired links which prohibits the robots from being mobile, at least in relation to each other. Disregarding the real-time aspect leads to robots that can be mobile but their ability to cooperate is limited, as tight control loops can not be closed between two robots.

One particularly popular application for ad-hoc networked robots are UAVs. Giordano et al. [17] target the challenge of loosing the connectivity inside a wireless network of UAVs. The UAVs in their network are calculating trajectories to maintain network connectivity in a decentralized manner. This even works when some of them are commanded to move to a certain position by a human operator. As Giordano et al. state, UAVs can be used in rescue missions. In this application cooperation between UAVs is required to search large area or even to lift objects that are too heavy to be lifted by one UAV. Especially lifting an object with multiple UAVs requires a tight real-time communication. Such tight and guaranteed interaction among the robots is not available so far and it is also not in the focus of Giordano's work. The methods we propose could enable multiple UAVs to lift one object that is too heavy for a single one. That would extent the range of applications by far.

Modular robots are a special case of cooperative robotics. Every module can be seen as a robot of its own. Multiple modules joined together will act as a larger robot while its parts are just smaller robots in very tight cooperation. Such combinations of robotic modules can often perform tasks which a single robot could not do. An example for highly modular robots is the *Distributed Flight Array* introduced by Oung et al. [37, 38]. This system is a modular multi-propeller UAV that can assemble itself from multiple single-propeller devices. Communication between the single-propeller devices is realized through infra-red transceivers [37] on each side of the hexagonal platform in the first iteration. In the redesigned platform Oung et al. [38] use pushpins for a direct electric connection. On this connection a data rate of 115.2 kbps is realized. If a wireless real-time communication system as we propose it in this work would be available for use with the *Distributed Flight Array* the physical connection between the different devices could be omitted. This would enable the devices that they do not have to be attached to each other, but instead they could attach directly to their payload, e.g., at opposite sides. In such a scenario, the payload could be a structural component of the UAV allowing for greater flexibility in applications.

Romanishin et al. [41] present another example for re-configurable robots. The proposed system consists of multiple robotic cubes that move by flipping over their edges via an internal spinning wheel. They plan to use *ANT™* for inter-cube communication. However, this does not guarantee timing for real-time communication.

While self-assembling robots consisting of small pieces are still futuristic, the idea can be transferred to robots with interchangeable end effectors. Currently, these end effectors communicate mostly over wired connections. A real-time wireless network as proposed by us in this work could offer greater flexibility when connecting different modules. Especially for robots with rotating parts, wireless communication can be of great advantage.

Baillieul et al. [4] forecast a variety of coordinated activities for mobile robotics. They also state that mobility and wireless connection are adding a new dimension of complexity. They identify delay and jitter as challenges for the system but not bandwidth because only tens of bytes are transmitted at a time. Classical buffering cannot be used in this scenario since it likely violates the delay requirements.

The state of the art in cooperative and networked robotics shows two things: First, communication in such robotics is not well investigated but required to reach the full potential of such robots. Second, mobile, cooperative robotics would benefit greatly from our communication architecture and the proposed methods being as modular and scalable as the robotic platforms.

2.2.2 Mobile Real-Time Networking

There is a large variety of Medium Access Control (MAC) protocols for wireless real-time networks. Many of them were developed in the realm of Wireless Sensor Networks (WSNs) and related topics. A survey is given by Teng et al. [50]. Their work shows that adaptability is not a requirement in the networks these protocols were developed for. However, enabling mobility and adaptability in real-time wireless networks became a research topic in WSNs recently. Due to the special limitations of WSNs, like the limited power supply, approaches in this field aim to save as much power as possible. So saving power is typically prioritized over meeting real-time requirements as in the work of Nabi et al. [35] and Ferrari et al. [13].

Nabi et al. [35] cluster mobile nodes and assign each node in a cluster to a unique slot. Problems can arise when two clusters move into the transmission range of each other. Therefore, contention based methods like CSMA and slotted ALOHA have to be used in each slot. However, by introducing these methods hard real-time requirements cannot be met.

A system that adapts features of traditional bus systems to a wireless network is presented by Ferrari et al. [13]. They basically try to ignore the mobility of nodes by flooding all messages in the network with an algorithm called Glossy flood [14]. This flooding is performed in a TDMA fashion that make this approach time aware and basically capable of real-time communication. Yet, flooding every message to all nodes in a network has several significant disadvantages. These include the overall load induced into the network and inefficient use of slots, i.e., if they are used to forward messages where no node awaits it in a certain region of the network. The scheme works on a centrally computed global schedule that is distributed for every round. Every node can specify certain traffic demands and request according resources that are then satisfied by a new global schedule. In addition to the slots assigned for specific transmissions, a contention slot is provided that can be used to send the traffic requests. Another method to request a slot is to piggyback the request onto another transmission. Ferrari et al. did not consider situations with two networks that have to join each other to fulfill a task. Another disadvantage is the centralized schedule calculation – that also shows that this approach is designed for a single network in one location. To the best of our knowledge there is no protocol, based on the same principle as Glossy, that considers frequent changes in the network, like the joining of two networks. Most of these protocols are designed to work in classical WSN environments where the network's topology is static. The focus is more on reliable communications than flexibility [27, 31, 48].

The Mobility-Aware Real-Time Scheduling for Low-Power Wireless Networks (MARS) system by Dezfouli et al. [9], is one of the first real-time schedulers that are aware of mobility and able to handle it. The presented system is based on a hierarchical network model 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*). This way it does not matter to which infrastructure node a mobile node transmits data because all of them are waiting for the respective node to transmit. This alone is sufficient to ensure that real-time constraints are met even though some nodes are mobile. However, it is rather inefficient due to the waste of slots which are currently not used by the node. As a result, the cycle time would be unnecessarily elongated. To mitigate this problem, several theorems and rules are postulated.

One downside of MARS is the fact that jitter is not considered at all. The deadlines are said to be met for the nodes which are integrated into the system. But as soon as a new node requests to enter the network, the schedule has to be inevitably re-calculated. This can introduce a large jitter if the old and the new slot are far away of each other in the frame. And it can cause severe errors if the system relies on the exact timing.

A further problem resulting from the hierarchical structure with one single gateway is the length of the paths. Even in a physically meshed network, the usable topology is always a star. This requires every packet to take at least four hops (mobile node $\rightarrow$ infrastructure node $\rightarrow$ gateway $\rightarrow$ infrastructure node $\rightarrow$ mobile node), instead of only one hop (mobile node $\rightarrow$ mobile node). Although MARS seems to be a big step towards the right direction, the aforementioned facts are still problematic.

A project that evaluates the fifth generation (5G) of cellular networks for the use in factory automation is *Koordinierte Industriekommunikation (KOI)*[1]. In associated papers [2, 57] the authors present simulations on the physical layer as well as on the system level. They show that with a future 5G radio interface end-to-end delays of less than 1 *ms* will be possible. Another simulation shows that with Long-Term Evolution (LTE) (4G) such short end-to-end delays are not possible. Here these delays are at 4 *ms* for semi-persistent scheduling and at 12 *ms* for dynamic scheduling. The simulations also show that the reliability of both LTE and 5G are suitable to meet 10^{-9} block error probability. However, as in MARS there is no investigation on the jitter in the transmission.

One very recent work by Mager et al. focuses on closing feedback control loops over multi-hop wireless networks. The goal is to reach control intervals of tens of milliseconds with low cost, low power wireless technology. To make their system able to cope with imperfections of these communication technologies the authors present a communication protocol, a scheduling framework, and a control design [34]. Baumann et al. details the scheduling in a later work [5]. Even tough the system presented by Mager et al. has to cope many of the challenges we described in Section 2.1, it is not able to handle topology changes or an altering of application parts during operations.

[1] http://koi-projekt.de/

2.2.3 Networked Feedback Loops

A research field that can not be ignored is research that addresses the question: how feedback loops could be closed using packet based networks and some even the internet. Rüth et al. present a concept that splits the control of an process into two parts, the control synthesis and the actual control of the process itself [43]. The control synthesis is a rather complex task in terms of computation but does not have such hard real-time requirements as the actual control. To fulfill both these requirements, the authors suggest to use cloud infrastructure to execute the control synthesis and to use in-network processing to move the actual control task as close to the process a possible. Even though processes controlled by such a system could be reorganized more frequently, due to the elasticity in the cloud computation resources and in-network deployments of the control tasks, this concept is not applicable to our use case, as it relies heavily on infrastructure.

Another approach to networked feedback loops is to adjust the controller to networks temporary performance. Gallenmüller et al. propose to use gain scheduling, which is a way to choose the controller closest to the current operation conditions[15]. The authors explore several parameters a wireless network needs to monitor to enable the gain scheduling to choose the right controller. Gain scheduling seems to be an interesting option to make feedback loops more resilient to the effects introduced by wireless communication. However, in this work we focus on minimizing these effects.

A work more focused on the coexistence of several feedback loops in one network is presented by Rosenthal et al. In [42] the authors propose a way how several feedback loops can share the limited network resources a shared medium offers. It is based on the idea to give each feedback loop a priority. Each controlled process also has a so called quality of control, the goal is to keep the overall quality of control as high as possible. If the network resources get to limited, the quality of control of the least prioritized feedback loops is degraded first. Rosenthal et al. do not consider changing network topologies. The lack of method to communicate to nodes that are not a part of the control system hinders the interference free inclusion of new nodes.

2.3 Task Cluster Management Operations

The main purpose of the proposed real-time networking system is to run control applications enabling mobile robotic devices to cooperate with each other. Therefore, the network must satisfy delay and jitter requirements needed for these types of control loops. To achieve adaptability in a real-time network, without exceeding the jitter and delay bounds, there needs to be a way to add or remove nodes and tasks from or to the network. While adding or removing tasks can be handled by a scheduler almost by it self, altering the nodes in a network is much more challenging. In our case that means adding and removing nodes to the TC. To do so we define three operations: merge,

split and synchronize. The merge operation is used to add nodes, as this adding of a single node is the same as adding of TC with only one node. The split operation can remove nodes by separating one TC into multiple ones. The synchronize operation is smoothing the transition between schedules.

2.3.1 Merge Operation

A merge operation is a procedure that combines two disjunct TCs to a bigger new one without harming the running real-time communication. If two TCs have to cooperate to fulfill a task, communication between both is required. Cooperation in this case might mean that two TCs make sure that their real-time communication does not interfere. To accomplish this, the network topology has to adapt to the new situation. We assume that only two clusters are involved at once. To merge several clusters the same process can be repeated until all TCs are merged. When two clusters are merged, the following steps are required:

Step 1 - Discovery After mutual detection of the other cluster, both clusters must first assure that the other cluster's schedule does not conflict with its own in terms of used network resources. Conflicts must be resolved by new schedules that do not conflict when clusters move physically close to each other. Second, the nodes from both clusters must ensure that all nodes are in communication range. A merge operation is now initiated.

Step 2 - Subscriptions The information of data a node could publish is forwarded to all nodes in both TCs. The nodes can subscribe to the published data to indicate that the information is needed for future operation. This information is used by the scheduler to plan concurrent transactions on different network resources. Each subscription can define a quality class that limits the maximum jitter tolerable for the control application.

Step 3 - Schedule calculation A new schedule is calculated based on the subscriptions gathered in step 2. The scheduling algorithm tries to fulfill all requirements from the subscribers. If multiple subscribers need the same data, the transmissions are combined to a multicast communication which allows reducing the allocation of scarce network resources. Data without any subscription is not scheduled for transmission. If the requirements cannot be met, an emergency schedule is proposed that fulfills the requirements as much as possible.

Step 4 - Schedule distribution The new schedule is distributed to all participating nodes. In case of an emergency schedule all nodes have to agree to use it. Using an emergency schedule without consensus may harm the operation or even worse lead to damage of material or humans. If a node disagrees, a new schedule is calculated or the current one continues to be the running schedule, in order to maintain a safe system state.

If no new schedule can be found, the control application must define a new way to cooperate that will result in a different schedule.

Step 5 - Switching schedules The new schedule takes effect at a predefined time in the future. To synchronize the clocks of all participating nodes we present a high precision clock synchronization protocol in Chapter 3. Besides that, switching the schedule must occur instantaneously and not introduce additional jitter to the communication. In Chapter 4 we present a protocol stack that meets these requirements.

2.3.2 Split operation

The split operation divides a TC when cooperation between nodes of a TC is no longer needed or possible, e.g., when moving out of transmission range. By just ignoring the resources allocated by tasks that are no longer part of the TC, the system would only need to notify subscribers of these tasks. However, this will waste network resources, as they cannot be allocated again. The difference between a merge and a split operation is that no discovery step is performed and that in both TCs the schedule must be generated separately. In case a node drops out of the TC unexpectedly, the resources can not be reallocated immediately, as it is not guaranteed that this node is not continuing its operation. A future split (or merge) operation will automatically not include a missing node in a new schedule. As the missing node is no longer part of the TC, no other node will subscribe and no transmission will be scheduled.

2.3.3 Synchronization

If clusters are merged, they need to be synchronized to a new shared time. Otherwise the transition to a new schedule cannot be atomic. It is further possible that the slots of the running schedules are not aligned, so the phases of both schedules must be corrected to allow for a smooth transition to the new schedule. Care must be taken to not interrupt running real-time communication. It is favourable to synchronize the nodes implicitly by employing already sent data via piggybacking.

2.4 Components

In this section we define the main components needed to implement the operations introduced in Section 2.3. To give a better overview of the components and the data they exchange, we depict this information in Figure 2.3. We define the components scheduling algorithm, time synchronization and real-time network management protocol, as follows.

2.4.1 Scheduling Algorithm

The scheduling algorithm has to fulfill all constraints imposed by the communicating nodes with respect to available network resources like bandwidth, channels, and possibly concurrent radio links. A subscriber can request certain data with a defined period,

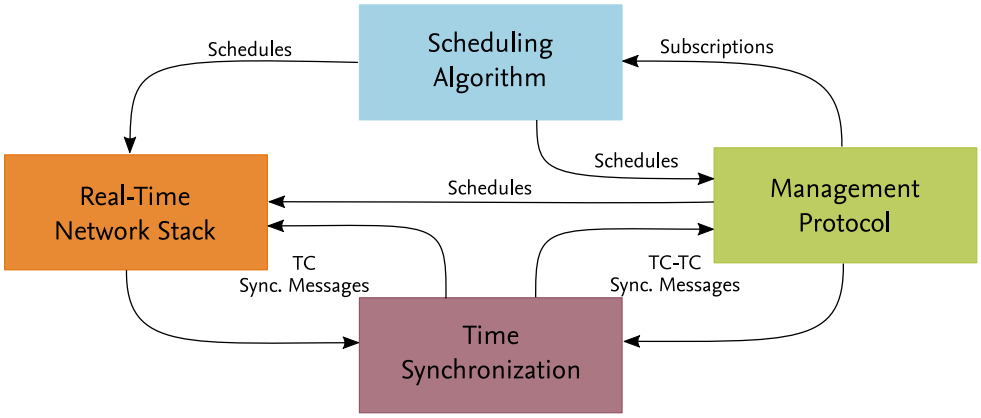


Figure 2.3: Overview of components and their interactions

maximum jitter and maximum delay between the date is created and the subscribing task is scheduled.

In cases where multiple subscribers request the same data but with different periods, jitter or delay bound, the scheduler can use the smallest period and schedule the subscribers in a way that their delay and jitter bounds are met.

The scheduler needs to be able to schedule transmissions in multiple dimensions such as time and channel to fulfill a request with minimum resource usage. For example, it may be more efficient to switch a transmission to a different (frequency) channel instead of waiting for a time slot in the current channel. A channel might be a frequency, a code, a PHY or even a combination of them. Different channels are always assumed to be interference free from each other.

When multiple independent TCs with conflicting schedules meet, the management components trigger a merge operation as described above. During the computation of the new schedule, care must be taken not to violate the jitter and delay guarantees in the old schedules. Therefore, the schedule algorithm must include the old schedules of both TCs in to the computation. Doing so, time slots in the new schedule can align better to the old one resulting in a smoother transition with smaller jitter. In Chapter 5 we introduce two approaches to calculate schedules fulfilling all these requirements, one based on mathematical optimization, the other one based on an heuristic for lower computational complexity.

2.4.2 Time Synchronization

To merge two TCs the time must be synchronized between all node in and across both TCs. This is especially important to be able to switch the schedules a the same point in time on all nodes. In applications without any central instance, the TCs most likely will have different time bases. Therefore, their clocks must be aligned to each other.

As clock alignment cannot be done by solely modifying one TC's clock because this would violate the real-time guarantees of tasks in that TC, both time bases must be modified in a way the respects the real-time guaranties each TC gave.

Time slots of schedules in the TCs to merge may not be aligned to each other, see Figure 2.4. This results in jitter during merging as one TC may need to wait for the other to finish a slot. Low latency constructive interference communication scheme like Glossy [14] are able to synchronize time in large scale networks over multiple hops. By utilizing such an approach several TCs can be synchronized over larger distances and the need to perform a new synchronization can be reduced.

Synchronization between members of a TC can be realized by a protocol based on our Precision Time Protocol approach, presented in Chapter 3.

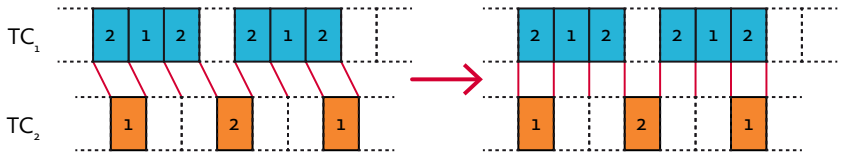


Figure 2.4: Time synchronization of TCs

2.4.3 Real-time Network Management Protocol

To support the functions derived above, a management protocol is needed. The management protocol is used to communicate above TC borders and triggers the operation described above. As this protocol must not harm running real-time communication, it should use distinct communication channels or even different PHYs. Including it into the real-time schedule does not work,TCs have different schedules and time bases. Therefore, management communication of one TC would cause interference in the real-time communication of another TC. The management protocol is also used to handle the publisher/subscriber system and needs to provide the gathered data to the scheduler. The resulting schedule will be distributed in the network using the same protocol.

Figure 2.5 shows a merge operation of two TCs after a new schedule is distributed. Before the merge operation, both TCs are not in transmission range and therefore use orthogonal network resources. Orthogonal network resources means, both schedules are interference free, this might be due to spacial separation or the use of different channels, codes, PHYs. However, when independent mobile TCs move towards each other, they can interfere. Before interference occurs both TCs must switch their schedule to the combined one simultaneously. In Figure 2.5 this time is marked by the red line. The previously orthogonal network resources (left side) from both TCs can be used more efficiently in the new schedule (right side). In this example, some time slots from

TC_2 can fill the unused slots in the upper network resource. To do so, the scheduler needs to exploit the allowed jitter for the task from TC_2.

In order to disseminate new schedules as well as the subscriptions as fast as possible to all nodes in a TC or to all nodes in a TC that is just forming, Glossy-based approaches are applicable. Utilizing the constructive interference capability of such approaches may help to reach other TCs earlier than with just one transmitter. We investigate how applicable Concurrent Transmission approaches are for our use case in Chapter 6.

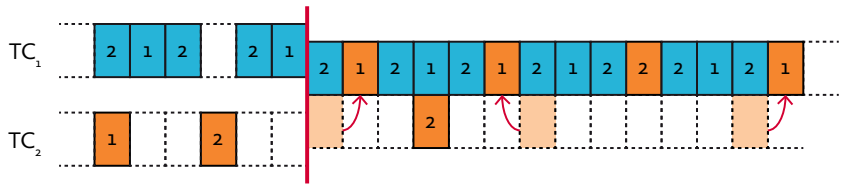


Figure 2.5: Merging of schedules

2.4.4 Real-Time Networking Stack

This is the core element, all the components discussed above are needed to enable this component guarantee flawless real-time communication between all nodes in a TC within the given bounds. To be able to guarantee certain delay bound the network stack be designed in a way that it does not introduce unnecessary delay between the generation of data and its transmission and between the reception of data and its usage. The other goal of the design of such a network stack is to minimize the jitter it introduces into the whole system. Both can be achieved by incorporating the transmission-scheduler into the task-scheduler of the operation system of the node.

Other needed features of the network stack are seamlessly switching from one to another schedule at a predefined time and switching between different network resources without adding any or adding only minimal jitter to the communication.

To fulfill all these requirements the network stack has to implement the whole range of the ISO/OSI-model, from the application-layer other the network-layer down to the MAC-layer and even the properties of thePHY-layer must be considered. By incorporating the transmission-scheduler with the task-scheduler it reaches into the operation system. In Chapter 4 we introduce our real-time network stack, designed to meet the requirements formulated above.

3 Time Synchronization

As we stated in Section 2.4 a precise time synchronization between TC members is needed to synchronize the execution of tasks on several nodes. From the communication point of view the time synchronization is needed to coordinate the medium access in the TDMA communication. Nevertheless, the application of this synchronization is much broader, as it can be used to synchronize actions nodes perform like lifting a work piece with two robots at the same time. Regardless of the application field, communication or synchronization of actions, a more precise time synchronization will lead to less errors. The most important task of the time synchronization in a TDMA communication is to ensure that all nodes have the same slot start-point, end-points and duration. A more precise synchronization enables smaller guard-times in between transmissions and thereby increases the number of messages that can be exchanged in a certain amount of time. Besides the actual transmission of data the guard-times are the limiting factor to the Round Trip Time (RTT). The time for data transmission depends only on the amount of data and the rate at which the PHY can transfer it. As the amount of data is application depended, reducing the guard-time is an important goal.

We stated above that time synchronization is not only important for the communication stack but also for the application a TC has to fulfill. In recent years multiple research projects and also industry standards approached wireless monitoring of factories or processes [36, 24, 49, 1]. The next step is wireless control of such factories and processes. To do so feedback loops for controllers must be closed over wireless links. Therefore, the time synchronization must be precise enough to allow feedback controllers to be spread over several wireless connected nodes. To accomplish that, several sub-microsecond synchronization schemes were proposed [32, 28, 14]. Most of these protocols do neglect that clocks on modern micro-controllers are drifting compared to each other. This is due to the combination of low-cost oscillators and the Phase Locked Loops (PLLs) used to gain higher frequencies from these oscillators. Both are introducing an error to the clock rate of the micro-controller that lead to the drift between clocks on different nodes.

In this chapter we present a time synchronization protocol based on the Precision Time Protocol (PTP) which we optimized for wireless connections. We focus on the accuracy of synchronization by compensating the clock drift of the nodes. As we discussed in [61], where we first introduced this method, this time synchronization protocol

enables the communication stack to minimize guard times and allows to close feedback loops over wireless links.

3.1 Related Work on Time Synchronization

In this section we give a brief overview of sub-microsecond synchronization protocols for wireless networks. Further, we describe PTP briefly to give the reader a background for the remainder of this chapter.

3.1.1 Precision Time Protocol (PTP)

PTP is a time synchronization protocol aiming at Local Area Networks (LANs) and providing microsecond accuracy, other than Network Time Protocol (NTP) which is used in the Internet and provides millisecond accuracy. It is standardized as IEEE 1588/IEC 61588 [21]. Although the protocol does not limit the medium to Ethernet, almost all implementations use it. But there are also working implementations using IEEE 802.11, for example [25]. When the protocol is started, the Best Master Clock (BMC) algorithm is started. Every node announces the capabilities of its clock in order to choose the best one among them as the master clock.

The messages exchanged to synchronize the clocks between the master and its slaves are depicted in Figure 3.1. The master occasionally multicasts a `SYNC` message to its slaves. The slave then records the reception timestamp t_1 as early and close to the hardware as possible. To do this, either a hardware timestamping unit inside the Network Interface Controller (NIC) can be used if available, or this can be done in software in the MAC layer. The master has to prepare the message and write the actual timestamp into the `SYNC` message at that time. Due to processing time, medium access, and propagation delay, sending timestamp t_0 is not precise. To mitigate this issue, a `FOLLOW_UP` message is sent afterwards. It contains the exact sending time as it has been recorded by the hardware or, if not available, by the Interrupt Service Routine (ISR) processing the TX-complete interrupt. The choice between hardware timestamping (if available) and software timestamping has a large impact on the possible precision. According to [10], software only solutions reach a precision of 5 to 50 μs, while hardware supported implementations reach a typical precision of $\pm 60\,ns$. Having now acquired both timestamps t_0 and t_1, the slave can calculate its offset to the master. Yet, this has the error of the propagation delay τ_{prop} which has to be subtracted in order to get the correct value. To determine τ_{prop}, the slave sends a `DELAY_REQUEST` message to the master. This answers with the reception timestamp t_3 in the `DELAY_RESPONSE`. Now knowing all the relevant information and assuming a symmetric delay, the slave calculates its propagation delay to the master. Both Ω and τ_{prop} are noted and the slave can correct its clock to be in sync with the master. Where Ω is the offset of the slave's clock.

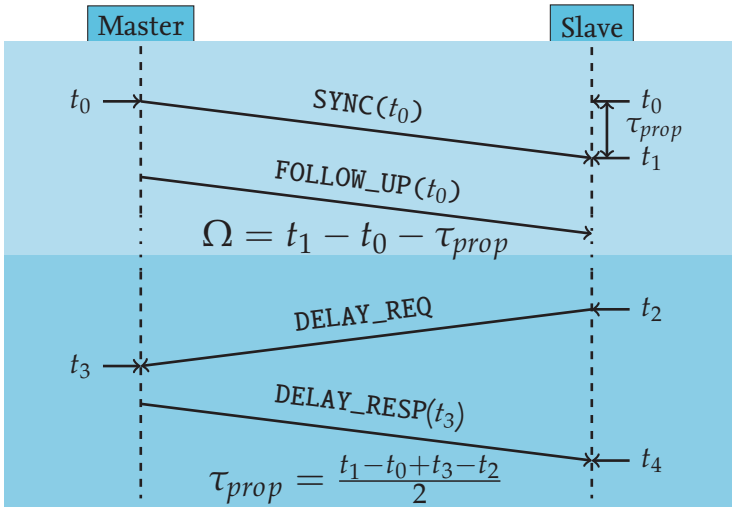


Figure 3.1: Messages exchanged in PTP

3.1.2 Glossy

Glossy is an efficient network flooding protocol proposed by Ferrari et al. [14]. Due to its special flooding mechanisms it also brings a synchronization protocol. The idea in Glossy is to forward certain packets as fast as possible without further processing. Thus, the time a node needs to forward a packet is constant and the error added in a multihop network can be predicted. Another advantage of constant packet processing times is the ability to transmit the packets by different nodes at the same time. This is only true if all nodes transmit the same data. Due to this ability there is no need for a MAC for this packet, this eliminates the delay introduced by clear channel assessments. Ferrari et al. do not propose a protocol for time synchronization but an implicit synchronization based on packets transmitted for other reasons. In their evaluation Glossy was able to synchronize a network with 8 hops with a mean signed deviation of 0.4 μs.

Although Glossy seems to be an elegant approach to time synchronization in wireless networks it suffers from some conceptional disadvantages. Due to the elimination of clear channel assessments it is not able to coexist with other networks in the same area and frequency. Another disadvantage is the need of precise crystal oscillators to meet 0.5 μs slot accuracy with all nodes that should transmit the same packet.

3.1.3 TPSN

TPSN [16] was one of the first time synchronization protocols designed for wireless networks. It works in two phases: first the fixed hierarchical structure is build up across the whole network. In the second phase a two-message protocol is used to synchronize the nodes pairwise among the edges. Thus, the synchronization propagates through the network. In comparison to Glossy, TPSN uses unicast messages, not broadcast,

to synchronize the pairs. The authors state that TPSN synchronizes a network with a precision of 20 μs, which does not meet the sub-microsecond barrier.

3.2 Time Synchronisation Protocol

In this section we describe our optimization to PTP and the design decisions that led to them in detail. As our main goal was to meet the requirements of wireless networked feedback loop controllers, we focused on the synchronization accuracy of our protocol. To do so, we had to put other possible goals out of focus. As discussed in Section 2.1 we assume that all nodes of one TC are in transmission range of each other. Therefore, multihop connections are not considered in our protocol. As a PHY we use the DW1000 by DecaWave [12]. This chip was designed for indoor localization applications and, therefore, has an internal clock with a resolution down to several pico seconds and other features that we use to synchronize the time on different nodes. Its transmission range of up to 250 m supports the assumption that multihop connections are not needed [12]. It is quite unlikely that a feedback loop controller is spread over more than 250 m and still needs a sub-microsecond synchronization. The DW1000 implements the Ultra Wide Band (UWB) PHY of IEEE 802.15.4, which uses a frequency band from 3 GHz to 6 GHz.

As explained Section 3.1.1 in PTP is a time synchronization protocol designed to work in wired LANs. It supports two mechanisms of timestamping packages: software timestamping by the Operation System (OS) kernel and hardware timestamping by the NIC. The DW1000 supports hardware timestamping of received packages, therefore we utilize this feature in order to adapt PTP to our requirements and thereby achieve an improved performance.

Like PTP our protocol has one master and several slaves. The master is used as the reference for all its clients and has to know which nodes are its slaves. This knowledge is distributed by the management protocol.

To initiate a time synchronization the master sends a `SYNC` message via multicast to its clients. This is done periodically to keep all nodes synchronized. The slave records the timestamp of the reception (t_1) with the timestamping unit in the DW1000. This guarantees a minimum error in the timestamping. In PTP the master would send a `FOLLOW_UP` message that contains the timestamp of the transmission (t_0) of the `SYNC` message. We do not need this `FOLLOW_UP` message, as we utilize the delayed transmission feature of the DW1000. To do so the master prepares the `SYNC` message with the timestamp t_0 when it wants to transmit this message and hands it over to the DW1000. Afterwards the DW1000 is advised to transmit this message at this exact timestamp. The DW1000 will transmit the message at that point in time without any further interaction with the micro-controller. By utilizing a TDMA network protocol as described in Chapter 4 the medium can be assumed free at this time, as the synchronization messages are embedded into the schedule.

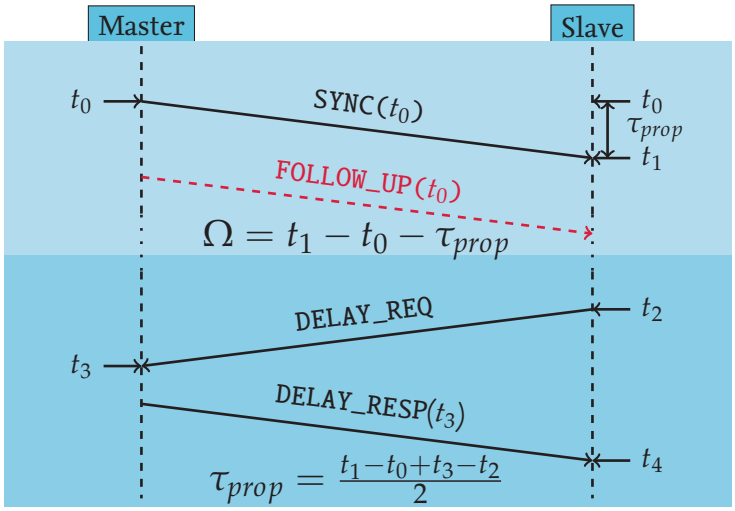


Figure 3.2: The proposed time synchronization protocol, the red FOLLOW_UP message is removed because it is not necessary due to precise timestamps.

It needs to be ensured that t_0 is far enough in the future to finish all necessary preprocessing of the SYNC message until t_0. This timestamp is precise due to the fact that the clock triggering the transmission is running inside the DW1000 and all computation needs be done beforehand [12]. In Figure 3.2 the removed FOLLOW_UP message is marked in red and dashed.

As the slave has recorded t_1 and read t_0 form the SYNC message, it calculates its offset Ω to the master.

$$\Omega = t_1 - t_0 - \tau_{prop} \tag{3.1}$$

The propagation delay τ_{prop} still needs to be determined in the way originally intended in PTP. Thus, DELAY_REQ and DELAY_RESP messages are exchanged just like in PTP.

3.2.1 Master Selection

In a TC nodes might have heterogeneous clock accuracy, in this case PTP's Best Master Clock algorithm can be implemented. For this work we used STM32f407 microcontrollers without an external reference clock. As it can be assumed that all clocks are of the same accuracy in such a homogeneous network and without a way to determine the accuracy of each node's clock, we had to implement our own master selection.

After a node is started, it listens for SYNC messages for a certain time. If that node does not receive any SYNC messages while it is listening, it starts transmitting SYNC messages and is therefore the master. If, on the other hand, a SYNC message is received the node becomes a slave by using the information in the received message. To resolve the conflicts in case several nodes start at the same time, all nodes will prefer the SYNC message with the lowest destination address.

