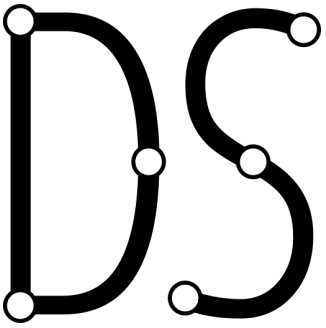
Bachelor thesis

Design and Evaluation of Deadline-Based Scheduling Algorithms for the Industrial Internet of Things

Michalski, Patrik Thomas



Technical Faculty
Department of Computer Science
Distributed Systems Research Group
Kiel University
Kiel, Germany 2020

Design and Evaluation of Deadline-Based Scheduling Algorithms for the Industrial Internet of Things
Michalski, Patrik Thomas

Advised by:
First supervisor: Prof. Dr. Olaf Landsiedel
Second supervisor: M.Sc. Oliver Harms

Bachelor Thesis
Technical Faculty
Department of Computer Science
Distributed Systems Research Group
Kiel University
D-24118 Kiel, Germany
Telephone: +49 431 880-00

Eidesstattliche Erklärung

Hiermit erkläre ich (Michalski, Patrik Thomas) an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel, den 28. September 2020

If I had more time,

I would have written a shorter letter.

Benjamin Franklin

Design and Evaluation of Deadline-Based Scheduling Algorithms for the Industrial Internet of Things
Michalski, Patrik Thomas
Department of Computer Science
Kiel University

Abstract

The Internet of Things is a rapidly increasing and promising technology which becomes more and more present in our everyday lives. It combines physical devices consisting of an embedded system with actuators, sensors, and network connectivity. It allows objects to collect and exchange data. This interconnection enables further integrating the physical world into computer-based systems and improves accuracy, economic benefit, and efficiency. Further research achievements allow a co-existence of an increasingly large number of Bluetooth and Wi-Fi devices, which is essential, especially in the Industrial Internet of Things [1]. Otherwise, this co-existence results in a massive amount of interferences, limited stability, and reliability [1]. Those networks often depend on deadlines and require stable communication with guaranteed low latency and high reliability. Appropriate scheduling algorithms are required to fulfill these requirements.

This thesis introduces our approach to dynamic schedulability analysis. We proposes three modified deadline-based scheduling solutions (i.e., Earliest Deadline First, Rate-monotonic Scheduling, and Conflict-aware Least Laxity First) that consider these requirements and extend MASTER to take into account deadlines, periods, and hyper-periods. We show how these solutions perform in terms of end-to-end latency, reliability, networking energy consumption, and schedulability by evaluating them in Cooja, a network simulator, and 20 nodes testbed at Kiel University. Our comparison with MASTER's Reverse Longest Path First algorithm shows that deadline-based scheduling outperforms regarding schedulability by a negligent latency increase.

Keywords: Industrial Internet of Things, Wireless Sensor-Actuator Networks, Time-Slotted Channel Hopping, MASTER, schedulability analysis, scheduling.

Acknowledgements

Typically, I would now take out a list of names and thank everyone in my life, starting with the midwife, who held me first. It would probably take at least the same number of pages as the thesis, and I already exceed the bachelor thesis boundaries. Therefore I will go for the sake of brevity and thank only the most related people. I want to start with my family and friends who have supported me all the time! - Mainly, my parents Katharina and Waldemar Michalski, for their endless love and support in all aspects. Without you, I would not be the person, and in the position I'm today! - Thank you for everything. I love you! I also would like to thank my friends, who have been so marvelous and supportive and have inspired me so much in the past. I'm ready for the next years with you! Certainly, I want to express my sincere thanks to my advisors Prof. Dr. Olaf Landsiedel and M.Sc. Oliver Harms for being an endless source of knowledge! - Thank you for your guidance and help whenever I ran in misconceptions or problems. Another great thank you goes to Oliver, who was a big help in finding and fixing bugs. I'm also very thankful for the relaxed and warm-hearted atmosphere in our working group! It was a big honor to be part of the Distributed Systems Research Group.

Finally, I would like to thank everyone who has somehow improved this thesis, helping me avoid mistakes and misunderstandings. You are all terrific!

Michalski, Patrik Thomas, Kiel, September 2020.

Contents

Acronyms

ACK Acknowledgement. 9
ALICE Autonomous Link-based Cell Scheduling. 15
AMUS Adaptive Multi-hop Scheduling method. 16
ARM Advanced RISC Machine. 37
ASN Absolute Slot Number. 7

C-LLF Conflict-aware Least Laxity First. 16, 17, 23, 27, 28, 30, 32, 39, 41
CSMA/CA Carrier-sense Multiple Access with Collision Avoidance. 4, 9
CSMA/CD Carrier-sense Multiple Access with Collision Detection. 4

DiGS Distributed Graph routing and autonomous Scheduling. 15, 16
DLL Data Link Layer. 3, 4

EB Enhanced Beacon. 8
EDF Earliest Deadline First. 23–25, 32, 38, 39, 41
ETX Expected transmission count. 10–13, 16, 37

FDM Frequency Division Multiplexing. 7
FDMA Frequency Division Multiple Access. 6
FHSS Frequency Hopping Spread Spectrum. 6
FSK Frequency Shift Keying. 6
FTDM Frequency-Time Division Multiplexing. 7, 8, 27

HART Highway Addressable Remote Transducer. 6
HSL Hopping Sequence List. 7

IEEE Institute of Electrical and Electronics Engineers. 3, 4, 6, 7, 9, 37
IIoT Industrial Internet of Things. 1, 13, 14, 16, 17, 30, 35
IoT Internet of Things. 1, 3, 37
ISA International Society of Automation. 5

LCM Least Common Multiple. 31
LLC Logical Link Control. 4
LLF Least Laxity First. 27

MAC Medium Access Control. 4, 7, 10, 16

MCU Microcontroller Unit. 3, 37

NACK Negative-Acknowledgement. 9

OSI Open Systems Interconnection. 31
OSI model Open Systems Interconnection model. 4

PC-LLF Path-Collision aware Least Laxity First. 16, 17
PDR Packet Delivery Ratio. 2, 40
PRR Packet Reception Rate. 11

QoS Quality of Service. 1, 16

R-LPF Reverse Longest Path First. 2, 11, 35, 38–41, 43
RAM Random-access Memory. 37
RM Rate-monotonic. 25
RMS Rate-monotonic Scheduling. 17, 23, 25, 26, 32, 39, 41
RPL Routing Protocol for Low-Power and Lossy Networks. 15
RTOS Real-Time Operating Systems. 23, 25

SJF Shortest Job First. 11, 16
SoC System on a Chip. 37
SPF Shortest Path First. 11, 16

TCP Transmission Control Protocol. 4
TDM Time Division Multiplexing. 7
TDMA Time Division Multiple Access. 6
TSCH Time-Slotted Channel Hopping. 3, 5, 7–10, 13, 15, 16, 27, 29, 31–33, 35, 37
TSM TSCH Schedule Matrix. 8

WCO Wireless Control Systems. 43
WSAN Wireless Sensor-Actuator Networks. 1

List of Algorithms

List of Figures

List of Tables

1
Introduction

Today, with an increasing number of devices connected to the Internet, the Internet of Things (IoT) is enormously growing and earning further acclaims in different directions, e.g., industry, self-driving cars, and smart homes. IoT needs low-power communication solutions that are able to satisfy changing application requirements and are easy to use. Especially in the Industrial Internet of Things (IIoT), where real-time embedded systems and Quality of Service (QoS) play an essential role [2], further research achievements allow building more efficient smart factories. Digitalization and exploitation of new technologies potentials should lead to significant improvements, such as an increase in automation and a shortened period between developing a new product and its market launch. Self-driven production lines are useful in real-time manufacturing processes and mass customizations of products. The car industry uses fully automated robots that communicate with each other to build complex components. These real-time systems depend on end-to-end deadline constraints and require stable wireless communication with guaranteed low latency and high reliability. End-to-End deadline constraints require scheduling algorithms. Such algorithms operate either in an autonomous, centralized, or distributed manner (see Section 2.6). Commonly a schedule is determined by a centralized scheduling algorithm that collects all information about the network topology, e.g., wireless link connectivity with possible interferences, the number of nodes, and their so-called (traffic) flows [1]. Other the past year's many different approaches for real-time scheduling algorithms are developed. IoT-devices have limited capabilities in terms of computation speed, latency, memory or power availability. Moreover, IIoT networks co-exist with an increasingly large number of Bluetooth and Wi-Fi devices [1]. This co-existence results in a massive amount of interferences, limited stability, and reliability of, e.g., Wireless Sensor-Actuator Networks (WSAN) [1]. Such limitations and non-interference-free channels are to take into consideration by calculating optimal routes and transmission.

After introducing the advantages, intention, and requirements of using end-to-end deadline constrained algorithms for scheduling real-time transmissions in IIoT, the theoretical background is given an overview of relevant concepts and discusses the circumstances in which our modified approaches operate.

The chapter on related work classifies this proposed work and shows different approaches for solving associated problems. In the next chapter, the assumptions and design choices are taken during the implementation and evaluation are discussed. Afterward, the evaluation shows how different scheduling strategies affect performance (i.e., end-to-end latency, reliability, networking energy consumption, and schedulability). A comparison with MASTER's Reverse Longest Path First (R-LPF) algorithm shows how a non-deadline based scheduler behaves in the same context. We show that deadline-based scheduling solutions outperform in terms of schedulability by an insignificant latency increase and no impact on the networking energy consumption. Furthermore, we explain the reason why different scheduling strategies have different PDRs. Finally, in conclusion, we summarize the thesis and providing possible research ideas extending this work.

2
Theoretical Background

In this chapter, all relevant concepts are covered, including the needed background knowledge of TSCH, a MAC layer specified in the IEEE 802.15.4e-2015 standard [3], MASTER [1], a high-level centralized scheduler designed for TSCH, and about the principle of scheduling, an NP-complete problem [4]. Overall, this chapter describes the following:

- Contiki-NG (see Section 2.1), an operating system for resource-constrained devices.
- The Data Link Layer (DLL) (see Section 2.2) and TSCH affiliation to it.
- Multi-hop Wireless Networks (see Section 2.3), an essential backbone for everything following.

2.1 Contiki-NG

Contiki-NG is a lightweight operating system developed for low-power Microcontroller Unit (MCU) and resource-constrained devices for the IoT. It contains an RFC-compliant, low-power IPv6 communication stack, enabling Internet connectivity by not supporting explicit power-saving abstractions [5]. Contiki-NG runs and supports many various platforms. These MCUs are limited in the CPU capacity and memory size. The code footprint has an order of 100 KB, and the memory usage can be as low as 10 KB [5]. To reduce memory usage while still supporting a multi-threaded environment, Contiki-NG builds upon an event-driven system. Contiki-NG implements processes as event handlers. All processes use the same stack and share the same memory resources, which lower the total memory usage. Protothreads simplify programming by reducing the abstraction of the complex state machine. They are providing a blocking wait condition to reduce the number of state machines. A program code runs in a cooperative mode, which means that it is executed sequentially concerning other cooperative codes. Contiki-NG does not have any interrupt handler for cooperative code, making it impossible to invoke it. Since interrupts are essential, Contiki-NG supports another mode called preemptive mode to executed preempting program code and reacting to external events. Preemptive code can stop the cooperative code by providing preemptive multi-threading on top of an event-driven kernel.

2.2 Layer two - The data link layer

The Data Link Layer is the second layer of the seven-layered Open Systems Interconnection model (OSI model). It is responsible for correcting possible errors made by the physical layer, identifying and transferring data between network entities (see Figure 2.1). The Data Link Layer provides (reliable) transmission of data frames between two nodes connected by a physical layer. It transfers the packet from the network to the physical layer by encapsulating data packets into frames and synchronizing them. The IEEE 802 LAN/MAN standards define the two sublayers, Logical Link Control (LLC) and Medium Access Control (MAC). MAC controls the hardware responsible for flow control, interacts, and multiplexes with the optical, wired, or wireless transmission medium. It determines the access to the media at any time using Carrier-sense Multiple Access with Collision Detection (CSMA/CD) protocols for collision detection and re-transmission and Carrier-sense Multiple Access with Collision Avoidance (CSMA/CA) protocol for collision avoidance in wireless networks. Moreover, it allows data packet queuing and physical addressing using MAC addresses and scheduling (see Section 2.4.2). Most data link protocols do not provide any acknowledgments, error checking, flow control, or re-transmissions, which need to be implemented by higher-level protocols, e.g., Transmission Control Protocol (TCP), to guarantee a reliable connection.

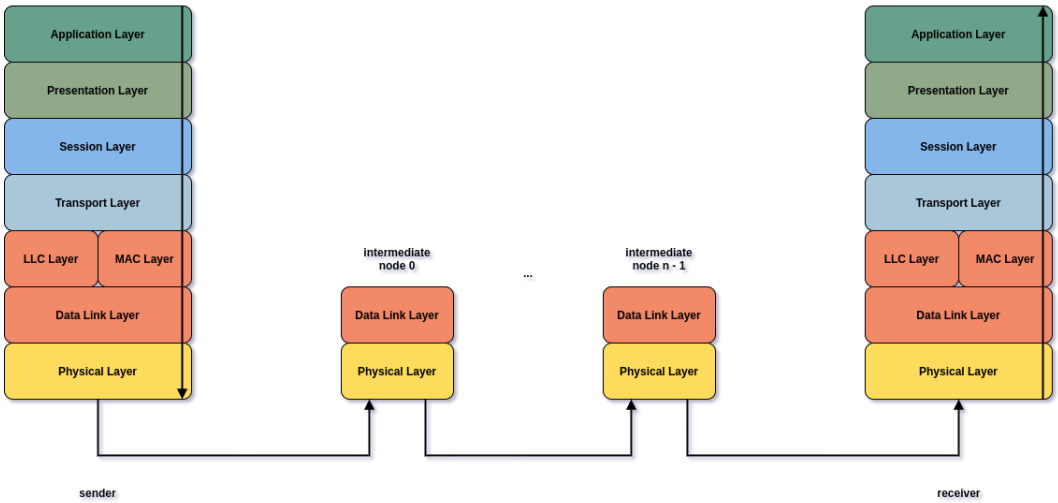


Figure 2.1: Example of the Open Systems Interconnection stack with communication over the Data Link Layer.

2.3 Multi-hop Wireless Networks

Multi-hop Wireless Networks are often called mesh networks because of the topology formed by the links and nodes. These networks consist of nodes that communicate over wireless connections, which commonly use digital packet radios [6]. Their communication range limits these nodes because the radio signal cannot reach all nodes directly, either because of interferences or power availability. Thus, nodes use other nodes to transmit their data over the network. These cooperating intermediate nodes act as relays which forward the data to the destination. A path represents all hops between the source and the destination. Each link between two nodes is referred to as a hop. Figure 2.2 shows an example of a Multi-hop Wireless Network. Node S (sender) is too far away to communicate directly with node R (receiver) and uses the intermediate nodes IN_0 to IN_7. S chooses one or more nodes from its neighborhood to route the data to R. These intermediate nodes forward the data further over their neighbors until R is reached (in Figure 2.2: IN_1, IN_2, IN_3, and IN_6).