3.2.2 Drift Compensation

As we stated before, PLLs and low cost oscillators add an individual error to each node's clock. This results in a drift between the clocks of these nodes and finally a drift between timed task executions on theses nodes. The magnitude of this error is determined in Section 3.3. This section describes how we mitigate the drift between nodes in the network. We designed our drift compensation in a way that does not require any extra communication between the nodes, as described in [61].

All synchronization clients measure the time span (ΔT_{Rx}) between the reception of two SYNC messages. This is the time between two instances of t_1 from Figure 3.2. Equation (3.2) gives the this time span with n as a SYNC message interval counter.

$$\Delta T_{Rx} = t_1(n) - t_1(n-1) \tag{3.2}$$

To calculate the drift to the clock master, the client compares ΔT_{Rx} with the time span between the transmission timestamps (ΔT_{Tx}) of the SYNC messages.

$$\Delta T_{Tx} = t_0(n) - t_0(n-1) \tag{3.3}$$

Where t_0 is the transmission time stamp as in Figure 3.2 and n is the SYNC message interval counter. The difference (ΔT) between ΔT_{Rx} and ΔT_{Tx} gives the drift of the two clocks over the SYNC message interval. A ΔT closer to zero means less drift between the clocks.

$$\Delta T = \Delta T_{Rx} - \Delta T_{Tx} \tag{3.4}$$

3.3 Evaluation

In this section we evaluate the accuracy of our time synchronization protocol. All experiments were performed in the setup shown in Figure 3.3. To have reproducible results throughout the whole evaluation we used $Node0$ as the master node. $Node0$ generates a reference signal by toggling a GPIO which is wired to a GPIO of all clients. This signal has a frequency of $1\,Hz$. The clients,$Node1$ and $Node2$, have an interrupt enabled on the GPIO the signal is wired to. Every time $Node0$ toggles the GPIO it records the timestamp. These timestamps are collected by a logging PC that is connected with the nodes via Universal Asynchronous Receiver Transmitter (UART). At $Node1$ and $Node2$ a timestamp is recorded each time the interrupt is triggered by the signal. After collecting these timestamps on the logging PC they can be compared with the ones from the master. The wired trigger signal was chosen to minimize jitter and delay in the reference.

All communication needed to synchronize the clocks of the nodes is wireless. The wired connections are not used to benefit the synchronization of the nodes in any way.

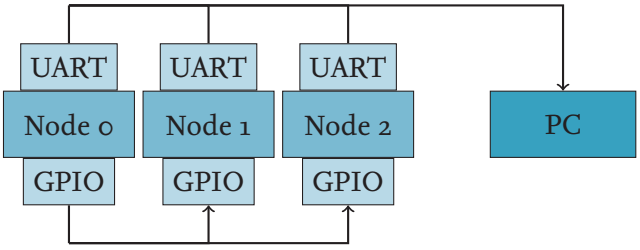


Figure 3.3: Measurement setup used to evaluate the clock synchronization

3.3.1 Ground Truth

In order to quantify the benefit achieved by our synchronization protocol between our evaluation nodes we take a ground truth measurement of the drift between the clock of *Node*1 and *Node*2 relative to *Node*0 without any synchronization. The results are shown in Figure 3.4.

From the results of the experiment we get two main observations: first the drift of the clocks is constant over time, as Figure 3.4 shows two straight lines. The second observation is that the drift differs a lot between nodes, *Node*2 drifts almost twice as fast as *Node*1. In our experiment we measured a relative clock error of 1.6 µs/s for *Node*1 and 2.8 µs/s for *Node*2. The results shown in Figure 3.4 underline the need for

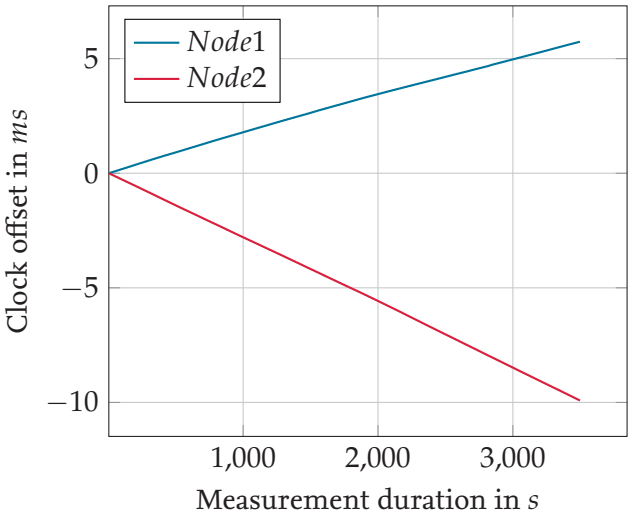


Figure 3.4: Clock drift measurement without synchronization. The clock of *Node*1 dirfts by 1.6 µs/s,the one of *Node*2 by 2.8 µs/s.

a drift compensation to achieve a sub-microsecond synchronization. As the drift of *Node*1 is 1.6 µs/s and for *Node*2 even worse with 2.8 µs/s, a `SYNC` message interval of less than one second would be necessary for the needed accuracy.

3.3.2 Time Synchronization without Drift Compensation

To evaluate how accurate our synchronization is without the drift compensation we used the same setup as in the ground truth measurement but enabled our synchronization. In order to limit the overhead of the synchronization to a reasonable amount we chose 1 *Hz* as the synchronization frequency. As Figure 3.5a shows the drift is still notable but it is around a constant offset of 0.5 µs. The measurements form a saw-tooth like shape. This is due to the fact that every `SYNC` message brings the clocks in sync and afterwards they are again drifting apart. The overlapping of the saw-tooth signal is due to the low sampling rate of 1 *Hz*. The measurements between the saw-tooth slopes most likely result from small variations in the sampling frequency. We only show a five minute slice of the half hour experiment to make the figure more clear. For *Node*2, shown in Figure 3.5b, the results are similar to *Node*1 but with a smaller offset.

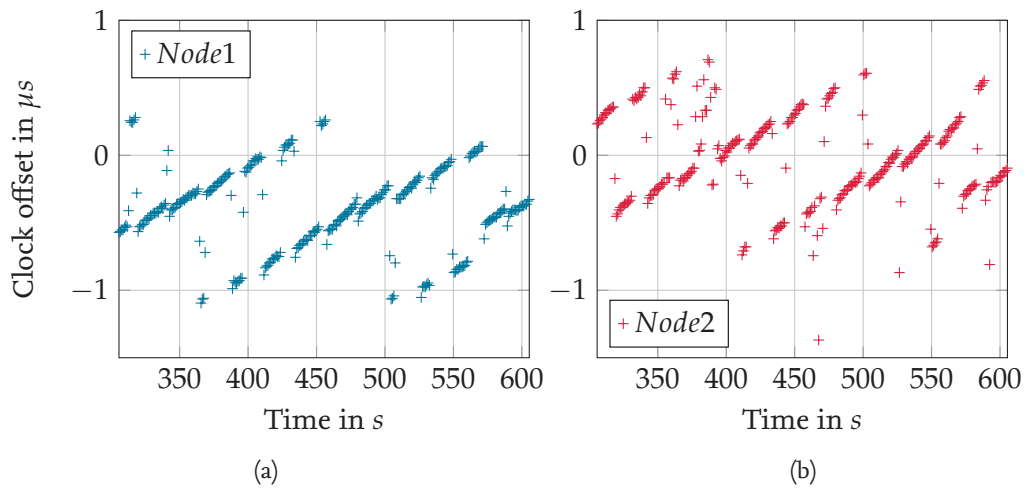


Figure 3.5: Exemplary five minutes slot of the 30 minute clock drift measurement with clock synchronization. The drifting clocks are still notable by saw-tooth like shape of the measurements

The mean absolute offset of the synchronization is 0.54 μs for *Node*1 and 0.33 μs for *Node*2. As the mean absolute offset is below one microsecond our system is able to synchronize drifting clocks in a sub-microsecond manner. The Standard Deviation (SD) of the offset is 0.411 μs for *Node*1 and 0.406 μs for *Node*2. Although the mean absolute synchronization error is below 1 μs there are still some outliers that exceed the 1 μs boundary. The saw-tooth shape means that tasks close to a `SYNC` message are better synchronized than the ones more far away. Tasks right before a `SYNC` message suffer from the biggest synchronization error which still might be more than 1 μs.

3.3.3 Time Synchronization with Drift Compensation

The results in Section 3.3.2 show that a our synchronization is able to synchronize nodes on a micro-second level. Never the less, the drifting clocks inhibit that the synchronization stays in the micro-second range for the whole measurement. To overcome this drift we use our proposed drift compensation from Section 3.2.2. The following evaluation uses the same parameters as the prior one but has the drift compensation enabled. As Figure 3.6 shows, there is no more slope in the clock offset, therefore the clock drift is compensated by our proposed drift compensation. This result is supported by smaller mean absolute offsets: 0.19 μs for $Node1$ and 0.22 μs for $Node2$.

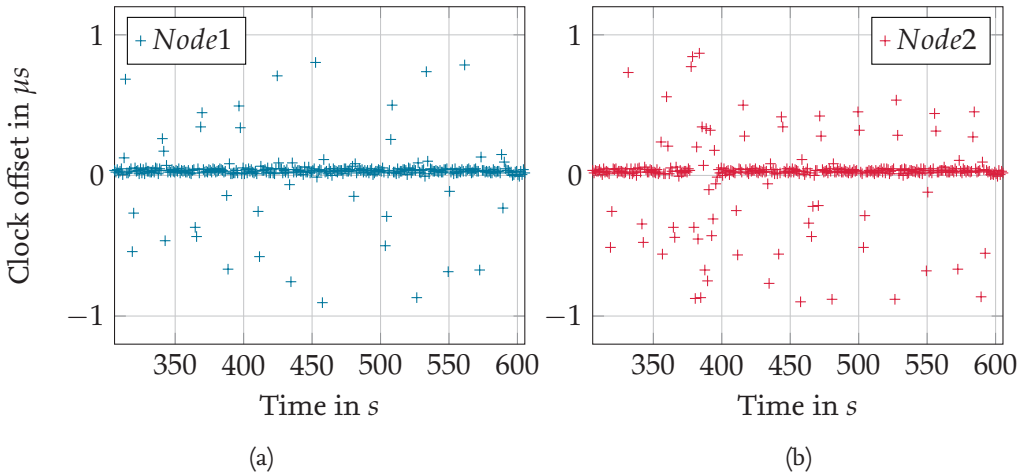


Figure 3.6: Exemplary five minutes slot of the 30 minutes clock drift measurement with clock synchronization. The drifting clocks are compensated and most measurements lie on a horizontal line near zero.

In Figure 3.6b there is time span between 370 s and 400 s with a bigger synchronization error. As this happened to both nodes in irregular periods and with various duration, we assume this to be caused by packet loss due to interference.

3.4 Conclusion

The evaluation showed that our synchronization protocol is able to synchronize nodes on a sub-microsecond level. We achieved this by utilizing the features of our hardware and were able to reduce the amount of messages needed for synchronization compared to PTP. By introducing a drift compensation we made our approach applicable for nodes with low-cost crystal oscillators. The mean absolute error was 0.19 μs. With such an accurate synchronization guard times in the TDMA communication can be reduced to a minimum. The other advantage of an accurate synchronization are smaller errors in the timing of feedback control loops which will result in smaller errors.

4 Real-Time Networking Stack

In this chapter we present our design of a TDMA network stack, capable of maintaining real-time communication even if the topology changes. As discussed in Section 2.1 maintaining real-time communication gets much more challenging if parts of the network's topology or parts of the network's traffic patterns change. To adapt to such changes the TDMA network stack needs to be able to apply a new schedule with no or very limited extra delay and jitter. In the following we discuss the abilities of several state of the art network stacks, focusing on their ability to adapt to changes in the network. Afterwards, we detail the architecture of our network stack, as presented in [62], in Section 4.2. Section 4.3 evaluates how well the network stack performs in stable network and also how well it adapts to changes.

4.1 Related Network Stacks

To structure this section, we separate industrial standards from academic research projects in two section. This is to give a better understanding what the industry is working with at the moment and what academic research is proposing to overcome future challenges.

4.1.1 Industrial Standards

ISA100.11a [1] is an industry standards that is based on the low-power wireless standard IEEE 802.15.4 [23]. In contrast to IEEE 802.15.4, ISA100.11a incorporates TDMA to achieve better real-time capabilities than IEEE 802.15.4 in its beacon enabled mode. It is not applicable to factory automation applications as its time-slots are longer than 10 ms [59].

WirelessHART is the wireless extension the Highway Addressable Remote Transducer (HART) standard. It specifies a self organizing mesh network that connects field devices via a robust, wireless communication stack. WirelessHART is based on IEEE 802.15.4 with a TDMA protocol that has fixed 10 ms slots. This TDMA protocol substitutes the token passing mechanism in HART. In terms of mobility, WirelessHART does support mobile handheld devices for diagnostics. Yet, it is not designed to support feedback loops while the network topology is changing [26]. WISA is the only standard designed to be used in factory automation [59]. It utilizes IEEE 802.15.1 [22] as a

PHY and achieves time-slots lengths below 10 ms. Therefore, it is able to close feedback loops over its wireless connections. By aiming for the highest possible timing accuracy and shortest time-slots, the adaptability of WISA is very limited. WISA is mostly used in fixed production cells with a fixed number of nodes and a fixed task to fulfill.

4.1.2 Other research

The MARS system by Dezfouli et al. [9] 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 [26] 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. [51] published a mobility tolerant TDMA-based MAC-protocol for 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. [13] published their Low-power Wireless Bus (LWB) which builds upon "Glossy" [14]. 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 [13].

The GINSENG project [36] 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.

4.2 Architecture

As the discussed state of the art shows, all approaches lack either the ability to guarantee timings accurate enough to close feedback loops over their links or they lack the ability to adapt to changes in the network's topology, needs of the applications or the environment. To combine both abilities we propose our TDMA network stack, that is

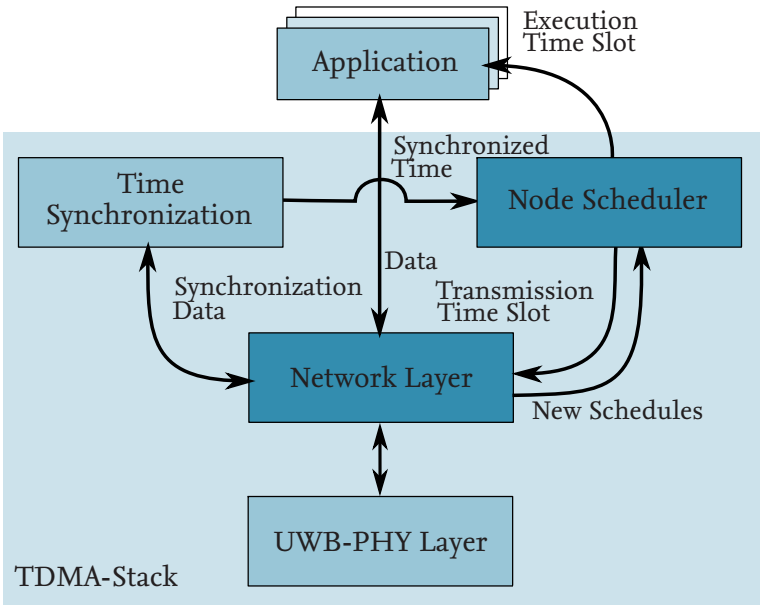


Figure 4.1: Network stack design overview with the information flows between its components

on the one hand flexible enough to allow the changes that occur in the applications but still is strict enough on its timings that feedback loops can be closed. In this section we describe the architecture of our TDMA network stack. Figure 4.1 depicts the different components and the flow of information between them. The proposed network stack is an enhancement to the Free Real Time Operating System (FreeRTOS), all its components, but the *Node Scheduler*, are implemented as standard FreeRTOS tasks.

4.2.1 Application Layer

We decided not to introduce a special layer for tasks that need to access the TDMA stack. The reason was to keep the application development as simple as possible. All tasks registered to FreeRTOS can use the stack. The only restriction a developer has in the design of a task is timing wise. Each task has to finish its execution in its assigned time-slot and needs to provide the data to the network stack. Otherwise the network stack is not able transmit the data to the receivers.

4.2.2 Time Synchronization

To synchronize the time on all nodes in a TC we use the synchronization protocol presented in Chapter 3. The synchronization never modifies any system timer, this way timed executions are never lost because the time was reset ahead the time of that executions. To synchronize the time on several nodes, an offset (Ω) and drift factor (ΔT) to the master's time is calculated as described in Section 3.2 and Section 3.2.2 accordingly. The drift factor is multiplied with the time span between the current time and the

planed execution time, the result gives the time span at the node. The offset is added to this time span, this gives the time the node has to execute the task.

$$T_e = (T_n - t) \times \Delta T + \Omega \tag{4.1}$$

Where T_e is the execution time at the node, t is the current time at the node and T_n is the synchronized time. Thus, all nodes will execute their tasks at the right time according to the master's time, even though their system timers might tell a completely different time. The offset and drift factor is updated each time the a synchronization is performed.

We implemented the time synchronization as a FreeRTOS task, therefore it can be embedded into the TDMA schedule of the TC. The frequency of re-synchronization has to be adjusted to the application's needs.

4.2.3 Node Scheduler

The *Node Scheduler* is responsible to take the schedules generated for the whole TC by the scheduling algorithm discussed in Chapter 5 and break it down into a schedule used on the node it is executed on. Further, it has to ensure that all tasks are executed at the right time and are stopped if they exceed their limit.

To achieve the most accurate timing for the execution of the tasks we had to replace the FreeRTOS scheduler. The standard FreeRTOS scheduler is working with 1 ms time slices. As it operates on the FreeRTOS system timer, which is not modified by our synchronization, the timing accuracy could not be higher than 1 ms.

Another reason to exchange the FreeRTOS scheduler was, that it works purely on the priorities of the tasks. It schedules the task with the highest priority that is in the so called `READY`-state and therefore waiting for execution. Such a model works well on isolated nodes where not all tasks are executed periodically and some are more important to be executed than others. In our applications all task need to be executed in order to fulfill the need of the application, as we discussed in Section 2.1.

To synchronize the medium access of all nodes in a TC and to achieve a smaller jitter between the aggregation/processing of data and its transmission/reception, we chose to combine the communication and processor scheduling. This way we can guarantee timings between the, e.g., measurement of a value and its transmission and also between its reception and its processing. Therefore, timings between the aggregation and the processing can be guaranteed which is necessary to be able a close a feedback loop over a wireless link.

How such a combined schedule looks like is depicted in Figure 4.2. It shows two rounds of the same schedule which in this case consists of three slots. We implemented the TDMA stack on the same hardware as the time synchronization presented in Chapter 3. Therefore, we can use the delayed transmission feature of the DW1000 to transmit

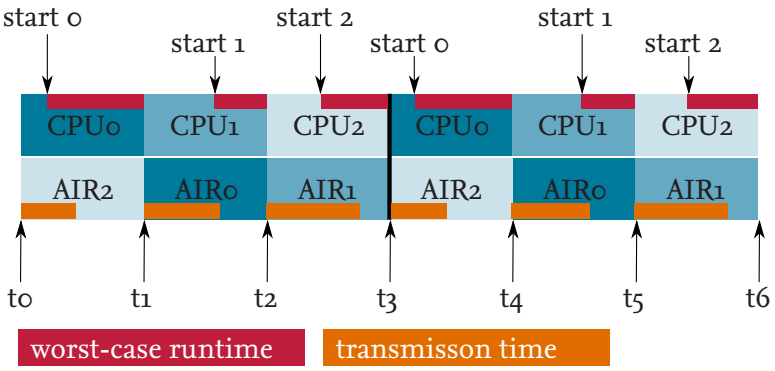


Figure 4.2: Example of a combined schedule of three tasks and two rounds. The execution times of the tasks are shown in red while the duration of the packet transmission is marked in orange. *Task*0 is executed in the slot labeled with *CPU*0, its data is transferred in the slot labeled *AIR*0. *Task*1 uses the slots *CPU*1 and *AIR*1 and so on.

packets at an exact point in time. By utilizing this feature for every transmission we free the CPU from any work related to the transmission during the transmission time. Therefore, we start the execution of a task before the transmission of the prior task's data is finished. This can be seen in Figure 4.2 in the first time-slot, the transmission of *task*2's data starts at t_0 and lasts as long as the orange bar, the computation of *task*0 starts at $start_0$, its worst case execution time is represented by the red bar.

A task that is supposed to transmit its data in the second time-slot will be scheduled to do its computation in the first time-slot. Therefore, the computation time-slot `CPU0` of *task*0 is the one time-slot right before its transmission time-slot `AIR0`, where the data of *task*0 is transmitted. To minimize the delay between the computation and transmission of data even further the computation does not start at the time-slot's start. It is timed based on its worst-case runtime before the end of the computation time-slot. That way it is guaranteed that the data is ready to be transmitted at the beginning of the transmission time-slot and that it is as recent as possible. To keep application development simple we use the vanilla FreeRTOS context switches and its Inter Process Communication (IPC). Our node scheduler notifies the task that is about to be executed next and sets it into `READY`-state. Afterwards, that task is executed at the time it is timed to.

To set the right task into `READY`-state the node scheduler must be executed at the beginning of each time-slot. The node scheduler is called by an ISR right at the time-slot start, then it calculates the start time of the task to be executed and sets a timer to wake up the task at that time from another ISR. In Figure 4.3 both ISRs are shown, it also shows t_I the start of the time-slot and t_{II} the start of the execution of the $Task_i$. The node scheduler also calculates t_{IV} the transmission timestamp for the packet which $Task_i$ might transmits.

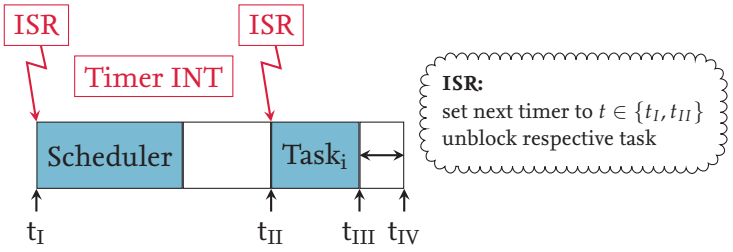


Figure 4.3: Procedure of $Task_i$ in its slot. At t_I the scheduler calculates the following slots, t_{II} is the defined transmission time t_{IV} minus the worst-case execution time of $Task_i$.

As t_{II} is calculated by Equation (4.2), where w is the worst-case runtime of the task, this worst-case runtime must be given by the application developer (or otherwise determined).

$$t_{II} = t_{IV} - w \tag{4.2}$$

Due to the fact that the runtime of tasks might differ for each execution, t_{III} might differ as well. By utilizing the delayed transmission feature of the DW1000, varying runtimes do not effect the transmission time. As long as the data is ready before the pre-calculated transmission timestamp, the radio ensures that the data is transmitted at that timestamp.

By combining the communication and processor scheduler into the node scheduler, we achieved the goal of minimizing the delay and jitter which our communication stack adds to a feedback loop. To achieve the goal of allowing our TDMA network stack to adapt to changes of every property of the network's configuration, it must be able to switch the schedule during runtime with the least additional jitter possible. Switching the schedule must happen at the same time on all nodes in a TC, otherwise interference and packet loss will occur, as several nodes might transmit in the same time-slot. This has to be avoided by all means. We use a designated message to signalize at what time all nodes have to 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 4.2, t_0, t_3 and t_6 are valid transition times.

4.2.4 UWB-PHY Layer

As mentioned in Chapter 3 we are using the DW1000 as our transceiver chip, it implements the IEEE 802.15.4 UWB PHY. UWB is more robust against narrow-band interference than communication standards utilizing narrow-banded channels. Such narrow-banded interference are common in factory automation scenarios due to electro magnetic emmissions originating from welding machines, generators etc.

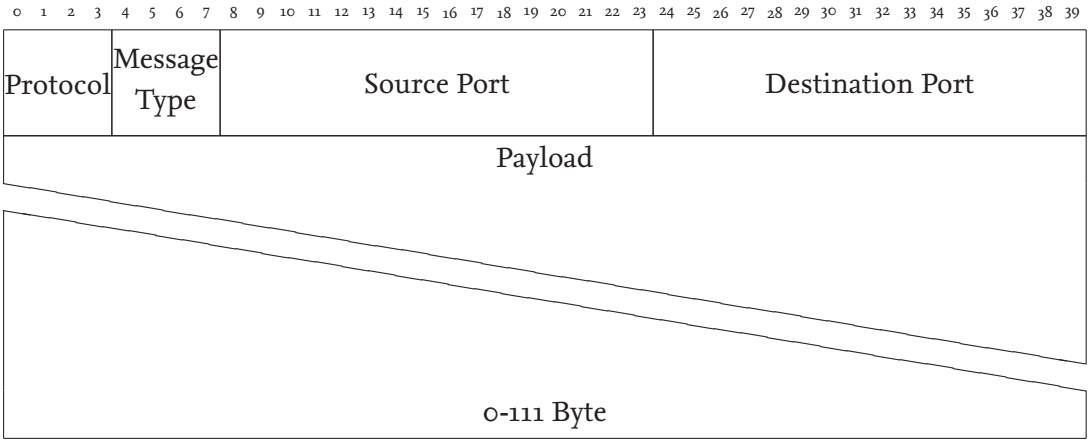


Figure 4.4: The message format of our network layer

The DW1000 was originally designed for indoor localization applications, to perform accurate time of flight measurements it is equipped with a high resolution clock. This clock runs with a resolution of 15.65 ps and is able to timestamp packet reception with this clock. An even more useful feature is the delayed transmission which uses the same clock to trigger the transmission of a packet that was previously loaded into its packet-buffer and configured to be transmitted at that time. We use both of these features heavily, the latter one allows to reduce the guard times between slots and to free the CPU during the transmission. The reception time-stamping is used in the time synchronization discussed in Chapter 3.

4.2.5 Network Layer

Figure 4.1 shows the network layer as the central instance which organizes all communication of that system. It provides the connection between all the components described above and controls the transmission and reception of all tasks. In this section we describe how packets are received by the transceiver and how they are passed to the tasks that are subscribed to the data included in that packet. Further we describe how tasks hand data over to the transceiver so it transmits them.

Whenever a task is scheduled to use the CPU, it is allowed to transmit data at the beginning of the next time-slot. The data has to be passed to the network layer before the current time-slot ends, so the network layer can prepare the packet including this data. Do to so the network layer needs some additional information, which is used to format a packet like shown in Figure 4.4. That newly constructed packet is then encapsulated in an IEEE 802.15.4 frame to stay compliant to the standard. Together with the transmission timestamp this frame is than given to the PHY, described in Section 4.2.4.

The protocol field is used to distinguish whether a packet is used for time synchronization or to transmit data generated by an application of the TC. For there are only these two protocols defined. To have the opportunity to add more protocols in the future we reserved a few extra bit in this field. To distinguish different messages for different purposes in a protocol we use the message type. For now four types have been defined:

- **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 time 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.

As we stated in Chapter 2 we use a publish-subscribe mechanism to decide which data needs to be delivered to which task. To make sure all subscribers receive the packet at the same time, all packets are sent to the broadcast address of IEEE 802.15.4. All task in a TC have a unique source port that they use to publish data. The destination port is a number to which tasks subscribe to, this enables the network layer to deliver the same data to different tasks at the same time without additional communications. Being able to deliver the same data at the same time to different tasks on different nodes is especially handy if several task need data from the same task and need to act according to this data simultaneously.

One drawback of this mechanism is that it needs a large centralized packet buffer on each node. This buffer has to store the data until each subscribed task had the opportunity to process the data. To keep memory usage and processing time as constant as possible we use a ring buffer to store all received packets. Using dynamically allocated memory could lead to an unpredictable runtime of the allocation. As the whole design of this network stack is based on the idea of predictable runtimes in any component, dynamically allocated memory was therefore not an option. The space in a ring buffer has to be defined at compile time, therefore a central buffer would introduce the risk that data was overwritten before it was processed by all subscribers. To overcome this we use a separate ring buffer for each destination port. The size of each ring buffer can be chosen by the developer of the application. That way it is up to the developer to decide how many old instances of the data are needed and after how many new instances the old one is overwritten. Another advantage of our ring buffer concept is that the system cannot be affected by a lack of memory caused by tasks that do not free the buffers after they processed their data.

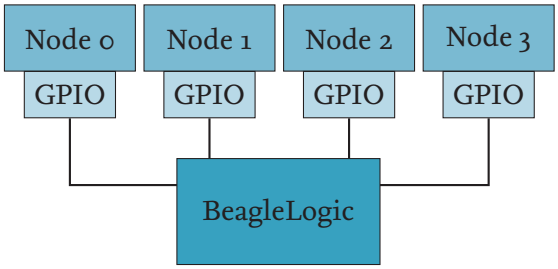


Figure 4.5: Evaluation setup with 4 nodes connected to the BeagleLogic to measure task executions timings

In order to subscribe to a certain port, tasks need to register to the network layer for this port. Packets received with this destination port are copied into the ring buffer. Afterwards, the tasks are notified that new data is available. When the task is executed the next time, it can process the payload of the received packet.

To publish data into the network, a task has to notify the network layer about the address and the length of the data. Additionally, destination port, protocol and message type need to be provided. The source port is included by the network layer based on the task it is notified by. The transmission timestamp is provided by the node scheduler for that task.

4.3 Evaluation

To evaluate the performance of our system we chose two main metrics: the timing accuracy of task execution between different nodes and the packet loss the system suffers from under different configurations. The first metric shows how much additional jitter and delay a feedback loop has to expect if it is closed via our TDMA network stack. The second gives an estimation on the reliability of links between nodes in a TC.

4.3.1 Evaluation Setup

To measure the timing accuracy we used a network of four nodes, one master and three slaves. The example application is to simultaneously set a GPIO 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 GPIO and let all clients act in the same slot.

The timings were measured by BeagleLogic[1] connected to the GPIO of all nodes. To have a good resolution of the different timings the BeagleLogic measured with a sample rate of 100 MSps. The setup is shown in Figure 4.5.

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

To make the results of all timings evaluations comparable, we use schedules of four slots of 5 ms, so a frame is 20 ms long.

4.3.2 Single Node Timing Accuracy

The easiest form of a feedback loop is one that is closed on a single node, so no network communication is needed. As one would expect this feedback loop to perform best in terms of timing accuracy, we evaluated this case first. The only impacts our system has to such a feedback loop are the time synchronization and the node scheduler. The time synchronization might re-synchronize between two executions and therefore add some jitter. The node scheduler is the component that manages when a task is executed, therefore, the evaluation of this use case gives us insight to the timing accuracy of the node scheduler.

In this evaluation we measured the time between two executions of the same task on one node. To get a better understanding how different master nodes influence the performance of the whole system, we executed this evaluation with each node as the master node.

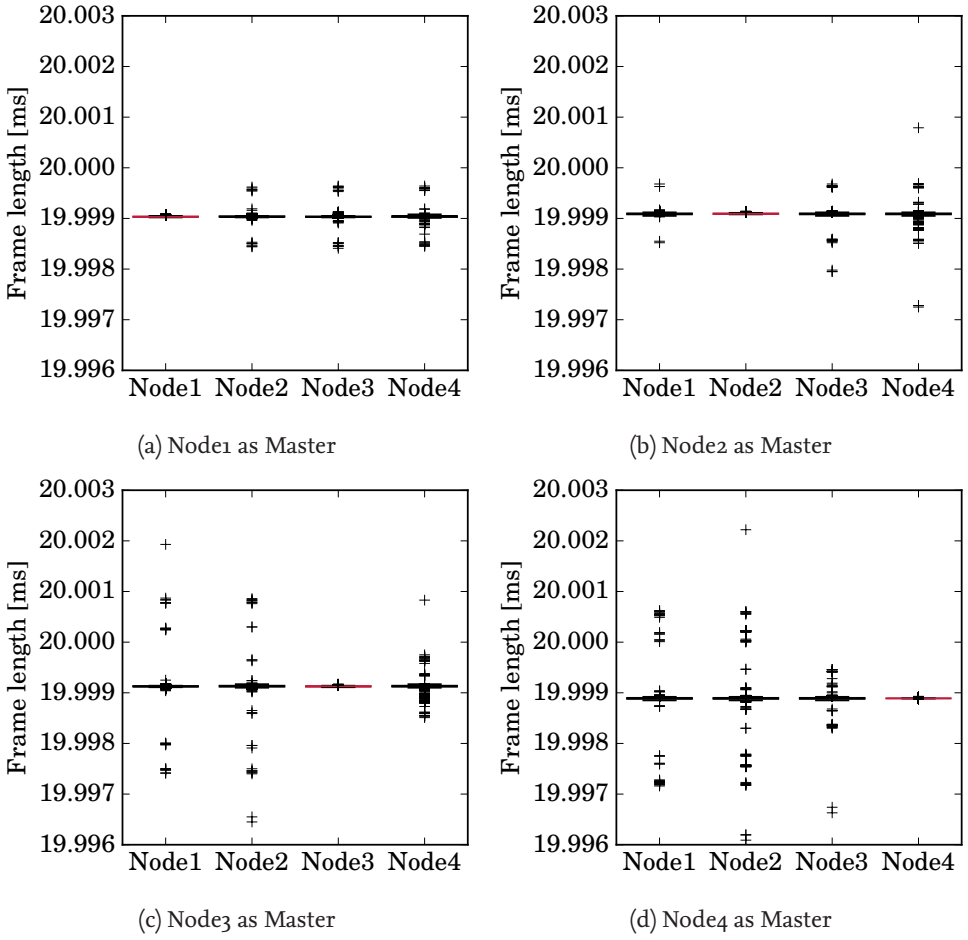
Figure 4.6 depicts the results to this evaluation. Each of the four figures shows another combination of master and slave nodes. The y-axis give the time between two consecutive executions of the same task and therefore the length of a frame. In all master-slave combinations the master has the best timing accuracy. The best performance has the network with $Node1$ as the master, the jitter is only about 1 µs. The worst performance shows $Node4$ as a master, the network jitters 6 µs. These differences are most likely due to manufacturing tolerances in the crystal oscillator of the nodes.