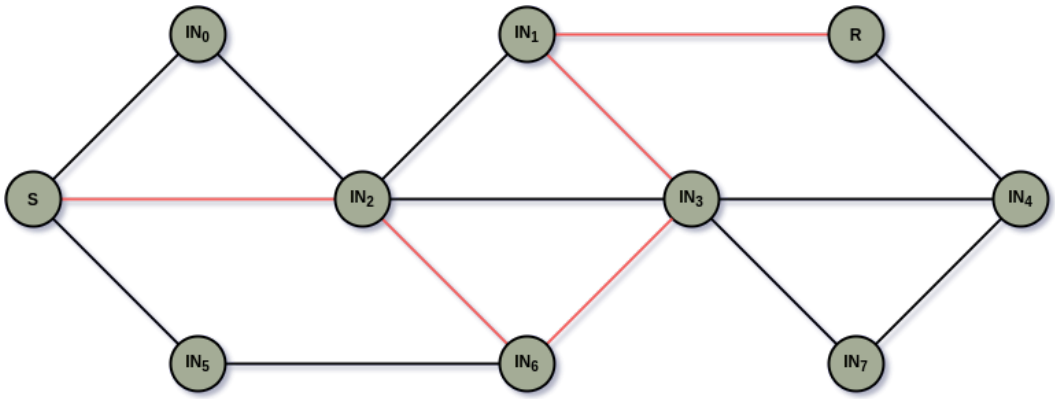


Figure 2.2: Example of a Multi-hop Wireless Network with communication between S, R, and intermediate nodes IN_1, IN_2, IN_3, and IN_6 (red lines).

2.4 Time-Slotted Channel Hopping (TSCH)

Time-Slotted Channel Hopping (TSCH) builds on two well-known technologies [7]: WirelessHART Networks and ISA 100.1a Wireless Networks, which build upon the ISA 100.1a standard introduced by the International Society of Automation (ISA)[1].

[1] visit: http://isa.org

2.4.1 (Wireless-)HART Protocol

WirelessHART builds on the HART Field Communication Protocol by the FieldComm Group™. It is an acronym for Highway Addressable Remote Transducer (HART), which uses Frequency Shift Keying (FSK), enabling two-way communication with smart field instruments on top of a 4–20 mA analog instrumentation current loops or a 2.4 GHz wireless instrumentation without compromising the integrity of the measured data [8], [9]. Thereby a current loop is a communication interface that uses current instead of voltage for signaling, allowing communication over long distances (tens of kilometers). Communicating by using the HART protocol needs at least two HART-enabled devices where up to two masters (primary and secondary) can be involved. The master/slave protocol allows various modes, e.g., multidrop communication or point-to-point, transmitting information between central control/monitoring systems and field instruments. In general, the HART protocol communicates at 1200 bps without interrupting the 4-20 mA signal and provides two simultaneous communication channels: the 4-20 mA analog signal and a digital signal. The first channel transmits the primary measured value, while the second channel contains information from the device, e.g., device status, diagnostics, and more. WirelessHART changed from the underlying protocol discussed above to the current wired and wireless technology known today and used in the IEEE 802.15.4 standard. It builds upon the HART protocol and complements it with features, e.g., advanced diagnostics, security, unsolicited data transfers, etc [8]. Moreover, it is an Frequency Division Multiple Access (FDMA) and Time Division Multiple Access (TDMA) based network, which minimizes collisions and reduces device power consumption [10]. To avoid collisions or loss of packets caused by inferences, e.g., co-existing wireless systems, WirelessHART networks use Frequency Hopping Spread Spectrum (FHSS) to hop across the 16 channels defined in the IEEE 802.15.4 standard (see next Section 2.4.2). Each network contains one schedule that is created and managed by the network manager. This schedule includes superframes with a fixed length and forms a network cycle with a fixed repetition rate. A superframe is a series of timeslots defining the communication schedule of a set of devices. The mesh topology is a typical network topology for a WirelessHART network. Not recommended but also possible is the star topology [10]. Routing avoids interference and broken links using redundant paths that allow packets to reach the destination with higher reliability. There are two different approaches for packet routing: Graph routing and Source routing. Graph routing uses predetermined paths to route a packet from a source to a destination device. A route through the graph consists of several paths between the source and destination devices and is the preferred way of routing packets both up- and down-stream in a WirelessHART network. Source routing uses ad-hoc created routes for the packet without providing any path diversity. The primary purpose of ad-hoc routes is for network diagnostics and not for real communication [10].

2.4.2 TSCH's specifications

TSCH, a MAC layer specified in the IEEE 802.15.4-2015 standard, combines Frequency Division Multiplexing (FDM) and Time Division Multiplexing (TDM) to Frequency-Time Division Multiplexing (FTDM) (see Figure 2.3). TDM allows splitting the time into timeslots, which TSCH uses as so-called slotframes. Each slotframe represents a (continuous) recurring of timeslots with a fixed size, a so-called slotframe cycle. Every slotframe has the same number of timeslots. FDM assigns all timeslots carrier frequencies, allowing to transmit several packets to non-conflicting nodes. These frequencies can change during a slotframe by using channel hopping. Channel hopping uses an Absolute Slot Number (ASN) and an Hopping Sequence List (HSL) to determine and assign the channels using channel offsets to the timeslots. TSCH determines for each new slotframe cycle, the ASN using the current cycle (represented by Sfc_i), the total number of timeslots (represented by M) in the slotted frame, and the current number of the timeslot (represented by $\mathrm{Ts}_{\mathsf{id}}$):

$$\mathrm{ASN}_{\mathsf{j}} = \mathrm{Sfc}_{\mathsf{i}} \cdot \mathrm{M} + \mathrm{Ts}_{\mathsf{id_j}} \quad \mathrm{i},\, \mathrm{j} \quad \mathsf{N} \tag{2.1}$$

The IEEE 802.15.4e standard does not specify a timeslot's duration, but typically 10 ms is used [7]. The specification is even more imprecise concerning the slotframe size, leading to advantages and disadvantages. I.e., short slotframes and often recurring timeslots result in a lesser latency and higher throughput by increasing the power consumption [7]. Contrary longer slotframes with less frequently recurring timeslots reduce power consumption and allow more data links to increase the potential network size. But increase the latency by a simultaneous decrease in the throughput in the network [7].

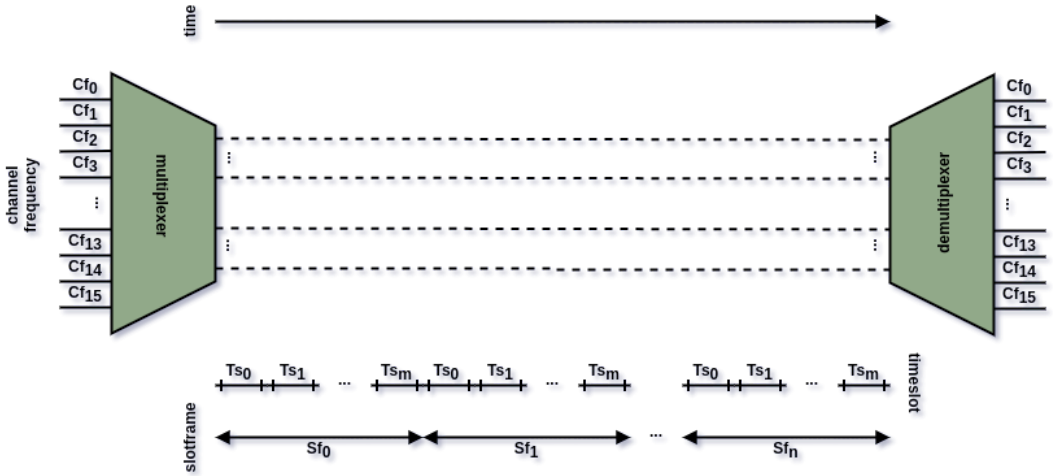


Figure 2.3: Example of a two-dimensional (de-)multiplexing scheme using Frequency-Time Division Multiplexing.

2.4.3 TSCH Schedule Matrix (TSM)

Using FTDM, a two-dimensional schedule in the form of a matrix called TSCH Schedule Matrix (TSM) allows representing the communication between nodes (see Figure 2.4). During the initialization, TSCH needs to build the TSM for each node and rebuilds it every time the network topology changes. TSM contains dedicated, empty, or shared cells, in which nodes either transmit or receive packets (see Figure 2.5). If a cell is empty, the radio remains off to avoid unnecessary energy consumption. A cell is empty if it is neither dedicated nor shared. Shared slots allow sending control packets, which contain essential control information about the network, e.g., the current system time or length of a timeslot, which newcomer nodes use to the association and synchronization. In this thesis, Contiki-NG's implementation of TSCH is used. It utilizes a dedicated beacon cell to transmit Enhanced Beacon (EB)s.

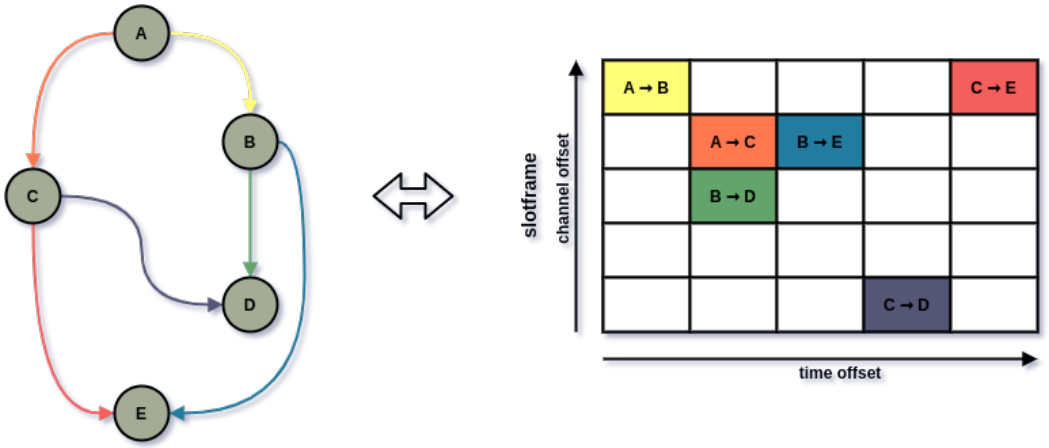


Figure 2.4: Relationship of a network transmission graph (left) and a corresponding TSCH Schedule Matrix (right).

2.4.4 Time synchronization in TSCH

A TSCH network needs synchronized clocks to perform its task. EBs or Acknowledgment- and Frame-based synchronization is used to synchronize the time between all nodes. EBs broadcast the current time to all nodes in the TSCH network. This time can either be adapted or used to determine the drift between the correct and own clock. Acknowledgment- and Frame-based synchronization allows in comparison to EBs an asynchronous synchronization by determining $\Delta\tau$ the time between the expected and the arrival time of a frame that the receiver calculates. Frame-based synchronization uses $\Delta\tau$ to adjust the receiver time. The Acknowledgment-based synchronization synchronizes the sender's time by sending $\Delta\tau$ to the sender in the acknowledgment frame.

2.4.5 TSCH's unicast communication

Each communication consists of two phases: sending/receiving and acknowledge of receiving. The response indicates either the communication was successful (ACK) or something went wrong (Negative-Acknowledgement (NACK)). Figure 2.5 shows a communication according to the IEEE 802.15.4e standard between a sender (S) and a receiver (R). In the first phase, S can send not until macTsTxOffset expires. Meanwhile, S checks if the channel to R is collision-free using the CSMA/CA back-off algorithm if the current timeslot is a shared slot. After macTsTxOffset expires S sends for macTsMaxTx and waits until macTsRxAckDelay expires to changes into the second phase and awaits R's acknowledgment. At the same time, R has to wait at least until macTsRx-Offset expires, where macTsRxOffset ≤ macTsTxOffset and the frame from S arrives. After macTsTxAckDelay expired, R responses with the (N)ACK within macTxMaxACK.

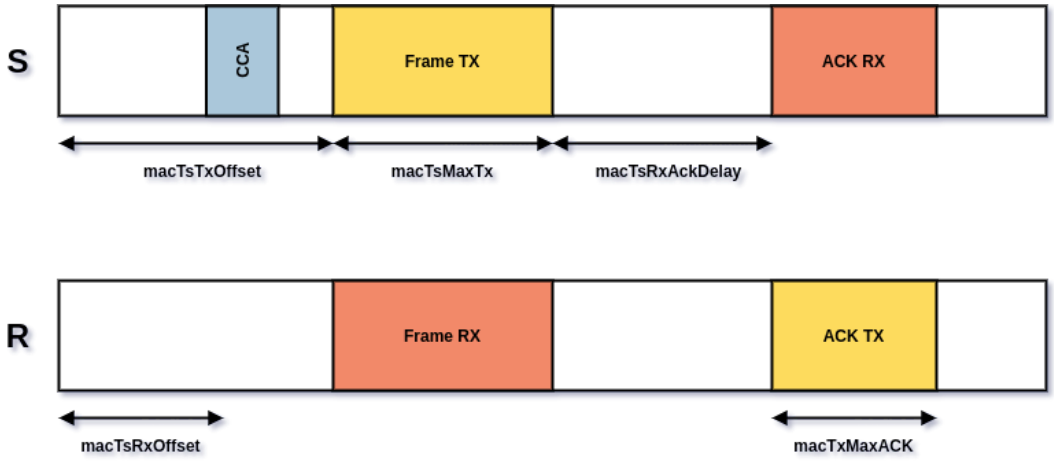


Figure 2.5: Example of a unicast TSCH communication, according to the IEEE 802.15.4e standard.

2.4.6 Summary

TSCH networks offer a globally-synchronized network by being more resilient against collisions and interferences. TSCH assigns different frequencies to timeslots while using channel hopping to increase the reliability of the communication [1], [7]. Further, by using channel offsets, the parallelization of the communication results in higher communication efficiency [7] because more channels are simultaneously available to transmit packets.

2.5 MASTER - A centralized scheduler

As described in Section 2.4, TSCH does not specify how to schedule the communication between nodes. Therefore many approaches, e.g., Orchestra (see Section 3.1) and later on MASTER, were introduced. MASTER combines centralized scheduling with a novel transmission strategy of Sliding Windows, which manages drawbacks, e.g., increasing latency, frequent rescheduling, and re-transmissions [1]. The system architecture works above the MAC layer and consists of an (external) scheduler and a routing layer.

MASTER consists of three building blocks, centralized routing, transmission strategy application, and centralized scheduling. These blocks are essential for MASTER's centralized scheduler and will be explained in the following sections.