The results are shown as boxplots but the vast majority of the executions is so close to 19.999 ms that the boxes appear as bars. As the outliers are far more important to this evaluation, we chose to show the whole range of outliers and sacrifice the details of the boxes for this purpose. From the outliers we can determine that the maximum jitter is 6 µs. At a cock rate of 168 MHz these 6 µs are about 1000 cycles. The majority of consecutive executions are performed with a difference below 168 cycles. This jitter most likely results 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.

4.3.3 Network Timing Accuracy

After showing that our systems is able to execute a given schedule very accurately on a single node, we will evaluate how our network stack performs. This is done by taking the same evaluation setup as before but measure the timing between two different



(a) Node1 as Master

(b) Node2 as Master

(c) Node3 as Master

(d) Node4 as Master

Figure 4.6: Results of execution interval accuracy with different master nodes.

nodes. We have chosen to measure between the master and the slave nodes. This setup emulates a feedback loop where the slave measures a value on which the master acts. Therefore, the slave is scheduled in the time-slot right before the master, as shown in Table 4.1. As we do not transmit any data from the slaves to the master, all slaves execute their evaluation task at the same time-slot. We chose to leave out the transmission of data as a simultaneously execution of the evaluation task gives us the opportunity to investigate the differences between the different slaves a little more. Additionally, the communication of the data has no impact on the execution timing of tasks. The received data would have an impact on the action the master would take but this is out of the scope of this evaluation. The master node in this evaluation is always *Node*1 as it had the best performance in the previous evaluation.

For better readability we subtracted the slot length of 5 ms from the delay in the following figures, therefore the displayed delay is negative in some cases.

Table 4.1: The two alternating 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

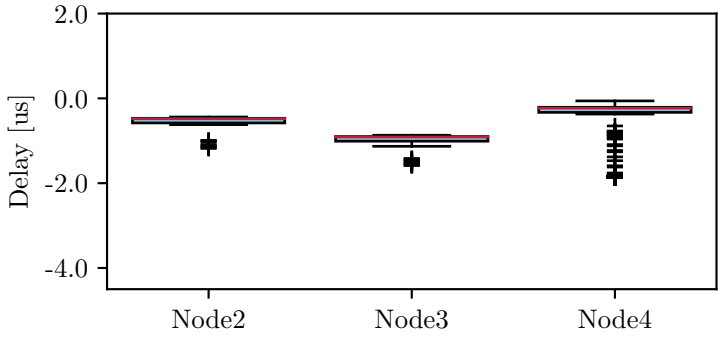


Figure 4.7: Network timing accuracy measurement without schedule or transceiver configuration changes.

For the first evaluation of timings on different nodes we used a static schedule, without any changes in the transceiver configuration or the schedule. The results shown in Figure 4.7 are similar to the ones in Figure 4.6. *Node*4 has the most jitter of all slave nodes, with up to 2 µs.

This evaluations can be seen as a base line for the following evaluations. These evaluations show how well our network stack can handle the different changes one can do to a schedule in our system. As we designed the network stack to be able to handle such changes, these evaluations are the most important ones.

In the next evaluation the transceivers had to change their communication channel periodically. In a real-word application this might happen when two TCs need to pass each other in transmission range. To resolve potential interference one of the TC has to change its communication channel.

The third evaluation of network timings will use two different schedules, shown in Table 4.1. We chose to switch these schedules periodically. In these schedules the evaluation task of the master changes from time-slot two to four and the slave's one from one to three. This changing of time-slots mocks a change in the application of the TC or its topology.

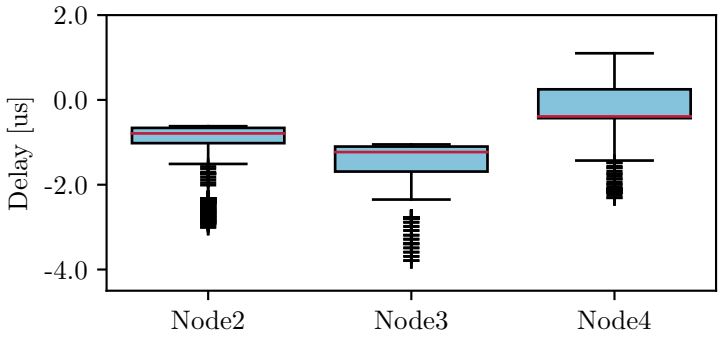


Figure 4.8: Network timing accuracy measurement with transceiver configuration changes.

In Figure 4.8 there is a significantly higher jitter as in Figure 4.7. Overall the jitter is about 5 µs, taking the results of *Node*3 and *Node*4. To reconfigure the transceiver blocking Serial Peripheral Interface (SPI)-transfers are needed. As these transfer cannot be interrupted to execute the evaluation tasks, they add jitter to the timings each time the configuration is changed. On our prototype hardware, see Figure 4.9, the SPI-bus cannot be clocked faster than 1.3 MHz. A custom PCB with shorter connections between the STM32 and the DW1000 would make faster SPI clock rates possible and thus reduce the jitter introduced by changing the configuration of the transceiver.

To evaluate how our network stack handles changes in the TC's application or the TC's topology we ran a continuous timing evaluation while switching the schedule every 10 frames (200 ms). We alternated between the two schedules in Table 4.1.

As the evaluation tasks of the master and the slave stay in consecutive tasks in both schedules, the jitter should not increase significantly. In Figure 4.10 the results of this evaluation are shown. Compared to the other evaluations, there is no significant difference. This shows, that our network 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.

4.3.4 Packet Loss

The prior evaluation shows that the timing accuracy in our system is good, therefore small guard times are possible in our network stack. Another finding was that additional SPI-transfers cause a significant jitter. To investigate why SPI-transfers are such a challenge we performed an evaluation in which we measured the packet loss between the nodes at different payload lengths, time-slot length and guard times.

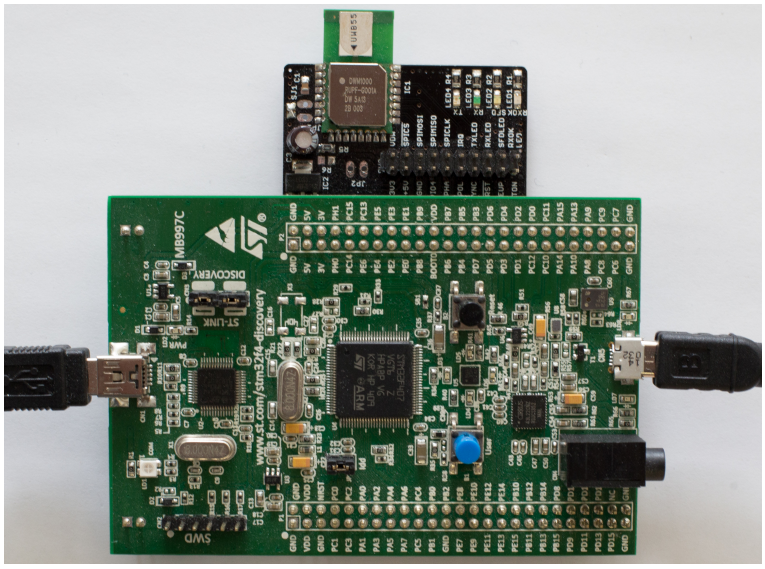


Figure 4.9: Prototype hardware used to evaluate our network stack. DW1000 with black PCB and STM32F407 evaluation kit with green PCB.

In this evaluation we used two nodes as senders and one as a receiver, one of the senders was *Node*1 and also used as time synchronization master. The two senders sent their packets alternating to each other to generate maximal load on the channel. In our evaluation we uses slot lengths from 2 ms to 100 ms, guard times from 10 ms down to 0.5 ms and payload sizes from 1 byte to 111 byte, which is the maximum payload size in our network stack.

The results of these measurements are shown in Table 4.2. Each cell in the table represents the packet loss in 20000 packets. For all combinations with a slot length longer than 4 ms the packet loss is below 0.01%. For 4 ms the packet loss rises only with

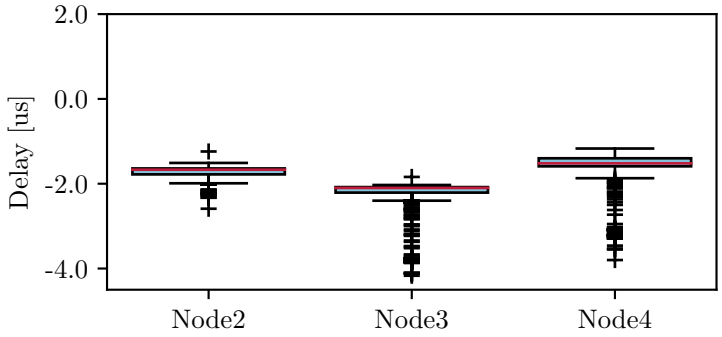


Figure 4.10: Network timing accuracy measurement with two alternating schedules switched every 200 ms.

a guard time of 0.5 ms. The same is the case for 3 ms, the fact that the packet loss is lower than for 4 ms seems to be coincidence. For slot lengths of 2 ms the packet loss rises for all payload sizes, for 74 byte and 111 byte all packets are lost. For a better visualization

Table 4.2: Resulting packet loss in percent for different configurations

		Packet Loss in % for Payload Size			
Slot Length [ms]	**Guard Time [ms]**	**1 Byte**	**37 Bytes**	**74 Bytes**	**111 Bytes**
100	10	0.024	0.005	0.010	0.010
	5.0	0.029	0.014	0.019	0.024
	2.0	0.024	0.010	0.000	0.029
	1.0	0.024	0.010	0.019	0.014
	0.5	0.075	0.062	0.062	0.024
50	10	0.043	0.057	0.052	0.067
	5.0	0.048	0.091	0.052	0.067
	2.0	0.057	0.067	0.052	0.043
	1.0	0.038	0.114	0.052	0.048
	0.5	0.051	0.014	0.062	0.029
10	5.0	0.058	0.091	0.052	0.067
	2.0	0.029	0.043	0.062	0.076
	1.0	0.071	0.086	0.033	0.033
	0.5	0.013	0.043	0.052	0.029
5	2.0	0.035	0.065	0.029	0.038
	1.0	0.010	0.038	0.052	0.086
	0.5	0.045	0.024	0.052	0.033
4	1.0	0.009	0.029	0.019	0.033
	0.5	0.043	0.010	0.024	7.012
3	1.0	0.048	0.000	0.038	0.010
	0.5	0.033	0.010	0.024	5.373
2	1.0	1.326	0.958	100.0	100.0

of the issue we show the last and important five rows of the table in Figure 4.11.

The behavior of loosing large packets in short slots and loosing smaller packets in even shorter slots indicates an issue with short slot lengths and large payload. Taking into consideration that the data rate of our wireless link is 6.8 Mbps the overall duration of a transmission cannot exceed 176 µs. This is, preamble, SFD, header and payload together. Therefore, the explanation for such behavior must lay in the SPI-transfers. As mentioned in Section 4.3.3 the SPI-clock had be reduced to 1.3 MHz to mitigate communication error on the SPI-bus. Therefore, a SPI-transfer for a maximum-sized packet of

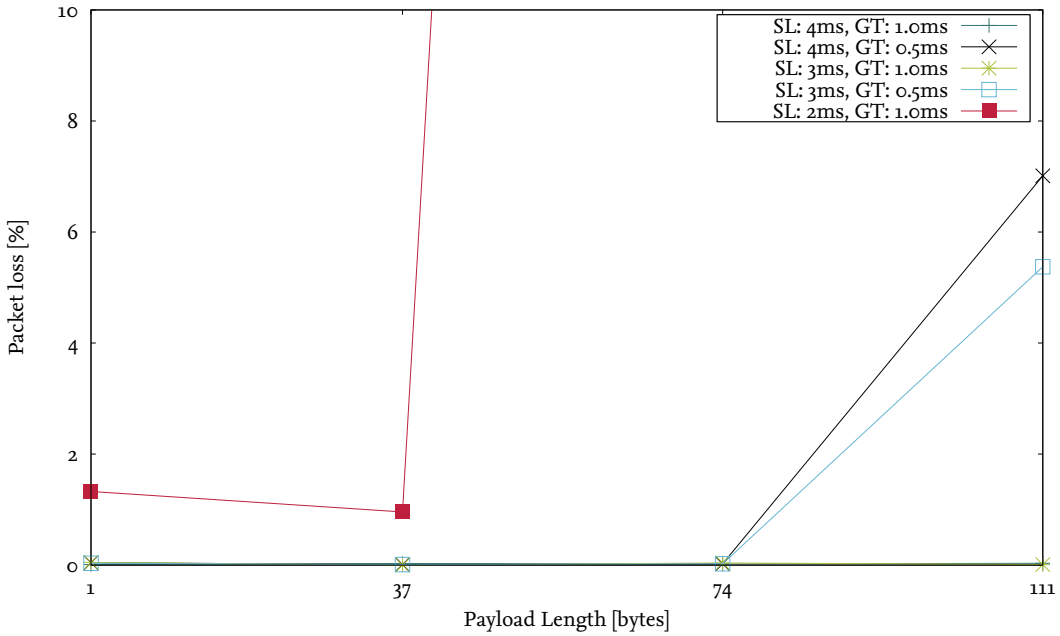


Figure 4.11: Packet loss at the critical configurations. SL stands for Slot-Length and GT for Guard-Time

127 byte takes 0.8 ms together with the SPI-transfer necessary to read or write the packet-buffer this takes longer than 1 ms. Figure 4.12 shows the SPI-transfer captured with a logic-analyzer, the long block on `Channel 1` a packet that is written to the transceiver during this transfer the `DW1000 IRQ` shows a rising edge that indicates the reception of a packet. Thus, the SPI-transfer took way too long to transmit the packet just transferred to packet buffer in time.

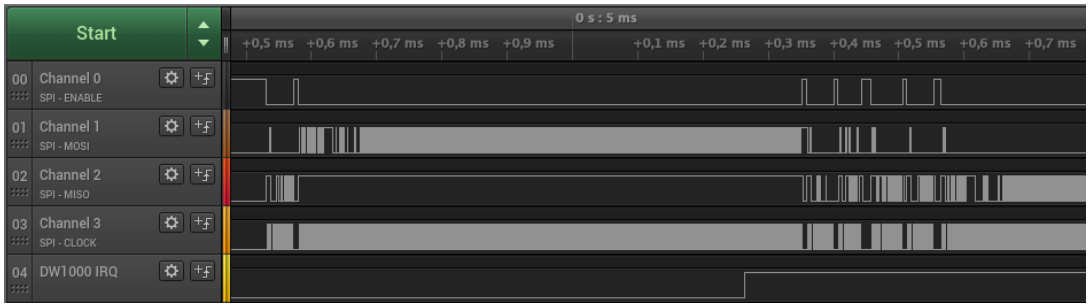


Figure 4.12: Screenshot of the logic analyzer showing overloaded SPI

As our network stack transmits all packets as broadcast messages, all nodes have to read the data from the transceivers packet-buffer to decide whether there is a task that subscribed to this message. For our evaluation setup that mean the second sender needs to read the packet sent by the first one. During this time it is not possible to prepare

the packet that need to be sent in the next time-slot. Thus, the packet will not be sent by the second sender and the receiver will count it as lost. Considering a 2 ms long slot there is simply not enough time in that slot to read a 127 byte packet and write one of the same size. This situation is depicted in Figure 4.13, together with a more relaxed situation with a slot length of 5 ms. In the beginning of a slot the receiver has to wait at least for the guard time (GT) and the radio transmission (RX on Air) before it starts to read the packet via SPI (RX on SPI). After reading the packet and writing the new one (TX on SPI). In the upper case, where the slot is 2 ms long, the deadline for the transmission is missed, the deadline (t_d) at the end of the guard time is reached before the transfer is finished at t_{tx}. For longer slots, e.g. 5 ms, t_{tx} is before t_d and the packet is transmitted at the right time.

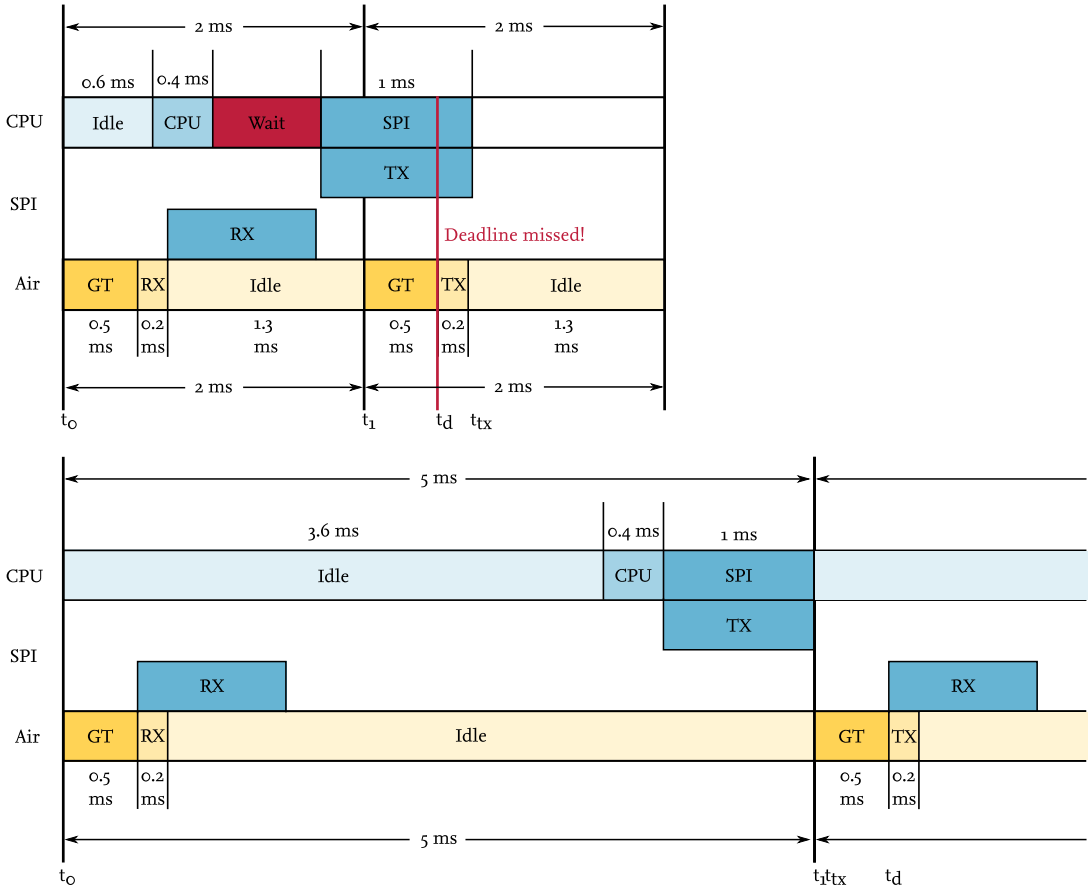


Figure 4.13: Analysis of SPI resource utilization timing for different slot lengths. SL stands for Slot-Length and GT for Guard-Time

Taking into consideration that no retransmissions were made in our evaluation a packet loss below 0.01% is competitive, as most other highly reliable network stacks use several transmissions to achieve the 99.99% packet reception rate [14]. tTo reduce packet loss for larger packets a PCB that enables higher SPI clock rates would be sufficient.

4.4 Conclusion

In conjunction with the time synchronization we presented in Chapter 3 our network stack introduces adaptability to networks which have previously be known for their static setups, like industrial networks based on TDMA-schedules. This was achieved by designing a node scheduler that combines the CPU and radio scheduler to minimize the jitter between the execution of a task and the transmission of its data. Further, we took great care in the whole design of the network stack to avoid unpredictable computational complexity but use methods with a static and known runtime. By utilizing multicast communication together with a publish-subscribe system and the delayed transmission feature of the DW1000 we are able to deliver data of one publisher to several subscribers on different nodes at the same time. With our approach such networks can be enabled to switch their schedules while maintaining the real-time communication between their nodes. Thus, they are able to adapt to changes in their environment, topology and applications. In a real-world evaluation we measured a total maximum jitter of 6 µs on a single node. The jitter between task executions on different nodes did not exceed 5 µs. Even changing the configuration of the transmitters and the schedule of the networks did not lead to a jitter larger than 5 µs.

5 Adaptive Real-Time Scheduling

The scheduling is one of the most critical tasks in a distributed real-time system, it provides the base for all operations of the system. Although, scheduling of distributed real-time systems is a vastly researched topic, introducing mobility in such systems adds challenges that are not addressed in existing research to date, see Section 5.3. To give a better understanding of these challenges we first declare the problem and reason the assumptions we make in detail in Section 5.1. In Section 5.2 we discuss the constraints a scheduling algorithm has to follow to generate valid schedules and explain them by using an example. Section 5.4 introduces our Mixed Integer Linear Programming (MILP)-model and its mathematical implementation of the constraints from Section 5.2. We evaluate its computational complexity in Section 5.5. In Section 5.6 we formulate a hypothesis on how a heuristic algorithm should be designed to generate schedules that can be adapted and investigate its validity. To give an alternative with a more foreseeable computational complexity we present a heuristic approach in Section 5.7. Section 5.8 compares the MILP- and the heuristic-based approach and shows the performance of our heuristic.

5.1 Problem Statement and Assumptions

In traditional real-time systems a schedule was calculated once and used until the system got another taskset. To switch the taskset of such systems they were stopped completely and started with the new schedule. Taking mobility into consideration, the assumption that a system can be stopped to load a new schedule is not applicable anymore. Therefore, mechanisms are needed to generate schedules that introduce the changes, needed to adapt to the new situation, without harming the real-time constraints of running tasks.

For our scheduling we assume that a distributed real-time system is a wireless network consisting of several nodes. Each node is capable of executing different tasks but only one of them per time-slot. Nodes can only communicate in a half-duplex manner on one channel in one time-slot but may switch channels between two consecutive time-slots. The whole network on the other hand is able to utilize multiple channels at the same time to transfer data in disjoint sets of nodes.

The problem of scheduling data transfers over multiple channels can be mapped to the problem of scheduling computation tasks to multiple processors. Coffman et al. [8] and Du et al. [11] showed that multiprocessor scheduling of non-preemptive task is NP-Hard. As we assume that the data transfer of a task is non-preemptive, our scheduling problem is NP-Hard as well.

All nodes are able to use the same set of interference free channels. Each channel has the same characteristics for all nodes in the TC. Further, we assume that all channels have the same characteristics. We also assume all nodes participating in a TC to be in each others transmission range, thus all communication is single-hop.

Several tasks from a job, the tasks of a job might be executed on several nodes and have dependencies between each other. Each job has a period which is inherited to the tasks. All jobs of a TC from the set ω. Jobs might share common tasks, these tasks have the shortest period of all jobs they are participating in. Besides the period (P_i), each task T_i has a maximum jitter (J_i) that describes how many time-slots a task might move between two consecutive executions. Additionally, each task T_i has a maximum age (d_{ij}) that describes how many time-slots the task might be executed before a task (T_j) depending on it. The tasks T_i depends on are called its dependencies and are stored in Γ_i. The matrix D, with dimensions $[|\tau| \times |\tau|]$, stores the maximum age d_{ij} and therefore how many time-slots a task T_i is allowed to be scheduled before task T_j. All tasks in a TC form the set τ. The execution time of each task is assumed to be at most one time-slot.

As we assume that the overall job of a system fails if one task is not executed in time we do not implement priorities in our scheduling.

All tasks of all jobs in a system form a directed graph without circles. This graph might have several entry tasks and leaf tasks. Each job in this graph forms a so called path that might have several entry tasks but only one leaf task. Entry tasks are tasks without dependencies, they form the set E. Leafs tasks are tasks without depended tasks, all leaf tasks form the set L.

The number of time-slots in which the schedule is not repeated is called Hyperperiod (H). It is defined as the least common multiple of all periods of jobs in ω.

We differentiate between time-slot and slot, a time-slot is a certain portion of time that has a defined start time and end time, all time-slots are of the same length. A time-slot can inhabit multiple slots, as a slot is a time-slot on a certain network resource. Therefore, a time-slot consists of as many slots as interference free network resources are available. An example for interference free network resources are the channels a communication standard defines. In a system with only one communication standard that defines three channels, each time-slot would consist of three slots.

Two tasks are called intersecting if they have common communication partners. This is the case if they have a direct dependency to each other, they are depending on the same task or they have the same depending task. Intersections between tasks are stored

in the matrix I with the dimensions $[|\tau| \times |\tau|]$.

Table 5.1 gives an overview of the symbols introduced above.

Symbol	Definition
T_i	i-th task
τ	Set of all tasks in a TC
J_i	Maximal jitter of T_i
d_i	Maximal age of T_i
P_i	Period of T_i
H	Hyperperiod of τ
ω	Set of all jobs in a TC
ω_k	k-th job in ω
$\|\omega_k\|$	number of tasks in ω_k or its length
M	Number of channels
S	Schedule: stating the task executed at time-slot and channel $[M \times \|\tau\|]$
s_{ij}	$s_{ij} \in S$ value of s gives the task to be executed at time-slot i on channel j
L	Set of all leaf tasks
E	Set of all entry tasks
Γ_i	Set of dependencies of T_i
D	Matrix of maximum number of time-slots between tasks $[\|\tau\| \times \|\tau\|]$
I	Matrix of intersections between all tasks $[\|\tau\| \times \|\tau\|]$
ι_{ij}	$\iota_{ij} \in I$ is 1 if T_i and T_j are intersecting, otherwise 0
N	Number of all scheduled executions of all tasks
e_{in}	time-slot of nth execution of task T_i
δ_i	Distance from T_i to the leaf task of the active job

Table 5.1: Notations of Symbols

Figure 5.1 depicts an example dependency graph that consists of two jobs. The first job has the entry task with id 5 and the leaf task with id 0, called job 0. It consists of the tasks: 5, 4, 3, 2 and 0. The second job has the entry task with id 5 and the leaf task with id 1, called job 1. Consisting of the tasks: 5, 4 and 1. Both of them share the common tasks 5 and 4. The graph is executed on a network with five nodes, the color of each task shows on which node it needs to be executed on. The arrows between tasks describe the dependencies, where the task the arrow is pointing to is depending on the task the arrow is pointing from, e.g. task 0 is depending on task 5.

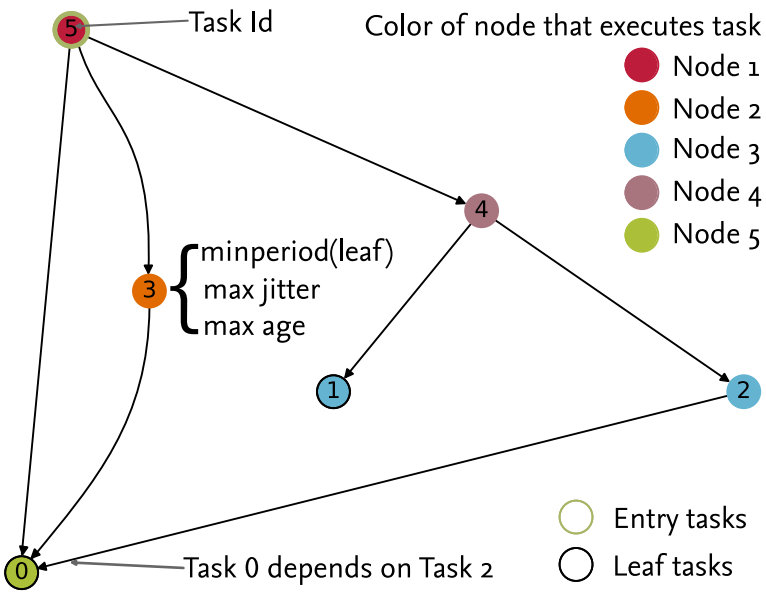


Figure 5.1: Example dependency graph with two jobs consisting in total of six task executed on a network with five nodes.

5.2 Scheduling Constraints and Objectives

This section discusses the constraints the scheduling has to fulfill in order to generate valid schedules.

Constraint 1 *It is not allowed that two or more tasks share the same slot.*

Sharing slots would lead to transmission interference and loss of data, therefore each task needs its own slot.

Constraint 2 *Tasks with a common participating node must not be executed in the same time-slot*

Constraint 2 prohibits the scheduler from scheduling two tasks at the same time that have the same node either receiving or transmitting data. In our example from Figure 5.1 there are different tasks that can not be scheduled at the same time: task 1 and task 2 as they are executed at the same node, task 2 and task 3 as both transmit data that is needed by task 0. By prohibiting these tasks to be scheduled at the same time, these constraint ensures that the communication can be handled by half-duplex transceiver.

Constraint 3 *All dependencies of a task must be scheduled before the depending task.*

Referring to our example in Figure 5.1 this constraint ensures that, e.g., task 5 is scheduled before task 0, task 3 and task 4. To be able to schedule task 0 it is necessary that task 5, task 3 and task 2 are scheduled before task 0.

Constraint 4 *Each dependency of a task must be scheduled no more than its maximal age before the task.*

As most data has an age at which it becomes less usable to the requesting task, the providing task or dependency must be scheduled less than this age before the requesting task. E.g., if the maximal age of data provided by task 3 is ten time-slots, task 3 must not be scheduled more than ten slots before task 0.

Constraint 5 *All depending tasks in one job must use the same execution of a common dependency.*

To ensure that tasks in one job use the same state of the system, it is necessary that tasks in one job that depend on the same task are scheduled after the same execution of that task. In our example: the job with entry task task 5 and leaf task task 0 is formed by the task ids 5, 4, 3, 2 and 0. Constraint 5 defines that task 5 must not be scheduled between the tasks 4, 3, 2 and 0.

Constraint 6 *Each leaf task must be scheduled once in its period.*

The different jobs might have different periods in which they have to be scheduled in, the leaf task of each jobs might have a different period. Therefore, some of these periods might differ form the hyperperiod, which is the least common multiple (LCM) of all periods. To fulfill the requirements of all jobs, the leaf might have be scheduled multiple times in one schedule. Together the Constraints 3 to 6 ensure that each job is executed the right amount of times per hyperperiod and all tasks in the jobs are executed in the right order.

Constraint 7 *Two consecutive periods of the same task must not exceed the defined jitter bound for this task.*

As described in Section 5.1, each task has a jitter bound that must not be exceeded. Therefore, Constraint 7 ensures that the period of a task does not vary more than its jitter bound. E.g., the scheduled period of a task with the defined period of five slots and a maximal jitter of two slots could be decreased to three slots or increased to seven slots. But two consecutive periods of that task may not vary more than two slots, thus, a period change from three slots to seven is not allowed.

A scheduler that enforces all the Constraints 1 to 7 will generate schedules applicable to a system described in chapter 2. To be able to adapt to topology changes, the scheduler needs to follow one more constraint:

Constraint 8 *The difference between the last period in the old schedule and the first period in the new schedule of a task must not exceed the defined jitter bound for this task.*

If this constraint is followed the network can switch its schedule to the new one without breaking any real-time constraints.