2.5.1 MASTER's initial phase

MASTER needs some information about the network topology (i.e., the ETX-value) (see Section 2.5.3) of each network link. MASTER uses bootstrapping with neighbor discovery to collect all these required information. Further, MASTER builds a neighbor discovery scheduler similar to the sender-based operation mode of Orchestra [1]. Thereby each node in the network does two alternating things: broadcasting in its active slot and listening in all other timeslots to find other nodes in its surroundings during their broadcasts. After deployment, TSCH's beaconing process replaced this scheduler, and the normal process starts as mentioned above.

2.5.2 MASTER's centralized routing phase

MASTER builds upon TSCH and implements a high-level scheduler whose input is represented by flows. These flows are specified by a source and destination, deadline and the periodicity. The routing process takes the long-term link reliability statistics provided by each flow to determine a directed weighted graph of the network topology. Paths with minimal ETX-value have the highest throughput. Thereby a higher ETX-power implies usually the selection of a highly reliable link over fewer links with lower reliability (see Section 2.5.3) [1]. MASTER uses this graph to perform the routing while using Dijkstra's Shortest Path First algorithm [11] for determining the shortest end-to-end routes from each node to all other nodes. All paths are optimal in terms of concerning the flow latency for each flow.

2.5.3 MASTER's strategy application phase

Wireless communication is in general unreliable because of interferences or network changes, and transmissions cannot be guaranteed to be received. Therefore, transmission strategies add re-transmissions to retry a failed transmission. So different approaches lead to a different number of re-transmissions, which determines the size of the schedule. A common way of adding re-transmissions is to duplicate single timeslots in which transmissions can fail. This slot-based approach increases the reliability by including multiple tries per-hop [1]. A different approach is the flow-based transmission strategy. It assigns a specific number of re-transmissions to flows instead of a per-hop basis as done in MASTER. The number of re-transmission can either be determined by ETX-based metrics or a fixed amount of timeslots. Expected transmission count (ETX) represents a wireless link's quality and finds high-throughput paths in Multi-hop Wireless Networks (see Section 2.3) [6]. The ETX-value of a route determines the required number of transmissions expected to transmit a packet over a wireless link successfully. It is the inverse of the Packet Reception Rate (PRR) of a link:

$$\mathrm{ETX} = \frac{1}{\mathrm{PRR}} \tag{2.2}$$

The ETX metric incorporates the effects of an asymmetry in the loss ratios between each link's two directions, interference among the successive links of routes, and link loss ratios [6]. MASTER's flow-based transmission strategy, Sliding Windows (see Section 2.5.5), chooses the number of re-transmissions for each flow based on the ETX metric by allowing the nodes of a flow to share these timeslots and use them as needed along the path. It uses link qualities, which result in better adaptability to a network's link characteristics during scheduling [1].

2.5.4 MASTER's scheduling phase

The standard algorithm used in MASTER to determine the schedule is a non-deadline-based R-LPF scheduling algorithm, which is a variation of the Shortest Path First (SPF) scheduling algorithm. It is based on the process scheduling algorithm Shortest Job First (SJF) and is especially suitable for best-effort, deadline-free and periodic systems [1]. The scheduling algorithm performs backward scheduling by starting with the end of the most extended flow instead of the shortest flow, resulting in a lower number of unused timeslots and therefore have lower latency [1]. In MASTER, it is possible to implement any scheduling algorithm, including deadline-based ones [1].

2.5.5 MASTER's Sliding Windows strategy

As mentioned in Section 2.5.3, MASTER's flow-based transmission strategy Sliding Windows uses for each flow the ETX metrics to determine the number of re-transmissions, which results in being (more) adaptable to network changes [1]. Such an advantage is typical for distributed and not centralized schedulers. MASTER determines the window size by considering the maximal number of transmission slots and each flow's hop-count. The window size limits the maximum number of active nodes in a flow. Additionally, MASTER provides two flow-based transmission policies to specify the size of the window. The window size can either be ETX-based or fixed for all flows. The first policy dismisses the length or link quality and determines one window size for all flows. On the contrary, the second policy determines the window size and the number of re-transmissions depending on the flow's or link's ETX-values, which allows accounting the number of hops and the individual's reliability links [1]. To avoid inefficiencies caused by the fact that large flows would lead to bigger window sizes with too many nodes being active simultaneously, MASTER splits a flow into so-called sub-flows once it exceeds a threshold, which is standardized by ten. Moreover, it uses a scaling factor to determine the conservativeness of the schedule. This concept allows shared transmission slots concerning all links of the flow (see Figure 2.6) where interferences impact communication.

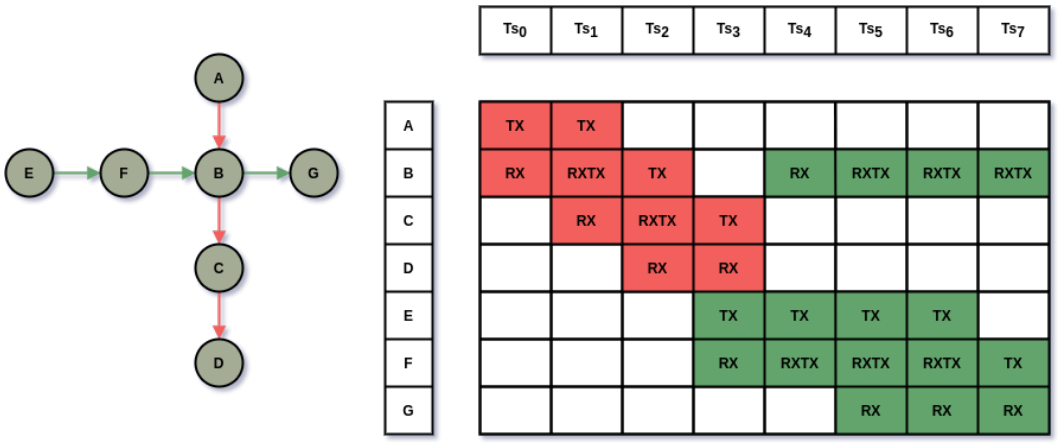


Figure 2.6: Example of a network transmission graph (left) and MASTER's Sliding Windows schedule with one transmission slot per-hop, one (red flow) and three (green flow) re-transmission slots to be shared among the nodes of a flow (right).

2.5.6 Time synchronization in MASTER

A TSCH network needs a stable time synchronization to avoid drift of clocks. As shown in Section 2.4.4. MASTER ensures this by building a clock synchronization tree while using the ETX metrics and starting with the TSCH network coordinator as the root as part of the scheduling process. This tree, built with Dijkstra's algorithm, assigns each node a parent node for clock synchronization.

2.5.7 Summary

Independent of knowing the optimal amount of needed re-transmissions per link, MASTER, and its flow-based transmission strategy, Sliding Windows is highly reliable by scheduling several re-transmissions for each flow concerning all flow links [1]. It reduces the overhead significantly compared to today's traditional scheduler concepts by being more adaptable to interferences while keeping the lowest possible latency without frequently rescheduling all flows. Both latency and reliability outperform in comparison to the state-of-the-art strategies: baseline and slot-based strategy [1]. In contrast to Orchestra, MASTER achieves similar reliability. It shortens the latency by not consuming significantly more energy. Moreover, the Sliding Windows algorithm accomplishes communication reliability, long-term schedule stability, and makes MASTER suitable for low-power systems while meeting the latency requirements of IIoT applications. Finally, MASTER's open and easily extendable modular design allows extending it with different metrics, routing-based, and deadline-based scheduling algorithms or even scheduling policies.