To make the operation of a network reliable the schedule stability should be maximized. This is especially important when existing schedules need to be adapted to changes in the topology or taskset of a network. Increasing the schedule stability reduces the complexity of switching the schedules and therefore reduces the probability of failures while switching. Therefore, we formulate the general objective for the scheduling as following:

Objective 1 *Time-slot changes between periods of tasks should be minimized.*

5.3 Related Work

Real-time scheduling is a vast topic, especially if the scope is widened to real-time multiprocessor scheduling. To keep this section of reasonable size we focus on the most applicable related work.

Conflict-aware Least Laxity First (C-LLF) was proposed by Saifullah et al. to schedule WirelessHART (WirelessHART) networks[44]. It is designed for wireless real-time networks with changing topologies. The priority of a transmission is determined by the time between it is released and its deadline and the number of conflicting transmissions in this time. The highest priority is given to the task with the highest number of conflicting transmissions and the shortest time between release and deadline. C-LLF needs the release time of each transmission prior to scheduling, this is not possible in systems where the release time of a transmission depends on a task that has dependencies. As it is unknown when a dependency is scheduled, the release time of the depended transmission is also unknown. Therefore, C-LLF is not applicable for the problem described in Section 5.1.

Another application of wireless real-time networks with changing topologies are LTE networks. Shakkottai and Stolyar introduced an algorithm to schedule a mixture of real-time and non-real-time traffic in LTE networks[46]. At each time-slot the algorithm calculates which of the packets in the transmission queue has the shortest deadline and schedules this packet into the slot at a certain channel. This done by the eNode-B for each transmission time interval which consists of multiple time-slots. The schedule determined in this manner is only valid for the down-link traffic from the eNode-B to

the user equipment. This technique is only possible, as the eNode-B buffers all downlink traffic. In a network such as in Section 5.1 there is not one central instance as the eNode-B that buffers all the traffic.

Wang et al. propose a two staged approach to adaptive scheduling in train communication networks[54]. These networks are often time-triggered and need to handle rapid topology changes in cases where two trains are coupled or decoupled. The first stage is the offline scheduling, it generates schedules for the whole train. The second stage is called online stage, it derives the schedule for the parts of the train during the coupling and decoupling process. A two staged design seems applicable to our system as we also expect rapid topology changes when two TCs need to be merged. On the other hand the characteristics of the network described by Wang et al. are in great contrast to our application. The authors describe train networks as strictly hierarchical multicluster networks with wired connections. Further, the approach does not have a concept of dependencies between different data flows.

A novel approach on how to close feedback control over wireless links is proposed by Baumann et al.[5]. They propose a system that reduces the communication between different parts of a distributed feedback controller. The reduction is achieved by a co-design called control-guided communication. Control-guided means that the controller tells the communication part of the system its communication demands ahead of time. The communication demand is decreased by a controller that estimates values in between communication. To benefit from the decreased communication demands the schedule needs to reschedule the communication frequently. Therefore, Baumann et al. choose an online scheduling approach. Like the other approaches this approach also lacks a concept of dependencies between tasks.

As the approaches discussed above do lack the ability to handle dependencies the scope is widened to scheduling in operations research. In assembly lines dependencies are very common and therefore scheduling approaches in this domain need to handle them from the beginning. The algorithm proposed by Hu forms Directed Acyclic Graphs (DAG) from a given set of tasks[20]. In these DAGs each node represents a task and each directed edge represents a dependency. The author assumes an assembly line with a number of equally skilled workers, each of this workers is able to fulfill one task at a time. The task have a predefined order in which they have to be fulfilled, the goal is to find the sequence of tasks for each worker that needs the shortest time to complete all tasks of the given set. To clarify why this is applicable to the challenges stated in Section 5.1 let the equally skilled worker be equally good channels and the predefined order gives a set of dependencies between tasks. This gives a model of a system which is quite close to the one we depicted above. However, a difference between our use case and assembly line scheduling is, that there is no concept of tasks that cannot be executed at the same time because of common child tasks. Further, the tasks in Hu's model

have no deadline and the goal is to finish as fast as possible, in contrast our goal is to meet the deadline of all task.

Each of the scheduling concepts mentioned addresses a part of the problem stated in Section 5.1. However, each one is missing some key features needed to fulfill the requirements for scheduling algorithms as they are needed in this work. Some of the solutions used in the following two approaches are inspired by the discussed concepts.

5.4 Mixed Integer Linear Programming Approach

In this section we introduce an approach to solve the problem described in Section 5.1 based on a MILP model. First we introduce the model for a decision problem that generates valid schedules regarding to the assumptions discussed in Section 5.1. Afterwards, we describe the constraints needed by the model to generate valid schedules. Then we introduce the objective that minimizes the jitter.

To ease the modeling we extend the schedule S by a third dimension which is the dimension of tasks in τ, the resulting Matrix is called A. Thus, the three dimensions of A are, (i) the considered task in τ, (ii) the channels and (iii) the time-slots. A has the dimensions $|\tau| \times M \times H$ and is defined as followed:

$$\forall a_{Tct} \in A : a \in \{0, 1\} \tag{5.1}$$

Where T is the task, c is the channel and t is the time-slot.

$$a_{Tct} = \begin{cases} 1, & \text{if task } T \text{ is scheduled in channel } c \text{ at time-slot } t \\ 0, & \text{if task } T \text{ is not scheduled in channel } c \text{ at time-slot } t \end{cases} \tag{5.2}$$

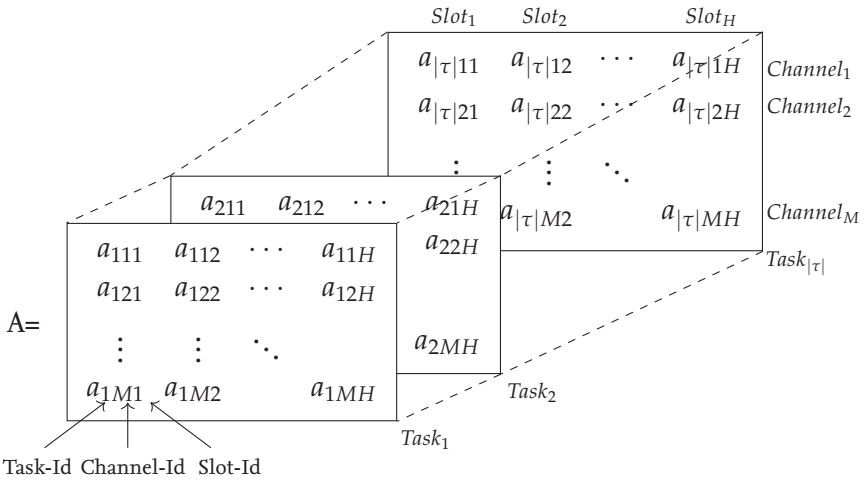


Figure 5.2: Graphical representation of the MILP-schedule A with H slots, M channels and $|\tau|$ tasks.

Figure 5.2 depicts the schedule used as a decision variable in the MILP-model. The layers represent the different tasks, the rows represent the channels and the columns represent the time-slots. In the following the constraints of the MILP-model are discussed in detail.

5.4.1 Constraints

MILP-Constraint 1 *Every slot (timeslot and channel) must at most have one scheduled task*

$$\sum_{T \in \tau} a_{Tct} \leq 1 \quad \forall c \in \mathbb{N} : 1 \leq c \leq M,\ \forall t \in \mathbb{N} : 1 \leq t \leq H$$

MILP-Constraint 1 guarantees that there are no direct collisions between tasks in the same time-slot and the same channel and therefore implements Constraint 1.

MILP-Constraint 2 *Tasks with common participating node must not be executed in the same time-slot*

$$(\iota_{UT} \times \sum_{c=1}^{M} a_{Tct}) + (\iota_{UT} \times \sum_{c=1}^{M} a_{Uct}) \leq 1 \quad \forall t \in \mathbb{N} : 1 \leq t \leq H,\ \forall U, T \in \tau$$

As task that have an intersection in their sets of nodes they are communication with must not be scheduled in one time-slot, MILP-Constraint 2 takes the sum of all scheduled distributions of the interfering tasks T and U in a time-slot t over all channels $1 \leq c \leq M$. This sum has to be less or equal one for all time-slots and all interfering pairs of tasks. Whether tasks T and U are interfering is determined from matrix I at position UT. It is the implementation of Constraint 2. The matrix I is defined by Equation (5.3) and Equation (5.4).

$$\forall \iota_{UT} \in I : \iota \in \{0, 1\} \tag{5.3}$$

Where T and U are tasks in τ .

$$\iota_{UT} = \begin{cases} 1, & \text{if task } T \text{ and } U \text{ have common communication partners} \\ 0, & \text{if task } T \text{ and } U \text{ do not have common communication partners} \end{cases} \tag{5.4}$$

MILP-Constraint 3 *All dependencies U of task T must be scheduled before T within the minimum of P_i or $t - d_U$ time-slots*

$$\sum_{i=max(1;t-d_U;\lfloor t/P_T \rfloor \times P_T)}^{max(1;t)} \sum_{c=1}^{M} a_{Uci} \geq \sum_{c=1}^{M} a_{Tct} \quad \forall t \in \mathbb{N} : 1 \leq t \leq H,\ \forall U \in \Gamma_T,\ \forall T \in \tau$$

MILP-Constraint 3 ensures that all dependencies Γ_T to a task T are executed at least as often as the depended task T. It sums up all scheduled executions of a dependency U in the P_i slots and channels before T is scheduled and ensures that this sum is larger than the sum of the scheduled executions of T in time-slot t. This guarantees that Constraints 3 and 4 are enforced.

MILP-Constraint 4 *Each leaf task must be scheduled once per its period*

$$\sum_{t=max(1;(p-1)\times P_T}^{p\times P_T} \sum_{c=1}^{M} a_{Tct} = 1 \quad \forall p \in \mathbb{N} : 1 \leq p \leq \frac{H}{P_T},\ \forall T \in \tau$$

To ensure that each job is executed once in its period (Constraint 6), MILP-Constraint 4 sums up all executions of task T during all possible periods and ensures the sum is always one for all tasks.

MILP-Constraint 5 *Execution of a T must be scheduled in its jitter bounds*

$$\sum_{i=max(1;t-P_T-J_T)}^{max(1;t-P_T+J_T)} \sum_{c=1}^{M} a_{Tci} \geq \sum_{c=1}^{M} a_{Tct} \quad \forall t \in \mathbb{N} : 1 \leq t \leq H,\ \forall T \in \tau$$

As a task T scheduled outside its jitter bound J_T could harm the operation of the system (Constraint 7), MILP-Constraint 5 prohibits that. Therefore it checks whether there is an execution of T scheduled in the jitter bound $\pm J_T$ one period P_T before the current execution.

The following MILP-Constraints 6 to 8 ensure that all tasks of one job use the same execution of a common dependency (Constraint 5). Referring to the graphs in Figure 5.3 they ensure that task 0, task 3 and task 4 all use the same execution of task 5. That means, task 5 must not be scheduled between task 0, task 3 and task 4. This might happen, as jobs can share certain tasks and the periods of some dependencies might be smaller than the period of the job and the depending task.

As an example, let job 0 have a period of ten slots and job 1 a period of five slots. With the constraints MILP-Constraints 1 to 5 there is nothing that prohibits to schedule the tasks 4 and 5 a second time in between the scheduled executions of task 3 and 0. Doing so could cause task 0 to operate on different data than task 3.

To cope with this we need to introduce a few more variables. Let $\Omega_{T_eT_l}$ be the set of tasks between the entry task T_e of a job and its leaf task T_l with the same period as T_l.

For the example above Ω_{50} would consist of the tasks 3, 2, 0.

On contrast let $\breve{\Omega}_{T_eT_l}$ be the tasks of the job that have a shorter period than T_e. Again, for the example that would mean $\breve{\Omega}_{50}$ consists of the tasks 5 and 4.

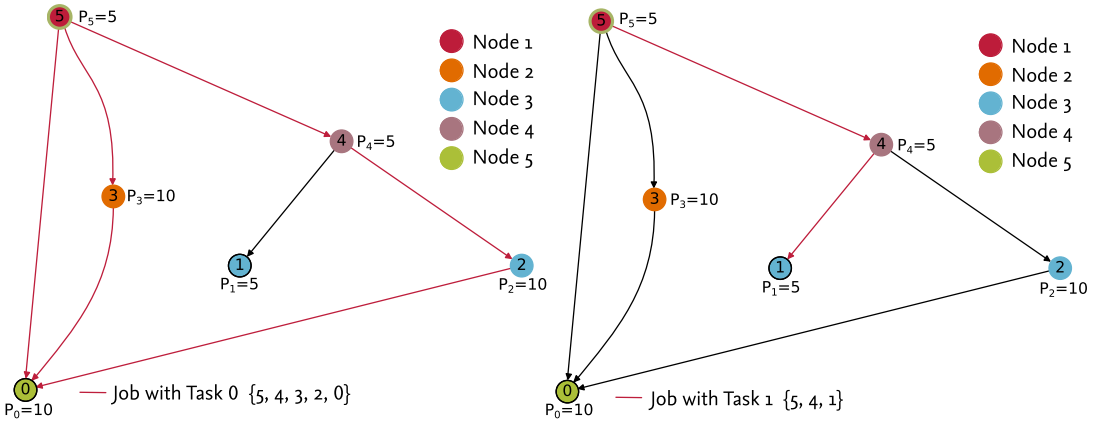


Figure 5.3: (a) shows the subgraph of job 0 and (b) shows the subgraph of job 1.

MILP-Constraint 6 *$\rho_{T_e T_l}$ is the sum of all executions of all tasks in $\Omega_{T_e T_l}$, $\check{\rho}_{T_e T_l}$ is the sum of all executions of all tasks in $\check{\Omega}_{T_e T_l}$*

$$\rho_{T_e T_l} = \sum_{T \in \Omega_{T_e T_l}} \sum_{c=1}^{M} \sum_{i=t}^{min(i+P_{T_l}, H)} a_{Tci} \quad \forall t \in \mathbb{N} : 1 \leq t \leq H, \forall T_e \in E, \forall T_l \in L$$

$$\check{\rho}_{T_e T_l} = \sum_{T \in \check{\Omega}_{T_e T_l}} \sum_{c=1}^{M} \sum_{i=t}^{min(i+P_{T_l}, H)} a_{Tci} \quad \forall t \in \mathbb{N} : 1 \leq t \leq H, \forall T_e \in E, \forall T_l \in L$$

MILP-Constraint 6 is only needed to give $\rho_{T_e T_l}$ and $\check{\rho}_{T_e T_l}$ a value that indicates how many tasks of $\Omega_{T_e T_l}$ and $\check{\Omega}_{T_e T_l}$ are scheduled in one period of T_l.

MILP-Constraint 7 *If the complete path or non of its tasks is scheduled then $\dot{\rho}_{T_e T_l}$ and $\dot{\check{\rho}}_{T_e T_l}$ are 1. If only parts are scheduled $\dot{\rho}_{T_e T_l}$ and $\dot{\check{\rho}}_{T_e T_l}$ are 0.*

$$\rho_{T_e T_l} = |\Omega_{T_e T_l}| \times \dot{\rho}_{T_e T_l} \quad \forall T_e \in E, \forall T_l \in L$$

$$\check{\rho}_{T_e T_l} = |\check{\Omega}_{T_e T_l}| \times \dot{\check{\rho}}_{T_e T_l} \quad \forall T_e \in E, \forall T_l \in L$$

$\rho_{T_e T_l}$ and $\check{\rho}_{T_e T_l}$ are used in MILP-Constraint 7 to determine whether all task in $\Omega_{T_e T_l}$ or respectively $\check{\Omega}_{T_e T_l}$ are scheduled. That is necessary as tasks in $\check{\Omega}_{T_e T_l}$ might be scheduled independently but tasks in $\Omega_{T_e T_l}$ must not be scheduled without the tasks in $\check{\Omega}_{T_e T_l}$. This is ensured by MILP-Constraint 8

MILP-Constraint 8 *The $\check{\Omega}_{T_e T_l}$ might be scheduled alone but $\Omega_{T_e T_l}$ must not*

$$\dot{\check{\rho}}_{T_e T_l} \geq \dot{\rho}_{T_e T_l} \quad \forall T_e \in E, \forall T_l \in L$$

Together the MILP-Constraints 1 to 8 define a model for the decision problem. This model is able to generate valid schedules in A.

5.4.2 Objectives

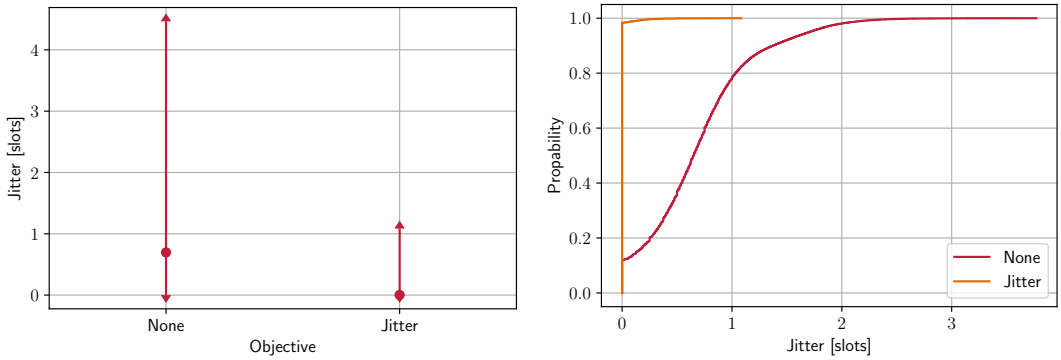
The objective of our model is to minimize the jitter between executions of a task. Minimizing the jitter in feedback loops is one of more obvious optimizations for a scheduling. This is due to the fact, that jitter leads to a bigger error in timing of the control task and therefore to larger error in the controlled process.

Minimizing the jitter in the model described above is challenging, as it lacks a concept of how many slot are between different executions of a task. To mitigate this, we minimize the number of tasks that change their relative time-slot in different periods of their job. With this optimization we additionally increase the schedule stability. The schedule stability gives a measure how much a schedule changes between periods.

MILP-Objective 1 *Minimize the number of tasks changing slots between their periods*

$$Minimize: \frac{1}{N} \times \sum_{T \in \tau} \sum_{t=1}^{H-P_T} \left(\left| \sum_{c=1}^{M} a_{Tct} - \sum_{c=1}^{M} a_{Tc(t+P_T)} \right| \right)$$

To evaluate the effectiveness of MILP-Objective 1 we scheduled the same set of 130000 tasksets once with MILP-Objective 1 and once without any objective. These tasksets were randomly generated under certain constraints. We defined five different hyperperiod lengths (8, 12, 16, 25, 35), four different numbers of dependencies between the tasks (9, 12, 16, 24) and three different numbers of jobs (1 ,3 ,6), the number of tasks were eight or twelve. In Figure 5.4 the jitter for the two different objectives is shown. "None" refers



(a) (b)

Figure 5.4: (a) mean, maximum and minimum jitter over all scheduled tasksets for the two objectives, "None" for no objective and "Jitter" for MILP-Objective 1. (b) depicts the Cumulative Distribution Function (CDF) of the jitter for both objectives.

to the results scheduled without any ojective, "Jitter" refers to the jitter in the schedules optimizes to a minimum jitter by MILP-Objective 1. The jitter shown in Figure 5.4 is

calculated using Equation (5.5), it is the mean of the mean jitter of all tasks scheduled in the schedule.

$$jitter = \frac{\sum_{T \in \tau} \frac{\sum_{i=1}^{H/P_T} (e_{Ti} - e_{T1}) \bmod P_T}{\sum_{t=1}^{H} \sum_{c=1}^{M} a_{tcT}}}{|\tau|} \tag{5.5}$$

As Figure 5.4a shows MILP-Objective 1 reduces the mean jitter to almost zero. The maximum jitter is also reduced. Together with the CDF depicted in Figure 5.4b this shows that mILP-Objective 1 reduces the jitter for almost all tasksets.

As an objective cannot harm the performance, in terms of schedulablility, of our MILP-model, we use the MILP-Objective 1 for all further evaluations if not stated otherwise.

5.4.3 Adapting Schedules

This section discusses one of the key contributions of this chapter, the adaptation of existing schedules to changes in the topology of the network or in the taskset. The adaptation needs to be done without harming the real-time requirements of jobs which are present in the existing schedule. Constraint 8 formulates this complex goal in a very brief way.

To achieve the goal of adapting schedules, several steps are needed in preparation. First the tasksets of the old schedule and the new tasks need to be merged. The new tasks can either be a second taskset of another TC or tasks of new job added to the existing TC. In this work we focus on the first case, where two TCs need to be merged into one. We consider this as the more complex case, as adding a new job is the same, despite the fact that the new job does not have the restrictions of an old schedule. While joining the tasksets (τ_1 and τ_2), task-ids must be kept unique throughout the new taskset τ'. The new hyperperiod H' is the LCM of all periods in τ'.

In the second step the two schedules (A_1 and A_2) are merged into one schedule C that violates Constraint 1. Thus, tasks of both networks might share one time-slot on the same channel. As C is never to be executed, this does not cause any harm to the networks. C is used in MILP-Constraint 9 in addition to MILP-Constraints 1 to 8 to generate the new, combined schedule A' that respects the Constraints 1 to 8.

MILP-Constraint 9 *The timeslot of $task_T$ must not differ more than J_T from C to A', changes of channels are ignored*

$$\sum_{j=t-J_T}^{t+J_T} \sum_{c=1}^{M} a'_{Tcj} \geq \sum_{c=1}^{M} c_{Tct} \quad \forall t \leq H', \forall T \in \tau'$$

Together with MILP-Constraint 9 we introduce the new MILP-Objective 2 that minimizes the amount of tasks that are shifted to other time-slots between C and A'. Thus, the schedule stability is maximized.

MILP-Objective 2 *Minimize time-slot allocation changes from C to A′*

$$Maximize: \sum_{T \in \tau'} \sum_{t=1}^{H'} (\sum_{c=1}^{M} a'_{Tct} \times \sum_{c=1}^{M} c_{TcT})$$

By multiplying the sum of all channels for a certain time-slot and a certain task in the new schedule with the sum of the channels of the same task and time slot in the combined schedule, we get one for each task that was not move and zero for each task that was moved. As each task can at most be scheduled once per time-slot the result of this multiplication can only have the two values, one and zero. By summing this result up over all tasks and time-slots, we get the number of the unchanged time-slot allocations. As we maximize this value, the schedule stability is maximized and the jitter is minimized.

5.5 Evaluation of Computational Complexity

As stated in Section 5.1 multi-channel real-time scheduling is NP-Hard. Therefore, it is important to evaluate whether the formulated MILP-Model can be solved in a reasonable time according to the use case. Besides that we will also investigate what parameters influence the time it takes to solve the MILP-Model.

As the first parameter we evaluated the number of slots in a hyperperiod. We scheduled more than 160,000 different tasksets with five different hyperperiods: 8, 12, 16, 25 and 35 slots on an Intel Xeon W-2195 CPU. The tasksets are the same as in Section 5.4.2. The time used to solve the model varies from 0.07 s to more than 600 s. Figure 5.5 shows the CDF of the five different hyperperiods. The right graph shows only the range from zero to ten seconds of the left graph to show more details. As expected, a longer hyperperiod leads to a longer solve-time, this is due to the larger solution space. The dots in both graphs mark the maximum solve-time. Even though some tasksets need over 600 s the majority is scheduled in less than 3 s.

As a second parameter we evaluated the influence of the number of dependencies in a taskset. To mitigate the influence of the hyperperiod length we only evaluate the solve-time of tasksets with 35 slots. Figure 5.6 shows the CDF for 9, 12, 16, and 24 dependencies, all together it shows 76000 tasksets. As the model gets more complex with more dependencies the solve-time increases as well.

To determine how TCs with multiple jobs would be handled in contrast to TCs with just one job, we evaluated the solve-time of tasksets with a hyperperiod of 35 slots and 16 dependencies. These roughly 20000 tasksets have either 1, 3 or 6 jobs. Figure 5.7 shows that a taskset with more complex jobs and more dependencies, takes longer to be scheduled than an easier one. As the number of tasks in all scheduled tasksets in this evaluation was the same, the most complex jobs were in the tasksets whit only one job. Therefore, these took the longest to be scheduled followed by three and then six jobs.

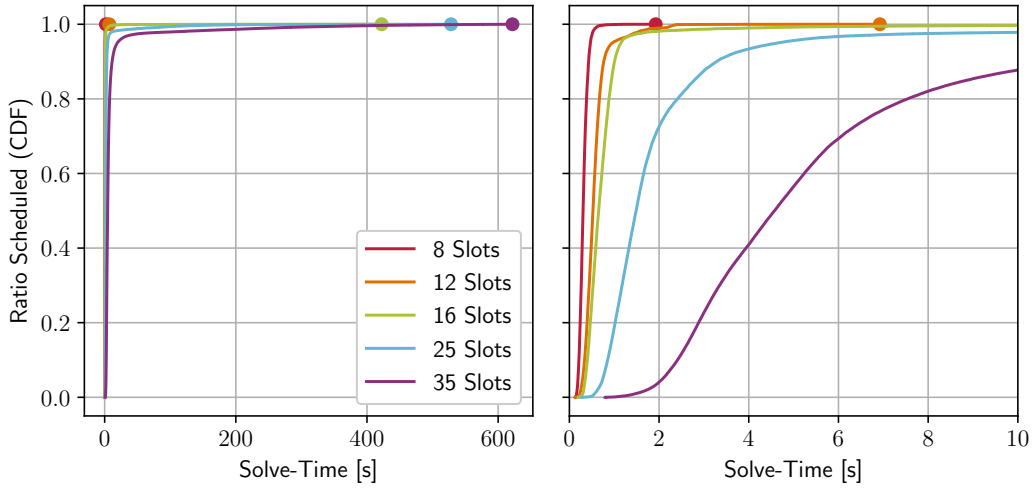


Figure 5.5: Impact of different hyperperiod length to the time needed to solve the schedule, shown as CDF

5.5.1 Applicability to Embedded Devices

As most CPSs consist of embedded devices with far less computation power than our Intel Xeon W-2195 (fast CPU), we assume the solve-time to be higher. To support this assumption, we used our 10 years old Intel Xeon E5520 (slow CPU) to schedule all tasksets with hyperperiod of 35. The results for the median solve-time are not that different: 5.5 s for the slow CPU and 4.5 s for the fast CPU. However, the mean and absolute maximum solve-times differ a lot: the fast CPU needs a mean solve-time of 11.7 s and a absolute maximum of 621 s, the slow CPU needs 72.3 s in mean and 14 332 s at absolute maximum. In Figure 5.8 we depicted all solve-times, the left figure shows all outliers, the left one is zoomed to 15 s to make the details visible. The left figure shows that the vast majority of tasksets in scheduled in less than 14 s, even with the slow CPU.

This huge variety in the solve-times is a problem considering real-time applications. Even if the scheduling has no hard time constraints, waiting up to 600 s or even 14 000 s for a schedule is unrealistic in most applications. Especially in cases where two mobile TCs need to be merged, here both TCs would have to stop and wait until the new schedule is calculated. Even more pressing is the issue that the decision whether it is possible to merge two tasksets also takes that long. Therefore, a way to calculate schedules in a more predictable time is necessary.

In Section 5.6 we formulate a hypothesis on the adaptability of schedules and validate it to get better insight on how to design an algorithm suitable for the described problem.

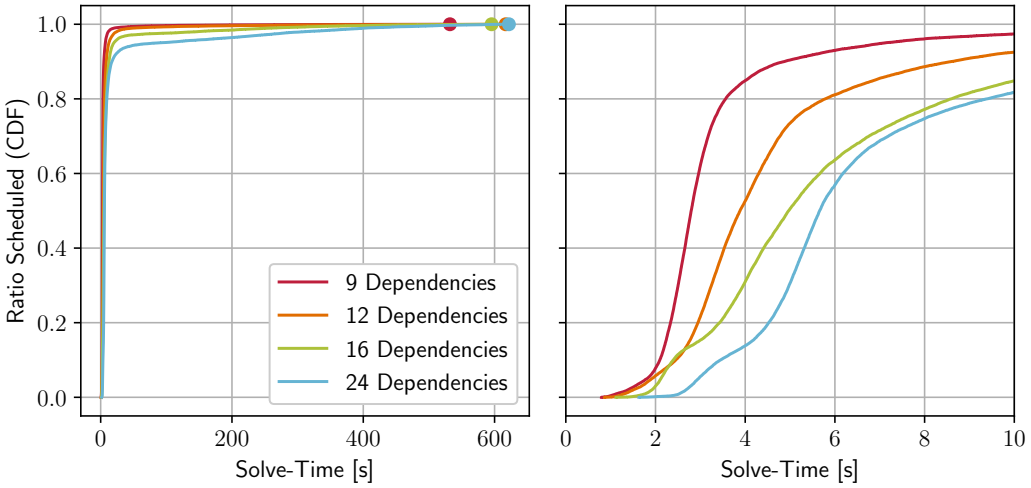


Figure 5.6: Impact of different numbers of dependencies to the time needed to solve the schedule, shown as CDF

5.6 Hypothesis on Adaptability of Schedules

In most scheduling applications it is preferable to schedule all tasks as dense as possible. So the taskset can be scheduled more often in the same amount of time or the executing machines can sleep or take other jobs. In a system where jobs have a defined period there is no need to schedule all task as dense as possible. In contrast, it might have advantages to schedule tasks as sparse as possible. That means, free slots are more uniformly distributed throughout the hyperperiod.

This is especially advantageous if adaptations are taken into consideration. Having free slots throughout the hyperperiod means, that tasks in a merged schedule must not be shifted to time-slots as far as in a dense schedule. Figure 5.9a depicts two dense schedules S_1 and S_2, S_1 has the tasks A_1 to A_3 and S_2 the tasks B_1 to B_3. The merged schedule S_{12} contains all tasks. In this example, in the separate schedules S_1 and S_2, all tasks are scheduled very densely in the first three of the six time-slots. To merge S_1 and S_2 the tasks are shifted into other time-slots. B_3 has to be shifted three time-slot, therefore Δ_{B_3} is 3.

Figure 5.9b on the other hand shows two sparse schedules that also contain three tasks in six time-slots. In contrast to the schedules in Figure 5.9a, this time the tasks are distributed equally over the six time-slots. To merge the schedules all tasks of S_2 have to be shifted only by one time-slot, therefore Δ_{B_3} is 1.

Figure 5.9 shows the two extreme cases but they illustrate why it might be a good idea to spread the tasks throughout the whole hyperperiod. In the following we introduce our metric for the degree of distribution and investigate under which circumstances

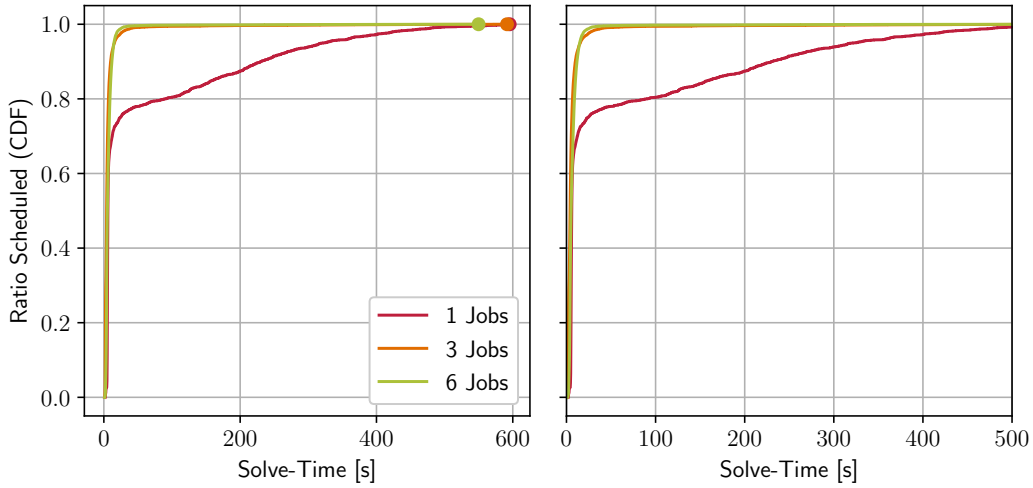


Figure 5.7: Impact of different numbers of jobs to the time needed to solve the schedule, shown as CDF

the hypothesis, that pairs of schedules which have a higher distribution are more likely to be combinable, is true.

5.6.1 Task Distribution

$$Distribution = \frac{\sum_{t=1}^{H} x_t}{\sum_{t=1}^{H} \sum_{c=1}^{M} \sum_{T \in \tau} a_{tcT}} \tag{5.6}$$

Equation (5.6) gives the distribution as a normalized function of number of used time-slots to unused time-slots divided by the number of all scheduled executions of all tasks. Where x_t indicates an unused time-slot following an used time-slot, as described in Equation (5.7).

$$x_t = \begin{cases} 1, & \text{if time-slot } t-1 \text{ is used and time-slot } t \text{ is unused} \\ 0, & \text{if time-slot } t-1 \text{ is unused or time-slot } t \text{ is used} \end{cases} \qquad \forall 1 \leq t \leq H \tag{5.7}$$

A time-slot is called used if there is a task scheduled on at least one channel.

5.6.2 Validity of the Hypothesis

To validate whether the hypothesis is true we generated pairs of two schedules for two different tasksets of similar form, in terms of hyperperiod length, number of jobs, number of dependencies, etc. The distribution of a pair lies between zero and two, as it is the sum of the distribution of both schedules. These pairs were than rescheduled using the MILP-model with MILP-Constraint 9 and MILP-Objective 2 enabled. Figure 5.10 shows that pairs with a higher distribution are much more likely to be schedulable than pairs with a lower distribution.

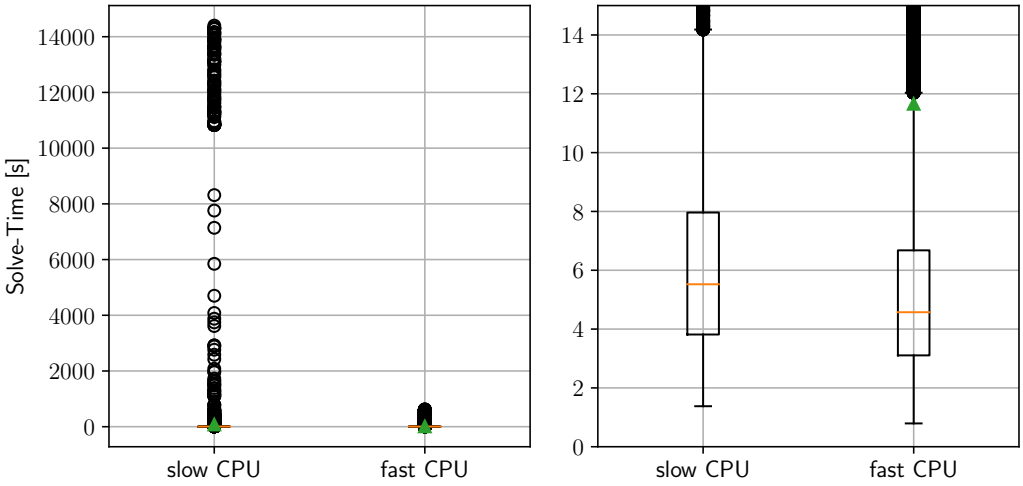


Figure 5.8: Comparison of two different CPUs and their scheduling performance

5.7 Heuristic Approach

As we have shown in Section 5.6 schedules with higher distribution can be merged more easily. Therefore, we propose an algorithm that has three main goals, 1. maximize the distribution, 2. minimize the jitter of tasks and 3. minimize the number of executions of each task. These three goals have to be united with the need for an algorithm that calculates schedules in a more predictable time than the MILP-model.

In general the algorithm schedules a whole job before it starts to schedule the next job. It starts with the job that has the longest path from the entry to the leaf task and continuous with the next longest job until all jobs are scheduled. We call the job that is currently scheduled active.

Within a job the scheduler starts with the leaf task and places it in the latest possible slot, this way the number of slots to schedule the rest of the jobs is maximized. The slot to schedule the leaf task for the k-th subperiod is calculated by utilizing Equation (5.8). A subperiod is one period of a job, its length is always a divider of the hyperperiod.

$$t_{l,k} = k \times P_l \tag{5.8}$$

As slots might be occupied by tasks which were scheduled prior to the current one, we introduce two solutions to choose another slot in which the current task is scheduled. One of them chooses to use other channels before moving to other time-slots. This should lead to a small jitter, as more task are scheduled in the calculated time-slot. The other one prefers to use other time-slots first and only uses different channels if no time-slot in the allowed jitter bound is free. This should be beneficial if TCs need to be merged. In this case the TCs can simply use another channel and most conflicts are

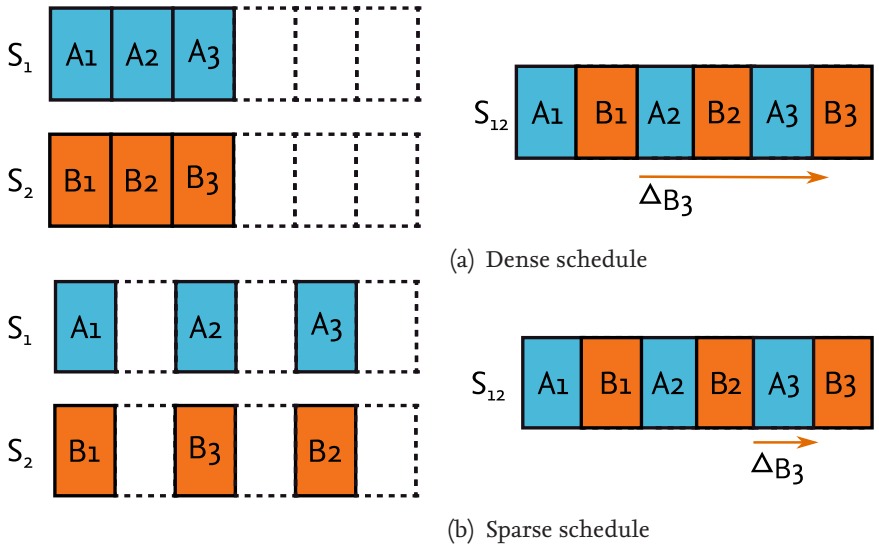


Figure 5.9: Examples of a dense schedules and a sparse schedules.

resolved. These solutions are described in more detail in Section 5.7.3 and Section 5.7.4. After a free slot is found, this slot in the two dimensional array S is marked occupied with the task. The dimensions of S are given by the number of channels M and length of the hyperperiod H.

After the leaf task is scheduled the algorithm uses the so called backward equation to find the slot for the dependencies of the leaf task, this equation is explained in more detail in Section 5.7.1. The backward equation maximizes the distance of the execution between the scheduled task and other dependencies but also between the dependency and its dependencies. This process is repeated until all entry tasks of a job are scheduled.

As it might occur that several jobs have common tasks, the backward equation would schedule a common task twice although an already scheduled execution of that task could be used to schedule the rest of the job. Therefore, the algorithm searches the schedule whether there are tasks of the active job scheduled in a certain range of slots. If such a task is found, the algorithm uses this execution and schedules the rest of the active job in the direction from that execution to the leaf task using the so called forward equations, described in Section 5.7.2.

While the scheduler is following the dependencies of tasks it might face tasks that have multiple dependencies or multiple tasks that depend on a task. In both cases the scheduler needs to decide in which order these task should be scheduled. We have two different approaches to this issue: the first one orders the task by ascending maximal age. We call this approach *age first*. The idea is to schedule the task first that have a smaller range of time-slots in which they can be scheduled.

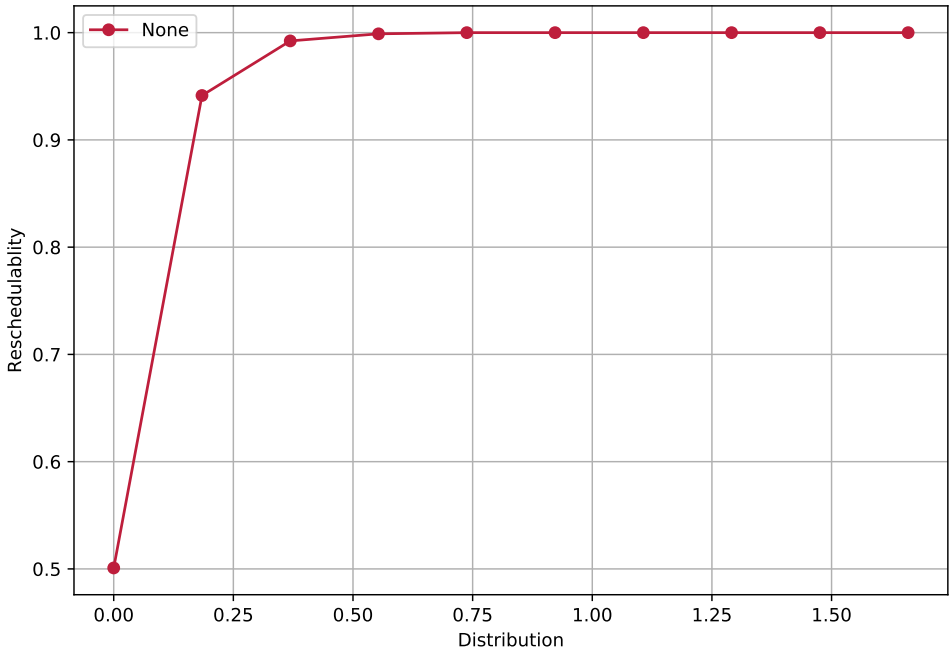


Figure 5.10: Reschedulablity over distribution

The second approach is, to sort the tasks by ascending maximal jitter. This guarantees that the tasks with the hardest jitter constraints are scheduled first. We call this approach *jitter first*. Both approaches will be compared in the evaluation Section 5.8. In Figure 5.11 we depicted the flow-graph of the algorithm described above.

5.7.1 Backward Equation

The Backward Equation (5.9) gives a time-slot in which the task should be scheduled, based on the execution time-slot of the tasks that are depending on the task to be scheduled. k gives the subperiod which is currently to be scheduled, T_c is the child (dependent task) of a task and T_p is the parent (dependency) of a task. Equation (5.9) needs a special case for the first subperiod $k = 1$. This is simply because in this case there is no prior execution of this tasks, so its time-slot can not be taken into the equation.

$$t_{p,k} = \begin{cases} t_{c,k} - min(\lfloor \frac{(t_{c,k} - 1)}{|\omega_i| - \delta_p} \rfloor, d_p), & \text{if } k = 1 \\ t_{c,k} - min(\lfloor \frac{t_{c,k} - t_{c,k-1} - 1}{|\omega_i| - \delta_p} \rfloor, d_p), & \text{if } 2 \leq k \leq H/P_l \end{cases} \tag{5.9}$$

With $|\omega_i| - \delta_p$ we calculate how many tasks of the job we have to schedule until the leaf task is reached. The divided $t_{c,k} - 1$ or $t_{c,k} - t_{c,k-1} - 1$ gives the number of slots left in the subperiod. The division of both gives a time-slot for the parent that has the

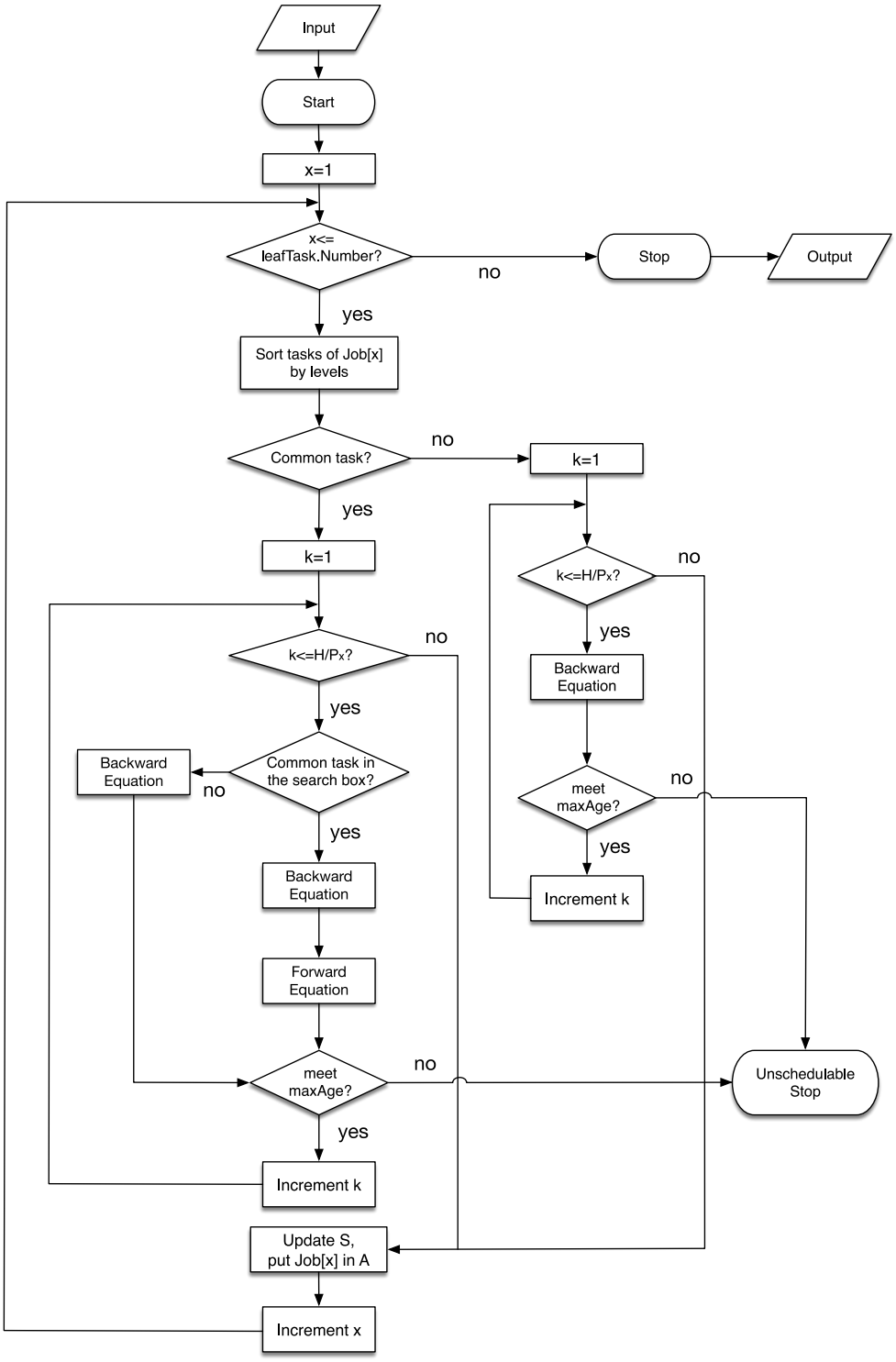


Figure 5.11: Flowchart of the discussed scheduling algorithm

maximal distance to the child but still leaves enough time-slots to schedule its parents. As the time-slot calculated this way might have a larger distance to the child than the maximal age of the parent the minimum of the division and the maximal age is taken.

If the calculated time-slot is occupied, an alternative is selected applying the solutions from Section 5.7.3 or Section 5.7.4.

5.7.2 Forward Equation

Jobs in a taskset might have tasks in common, an example is depicted in Figure 5.12. In the example the jobs with the leaf task T_1 and T_2 have the common task T_{com}. This

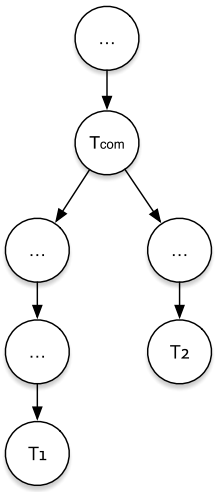


Figure 5.12: Jobs with common tasks

fact can be used to reduce the total number of task executions in a schedule. To do so we need to find these common tasks in the schedule and decide whether the found execution of such a common task fulfills the timing requirements of the child task that is to be scheduled based on the common task. Therefore, the algorithm defines a range of time-slots in which the execution of a common task has to be located in order to use it. The lower bound of this range $Lower_{com,k}$ for the k-th subperiod of T_{com} is defined as the maximum of three values: first, the execution time-slot of its child task in the $k-1$-th subperiod. This guarantees that the order of executions is not altered for prior subperiods. The second value is the time-slot of T_{com}s execution in the $k-1$-th subperiod on which the period of the leaf task of the active job P_l is added. To be able to use extra time-slots the maximal jitter J_{com} is subtracted. The last value is the end of the current subperiod $k \times P_l$, to make use of allowed jitter J_l is subtracted as well as the sum of all maximal ages of the tasks from the common task to the leaf task $\sum_{i=com}^{l-1} d_i$.

Thus, $Lower_{com,k}$ is defined as follows:

$$Lower_{com,k} = max(t_{c,k-1} + 1, t_{com,k-1} + P_l - J_{com}, k \times P_l - J_l - \sum_{i=com}^{l-1} d_i) \tag{5.10}$$

The upper bound of the range is defined as the minimum two values: $t_{l,k} - \delta_{com}$ where $t_{l,k} = k \times P_l + J_L$, which gives the last slot far enough for the end of the subperiod to schedule all tasks between T_{com} and the leaf task, and where δ_{com} is the distance T_{com} and T_l. The second value is the period of the leaf task P_l and maximal jitter of T_{com} added to the time-slot of the last execution of T_{com}.

$$Upper_{com,k} = min(t_{l,k} - \delta_{com}, t_{com,k-1} + P_l + J_{com}) \tag{5.11}$$

As there are the corner cases of the first and last subperiod, the range in which the algorithm searches for T_{com} is defined as following:

$$t_{com,k} \in \begin{cases} [|\omega_i| - \delta_{com}, k \times P_l + J_l - \delta_{com}] & \text{if } k = 1 \\ [max(t_{c,k-1} + 1, t_{com,k-1} + P_l - J_{com}, k \times P_l - J_l - \sum_{i=com}^{l-1} d_i), & \text{if } 1 < k < H/P_l \\ \qquad min(k \times P_l + J_l - \delta_{com}, t_{com,k-1} + P_l + J_{com}))] & \\ [max(t_{c,k-1} + 1, t_{com,k-1} + P_l - J_{com}, k \times P_l - J_l - \sum_{i=com}^{l-1} d_i), & \text{if } k = H/P_l \\ \qquad min(t_{com,k-1} + P_l + J_{com}, H - \delta_{com})] & \end{cases} \tag{5.12}$$

In the last subperiod the algorithm must take care not to search outside or too close to the border of the hyperperiod, therefore $Upper_{com,k}$ limits at $H - \delta_{com}$.

If an execution of T_{com} was found in the range defined by Equation (5.12) the scheduler uses Equation (5.13) to calculate the time-slot in which the child T_c of T_{com} should be executed.

$$t_{c,k} = \begin{cases} t_{p,k} + min(\lfloor \frac{min(k \times P_l, t_{p,k+1}) - t_{p,k}}{\delta_p} \rfloor, d_p) & \textit{if } T_p = T_{com} \textit{ and } k < H/P_l \\ t_{p,k} + min(\lfloor \frac{k \times P_l - t_{p,k}}{\delta_p} \rfloor, d_p) & \textit{else} \end{cases} \tag{5.13}$$

As Equation (5.13) describes the k-th execution of T_c is scheduled after the k-th execution of T_p but before the $k+1$-th execution of T_p. Therefore, the execution order defined in the taskset is respected. The other limiting factor for the forward equation is the maximal age d_p of T_p, the child must be scheduled before d_p, otherwise the data produced by T_p is useless. As for the backward equation it is possible that $t_{c,k}$ is already occupied, strategies to handle such situations are described in Section 5.7.3 and Section 5.7.4.

5.7.3 Time First Shifting

In the sections above we described how the algorithm determines in which time-slot a task should be scheduled. If this slot is occupied on one channel the algorithm needs to find another slot either on another channel or at another time. This section describes a solution to this challenge, that tries to find another time-slot before it uses other channels. Figure 5.13 shows an example where a task T_i is supposed to be scheduled in $t_{i,k}$ which is occupied by the task T_j. As the algorithm normally schedules the tasks

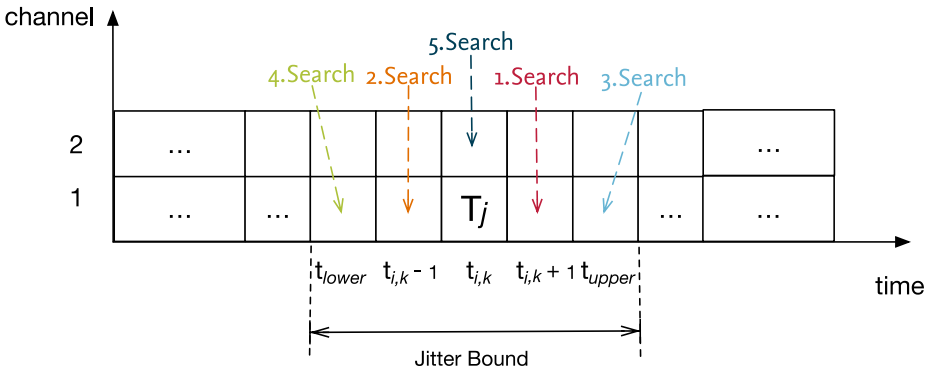


Figure 5.13: Example how the time first shifting reacts if a slot is occupied

from the leaf to the entry task and thus from right to left in the example, the algorithm first tries to put T_i into $t_{i,k} + 1$ on the same channel. This is the time-slot next to $t_{i,k}$ on the right, by going right first the algorithm leaves potentially more space where the yet unscheduled tasks need to be scheduled. If $t_{i,k} + 1$ would be occupied as well the algorithm would go to $t_{i,k} - 1$ and not $t_{i,k} + 2$ to minimize the jitter. After all slots in the jitter bound of T_i are tested and found occupied the algorithm would test $t_{i,k}$ at another channel (5. search in Figure 5.13) and repeat the same search pattern if it is occupied as well. Another reason not to take $t_{i,k}$ on the second channel would be that T_j and T_i are interfering tasks, in this case this slot would be handled as occupied.

In this mode the algorithm tries to fit all tasks on one channel. This is done under the hypothesis that two schedules that use one channel primarily are easier to merge, as one of the schedules could be shifted to another channel and most of the conflicts would be resolved.

5.7.4 Channel First Shifting

This mode of the algorithm solves the same issue as the one described in Section 5.7.3 but by using all available channels before shifting the execution of T_i in time. Figure 5.14 shows the same situation as Figure 5.13 but in this mode the algorithm tries to schedule

T_i on channel 2 in $t_{i,k}$ as its first option. After all channels, in the example two, are found occupied for $t_{i,k}$ the algorithm would try then all channels at $t_{i,k} + 1$. The pattern the algorithm walks through the time-slots is the same as described in Section 5.7.3. This

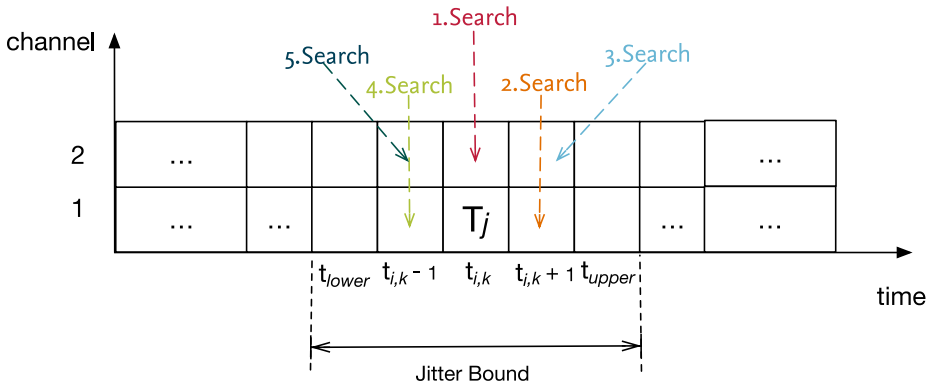


Figure 5.14: Example how the time channel shifting reacts if a slot is occupied

mode has the advantages that it minimizes the jitter in the schedule and that it potentially leaves more time-slot entirely empty. Having time-slots empty on all channels might be of advantage when two schedules need to be merged, as tasks can be shifted there to make room for tasks that can not be shifted due to stricter jitter bounds.

In Section 5.8 we will evaluate the proposed shifting mechanisms.

5.7.5 Schedule adaption

To adapt a schedule to changes in the taskset or the network's topology, we use the same method as described in Section 5.4.3 to merge tasksets. After the tasksets are merged, the same algorithm is used to generate the new schedule.

The result of the backward Equation (5.9) has no dependencies to tasks outside of the active job, therefore it will not changes unless the job is changed which results in a new job. Thus, the backward equation does not harm the real-time requirements while adapting schedules.

The search range definition Equation (5.12) for $k = 1$ sets the upper search bound to $k \times P_l + J_l - \delta_{com}$, and the lower bound to $|\omega_i| - \delta_{com}$. As the results are not effected by any task outside the job, it will not harm real-time requirements during adaption. For all subperiods where $k > 1$, the search box is limited by the previous subperiod, that means that no subperiod with $k > 1$ could harm the real-time requirements of a task if the first subperiod did not harm them. The same is true for the forward Equation (5.13) as it is not effected by tasks outside the job.

As discussed the calculations of the time-slots cannot give results outside the jitter bound and the shifting methods described in Section 5.7.3 and Section 5.7.4 do not shift

further than the allowed jitter. Thus, no task can be effected by a jitter more than the allowed one during schedule changes. Therefore, the same algorithm can be used to schedule a TC initially and to adapt its schedule to new topologies or tasksets.

5.8 Evaluation

In this section we evaluate the algorithm discussed in Section 5.7 and compare it with the results of the MILP-model introduced in Section 5.4. First we show that the presented algorithm has a much more predictable execution time than the MILP-model. Afterwards we compare the different approaches in terms of jitter and the percentage of scheduleable tasksets. In Table 5.2 we list the short names used to distinguish the different modes of the algorithm in this section.

Table 5.2: The four different modes of the scheduling algorithm

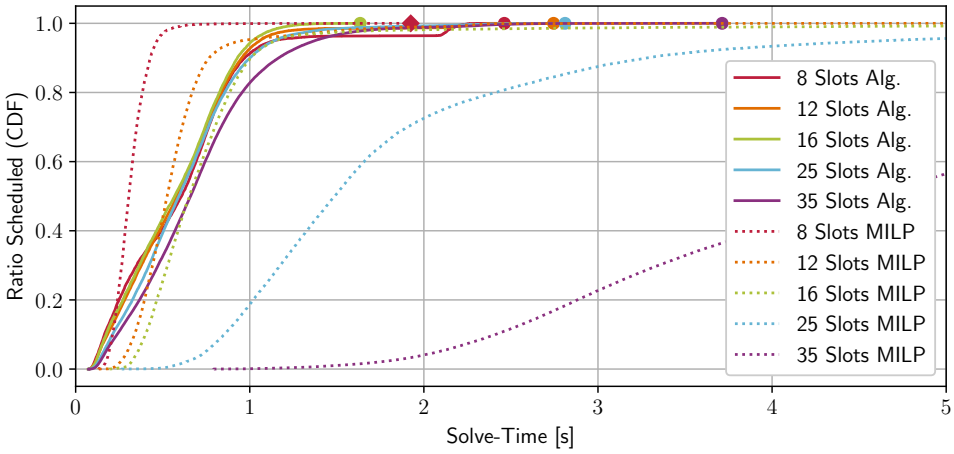
Approach	Sort tasks on same level by	Mode
Time First Shifting	Age First	(0, 0)
	Jitter First	(0, 1)
Channel First Shifting	Age First	(1, 0)
	Jitter First	(1, 1)

5.8.1 Computational Complexity Comparison

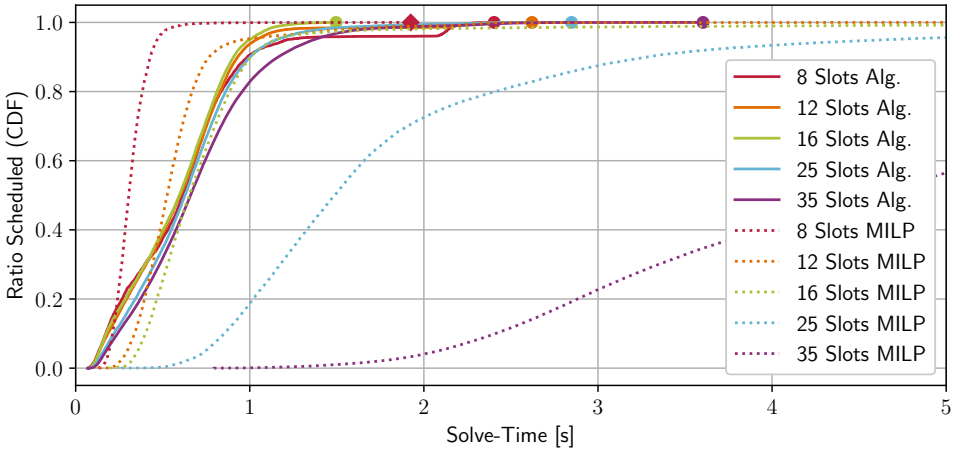
Figures 5.15 and 5.16 show the comparison of the time needed to schedule tasksets with a hyperperiod of 8, 12, 16, 25 and 35 slot with the proposed algorithm and the MILP-model. The dotted lines represent the MILP-model and solid lines the heuristic algorithm, the maximum solve time is marked with a diamond for the MILP-model and a dot for the algorithm. As the results for the MILP-model are discussed in Section 5.5, Figures 5.15 and 5.16 are zoomed to a five seconds range to see the algorithm's results in detail. The results for the four different modes are very similar, therefore we only discuss the results of the *time first shifting and age first* mode [0, 0]. Figure 5.15a shows that the solve times of the algorithm are much closer to each other and, thus, are much less influenced by the length of the hyperperiod. With the vast majority of tasksets scheduled in less than one second, the algorithm is much more likely to finish scheduling in time even on embedded devices. Nevertheless, for short hyperperiods the MILP-model is faster, that suggests that there is room for optimizations, at least in the implementation of the algorithm.

5.8.2 Influence of Taskset Parameters to Scheduling Success

To determine under which conditions the algorithm is able to schedule what percentage of the tasksets we evaluated four parameters: hyperperiod length, number of nodes in a TC, number of dependencies and number of jobs. As not all of our test tasksets



(a) [0, 0]

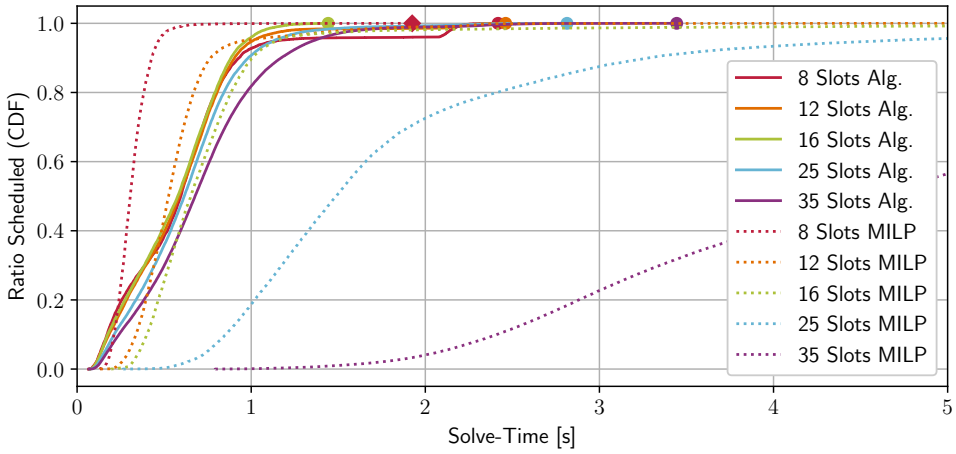


(b) [0, 1]

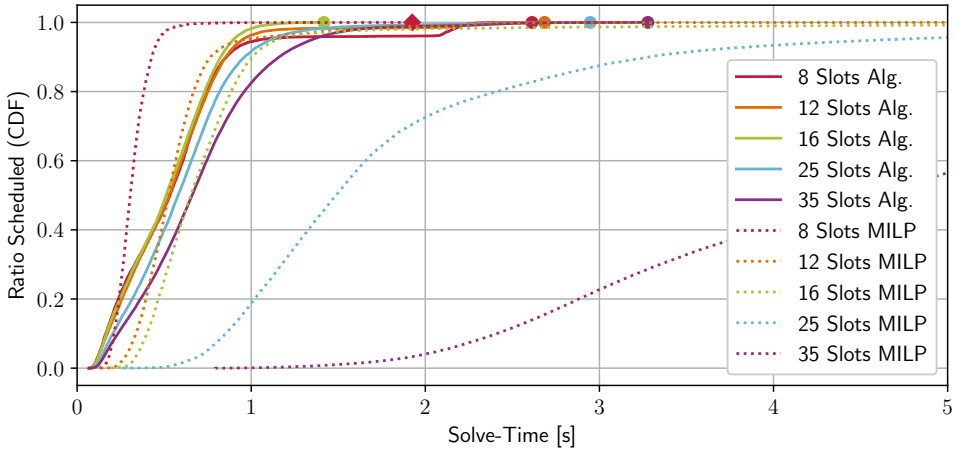
Figure 5.15: Influence of hyperperiod length on the schedulability, comparing the four algorithm modes and the MILP-model. For three jobs, nine dependencies and twelve nodes for the two *time first shifting modes*

are schedulable, we compare the percentages of the heuristic-based algorithm with the results using the MILP-model. Figure 5.17 shows the influence the hyperperiod length has, for the hyperperiods 24 and 35 slots. In both cases the *channel first shifting* performs better than the *time first shifting*, especially in the case of 35 slots. It is able to schedule over 90% of the tasksets. The order in which tasks are chosen to be scheduled does not seem to have a big influence on whether a taskset is schedulable or not. In general, the hyperperiod length does not have a large impact on the performance of the algorithm.