2.6 Principle of scheduling

Today, where more and more real-time computing systems are part of aircraft and airports, industry, modern telecommunication systems, and even space stations, schedulers are inevitable and ubiquitous [12]. The correctness of such systems depends on calculation and timing requirements that must be guaranteed [13]. Therefore scheduling is part of numerous studies aiming to develop more efficient scheduling strategies for the industry [14]. Scheduling describes the process by which distribute valuable computing resources are allocated to transmissions by respecting given constraints, e.g., deadlines. This process uses sets of transmissions with a finite or infinite stream of jobs and some policies, including dynamic or fixed priority algorithms, to generate a schedule. A flow set is schedulable if and only if a scheduler has at least one solution. Thereby all transmissions complete their executions before their deadlines expire. There are three major approaches to build a schedule either in an autonomous, centralized, or distributed manner.

2.6.1 Autonomous scheduling

An autonomous scheduling process provides autonomous agents with a set of minimal constraints on tasks. Based on this set of minimal constraints, each agent makes a schedule for its tasks independently from the others. This independent scheduling of tasks should be capable even if the agent's tasks are dependent upon the completion of tasks given to other agents. By constructing constraints, this coordination mechanism ensures a feasible joint schedule whenever the individual agent's schedules meet its conditions [15]. This approach is useful whenever a set of tasks has to be completed by several autonomous agents. Flexibility criteria that aim to maximize the range of scheduling choice of the participating agents have become highly critical besides efficiency criteria [15]. Thereby, autonomy is achieved by not forcing each agent to comply within a given schedule instead of letting autonomous agents choose from a set of schedules for the proposed tasks.

2.6.2 Centralized scheduling

The centralized scheduling approach uses global topology knowledge. This global knowledge with a more consistent interpretation of scheduling rules and collective agreements result in a significant advantage over distributed solutions [1], [12]. Moreover, centralized algorithms compute an optimal solution if it exists and allows using well-known and established algorithms, such as Dijkstra's Shortest Path First algorithm [1], [16]. Centralized scheduling needs to continuously adapt the received sensor measurements and rebuild parts of the scheduler, which may result in much communication overhead. This potential problem is for many practical applications challenging to solve [16], [17].

2.6.3 Distributed scheduling

In the context of distributed schedulers, smaller parts of a scheduling problem are solved independently and locally by different machines of the same system by performing cooperative scheduling (i.e., 6TiSCH MSF). Each machine locally uses the most recent system information by locally adapting to link changes. The solutions are then coordinated under a global objective. Specific communication mechanisms, e.g., sender-, receiver-, and symmetrically initiated, are adapted to achieve a comprehensive system agreement [18]. Because of the non-availability of shared states amongst nodes, high communication costs, heterogeneous hosts and networks, and low node resilience scheduling are more complicated in distributed systems [18]. Scheduling in a distributed manner increases system responsiveness to dynamic and unpredictable events, which may be especially crucial for scheduling in IIoT [14]. This approach, in combination with load distributing algorithms, allows handling machine breakdowns, load unbalancing, or order cancellations [14], [18].

3
Related Work

This chapter summarizes current work concerning autonomous, centralized, and distributed (non-)deadline-based scheduling approaches. A discussion about Orchestra's autonomous and MASTER's non-deadline based centralized scheduler solution, followed by a different approach for laxity-based algorithms, will give a broader classification of this thesis. Following this, general strategies made in the context of schedulability will close this chapter and lead to the design and implementation chapters.

3.1 Autonomous/Distributed scheduling

Autonomous scheduling earns a significant focus in recent work in the past [1]. This concept introduced by Orchestra, an autonomous scheduling solution [19], requires neither central infrastructure nor multi-hop path reservation, negotiation, or signaling among nodes to build a schedule. It achieves significant improvement in energy efficiency, failure tolerance, latency, and network reliability [20]. Duquennoy et al. [19] introduce with Orchestra an approach that performs an autonomous mapping of links to resources. Nodes adapt and maintain their schedule autonomously and locally by exploiting information based on their RPL neighbors and parents and a set of scheduling rules, which result in not predefined activity patterns, e.g., periodic. Using there deployment-specific scheduling rules that describe how to maintain TSCH slotframes and timeslots as a function of the routing topology allows nodes to select timeslots. Kim et al. [21] introduce Autonomous Link-based Cell Scheduling (ALICE), which is an autonomous link-based TSCH cell scheduler. ALICE allocates a TSCH cell, not for each node but each directional link (i.e., a pair of nodes and traffic direction). It interacts with the RPL routing by allowing a node with more RPL neighbors (parent and children) to have more unique TSCH cells. Further, it prevents up- and downstream traffic from interrupting each other and simultaneously utilizes multiple channels by not adding more overhead for cell scheduling. Shi, Sha, and Yang [20] introduce Distributed Graph routing and autonomous Scheduling (DiGS) solution that allows the field devices to compute their own graph routes and transmission schedules in an autonomous/distributed fashion by providing two scheduling approaches.

The distributed approach uses the ETX-values of all links and a rank system to determine the primary and secondary parent for routing. However, the autonomous scheduling approach first assigns timeslots in three slotframes (i.e., application traffic, routing traffic, and synchronization traffic) for transmissions. It combines them into a single one for run-time execution. DiGS assigns different priorities to different types of traffic to resolve slot assignment conflicts during the combination. The application traffic has the lowest priority, while the most critical synchronization traffic has the highest priority. The slotframe for traffic with lower priority gives up its transmission for traffic with higher priority in the slot.

3.2 Scheduling algorithms

Some papers assume that each node generates a single data packet at the beginning of each superframe for its centralized scheduling approach. The delivery of this packet has to be before the end of the superframe. Zhang, Soldati, and Johansson [22] propose an example of an approach that uses such an assumption for tree-graphed WirelessHART networks. MASTER uses R-LPF, a non-deadline-based centralized scheduling algorithm, which is a variation of the SPF algorithm but based on SJF. It is suitable for best-effort, deadline-free and periodic systems. As described in Section 2.5.4, the scheduler performs backward scheduling, starting with the end of the most extended transmission. The fact that deadline constraints are missing makes it unsuitable for industrial applications with deadline constraints or IIoT. Jin et al. [23] propose Adaptive Multi-hop Scheduling method (AMUS), a centralized scheduling solution based on TSCH. AMUS enables sequential multi-hop scheduling to provide a low latency guarantee for time-critical applications. Its experimental cell allocation method allocates backup slots in empty cells of the schedule and additional resources to vulnerable links. Possible MAC re-transmissions can be accommodated within the same slotframe by significantly reducing the delay caused by collisions or interference. The main drawback of simplifications is that they do not consider deadlines and periodic transmissions with generation rates, which are part of QoS constrained applications [24]. Path-Collision aware Least Laxity First (PC-LLF), proposed by Darbandi and Kim [24], is a heuristic algorithm for centralized link scheduling by considering periodic transmissions. It dynamically prioritizes the laxity time of the end-to-end deadlines and the number of collisions that might occur along their designated paths to the destination nodes [24]. It dynamically predicts path latency by returning a good feasible solution that minimizes packet lateness and showing a significant advantage in handling end-to-end latencies and dynamic changes in network workloads [24]. Instead, C-LLF considers the remaining time before the deadline expires and the number of collisions at the current local link. PC-LLF prioritization allows handling delay issues more efficiently and

avoids a possible drawback of C-LLF being less successful when handling delay guarantees in highly congested routes [24]. Moreover, by increasing the difficulty of holding the deadline and the network scale, it outperforms C-LLF in performance [24]. It makes it more suitable for tight deadline constrained applications. RMS prioritizes each packet, based on a monotonic rate strategy. It assigns priorities according to the period. However, RMS is a static prioritization approach that does not dynamically prioritize packets as they deal with high collisions in their designated paths. PC-LLF would also outperform these approaches by boosting transmissions priority with a shorter remaining time to their deadline.