A bigger influence is caused by the number of nodes in a TC, as shown in Figure 5.18. In general the results show that a taskset, with all the same parameters except for the



(a) [1, 0]



(b) [1, 1]

Figure 5.16: Influence of hyperperiod length on the schedulability, comparing the four algorithm modes and the MILP-model. For three jobs, nine dependencies and twelve nodes for the two *channel first shifting modes*

number of nodes, is harder to schedule if less nodes are in the TC. This is due to the fact that in such tasksets there are more intersecting task which makes it more likely that a taskset is unschedulable. Therefore, the algorithm mode should be compared to the MILP-model. This comparison still shows that the algorithms performance deteriorates if fewer nodes are in a TC. The results of this evaluation confirm the assumption further that *channel first shifting* is the superior mode. They also give rise to the assumption that the algorithm is strongly effected by the number of dependencies.

To confirm the assumption we evaluated the performance of the algorithm on tasksets with the same parameters except for the number of dependencies. As for the evaluation

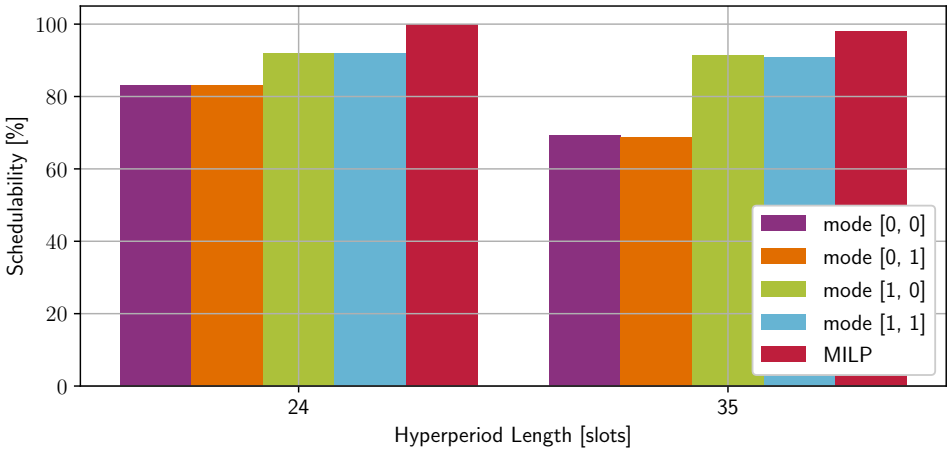


Figure 5.17: Influence of hyperperiod length on the schedulability, comparing the four algorithm modes and the MILP-model. For three jobs, nine dependencies and twelve nodes

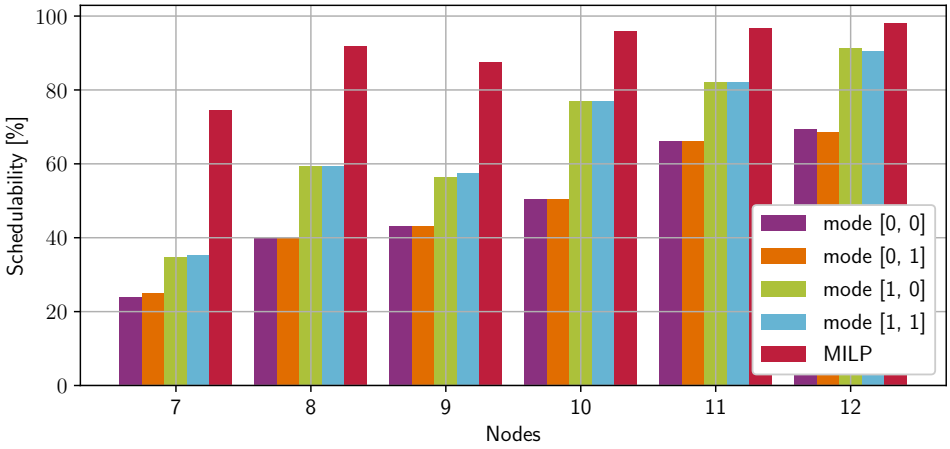


Figure 5.18: Influence of TC-size on the schedulability, comparing the four algorithm modes and the MILP-model. For three jobs, nine dependencies and 35 slots

above, the algorithms performance needs to be compared with the MILP-model. The results depicted in Figure 5.19 show a dramatic decline in the percentage of schedulable tasksets between nine and twelve dependencies.

As the algorithm tries to schedule the leaf task of each job in the last slot of the subperiod, we assume that more jobs lead to a lower performance. To confirm the assumption we evaluated tasksets with the same parameters but varied the number of jobs. Figure 5.20 show that this assumption is true, it also shows that the performance of *channel first shifting* declines more than the performance of *time first shifting*.

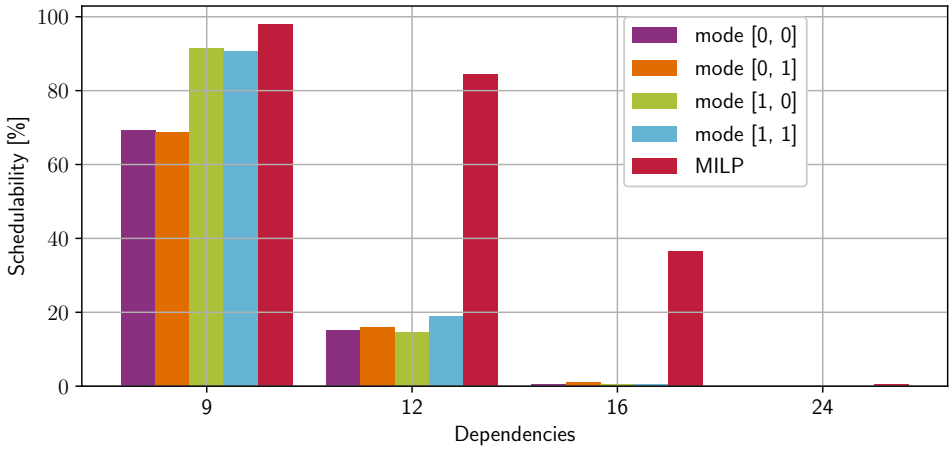


Figure 5.19: Influence of the number of dependencies on the schedulability, comparing the four algorithm modes and the MILP-model. For three jobs, and 35 slots and twelve nodes

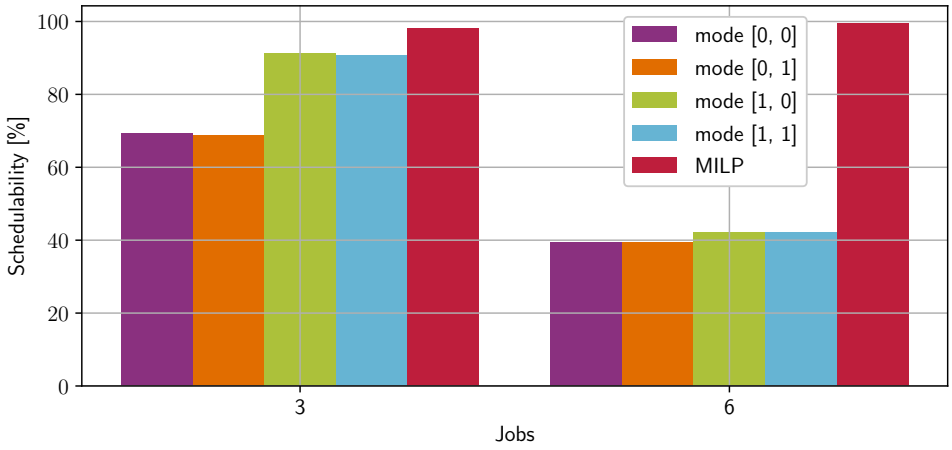


Figure 5.20: Influence of the number of jobs on the schedulability, comparing the four algorithm modes and the MILP-model. For nine dependencies, and 35 slots and twelve nodes

All evaluations show that the *channel first shifting* mode was superior to the *time first shifting*, only in some cases the performance of both approaches were close to each other. The evaluation also shows that the order in which the tasks are chosen to be scheduled does not make a noticeable difference in most cases and if there is a difference there is no pattern behind which of them performs better. In general, the evaluation shows that the heuristic algorithm is more suitable for taskset with only few jobs.

5.8.3 Slot Allocation Probability

To investigate further why *channel first shifting* suffers more from an increased job number than *time first shifting* we compare the probability of each time-slot to be allocated for a task. Figure 5.21 shows these probabilities for both shifting approaches and the MILP-model, it also depicts the value for an equal distribution as a reference. As we discussed above the algorithm always tries to allocate the last time-slot in the subperiod of a task. For the *channel first shifting* this behavior is evident from the results, time-slots that are divisors of 24 have a higher probability to be allocated. The higher the divisor is, the higher is the probability, as there are more jobs that share the end of its subperiod here. For the *time first shifting* this effect is still noticeable but it is mitigated. The MILP-model does not show this effect, as it is not bound to the limitations in the heuristic, but it shows the tendency to allocate the first time-slot.

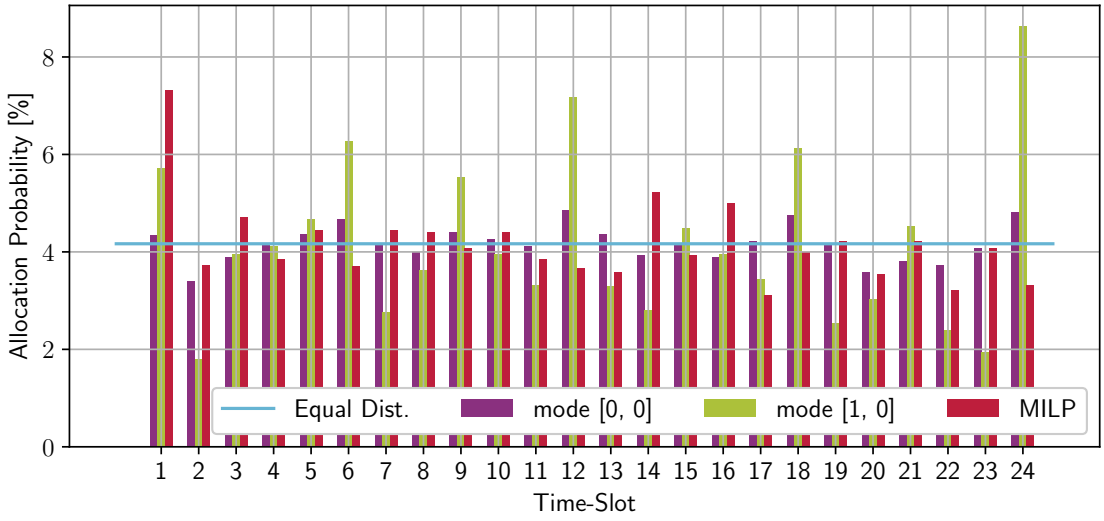
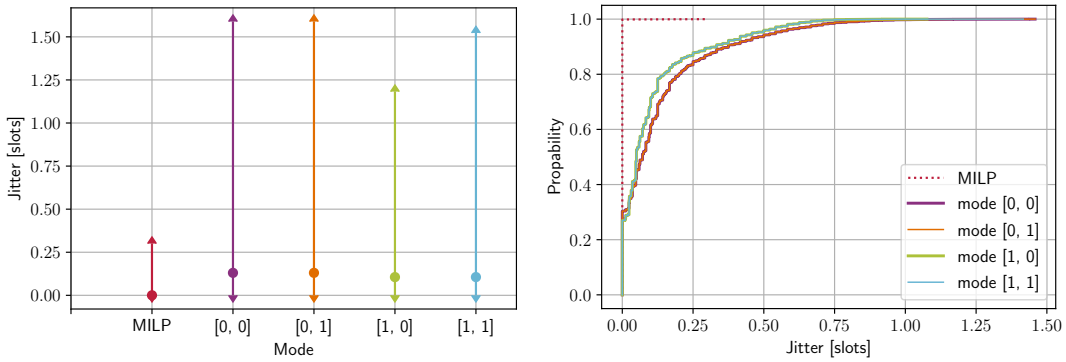


Figure 5.21: Comparison of the probability that a certain time-slot is allocated by the different scheduling approaches, for tasksets with hyperperiod length 24

5.8.4 Allocation Introduced Jitter

The jitter a scheduling approach introduces is, after its ability to schedule tasksets, one of the most important performance factors of a real-time scheduling approach. To evaluate this factor we compare the jitter introduced by the four modes of the proposed algorithm with the MILP-model in Figure 5.22. Figure 5.22a shows the minimum, mean and maximum jitter each mode introduces into a set of 56000 tasksets which are all scheduleable by all modes. As in the previous evaluations the *channel first shifting* outperforms the *time first shifting*. Especially with the *ascending age* order the maximum jitter is lower. The mean jitter on the other hand does not differ significantly over all

modes. The MILP-model with jitter optimization outperforms all algorithm modes, as it tries to find the minimal possible jitter.



(a) (b)

Figure 5.22: (a) depicts the mean, minimum and maximal jitter of the four algorithm modes in comparison to the MILP-model. (b) shows the CDF plot of the jitters, the two lines for the scheduling orders are overlapping for the two shifting approaches.

Figure 5.22b shows that the impact of the ordering is less significant than the impact of the shifting. The impact of the ordering is so small that the lines are actually overlapping in the graph.

The results of this evaluation are very promising especially for an algorithm that is not actively reducing the jitter but only relies on its allocation methods.

5.8.5 Performance of Rescheduling

To evaluate how good the different approaches are able to schedule combinations of tasksets we generated over two million combinations of the scheduleable tasksets. From these we randomly chose a set of 250 thousand combinations. This set was scheduled by all four modes of the proposed algorithm and the MILP-model as a reference. Figure 5.23 depicts the percentage of combinations that were scheduleable by each mode. As all evaluations above this shows that the *channel first shifting* is superior to the *time first shifting*.

The *channel first shifting* managed to schedule over 50 % of the combinations. This seems to be a quiet low percentage, but taking into consideration that we formed the combinations from tasksets with the same hyperperioid, job number and node number, the effect discussed in Section 5.8.3 is amplified here. As even more tasks need to be scheduled in the same time-slots, these combinations are the worst case combinations. Together with the fact that our tasksets are quite dense, Figure 5.21 shows that almost all time-slots are used at on least one channel, a success rate of 50 % is a promising result.

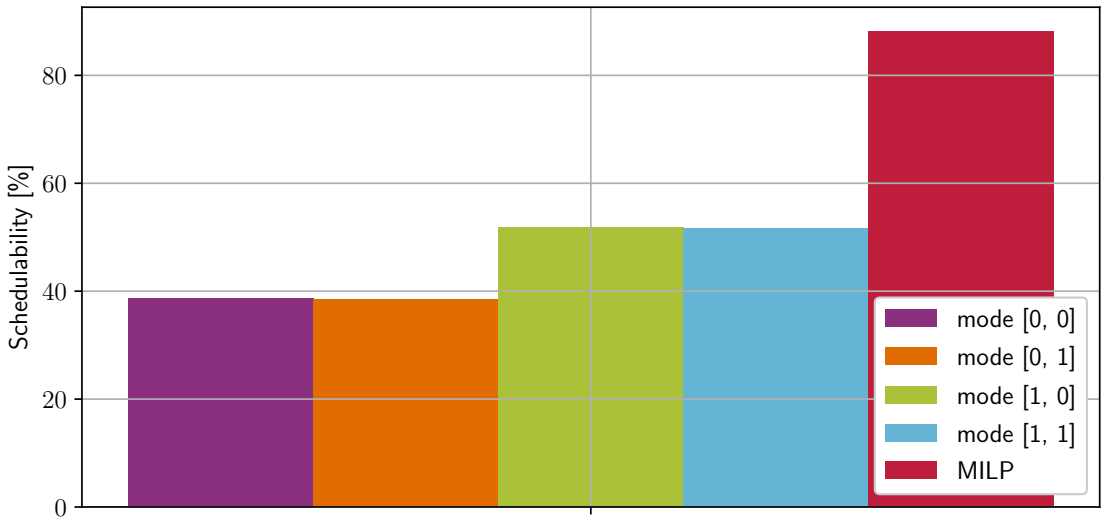


Figure 5.23: Comparison of the scheduling performance for taskset combinations

5.9 Conclusion

As cooperative CPSs are emerging in real-time applications, wireless connections become more and more important. These applications need scheduling algorithms which are able to adapt schedules to new network topologies and application requirements. In Section 5.1 we described how a cooperative CPS might be modeled and what the special challenges in these systems are. We discussed the constraints a scheduling algorithm for such systems needs to fulfill in Section 5.2. After a review of existing work in related areas we developed the MILP-model that generates schedules following the constraints discussed earlier. As it can be time consuming to solve the proposed MILP-model, especially on embedded devices, we propose an algorithm to generate which schedules has a lower computation time for almost all cases. To generate schedules that are easy to combine with other schedules we stated the hypothesis that two more sparse schedules are easier to combine than two dense schedules, even with the same hyperperiod and number of tasks. We proof the hypothesis in Section 5.6. In Section 5.7 we discuss the proposed algorithm and its four different modes in detail. The evaluations in Section 5.8 compares these four modes to each other and to the MILP-model. We show that the *channel first shifting* is the preferable of the two shifting modes. The order in which tasks are scheduled, on the other hand has no distinguishable influence on whether a taskset is scheduleable by the algorithm or not. The heuristic algorithm is clearly outperformed by the MILP-model, which is no surprise considering that the MILP-model searches for an optimal solution. However, it fulfills the need for a predictable runtimes much better than the MILP-model.

6 Investigating Concurrent Transmission

In a TC there is a lot of data that is transferred to close feedback loops and to fulfill other real-time applications' needs. For this communication we presented our TDMA network stack in Chapter 4. To enable a TC to fulfill its application in a changing environment we need to be able to communicate in a sporadic manner with other nodes in the TC and also to other TCs. This communication is needed to disseminate schedules to the whole TC or to manage the merge process between two TCs. For this traffic we proposed in Section 2.4 to use a Glossy [14] based protocol. These protocols are proven to be able to disseminate data fast and reliable through a network [13]. They do not need knowledge about the topology of the network to disseminate data, therefore they are well suited for the communication between different TCs.

Even though these protocols a perfect match for the management protocol of the system we discussed in Chapter 2, there are open questions. In the following we will present these questions and detail on our contributions to their answers, we give detailed insight in the results we presented in [60].

Glossy is a protocol that tries to exploit Constructive Interference (CI) of IEEE 802.15.4 symbols by using Concurrent Transmission (CT) of several transmitters [14]. Whether Glossy is really able to exploit CI or whether it just does not add destructive interference is an ongoing debate [40, 30, 27].

Another open question is how well CT-based protocols work on transceiver chips other than the CC2420 [7] and its successor the CC2520, both produced by Chipcon. To the best of our knowledge all research on CT using real-world test-beds is based on the CC2420. Only Brachmann et al. [6] used the CC2520 for some first work on CT using this chip. As the CC2420 is not recommended for new designs [7], results based on this chip should not drive the design of a novel communication architecture. To generalize the statements done on CT, we implemented a CT protocol on the Atmel AT86RF233 [3] which we will present and evaluate in this chapter.

6.1 Related Work

The idea to use flooding to disseminate data in a network was brought into WSN research community by protocols like Spin [19], Trickle [29] or Flash [33]. Compared to traditional routing protocols like CTP [18] or RPL [56], these approaches do not need to maintain a network graph to transfer data to its destination. This is an advantage if energy or time is limited, as their is no overhead of finding the right route. A disadvantage is that all data is transferred to all nodes in the network, which adds its own overhead to the communication. As we want to transfer the data, like schedules, to all nodes, this is no disadvantage in our use case.

In general all flooding protocols work according to the same method, they retransmit all messages they received via broadcast. To limit the transmissions each node counts how often it transmitted a certain message, if a predefined number is reached it stops retransmiting this message. Especially in dense networks this can lead to a massive amount of transmissions which leads to collisions. This phenomenon is called broadcast storm and increases the Packet Error Rate (PER) [65] due to collisions [52].

Glossy overcomes the issue by exploiting CI, therefore the collisions are not increasing the PER. The CI Glossy uses is not on the carrierband, as it is not possible to synchronize 2.4 GHz signals on WSN sensor nodes which are typically operating with a CPU clock rate between 1 MHz to 200 MHz. Thus, the signal is synchronized on the baseband. In IEEE 802.15.4 each group of four bit is encoded into pseudo-random noise sequence of 32 physical-layer bits called chips. These chips are Offset Quadrature Phase-Shift Keying (O-QPSK) modulated by the PHY. At 2.4 GHz each chip is $T_c = 0.5\,\mu s$ long, which results in a bit rate of 250 kbps. Therefore, Ferrari et al. argue that transmitters need to have a synchronization error smaller T_c to generate CI [14], although Wang et al. argue that it is not CI but non-destructive interference [55]. Non-destructive interference means that the interference an additional transmitter introduces to the signal neither deteriorates nor raises the quality of that signal.

6.2 Background on Concurrent Transmission (CT)

All CT protocols work based on the same principle, an initiator called node transmits a packet to all nodes in its transmission range. The receivers of that packet check whether they have to retransmit the packet and do so if needed. This retransmission is done as synchronized, by all transmitters, as possible to prevent destructive interference.

6.2.1 Glossy

The CT part of the different protocols differ in the way they decide whether a packet needs to be retransmitted and the way they synchronize the retransmission. Glossy introduces so called phases in which a packet is retransmitted for a predefined number N times. If the packet is received for the first time in a phase its data is provided to

the application on the node, after it was retransmitted. To retransmit a packet it is read from the transceiver buffer. Then it is checked whether the packet was retransmitted less than N times, if so, a counter is increased by one and the packet is written back to the transceiver and transmitted. If the packet was retransmitted N times the transceiver is shut down to avoid further receptions of that packet. This prevents the network from being occupied by this packet for eternity.

The receivers in Glossy can be seen as situated on rings, where all nodes in the transmission range to the initiator are on $Ring_1$. The nodes in transmission range to the nodes in $Ring_1$ are in $Ring_2$ and so on. Figure 6.1 shows the procedure of a transmission in Glossy with $N = 2$ and two of the rings. The layered boxes indicate the existence of more than just one node in a ring.

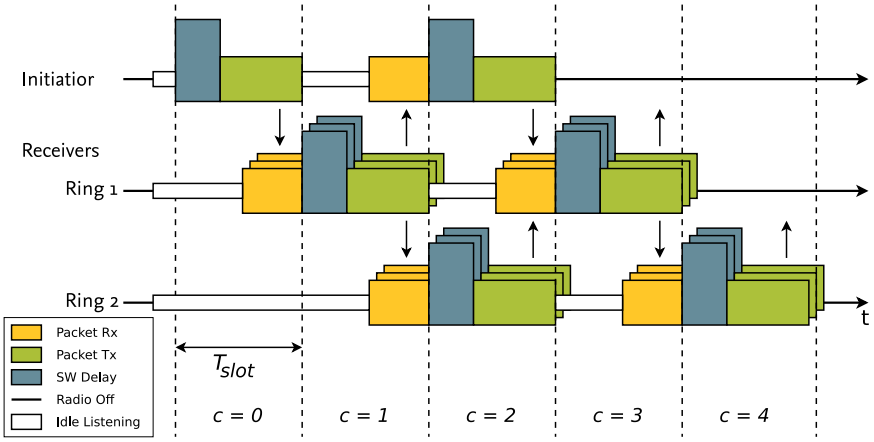


Figure 6.1: Example Glossy phase with $N = 2$ and two rings of receivers

To synchronize the transmission of all nodes, Glossy uses a relay-counter. This relay-counter is increased by each node that receives the packet for the first time, and is embedded into the packet. Therefore, nodes on the same ring should increase the relay-counter to the same value. Together with the slot-length T_{slot} and the relay-counter nodes can estimate how long the packet needed to reach them. This can be used for time synchronization.

6.2.2 Constructive Baseband Interference

In IEEE 802.15.4 data is encoded into chip sequences of 32 bit which represent 4 bit of payload data. At a data rate of 250 kbps each chip is therefore seems to be 0.5 µs long. To explain why Ferrari et al. stated that CI can happen as long as all senders start transmitting the packet within a span of 0.5 µs [14], we need to take a look what happens on the PHY.

IEEE 802.15.4 uses an O-QPSK modulation to bring the chip sequences to the baseband signal. An O-QPSK uses a complex signal, which consists of an in-phase signal (I) and a quadrature signal (Q), in its modulation. These signals are shifted by 90°, therefore the Q-signal is $1T_C$ (0.5 µs) following behind the I-signal. The chips are modulated to the I- and Q-signal in an alternating manner, starting with the I-signal. As the I- and Q-signal are non interfering by definition, one could understand them as completely independent, for now. To achieve the data rate of 250 kbps each, I and Q, only needs to transfer half of the data. Therefore, each chip is $2T_C$ (1 µs) long and if the interference is shorter than T_c less than half of the chip is interfered. Figure 6.2 depicts the whole process, from the bits in the MAC-layer over the chips in the PHY to the IQ-signal in the medium.

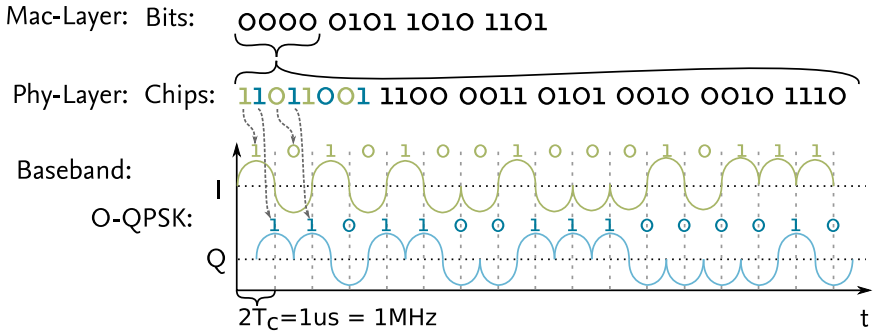


Figure 6.2: IEEE 802.15.4 modulation example, showing how four bits are modulated to the baseband signal using chip sequences and O-QPSK

.

6.3 Concurrent Transmission on AT86RF233

As mentioned before, as far as we know, all implementations of the CT approach are based on the CC2420 or the successor CC2520. To study whether CT would be applicable to other radios, we developed an implementation for another type of radio, the AT86RF233. This section will give insight to the benefits and challenges the AT86RF233 has compared to the CC2420 in CT applications. The AT86RF233 has a shared receive and transmit buffer, which stores the last received packet or the packet to be sent. By utilizing this shared buffer, the delay between the reception and the transmission of a packet should be shorter and more constant, compared to the CC2420. Using the CC2420 the entire packet has to be read from the transceiver's receive buffer to the CPU's RAM via SPI and then needs to be written back into the transmit buffer of the transceiver via SPI before it can be transmitted. In contrast, with the AT86Rf233 only a few bytes in the packet need to be manipulated inside the transceivers buffer. As this manipulations are independent from the length of the packet, the time needed to

process a packet before retransmiting should be more constant. This should lead to a better synchronization of the retransmissions of different nodes.

6.3.1 Implementation

Our implementation is focused on the basics needed to realize CT with the AT86RF233. We designed the implementation in a way that it needs minimum processing time to forward packets and that the forwarding time has minimum variance.

In Algorithm 1 it is shown which steps are executed to retransmit a packet following its reception. Our approach reads two byte from the frame buffer for each received

Algorithm 1 CT implementation on AT86RF233

```
 1: read(protocol_identifier)
 2: read(destination_address)
 3: if protocol_identifier == CT && destination_address == broadcast then
 4:     read(seq_num)
 5:     read(hop_count)
 6:     if seq_num not transmitted N times then
 7:         write(hop_count+1)
 8:         transmit packet
 9:         increment transmit_counter(seq_num)
10:     else
11:         read(payload)
12:         deliver(payload)
```

packet, destination address and protocol identifier. We check whether a packet is a broadcast packet and has the right protocol identifier to make sure we do not forward packet from other networks and therefore introduce unexpected behavior into them. If a packet is a broadcast packet and a CT packet we read the sequence number and the hop count. Based on the sequence number we decide whether the packet is retransmitted, if this sequence number was retransmitted less then `N` times it is retransmitted. Before retransmission the hop count is increased by one and written back to the frame buffer in the transceiver. Therefore, we read and write a constant number of byte per packet, five bytes are read and one byte is written. This is independent from the payload size, thus the time between reception and retransmission should be shorter and should not vary based on the payload size, in comparison to Glossy based on CC2420. This is due to the fact that Glossy needs to read the whole frame buffer each time a packet is received, as described in Section 6.2.2.

6.4 Comparative Evaluation

To determine whether there are characteristics of one of the transceivers, AT86RF233 or CC2420, that would hinder the usage in CT applications, we compare them in several ways. First we compare the differences in the timings between a transmission is

triggered by the CPU and the transceiver starts to transmit the packet into the medium. This is followed by a comparison of the Received Signal Strength (RSS) and Link Quality Indicator (LQI) under the influence of CT in a minimal setup. To verify that the transceivers do not add too much jitter in their packet reception path we did an evaluation on this aspect. At last we evaluated the ability to synchronize the time on different nodes of our protocol.

6.4.1 Transmission Start Timing

We start our comparison of the AT86RF233 and the CC2420 by comparing how long it takes the two transceivers to start a transmission after the CPU triggered it. This is an important metric; especially the variance of this time is interesting to be able to determine whether a transceiver is suitable for CT or not.

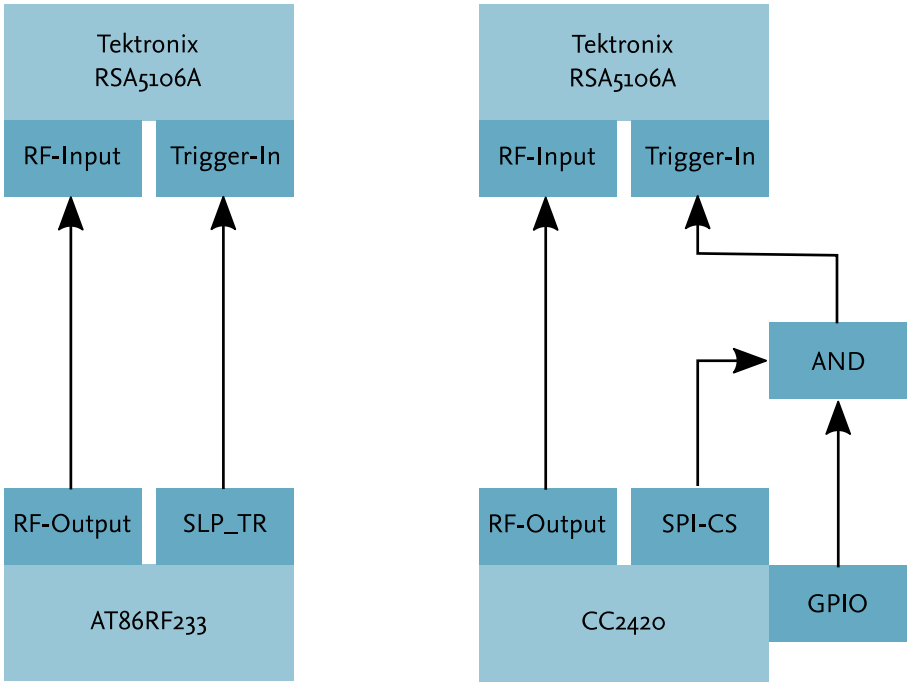
To measure the time we first had to define a point in time at which the CPU triggered the transmission. For the AT86RF233 this is quite simple, as a transmission is triggered by toggling the `SLP_TR` pin to its `HIGH`-state. This event can be triggered by a measurement device. For the CC2420 it is more complicated, as the transmission is triggered by a SPI command. To trigger on this command we chose to use the SPI-Chip Select (CS) pin, as it signals the client that the communication is over by returning to its `HIGH`-state. As the SPI-CS pin is toggled for each communication with the CC2420, reading/writing the frame buffer, configuration, etc., we needed to prevent theses false triggers. Therefore, we used a GPIO pin of the CPU which is only in `HIGH`-state when a transmission is about to be triggered. Both pins, SPI-CS and the GPIO are connected by a logic `AND` to generate a rising edge each time a transmission is triggered.