Besides autonomous and centralized approaches, Tinka, Watteyne, and Pister [25] proposed a distributed scheduling solution that uses an advertisement and rendezvous scheme. They establish two-way connections between neighbor nodes, subject to the superframe structure's constraints, and the physical connectivity by using rendezvous slots for discovery and slot installation. Nodes continuously advertise their presence to allow neighbor nodes to discover and contact each another. The Aloha-based scheduling allocates one channel for broadcasting advertisements for new neighbors. Additionally, the reservation-based scheduling extends Aloha-based scheduling with a dedicated timeslot for targeted advertisements based on network information.

3.3 Schedulability analysis

Schedulability analysis runs in pseudo-polynomial time (i.e., exponential in the length of the input) [26]. While it is well-explored in the CPU scheduling domain, schedulability analysis for multi-hop wireless networks has seen little progress [26]. A set of (periodic) transmission is schedulable if and only if all transmissions meet their deadlines. A schedulability test is sufficient if any set of transmissions is schedulable. Existing work mostly focuses on worst-case delay analysis that runs in pseudo-polynomial time, making it less suitable under many network dynamics, typical in IIoT. Schedulability analysis for wireless sensor networks, which use end-to-end delay bounds, is already discussed in several works [27]–[29]. Their key focus relies on data collection through a routing tree but does not consider multiple channels. Therefore, Modekurthy et al. [26] propose a utilization-based approach for schedulability analysis, which remarkably has a low run-time overhead. A utilization bound analysis specifies the maximum possible utilization of all transmissions in the network. It determines the transmissions as schedulable if the total utilization does not exceed the maximum possible utilization. Its extremely low run-time overhead is considered to be one of the most effective and efficient schedulability tests [26]. It is incredibly valuable for IIoT [26].

4

Design

In this chapter, we present the design goals of our extension for MASTER. We overview the system and workflow (see Figure 4.1) and explain our assumptions during the design, including our proposed algorithms and schedulability test. Additionally, we justify our design decisions. We start with the problem formulation and proof of NP-completeness.

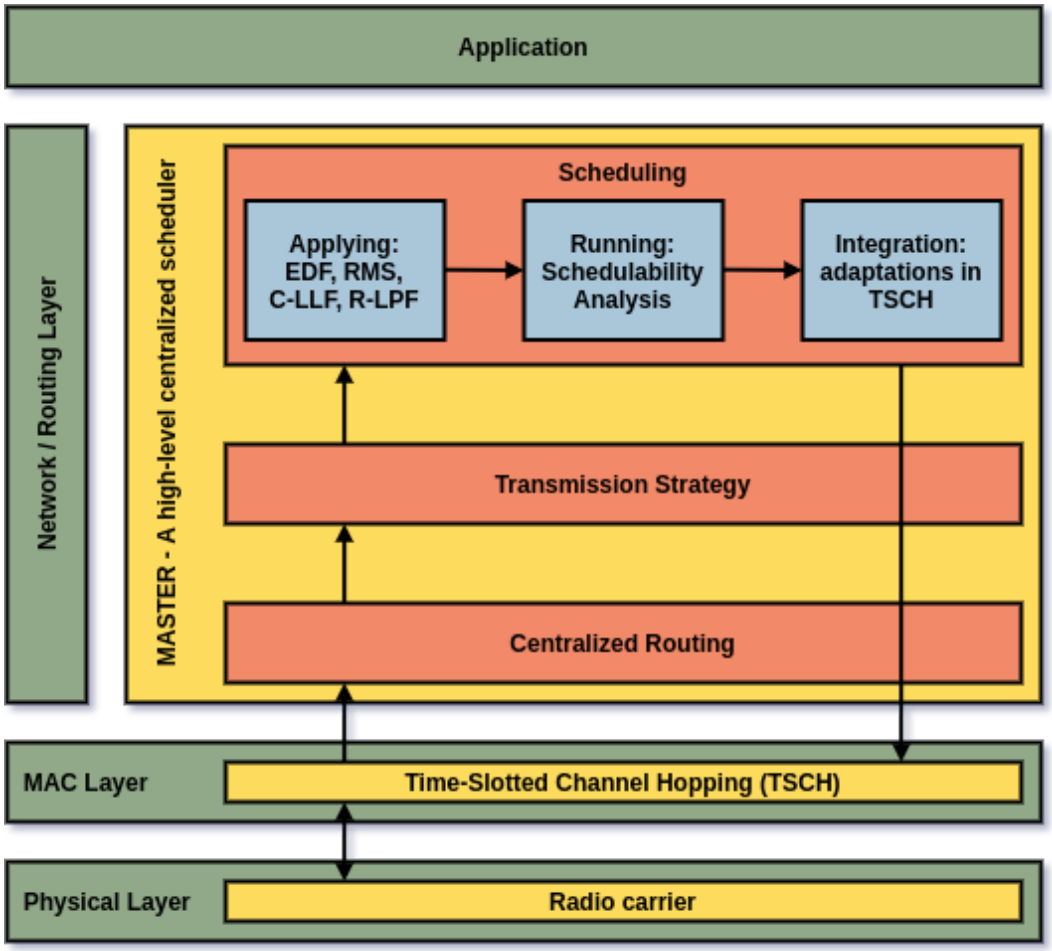


Figure 4.1: MASTER's system architecture with workflow overview.

4.1 Problem formulation

The main aim is to compute an optimal schedule for a given set of transmissions such that each transmission meets its deadline within one hyper-period. Each transmission T_i has a deadline D_i, a period P_i where $D_i \leq P_i$ and a set of routes Φ_i, connects the source node to its destination node. A packet can take many different paths to its destination via the routing graph. Each source node forwards each packet through each possible route ϕ with $\phi \quad \Phi_i$. All periods together determine the hyper-period H. The release time R_i of a packet is the earliest timeslot when it is ready to be scheduled. The number of R_i depends on the period of T_i. Formally, a set of transmissions T is schedulable if and only if, $T_i \quad \mathsf{T}$: $T_{R_i} \leq T_{D_i} \qquad T_{P_i} \quad T_i$: $T_{P_i} \leq \mathsf{H}$, holds. More constraints are possible but will neglect the rest of the thesis. In the next section, we prove that the problem mentioned above is NP-hard by proving that its decision version is NP-complete.

4.2 Scheduling is NP-complete

The complexity theory, a part of theoretical computer science, contains many different complexity classes. A complexity class is a set of computational problems of related resource-based complexity. Two of them are very important for implementing algorithms for real-life scenarios, P and NP. P, also known as DTIME or PTIME, is a fundamental complexity class. It contains all decision problems that can be solved by a deterministic Turing machine using a polynomial amount of computation-time or polynomial time. In comparison, NP is a complexity class used to classify decision problems. It is the set of decision problems for which the problem instances, where the answer is "yes", have proofs verifiable in polynomial time by a deterministic Turing machine. An NP's equivalent definition is the set of decision problems solvable in polynomial time by a non-deterministic Turing machine. Generally, all scheduling problems cannot be solved optimally in polynomial run-time except, P = NP. The general scheduling problem is NP-complete [30], which means that if an NPC problem has a deterministic polynomial-time algorithm, every NP problem will have a polynomial-time solution. Hence most computer scientists are convinced that P = NP is probable. That means that all these problems have, in the best case, at least an exponential-time algorithm.

The following proof shows that the real-time scheduling problem is NP-complete by reducing it to k-edge-coloring. This proof is orientated on the proof shown in the paper "Real-Time Scheduling for WirelessHART Networks" by A. Saifullah et al. [4].