The second point in time we had to define is, when did the transmitter start to send. We chose it to be when the transmission reached 90% of its maximum energy. The energy was measured by a Tektronix RSA5106A real time signal analyzer.

To be able to measure the time between these two time points we connect the first signal to the `Trigger-IN` of the signal analyzer and the RF-signal to the `RF-Input`, as depicted in Figure 6.3.

Table 6.1: Transmission start timing results of cc2420 and AT86Rf233

	AT86RF233	**CC2420**
minimum delay [µs]	156.9333	190.7667
mean delay [µs]	156.9685	190.8539
maximum delay [µs]	157.0067	190.9400
standard deviation [µs]	0.0178	0.0370



(a) AT86RF233 (b) CC2420

Figure 6.3: Setup to measure the time between a transmission is triggered and the transceiver starts transmitting

For each transceiver we measured 500 transmissions and found that the standard deviation of the transmission start timing for both is below 40 ns. As this is less than a tenth of the allowed 0.5 µs, we consider both transceivers as suitable for CT. This assumption is supported by the other results listed in Table 6.1. The difference between the maximum and minimum delay is also below 0.5 µs for both transceivers. In general the AT86RF233 seems to start the transmission faster, this is most likely because it does not have to decode the SPI command. This might also explain the difference in the standard deviation, this however could also be introduced by our logic in the trigger path.

6.4.2 Minimal Concurrent Transmission (CT) comparison

To compare both protocols, ours and Glossy, in the most controlled environment, we chose a minimal setup, as depict in Figure 6.4. It consists of one initiator node and two retransmitters (Node1 and Node2). In this evaluation the retransmitters send the packet back to the initiator. To minimize external factors from disturbing our measurements, the retransmitters work in a round-robin manner. As shown in the right part of Figure 6.4, the first packet is only retransmitted by Node1, the second one only by Node2 and the third one is retransmitted by both. For all following packets this scheme is

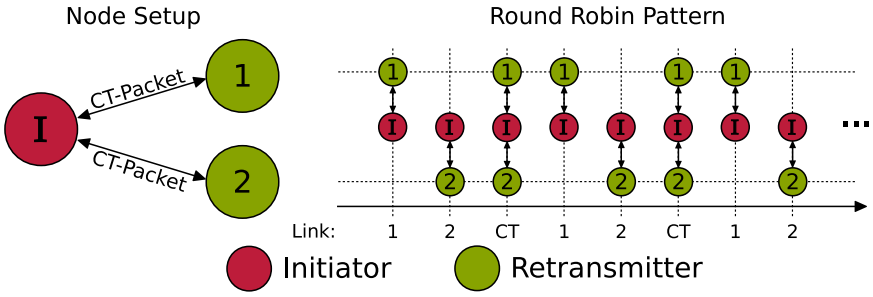


Figure 6.4: Evaluation setup with one initiator (Node I) and two retransmitters, Node1 and Node2.

repeated. The decision which node retransmits the packet is based on the sequence number of the packet and the node's id. Each time X in Equation (6.1) is zero, with the node_ids of the retransmitters being two and three, the packet is retransmitted.

$$X = (seq\_num - (node\_id - 2)\%3) \tag{6.1}$$

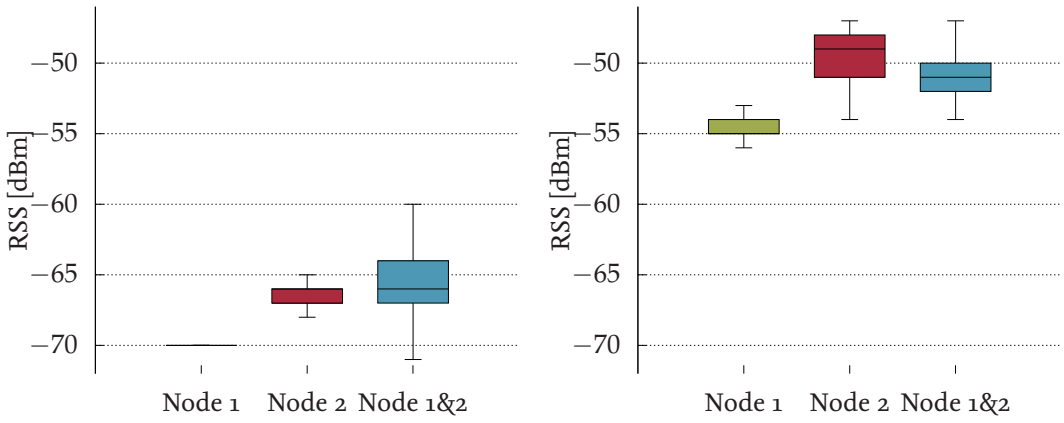
That way we have the two Single Transmission (ST) as a base line to the CT. All nodes in the setup are AT86Rf233 equipped if our protocol is evaluated and if Glossy is evaluated all nodes are equipped with a CC2420 transceiver.

To determine the overall performance of both protocols we evaluated the packet loss, with and without CT. In order to measure the influence of CT each packet was only retransmitted once ($N = 1$). We defined the packet loss as the number of packets that are not received by the initiator. Therefore we assume the link from the initiator to the retransmitters to be of equal quality for both transceivers. To get the most reliable results we placed the nodes with a spacing of ≈ 50 cm between each other and let the initiator transmit 5000 packets.

Table 6.2: Packet loss comparison between CT and no CT with only one retransmission for AT86RF233 and CC2420

Sending Nodes		**Packet Loss [%]**	
Node1	Node2	AT86RF233	CC2420
✓		5.764	5.096
	✓	6.715	2.166
✓	✓	6.603	4.841

The results in Table 6.2 show that our implementation has a slightly higher packet loss. As the packet loss is also higher for both nodes as a single transmitter, it is likely that this loss results from the AT86RF233 internals, the HF-path or antennas used. For both transceivers the packet loss is lower when CT is used compared to the ST packet



(a) AT86RF233 (b) CC2420

Figure 6.5: Comparison of RSSI values for the three different retransmission scenarios: only Node1 or Node2 are retransmitting or both (CT).

loss of one of the nodes but higher than the ST packet loss of the other node. Therefore, this evaluation shows that CT was not able to lower the packet loss significantly.

The RSS or Received Signal Strength Indicator (RSSI) is another value that is often used to determine whether CT is beneficial to a transmission [14, 55, 58]. As the RSSI is a transceiver specific value, we use the RSS which is more comparable. To calculate the RSS form the RSSI we used the suggested way from the CC2420 [7] or AT86RF233 [3] datasheet respectively. For the CC2420 we used Equation (6.2) and for the AT86RF233 we used Equation (6.3).

$$RSS_{CC2420} = RSSI - 45 \tag{6.2}$$

$$RSS_{AT86RF233} = -94 + RSSI \tag{6.3}$$

The RSSI can be read from both transceivers for every received packet. It gives the energy received by the antenna during the packet reception. The values shown in Figure 6.5 are measured at the initiator, the network was setup as for the packet loss measurements. For the AT86RF233 Figure 6.5a shows RSS values as one would expect them to be: in the case of multiple transmitters the RSS rises. Packets transmitted by Node1 had a very constant RSS of -70 dBm. However, the minimum RSS is lower for Node 1&2 than only Node2. An interesting finding is that the standard deviation of the RSS gets much bigger for the Node 1&2 CT case.

The results for the CC2420 look vastly different, shown in Figure 6.5b , here the CT configuration has a lower mean RSS than the Node2 only configuration. Even though the mean RSS for Node 1&2 is higher than the one for Node 1, we can not confirm the

hypothesis that CT leads to a higher RSS.

IEEE 802.15.4 defines a second metric for the quality of a received signal, besides the RSS, it is called Link Quality Indicator (LQI). It give a more complex insight whether the link between two nodes is good or not, it describes how well the receiver was able to decode the packet. To do so, it gives an estimation of the sameness between the received chip sequence and the most similar one from the IEEE 802.15.4 standard [23]. In the following we explain the LQI in detail to give a better understanding why it is supposed to qualify links under CT influence better than the RSS.

As mentioned in Section 6.2.2 IEEE 802.15.4 uses chip sequences to spread the bits and thus be more resilient to interference. There are 16 chip sequences defined to represent all states of the 4 bit symbols. In Section 6.2.2 we described how a symbol is translated into a chip sequence. To understand the LQI we need to understand how a received chip sequence is translated back into the corresponding symbol.

After a chip sequence is received, the receiver compares it to the 16 defined sequences. It will translate the received sequence to the symbol whose chip sequence is most similar to the received one. In Figure 6.6 we give an example of a chip sequence where the fourth chip is flipped to zero. With this spreading technique the receiver is able to decode the symbol correctly and to mask the flipped chip. The LQI describes the

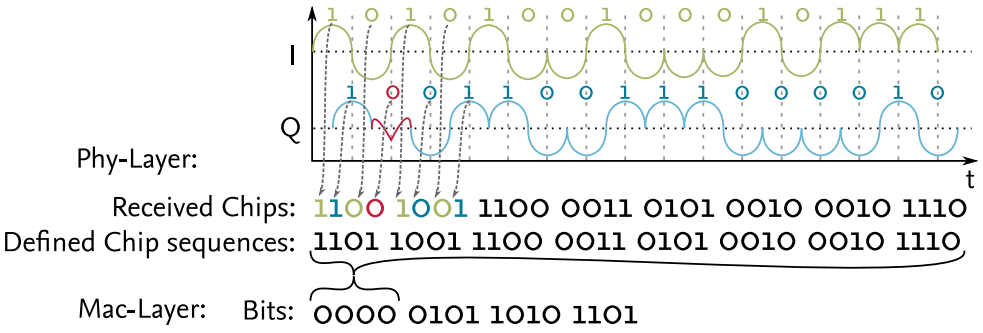
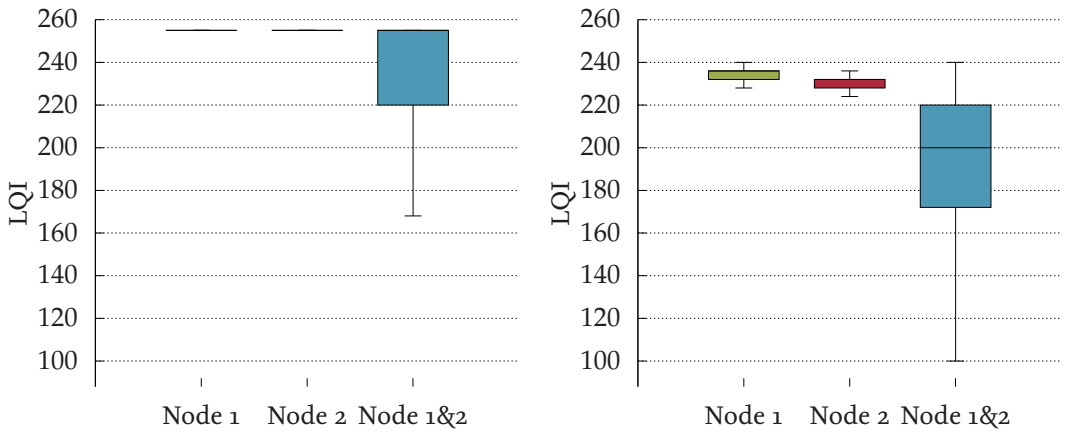


Figure 6.6: IEEE 802.15.4 demodulation example, how a chip sequence demodulated from the IQ-signal and translated into a four bit symbol.

difference between the received sequence and the defined sequence, therefore it gives a deeper insight to how well a signal was received. In contrast to the RSS the LQI is able to distinguish between an interfering and a contributing signal, under the assumption that an interfering signal would flip some of the chips in a sequence. Therefore, the LQI is the more appropriate metric to analyze the potential effect of CI during CT.

While recording the RSS we also recorded the LQI, the results are shown in Figure 6.7. For the AT86RF233 in the ST configurations the LQI was always 255 which is the maximum value and represents the best link quality. Therefore, the boxes for Node1 and Node2 are very thin in Figure 6.7a. The results for the CT configuration show a



(a) AT86RF233 (b) CC2420

Figure 6.7: Comparison of LQI values for the three different retransmission scenarios: only Node1 or Node2 are retransmitting or both (CT).

different situation, the maximum is still 255 but the minimum LQI is 167. Thus, the LQI shows that in fact it is interference which harms the signal and leads to errors in the chip sequences.

The results for the CC2420 show a very similar situation, shown in Figure 6.7b. Even though the LQI is not at its maximum for all transmission in the ST, the variance is a lot bigger in the CT configuration. The fact that the ST configuration did not result in the maximum value gives the opportunity to check whether the CT is able to increase the quality of some transmission. As shown in Figure 6.7b the maximum of Node1 and Node1&2 are at the same level. This indicates that the original Glossy was not able to introduce the intended CI in our setup.

6.4.3 Reception Start Timing

While implementing and testing our implementation we noticed a significant variation in the timing of the reception start interrupt (*rx_start*) on different AT86RF233 receiving the same message. This *rx_start* indicates the start of reception, it is triggered right after the PHY header was detected by the transceiver. To measure the difference in these timings we connected a logic analyzer to the *rx_start* pin of the transceivers.

In Figure 6.8 we illustrate the *rx_start* jitter (J_{rx_start}). Assuming a static processing time this jitter effects the *tx_start* jitter (J_{tx_start}). To measure the jitter our implementation adds while processing the packet ($J_{processing}$), we also connected the *tx_start* pin to our logic analyzer. The results of the evaluation are shown in Table 6.3 for both transceivers, the AT86RF233 and the CC2420.

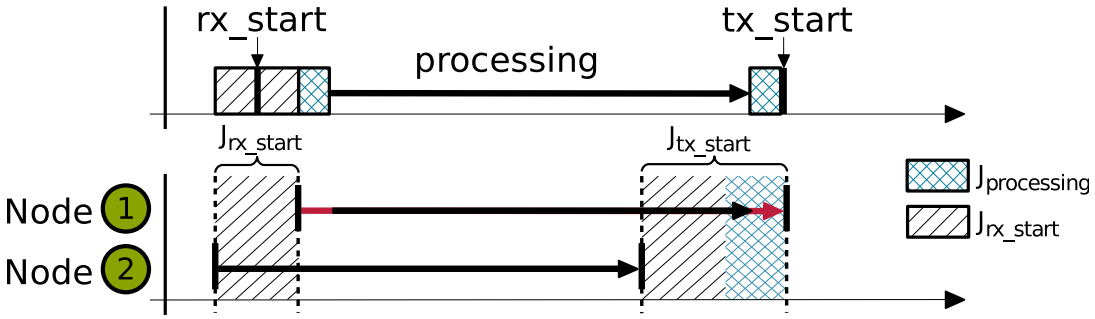


Figure 6.8: Illustration of the *rx_start* and transmission start signal (*tx_start*) jitter.

Table 6.3: Reception-Jitters for AT86RF233 and CC2420

Transceiver	**Signal**	**Jitter** [ns]	$_{min}$**Jitter**[ns]	$_{max}$**Jitter**[ns]
AT86RF233	J_{rx_start}	2±398.689	-1063	1063
	J_{tx_start}	40±429.083	-1313	1313
CC2420	J_{rx_start}	0±59.439	-188	188
	J_{tx_start}	61±134.685	-376	500

Both implementations show mean jitters which are within the range of 0.5 µs. However, the J_{rx_start} of the AT86RF233 has a jitter of over 1 µs, which also lead to a J_{tx_start} of the same magnitude. As $J_{processing}$ is defined by Equation (6.4) as the difference between J_{tx_start} and J_{rx_start}, both implementations show negligible values for $J_{processing}$.

$$J_{processing} = J_{tx\_start} - J_{rx\_start} \tag{6.4}$$

The more daunting jitter is J_{rx_start}, as it is solely caused by hardware internals of the AT86RF233. As there are no configuration variables to manipulate this logic, a mitigation of J_{rx_start} in software is impossible.

Although the jitter for the AT86RF233 is close to the 500 ns and therefore close to the threshold of CI [14], our evaluation in Section 6.4.2 shows that the packet loss is not significantly higher as for one transmitter.

6.4.4 Synchronization Comparison

In Glossy there is a synchronization method, as we mentioned in Section 6.2.1. To complete our comparison we did a brief evaluation how well our implementation is able to synchronize nodes. We used the same method and parameters as Ferrari et al. in [14] and set our retransmission counter to $N = 3$. During the evaluation we measured the time difference between the initiator and the retransmitters after each hop.

In Figure 6.9 we show the mean value of the time synchronization with the standard deviation for both retransmitting nodes separately. As a result, both nodes stay within

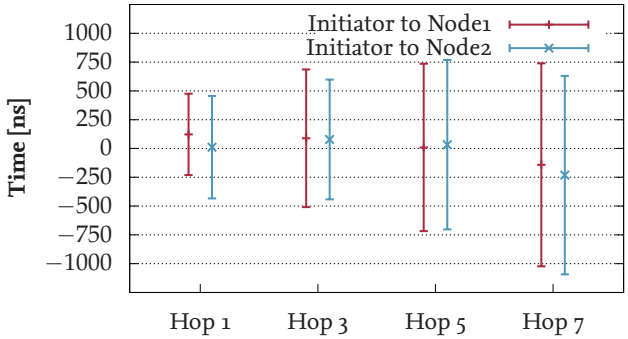


Figure 6.9: Time synchronization accuracy over multiple hops.

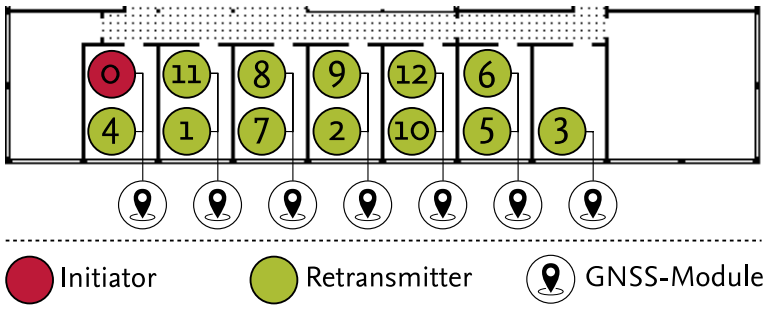


Figure 6.10: Testbed setup in our offices with connected GNSS-Modules for ground truth time synchronization.

the range of the synchronization error of Glossy given by Ferrari et al., in particular, $\leq 0.4 \pm 4.8\,\mu s$ [14].

6.5 Testbed Evaluation

In Section 6.4 we showed that our implementation has comparable performance to the original glossy. To evaluate the TC performance of the AT86RF233 in a real-world scenario, we deployed a testbed in our office building that is shown in Figure 6.10. As a reference time signal we used the GPS time, therefore we equipped each node with an ublox NEO-M8Q GNSS-Module [53]. To evaluate the packet delivery rate we used five different configurations for N ($N = 1, \cdots, 5$). For each configuration of N we gathered how many of the packets transmitted by the initiator were received by the different nodes. The average packet delivery rate as a function of the retransmission value `N` is summarized in Figure 6.11. The nodes that are located far away from the initiator need more retransmissions to be reached. This is due to the fact that more hops need to be made by the packet. Therefore, the chance to a packet being lost is higher. As an example, for $N = 1$ a delivery ratio of only 82 % is achieved for node 3, for node 4 it is higher than 99 %. For $N \geq 3$ the delivery ratio reaches 99 % for all nodes.

An interesting fact is that all nodes behind node 10 have a significant lower packet

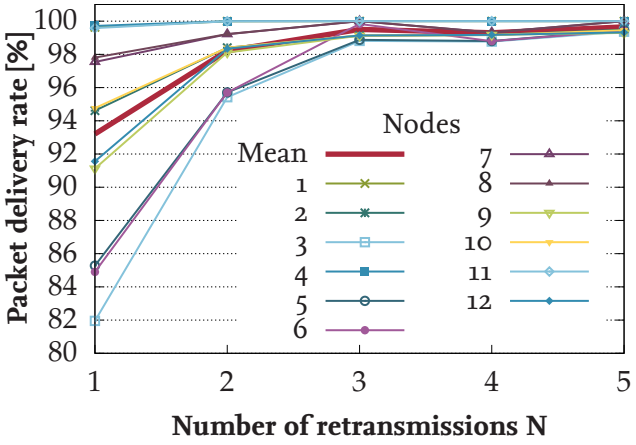


Figure 6.11: Reliability for different numbers of retransmissions.

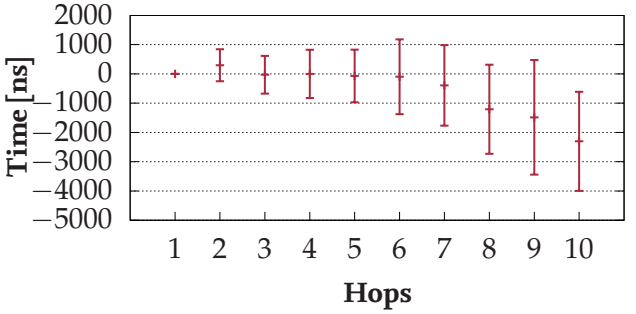


Figure 6.12: Time synchronization accuracy over number of hops.

reception rate than the other nodes. As all nodes are identical and node 10 is a lonely node we assume this might be an effect of not being able to perform CT as the other nodes.

To evaluate the time synchronization in real multihop scenarios we used the same testbed. For the ground truth time we connected the Pulse Per Second (PPS)-signal of the NEO-M8Q to an interrupt pin of our mirco-controller. Each time the interrupt was triggered the microcontroller gathered the timestamp. By comparing the timestamps we can calculate the error of the time synchronization. The PPS-signal of all GNSS-Modules is supposed to generate a pulse at the same time around the world. As the NEO-M8Q's PPS-signal has an accuracy of 60 ns [53] and our microcontroller runs at 8 MHz, Shannon's sampling theorem [47] says we are able to measure the timing accurately.

The results in Figure 6.12 show that on the one hand the mean time difference to the initiator stays in the same range as for the evaluation in Section 6.4. On the other hand, the standard deviation is larger due to the larger number of nodes tested.

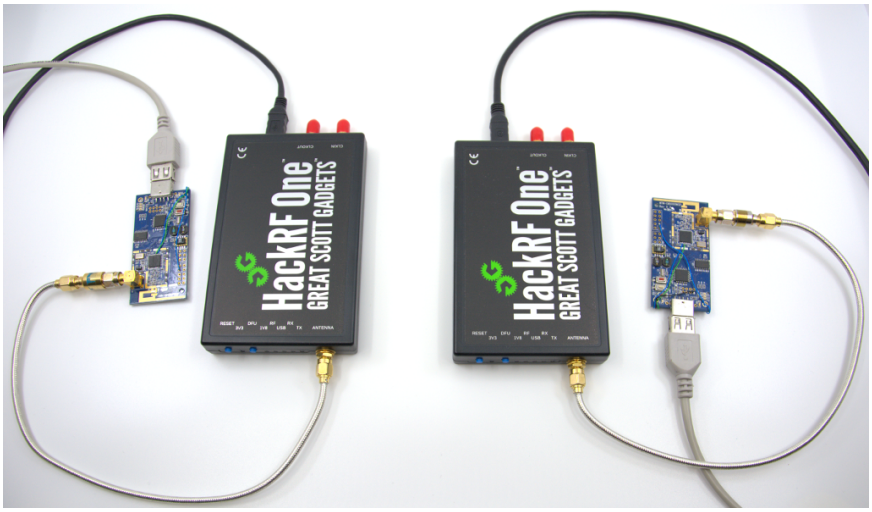


Figure 6.13: Emulator setup used for our emulations with two CC2420 based sensor nodes connected to the HackRFs by SMA-cables.

6.6 Concurrent Transmission Emulation

As the measurements taken in Section 6.4.3 raised doubts whether the AT86rf233 can be synchronized accurate enough to perform CT but the evaluation in Section 6.5 and Section 6.4.2 showed good results, a further investigation in CT was necessary. Therefore, we created a CT-emulator to be able to evaluate under which conditions CT is beneficial to the transmission.

6.6.1 Emulator Setup

The emulator consists of two *HackRF One*[1] Software Defined Radios (SDRs) and two transceivers of the type to be evaluated. The SDRs are connected to a PC running *GNURadio*[2]. As shown in Figure 6.13 each transceiver is connected to one SDR. In our emulation we define one transceiver as the receiver and the other one as the sender. The sender's signal is captured by the connected SDR and send to the PC. On the PC the signal can be manipulated using *GNURadio*. The manipulated signal is than transferred to the second SDR and sent to the receiver. To mitigate external influences we use coaxial cables to connect the transceivers to the SDRs.

To have the most similar signals in our CT emulations, we use only one sender. The signal of that sender is duplicated in *GNURadio*. Together with a *Delay*-block the duplication emulates the second sender that is transmitting the same signal a little delayed. In Figure 6.14 we depicted how a real-world CT scenario transforms into our emulations. By changing the delay we can emulate different synchronization accuracy

[1] https://greatscottgadgets.com/hackrf/
[2] https://www.gnuradio.org/

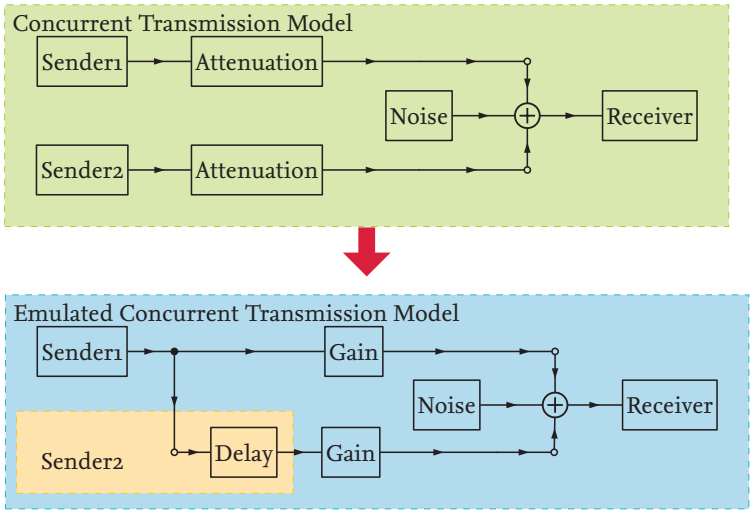


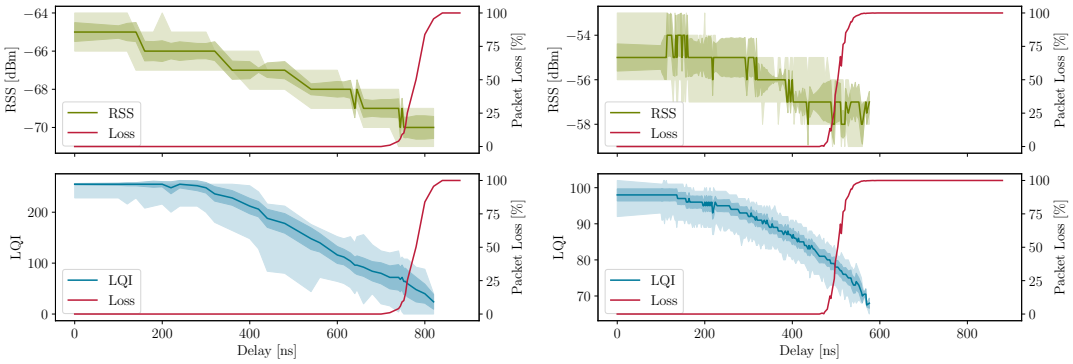
Figure 6.14: Conversion from the real world CT model to our emulated CT model, using a *Delay*-block to emulate a second, delayed transmitter.

between the senders. To emulate two links that are effected by different attenuation, we use the two *Gain*-blocks. As our emulation can be considered noise free we added a *Noise*-block to add different amounts of noise to the signal. To the best of our knowledge this is the first setup that can evaluate large numbers of packets under the same CT conditions with signals being transmitted and received by real WSN-hardware.

6.6.2 Noiseless Concurrent Transmission (CT) Emulation

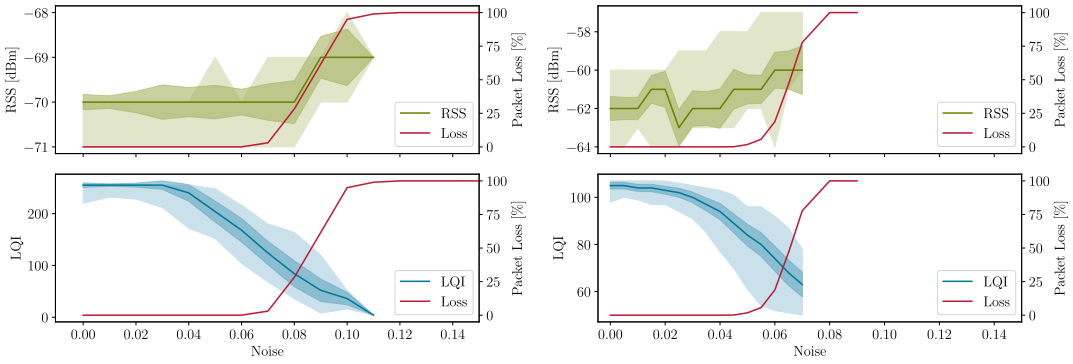
To evaluate CT under ideal conditions we started with emulating a noiseless channel between the senders and the receiver. The goal of this evaluations was to see what delay leads to a significant increase of the packet loss. Additionally we will show how the different transceivers handle different amounts of delay. For each delay setting we transferred at least 100 packets. For each received packet we recorded the RSS and LQI. In Figure 6.15 the packet loss, LQI and RSS are plotted over the delay we introduced into the emulation. The plots of RSS and LQI end at the point no more packets were received.

In Figure 6.16a the results for the AT86RF233 are plotted and show a non-surprising decline in the RSS and LQI for raising delays. Noticeable is the packet loss plot, it rises at about 700 ns, this is far later than the expected 500 ns known from the literature, e.g. by Ferrari et al. [14]. The LQI on the other hand starts to degrade from 250 ns. We argued before that the LQI shows how destructive the interference of the second sender is, therefore we can assume non-destructive interference stops at 250 ns. Another reason for a degrading LQI could be noise. However, as we did not add any noise in our emulator, the reason for the decreasing LQI must be the signal of the second sender.



(a) AT86RF233 (b) CC2420

Figure 6.15: AT86RF233 and CC2420 CT performance in an ideal evironment without noise.



(a) AT86RF233 (b) CC2420

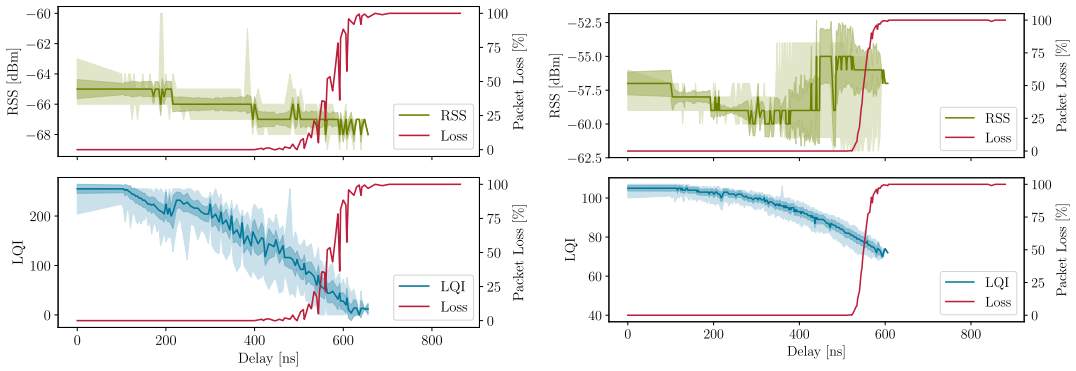
Figure 6.16: Single transmission under noise influence. Noise is given without unit due to emulator limitations.

In contrast to the evaluation of the AT86RF233 the packet loss results for the CC2420, shown in Figure 6.15b, support the assumption that CT becomes destructive interference for delays > 500 ns. The LQI data reveals that the signal deteriorates much earlier, in particular at 150 ns.

6.6.3 Noise Effected Concurrent Transmission (CT) Emulation

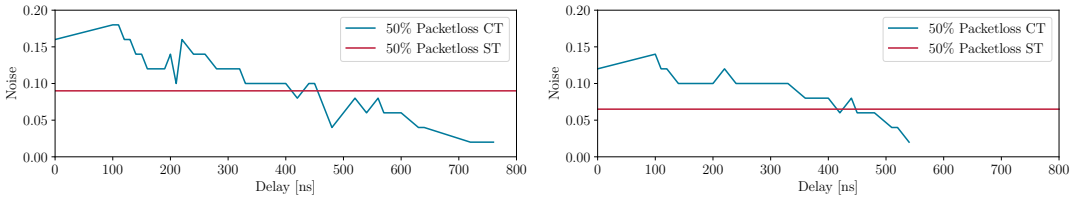
Assuming a noise free channel is a rather unrealistic assumptions. To get a baseline how the noise affects the signal from a single transmitter we disabled the second path in our emulator and measured packet loss, LQI and RSS for several noise levels. Figure 6.16 shows the results for this evaluation. As our emulator is not able to measure the incoming signal's energy in physical units, our noise level is also unit free. It is given as a relation to the signals amplitude, which was 0.5 on the same scale.

Both RSS plots, for the CC2420 and the AT86RF233, show that the RSS increases with



(a) AT86RF233 (b) CC2420

Figure 6.17: Concurrent Transmission (CT) under noise influence. The noise level was chosen to the level where a single transmitter had 99 % packet loss (0.07). Noise is given without unit due to emulator limitations.



(a) AT86RF233 (b) CC2420

Figure 6.18: 50 % packet loss threshold over different noise levels and transmission delays. The red line shows the noise level at which a ST has 50 % packet loss. Noise is given without unit due to emulator limitations.

the noise. The LQI plots show how an increased noise level decreases the LQI, as discussed in Section 6.4.2. This decreased signal quality lead to a higher packet loss, when the transmission error cannot be masked by the transceiver. As the packet loss in Figure 6.16b reached 100 % at a noise level of 0.07 we stopped the measurements at a noise level of 0.09. Interestingly, the AT86RF233 is able to receive packets effected with a higher noise level than the CC2420, see Figure 6.16a. This might be an explanation why the AT86RF233 is able to decode CT signals with a higher transmission delay than the CC2420.

Based on the results of the prior evaluation we determine the noise level at which the receiver was only able to decode 1 % of the sent packets. This step was done for each transceiver individual, afterwards the lower one (0.07) was taken to evaluate both transceivers. Having disturbed signals, CT should improve the signals quality at the receiver, as the two signals add up to an amplitude of 1.0 in the best case. We chose to

use the same noise level for both transceiver chips, CC2420 and AT86RF233, to make the evaluation more comparable.

Figure 6.17 shows the results of the same evaluation as in Section 6.6.2 but whith a noise level of 0.07. The RSS has a higher variance for both transceivers, for the AT86RF233 it is decreasing monotonically but for the CC2420 it decreases between 0 ns and 200 ns but increases again from 500 ns. Interestingly, the delay threshold at which the packet loss start to rise is almost the same for the CC2420, comparing the noise effected evaluation (Figure 6.17b) and the noiseless one (Figure 6.15b). For the AT86RF233 this threshold is about 200 ns lower compared to the noiseless evaluation (Figure 6.15a). This observation leads to the assumption that, at least for the AT86RF233, the ability to decode CT signals depends on the noise level on the channel.

To test this assumption we evaluated how the packet loss is linked to the transmission delay in combination to the noise level. Figure 6.18 depicts the highest noise level where at least 50 % of the packets were received as a function of the transmission delay. The red line represents the noise level at which a ST suffers form 50 % packet loss. The ST packet loss describes a vertical line as it is not effected by the transmission delay. We chose to plot 50 % packet loss as we see it as a border where links must be considered unreliable. Every other percentage would show a similar graph only shifted, squeezed or stretched. This evaluation proves the ability to decode CT signals depends on the noise level in the channel. The behavior is similar for both evaluated transceivers. Figure 6.18 also shows that a CT signal with more than 400 ns transmission delay is less likely to be decoded correctly than a ST signal, even though the CT signal has twice as much energy. Therefore, to get a real gain from CT the transmission delay should be far below 500 ns.

6.7 Conclusion

In this chapter we presented our minimalist implantation of a CT protocol on an AT86RF233. To verify that it performs CT as well as Glossy on the CC2420 we performed a comparative evaluation between the two implementations on their corresponding transceiver chip. In this evaluation we used the minimal setup of three nodes and compared the packet loss, RSS and LQI of both transceivers in CT and ST configurations. This evaluation showed a similar performance for both transceivers. To verify that our implementation is able to operate in larger networks we performed an evaluation in a testbed in our office envirionment. The results of this evaluation also showed similar results for our implementation as the original evaluation of Glossy [14].

The LQI values of the comparative evaluation did raise doubts whether CT is beneficial to the quality of the received signal. These doubts were strengthened by the evaluation of the jitter of the reception start timing. To investigate under which conditions CT is beneficial we developed an emulator to evaluate CT under the most controlled conditions possible, using real hardware transceivers. These evaluations showed that

the manageable delay between the two concurrent transmissions that does not lead to packet loss, differs between the two transceivers. Further we were able to show that the acceptable delay also depends on the noise level in the channels used for the CT.

As we have shown, it is possible to use CT with transceiver chips other than the CC2420. Utilizing such a technique to coordinate several TCs is beneficial, as for example schedules can be transferred to several TCs that need to be merged. However, to use CT in an industrial environment the ability for heterogeneous networks, consisting of nodes with different transceiver chips, would be necessary to ensure interchangeability between robots and machines of different manufactures. To form such networks a protocol that evens out the various timing differences would be necessary. One way to realize such a protocol would be to measure all timings, like the *rx_start* and *tx_start* delay or the time to copy bytes into the packet buffer, of all the different transceivers in the network and delay all timing to fit the slowest one. As this a labor-intensive, a way to automate the adjustments would be necessary to deploy networks with a number of different transceivers.

7 Conclusion

The use cases for CPSs are ranging from industrial automation to search-and-rescue applications. For now it is not possible to use mobile CPSs in applications that require tight cooperation of multiple CPSs. One reason is the lack of a communication infrastructure that supports real-time communication and that is still flexible enough to adapt to changes in the network's topology and configuration.

7.1 Contributions

In this thesis we proposed an architecture for wireless real-time networks which are able to adapt to application, topology and environmental changes without any interruption in executing the applications with performance guarantees? Most importantly we designed the system in a way which ensures that real-time requirements of the communication are not harmed during the adaption to the changed situation. In Chapter 2 we introduced a scenario of mobile robots which need to cooperate to fulfill their task, we called such a group of robots TC. Further, we motivated why such TCs need to be able to change their topology. Based on the identified challenges, we proposed an architecture consisting of four main components: time synchronization, the real-time network stack, the scheduling algorithm and the management protocol. This architecture needs to be able to perform three operations to manage a TC. The first one is the synchronization of several TCs in order to make sure they share a common time base. This is needed to perform the second operation: the merge operation. This operation is used to integrate nodes or TCs into another existing TC. The third operation is the split operation, which is used to separate TCs. Following to the architectural overview we took a closer look to the different components and how they are designed to cooperatively execute these three operations.

To achieve a time synchronization accurate enough to close feedback loops over several nodes, we propose a time synchronization, in Chapter 3, which is bases on PTP. By taking well proven principles from PTP and optimizing them to the special features and challenges in our system we were able to reduce the communication overhead. As nodes that are equipped with low-cost crystal oscillators show a significant drift between their time, we introduce a drift compensation that is able to minimize the drift to a minimum. In a real world evaluation we were able to show that the time synchronization is able to synchronize the time on all nodes on a sub-microsecond range. Utilizing such an accurate time synchronization offers the ability to develop a TDMA-based network stack that needs minimal guard times.

In Chapter 4 we present such a TDMA-based network stack to handle the real-time communication needed between the nodes in one TC. The main design goal for this stack was to keep the jitter and delay it introduces into the communication between tasks as low as possible. To guarantee that the time between a task is executed and the transfer of the data needed or provided by that task is constant, we introduced the node scheduler. This node scheduler combines the scheduling of a task's execution and the scheduling of the data transfer in one schedule. As each data transfer has a defined time-slot and data is supposed to be transferred at the beginning of this time-slot, the task execution is schedule at the end of the previous time-slot. Thus, the transmission start time as well as the execution start time have a minimum jitter between two executions of the same task. While developing the network layer great care was taken to make sure that the time, needed to deliver packets to a local task and to load packets into the transceiver buffer, is as constant as possible. Another goal was to utilize multicast traffic to be able to deliver data to several tasks at different nodes at the same time. This feature makes it easier to use the same feedback on different nodes to close one ore several loops. Moreover, it helps to save resources. The whole network stack was designed in a way that changing the schedule or the configuration of the transceiver adds only a minimum amount of jitter to the communication and execution of tasks. In a real world evaluation we proved that our network stack is able to keep the jitter in the realm of few microseconds, even if schedules and configurations were changed.

The scheduling of all tasks needed to fulfill an application on the different nodes in a TC is challenging. Especially the dependencies between theses tasks and the fact that nodes can only listen to one communication at a time makes the scheduling challenging. The scheduling problem gets even more complex if mobility is introduced and the schedules need to be adapted to certain changes without harming real-time requirements of applications that have to continue while the changes are applied. In Chapter 5 we present the constraints a scheduling algorithm needs to follow to create schedules that fulfill all requirements needed by the TC. To get a baseline on how much jitter certain tasksets have in an optimal schedule and how many tasksets in our evaluation set of tasksets are scheduleable, we implemented a MILP based scheduler. Through extensive evaluation we found that the computational complexity of solving our MILP-model on embedded devices is too high. It is important to notice here that the new schedule has to be generated between the time two TCs recognized each other until they are in transmission range of the TDMA network.

To generate schedules with a lower computational complexity, we introduced a heuristic approach. This approach is based on the hypothesis that schedules with evenly distributed tasks have a higher possibility to be mergeable than schedules in which tasks are scheduled very densely. To proof this hypothesis we used our MILP-model and merged the tasksets in our evaluation set. In a comparative evaluation we showed

that the heuristic has a much more constant runtime than the MILP-model especially over different schedule lengths. Not only is the runtime more constant, it is also much shorter in almost all cases, only for very short schedules the MILP-model is faster. By setting constant and short runtimes as the design goal for the heuristic the performance in terms of minimum jitter and schedulable tasksets suffered, as our evaluation showed. Nevertheless, both approaches generate valid schedules and the decision which one is more applicable depends on the application, e.g. the speed at which robots operate and their computation power.

On of the most important tasks of the management protocol is to distribute important information in a TC and also to neighboring TCs. In Chapter 6 we identified that protocols based on CT, like Glossy, are a good option to disseminate information in a TC. As protocols from this class lack the concept of a network topology, they can also be used to send this information to neighboring TCs using multihop communication. Their ability to synchronize time in multihop scenarios helps to prepare the merge of TCs. Saving the time for routing and other forwarding computation is beneficial, as this time can be used to react to the received information. As all research on data dissemination protocols utilizing CT is based on the same transceiver family, it was unknown how such protocols would perform on other hardware. To determine the generality of the statements made about CT, we implemented a minimalist CT protocol for the AT86RF233 transceiver. In a comparative evaluation we determined that the performance of the original Glossy and our implementation is similar in terms of packet loss and synchronization ability. These results were strengthened by a real-word testbed evaluation in our offices. Additionally, we evaluated the effect which CT has on the LQI. This evaluation raised serious doubts on the claimed non-destructiveness of the interference which CT adds to a signal. To investigate the influence CT has on a signal, we introduced an emulation setup that mitigates influences from the environment to the signal. Based on these evaluations we showed that even small timing differences between the senders deteriorate the signal quality. This deterioration is masked by the chip spreading technique used in IEEE 802.15.4. Therefore, CT-based protocols are applicable for information dissemination in and between TCs. However, for now such networks must be very homogeneous as timings of different transceiver types might differ vastly and present protocols cannot handle such differences.

7.2 Outlook

Beside the contrition we made, there are still challenges to solve until real-time networks can adapt to changing environments autonomously and do this fully transparent to the application. In this section we discuss some of the challenges which are left open and also some questions that arose throughout this work.

The most important challenge in the time synchronization is to generalize our

approach to different hardware platforms. To make our approach applicable to an industrial application, it needs to be manufacturer independent. This is to a certain degree also true for our TDMA network stack, as it relies on the same hardware features. However, most of the mechanisms used to guarantee the timing accuracy are hardware independent.

A more important question to be answered is: how much can applications influence the accuracy if they behave in unexpected ways or even maliciously? This leads directly to the challenge of hardening the system against such behavior and also to general questions about security in such networks. This comprises not only encryption of the transmitted information and how this effects the jitter of transmission, receptions, and delivery to tasks, but also about access control and how to incorporate nodes into a TC and how/whether TCs can be merged even though their communication needs to be confidential to each other.

Although these questions are reaching into the realm of the scheduling algorithm, there are further challenges in the scheduling of TCs. One would be to gather real-world application task sets and to optimize the heuristic approach to these applications in order to minimize the jitter and maximize the ability to combine the tasksets. Another way to enhance the scheduling performance would be to minimize the MILP-model and therefore reduce the complexity to solve it. The most promising way to enhance the ability to merge TCs for both scheduling approaches is to allow schedules that are only used in order do make the original schedules mergeable. For now, two schedules are merged in only one step. Using several iterations of schedules in both TCs would loosen the jitter and maximal data age constraints and, thus, ease the task of merging TCs.

To evaluate the risk of such frequent schedule changes, a CT-based management protocol needs to be integrated into the node scheduling system of the TDMA network stack. An open question regarding CT is: why do some transceivers decode CT signals with certain inter-transmission delay correct while other do not. To answer this question in-depth knowledge of the differences in the internals of these transceivers would be necessary. As this knowledge is intellectual property of the manufactures, it is most likely very hard to come by. A more interesting challenge from a research perspective is how heterogeneous CT networks could adjust their timings autonomously so that all transceivers can participate in a CT.

The most crucial questions however is: how accurate can an application developer estimate the requirements of the applications and how do the variations between the estimation and reality influence the performance of an application as a whole.

Acronyms

rx_start reception start interrupt. 91, 92

tx_start transmission start signal. 91, 92

5G fifth generation. 12

C-LLF Conflict-aware Least Laxity First. 52

CDF Cumulative Distribution Function. 58–63, 78

CI Constructive Interference. 81–83, 90–92

CPN Cyber-Physical Network. 1, 2

CPS Cyber-Physical System. v, 1, 4, 5, 61, 79, 101

CS Chip Select. 86

CT Concurrent Transmission. v, xi, 4, 18, 81, 82, 84–91, 94–100, 103, 104

DAG Directed Acyclic Graphs. 53

FreeRTOS Free Real Time Operating System. 31–33

HART Highway Addressable Remote Transducer. 29

IPC Inter Process Communication. 33

ISR Interrupt Service Routine. 20, 33

KOI Koordinierte Industriekommunikation. 12

LAN Local Area Network. 20, 22

LCM least common multiple. 51, 59

LQI Link Quality Indicator. 86, 90, 91, 96–99, 103

LTE Long-Term Evolution. 12, 52

LWB Low-power Wireless Bus. 30

MAC Medium Access Control. 11, 18, 20, 21, 30

MARS Mobility-Aware Real-Time Scheduling for Low-Power Wireless Networks. 11, 12, 30

MILP Mixed Integer Linear Programming. v, x, 4, 47, 54–56, 58–60, 63, 64, 72–79, 102–104

NIC Network Interface Controller. 20, 22

NTP Network Time Protocol. 20

O-QPSK Offset Quadrature Phase-Shift Keying. 82, 84

OS Operation System. 22

PER Packet Error Rate. 82

PHY Physical Layer. x, 5, 9, 16–19, 22, 30, 34, 35, 82–84, 91

PLL Phase Locked Loop. 19, 24

PPS Pulse Per Second. 94

PTP Precision Time Protocol. ix, 19–23, 27, 101

RSS Received Signal Strength. 86, 89, 90, 96, 97, 99

RSSI Received Signal Strength Indicator. 89

RTT Round Trip Time. 19

SD Standard Deviation. 26

SDR Software Defined Radio. 95

SPI Serial Peripheral Interface. 41, 43–46, 84, 86, 87

ST Single Transmission. 88–91, 98, 99

TC Task Cluster. 2–9, 13–19, 22, 23, 31, 32, 34, 36, 37, 40, 41, 48, 49, 53, 59–61, 64, 72–75, 81, 93, 100–104

TDMA Time Division Multiple Access. 4, 5, 11, 19, 22, 27, 29–32, 34, 37, 46, 81, 101, 102, 104

UART Universal Asynchronous Receiver Transmitter. 24

UAV Unmanned Aerial Vehicle. v, 1, 9, 10

UWB Ultra Wide Band. 22, 34

WirelessHART WirelessHART. 1, 29, 30, 52

WISA Wireless Interface for Sensors and Actuators. 3, 29, 30

WSN Wireless Sensor Network. 11, 30, 82, 96

Bibliography

[1] *ANSI/ISA-100.11a-2011 Wireless systems for industrial automation: Process control and related applications*. International Society of Automation. 2011.

[2] S. A. Ashraf, I. Aktas, E. Eriksson, K. W. Helmersson, and J. Ansari. "Ultra-reliable and low-latency communication for wireless factory automation: From LTE to 5G." In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. Sept. 2016, pp. 1–8. DOI: 10.1109/ETFA.2016.7733543.

[3] *AT86RF233 Datasheet: Low Power, 2.4GHz Transceiver for ZigBee, RF4CE, IEEE 802.15.4, 6LoWPAN, and ISM Applications*. 8315E–MCU Wireless–07/14. Atmel Corporation. San Jose, July 2014.

[4] J. Baillieul and P. Antsaklis. "Control and Communication Challenges in Networked Real-Time Systems." In: *Proceedings of the IEEE* 95.1 (Jan. 2007), pp. 9–28. ISSN: 0018-9219. DOI: 10.1109/JPROC.2006.887290.

[5] D. Baumann, F. Mager, M. Zimmerling, and S. Trimpe. "Control-Guided Communication: Efficient Resource Arbitration and Allocation in Multi-Hop Wireless Control Systems." In: *IEEE Control Systems Letters* 4.1 (Jan. 2020), pp. 127–132. ISSN: 2475-1456. DOI: 10.1109/LCSYS.2019.2922188.

[6] M. Brachmann, D. Becker, and S. Santini. "Towards Enabling Concurrent Transmissions in Heterogeneous Networks." In: *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*. IPSN '15. Seattle, Washington: ACM, 2015, pp. 354–355. ISBN: 978-1-4503-3475-4. DOI: 10.1145/2737095.2737164.

[7] *Chipcon CC2420 Datasheet*. SWRS041c. Texas Instruments. 2017.

[8] E. G. Coffman and R. L. Graham. "Optimal scheduling for two-processor systems." In: *Acta informatica* 1.3 (1972), pp. 200–213.

[9] B. Dezfouli, M. Radi, and O. Chipara. "Mobility-aware real-time scheduling for low-power wireless networks." In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. Apr. 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524594.

[10] A. Dreher and D. Mohl. *Präzise Uhrzeitsynchronisation - IEEE 1588 (White Paper)*. Tech. rep. Neckartenzlingen, Germany: Hirschmann Automation and Control GmbH, 2005.

[11] J. Du and J. Y.-T. Leung. "Complexity of scheduling parallel task systems." In: *SIAM Journal on Discrete Mathematics* 2.4 (1989), pp. 473–487.

[12] *DW1000 User Manual - How to use, configure and program the DW1000 UWB transceiver.* Decawave Ltd. 2015.

[13] 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.* SenSys '12. Toronto, Ontario, Canada: ACM, 2012, pp. 1–14. ISBN: 978-1-4503-1169-4. DOI: 10.1145/2426656.2426658.

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

[15] S. Gallenmüller, R. Glebke, S. Günther, E. Hauser, M. Leclaire, S. Reif, J. Rüth, A. Schmidt, G. Carle, T. Herfet, W. Schröder-Preikschat, and K. Wehrle. "Enabling Wireless Network Support for Gain Scheduled Control." In: *Proceedings of the 2Nd International Workshop on Edge Systems, Analytics and Networking.* EdgeSys '19. Dresden, Germany: ACM, 2019, pp. 36–41. ISBN: 978-1-4503-6275-7. DOI: 10.1145/3301418.3313943.

[16] S. Ganeriwal, R. Kumar, and M. B. Srivastava. "Timing-sync Protocol for Sensor Networks." In: *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems.* SenSys '03. Los Angeles, California, USA: ACM, 2003, pp. 138–149. ISBN: 1-58113-707-9. DOI: 10.1145/958491.958508.

[17] P. R. Giordano, A. Franchi, C. Secchi, and H. H. Bülthoff. "A passivity-based decentralized strategy for generalized connectivity maintenance." In: *The International Journal of Robotics Research* 32.3 (2013), pp. 299–323. DOI: 10.1177/0278364912469671.

[18] O. Gnawali, R. Fonseca, K. Jamieson, M. Kazandjieva, D. Moss, and P. Levis. "CTP: An Efficient, Robust, and Reliable Collection Tree Protocol for Wireless Sensor Networks." In: *ACM Trans. Sen. Netw.* 10.1 (Dec. 2013), 16:1–16:49. ISSN: 1550-4859. DOI: 10.1145/2529988.

[19] W. R. Heinzelman, J. Kulik, and H. Balakrishnan. "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks." In: *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking.* MobiCom '99. Seattle, Washington, USA: ACM, 1999, pp. 174–185. ISBN: 1-58113-142-9. DOI: 10.1145/313451.313529.

[20] T. C. Hu. "Parallel Sequencing and Assembly Line Problems." In: *Operations Research* 9.6 (1961), pp. 841–848. DOI: 10.1287/opre.9.6.841.

[21] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems." In: *IEC 61588:2009(E)* (Feb. 2009), pp. C1–274. DOI: 10.1109/IEEESTD.2009.4839002.

[22] "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)." In: *IEEE Std 802.15.1-2005 (Revision of IEEE Std 802.15.1-2002)* (2005). DOI: 10.1109/IEEESTD.2005.96290.

[23] *IEEE Standard for Local and metropolitan area networks – Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs).* Piscataway: IEEE, Sept. 2011.

[24] "Industrial Communication Networks - Wireless Communication Network and Communication Profiles - WirelessHART." In: *IEC Standard 62591* (2010).

[25] J. Kannisto, T. Vanhatupa, M. Hännikäinen, and T. D. Hämäläinen. "Precision Time Protocol Prototype on Wireless Lan." In: *Telecommunications and Networking-ICT 2004*. Springer, Aug. 2004, pp. 1236–1245.

[26] 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*. Sept. 2008, pp. 899–907. DOI: 10.1109/ETFA.2008.4638503.

[27] O. Landsiedel, F. Ferrari, and M. Zimmerling. "Chaos: Versatile and Efficient All-to-all Data Sharing and In-network Processing at Scale." In: *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. SenSys '13. Roma, Italy: ACM, 2013, 1:1–1:14. ISBN: 978-1-4503-2027-6. DOI: 10.1145/2517351.2517358.

[28] M. Lévesque and D. Tipper. "A Survey of Clock Synchronization Over Packet-Switched Networks." In: *IEEE Communications Surveys & Tutorials* 18.4 (2016), pp. 2926–2947.

[29] P. Levis, N. Patel, D. Culler, and S. Shenker. "Trickle: A Self-regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks." In: *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*. NSDI'04. San Francisco, California: USENIX Association, 2004, pp. 2–2.

[30] C. H. Liao, Y. Katsumata, M. Suzuki, and H. Morikawa. "Revisiting the So-Called Constructive Interference in Concurrent Transmission." In: *2016 IEEE 41st Conference on Local Computer Networks (LCN)*. Nov. 2016, pp. 280–288. DOI: 10.1109/LCN.2016.56.

[31] R. Lim, R. Da Forno, F. Sutton, and L. Thiele. "Competition: Robust Flooding using Back-to-Back Synchronous Transmissions with Channel-Hopping." In: *EWSN*. 2017, pp. 270–271.

[32] R. Lim, B. Maag, and L. Thiele. "Time-of-Flight Aware Time Synchronization for Wireless Embedded Systems." In: *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*. EWSN '16. Graz, Austria: Junction Publishing, 2016, pp. 149–158. ISBN: 978-0-9949886-0-7.

[33] J. Lu and K. Whitehouse. "Flash Flooding: Exploiting the Capture Effect for Rapid Flooding in Wireless Sensor Networks." In: *IEEE INFOCOM 2009*. Apr. 2009, pp. 2491–2499. DOI: 10.1109/INFCOM.2009.5062177.

[34] F. Mager, D. Baumann, R. Jacob, L. Thiele, S. Trimpe, and M. Zimmerling. "Feedback Control Goes Wireless: Guaranteed Stability over Low-power Multi-hop Networks." In: *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*. ICCPS '19. Montreal, Quebec, Canada: ACM, 2019, pp. 97–108. ISBN: 978-1-4503-6285-6. DOI: 10.1145/3302509.3311046.

[35] M. Nabi, M. Geilen, T. Basten, and M. Blagojevic. "Efficient Cluster Mobility Support for TDMA-Based MAC Protocols in Wireless Sensor Networks." In: *ACM Trans. Sen. Netw.* 10.4 (June 2014), 65:1–65:32. ISSN: 1550-4859. DOI: 10.1145/2594793.

[36] T. O'donovan, J. Brown, F. Büsching, A. Cardoso, J. Cecílio, J. Do Ó, P. Furtado, P. Gil, A. Jugel, W.-B. Pöttner, U. Roedig, J. Sá 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." In: *ACM Transactions on Sensor Networks* 10.1 (Dec. 2013), 4:1–4:40. ISSN: 1550-4859. DOI: 10.1145/2529975.

[37] R. Oung and R. D'Andrea. "The Distributed Flight Array." In: *Mechatronics* 21.6 (2011), pp. 908–917. ISSN: 0957-4158. DOI: 10.1016/j.mechatronics.2010.08.003.

[38] R. Oung and R. D'Andrea. "The Distributed Flight Array: Design, implementation, and analysis of a modular vertical take-off and landing vehicle." In: *The International Journal of Robotics Research* 33.3 (2014), pp. 375–400. DOI: 10.1177/0278364913501212.

[39] N. Rajagopal, P. Lazik, N. Pereira, S. Chayapathy, B. Sinopoli, and A. Rowe. "Enhancing Indoor Smartphone Location Acquisition Using Floor Plans." In: *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*. IPSN '18. Porto, Portugal: IEEE Press, 2018, pp. 278–289. ISBN: 978-1-5386-5298-5. DOI: 10.1109/IPSN.2018.00056.

[40] V. S. Rao, M. Koppal, R. V. Prasad, T. V. Prabhakar, C. Sarkar, and I. Niemegeers. "Murphy loves CI: Unfolding and improving constructive interference in WSNs." In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. Apr. 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524539.

[41] J. W. Romanishin, K. Gilpin, S. Claici, and D. Rus. "3D M-Blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions." In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 1925–1932.

[42] F. Rosenthal, M. Jung, M. Zitterbart, and U. D. Hanebeck. "CoCPN – Towards Flexible and Adaptive Cyber-Physical Systems Through Cooperation." In: *2019 16th IEEE Annual Consumer Communications and Networking Conference (CCNC)* (2019), pp. 1–6.

[43] J. Rüth, R. Glebke, K. Wehrle, V. Causevic, and S. Hirche. "Towards In-Network Industrial Feedback Control." In: *Proceedings of the 2018 Morning Workshop on In-Network Computing*. NetCompute '18. Budapest, Hungary: ACM, 2018, pp. 14–19. ISBN: 978-1-4503-5908-5. DOI: 10.1145/3229591.3229592.

[44] A. Saifullah, Y. Xu, C. Lu, and Y. Chen. "Real-time scheduling for WirelessHART networks." In: *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*. IEEE. 2010, pp. 150–159.

[45] Y. Schröder, D. Reimers, and L. Wolf. "Accurate and Precise Distance Estimation from Phase-based Ranging Data." In: *2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. Nantes, France, Sept. 2018.

[46] S. Shakkottai and A. L. Stolyar. "Scheduling algorithms for a mixture of real-time and non-real-time data in HDR." In: *Teletraffic Science and Engineering*. Vol. 4. Elsevier, 2001, pp. 793–804.

[47] C. Shannon. "Communication in the Presence of Noise." In: *Proceedings of the IRE* 37.1 (Jan. 1949), pp. 10–21. ISSN: 0096-8390. DOI: 10.1109/JRPROC.1949.232969.

[48] P. Sommer and Y. A. Pignolet. "Competition: Dependable Network Flooding using Glossy with Channel-Hopping." In: *EWSN*. 2016, p. 303.

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

[50] Z. Teng and K.-I. Kim. "A Survey on Real-Time MAC Protocols in Wireless Sensor Networks." In: *Communications and Network* 2.2 (2010), pp. 104–112.

[51] 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)*. Mar. 2011, pp. 515–519. DOI: 10.1109/ISCI.2011.5958969.

[52] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu. "The Broadcast Storm Problem in a Mobile Ad Hoc Network." In: *Wirel. Netw.* 8.2/3 (Mar. 2002), pp. 153–167. ISSN: 1022-0038. DOI: 10.1023/A:1013763825347.

[53] *ublox NEO-M8 Datasheet*. 802.15.1-2011. ublox. Aug. 2016.

[54] N. Wang, Q. Yu, H. Wan, X. Song, and X. Zhao. "Adaptive Scheduling for Multi-cluster Time-Triggered Train Communication Networks." In: *IEEE Transactions on Industrial Informatics* 15.2 (Feb. 2019), pp. 1120–1130. ISSN: 1551-3203. DOI: 10.1109/TII.2018.2865760.

[55] Y. Wang, Y. Liu, Y. He, X. Y. Li, and D. Cheng. "Disco: Improving Packet Delivery via Deliberate Synchronized Constructive Interference." In: *IEEE Transactions on Parallel and Distributed Systems* 26.3 (Mar. 2015), pp. 713–723. ISSN: 1045-9219. DOI: 10.1109/TPDS.2014.2312198.

[56] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and J. V. andR. Alexander. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. RFC 6550 (Proposed Standard). IETF, Mar. 2012.

[57] O. N. C. Yilmaz, Y. P. E. Wang, N. A. Johansson, N. Brahmi, S. A. Ashraf, and J. Sachs. "Analysis of ultra-reliable and low-latency 5G communication for a factory automation use case." In: *2015 IEEE International Conference on Communication Workshop (ICCW)*. June 2015, pp. 1190–1195. DOI: 10.1109/ICCW.2015.7247339.

[58] D. Yuan and M. Hollick. "Let's talk together: Understanding concurrent transmission in wireless sensor networks." In: *38th Annual IEEE Conference on Local Computer Networks*. Oct. 2013, pp. 219–227. DOI: 10.1109/LCN.2013.6761237.

[59] P. Zand, S. Chatterjea, K. Das, and P. Havinga. "Wireless industrial monitoring and control networks: The journey so far and the road ahead." In: *Journal of sensor and actuator networks* 1.2 (2012), pp. 123–152.

[60] G. von Zengen, A. Baumstark, A. Willecke, U. Kulau, and L. C. Wolf. "How Different Transceiver Hardware Effects Concurrent Transmissions in WSNs." In: *International Conference on Distributed Computing in Sensor Systems (DCOSS 2018)*. New York, USA, June 2018, pp. 139–146.

[61] 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. DOI: 10.1109/icit.2017.7915545.

[62] G. von Zengen, K. Garlichs, and L. C. Wolf. "AIRCoN-Stack - Introducing Flexibility to Wireless Industrial Real-Time Applications." In: *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies*. NEAT '18. Budapest, Hungary: ACM, 2018, pp. 39–44. ISBN: 978-1-4503-5907-8. DOI: 10.1145/3229574.3229578.

[63] G. von Zengen, Y. Schröder, S. Rottmann, F. Büsching, and L. C. Wolf. "No-Cost Distance Estimation Using Standard WSN Radios." In: *The 35th Annual IEEE International Conference on Computer Communications (INFOCOM 2016)*. San Francisco, USA, Apr. 2016. DOI: 10.1109/INFOCOM.2016.7524540.

[64] G. von Zengen, Y. Schröder, and L. C. Wolf. "A Communication Architecture for Cooperative Networked Cyber-Physical Systems." In: *Proceedings of the 1st IEEE Workshop on Cyber-Physical Networking (CPN)*. CPN 2019. Las Vegas, USA: IEEE, Jan. 2019.

[65] J. Zhao and R. Govindan. "Understanding packet delivery performance in dense wireless sensor networks." In: *Proceedings of the 1st international conference on Embedded networked sensor systems*. ACM. 2003, pp. 1–13.