To prove that the scheduling problem (represented by RT-SP) is NP-complete, we show (1) RT-SP NP and (2) the NP-hardness of RT-SP.

Proof. (1) We prove RT-SP NP by giving a verifier (represented by V), which verifies a given instance in a deterministic way in polynomial run-time. For every given instance of RT-SP with n transmission, V works as follows:

1. Verify if the given input (represented by S) is a valid instance of RT-SP. If not, discard the input.
2. For every *trm* S:
   2.1 Verify if *trm* meets its deadline. If not, discard the input.
3. Accept the given input as valid.

V works in a deterministic way by verifying each transmission of the schedule, starting with the first transmission, and stopping until the last one is verified. V's computation is polynomial bounded by the schedule's size with a worst-case run-time of $O(|S|)$.

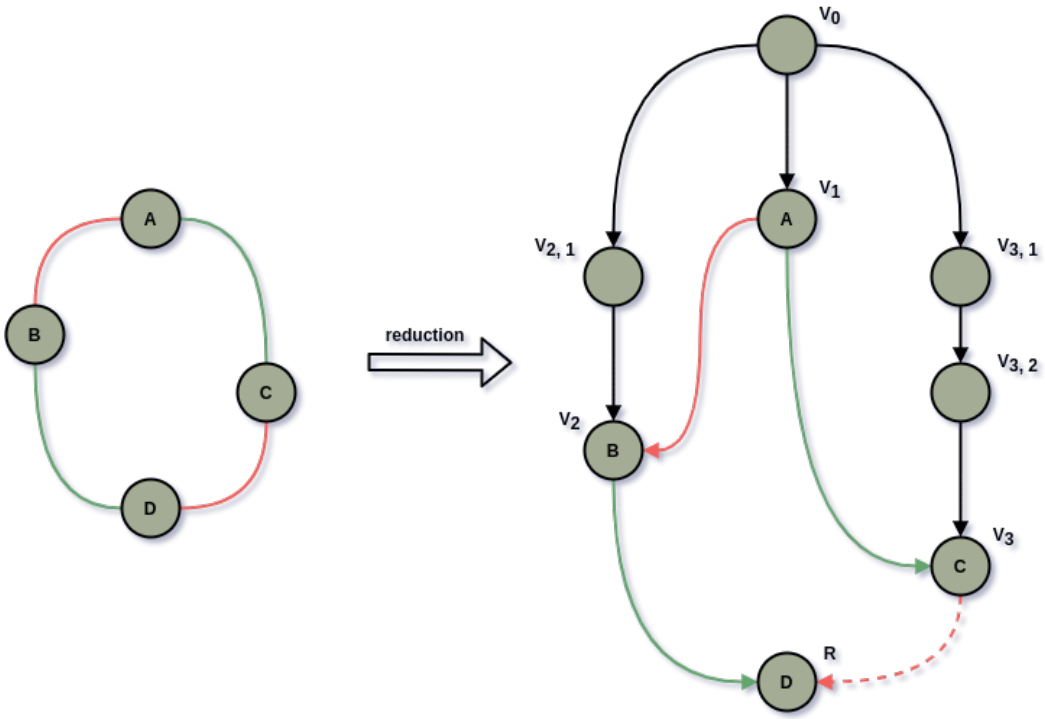


Figure 4.2: Example of a *2*-edge-coloring graph G (left) and the reduced network graph G' (right) after applying the reduction function.

(2) Now, we will show by reducing an arbitrary instance S of RT-SP to an instance G, k of the k-edge-coloring problem that graph G is k-edge-colorable if and only if S is schedulable. Further, an invalid instance of the RT-SP will result in an invalid instance of k-edge-coloring. The following procedure will transform each valid instance of the k-edge-coloring problem to an instance of RT-SP in $O(n^2)$ with n N and n as the number of nodes in G.

Let $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ has n nodes with $n = |\mathsf{V}|$.
The following procedure does the transformation:

1. Create a depth-first search tree of G, rooted at an arbitrary node r V.
2. Create a node v_0.
3. For every node v $\mathsf{V} \setminus \{r\}$:
   - 3.1 A tree edge is directed from v to its parent.
   - 3.2 A non-tree edge is directed with zero or more ancestors from v to its virtual parents.
   - 3.3 Label v with a unique identifier (represented by v_i), with i N and $1 \leq i \leq n - 1$.
   - 3.4 Create a path with $i - 1$ additional nodes $v_{i,1}, v_{i,2}, \ldots, v_{i,i-1}$ and connect node v_0 with this path (i.e., $v_0, v_{i,1}, v_{i,2}, \ldots, v_{i,i-1}, v_i$).

This procedure creates a network with $\frac{n^2}{2}$ nodes (see Figure 4.2). Each packet trm_i is routed through the gateway v_0 and $v_{i,1}, v_{i,2}, \ldots, v_{i,i-1}, v_i$ and forwarded by v_i to its (virtual) parents. These parents within $1 \leq i \leq n - 1$, are the destination nodes. Thereby, the path represents the $(n - i) \cdot j$-th timeslot in which trm_i should be released. Further, the deadline and release time of each transmission can be determined with $n - 1 + k$ and $n - i$, respectively.
Let graph G be k-edge-colorable and T be the set of all last one-hop transmissions that involve E, where E E' from G', generated by above-mentioned procedure. All transmissions in G' are schedulable within the first $n - 1$ slots, except those in T. Since these last transmissions have the same color, they are schedulable in the next k slots. All packets meet their deadline, S is feasible.
Let schedule S be schedulable by some algorithm. Each transmission starts from the earliest timeslot in T, where it is schedulable. Since all packets meet their deadlines and the release times prevent all transmissions in T in the first $n - 1$ slots, it follows that since the schedule is feasible, all transmissions in T happen in $(n - 1 + k) - (n - 1) = k$ slots. Thereby, G is k-edge-colorable. It follows that RT-SP is NP-hard and with (1), NP-complete. □

The scheduling's dynamic and stochastic nature makes it very hard to obtain optimal scheduling solutions [30]. To still handle scheduling problems, various approximate methods (i.e., both constructive and iterative heuristics) are developed to cope sufficiently with real-life scheduling problems.

4.3 Design goals

In the following sections, we will justify our design decisions of EDF, RMS, and C-LLF. We start with a short introduction, followed by our design goals.

4.3.1 Earliest-Deadline First (EDF)

Earliest Deadline First (EDF) is a dynamic priority scheduling algorithm used in Real-Time Operating Systems (RTOS). Typically it runs on preemptive uniprocessors. It uses a so-called ready queue to determine the next task to schedule. The ready queue is sorted according to decreasing priorities whenever there is a priority change. The task closest to the end of its period is assigned the highest priority. EDF is an optimal scheduling algorithm. It means that if a set of independent tasks with deadline and release time can be scheduled by any algorithm to ensure all the tasks do not miss their deadline, then EDF will schedule this set of tasks to complete before their deadline. Even though EDF is an optimal scheduling solution with a run-time of $O(n \cdot \log(n))$, it is not commonly in industrial real-time computer systems due to the dynamic priority assignment. Moreover, it takes longer to sort the ready queue on-line, and it is not stable. If any task instance fails to meet its deadline, any instance of any task may fail.

4.3.2 Design goals: EDF

Our approach of EDF introduces the first deadline-based scheduling solution for MASTER, which allows considering deadlines during the building process. EDF uses a priority queue to determine the next task. MASTER's scheduling process only supports schedule built during compile time. Thus our approach does not require any updated queue at run-time and, finally, no ready queue. We determine for all transmissions their corresponding pair of channels and timeslots during the compile time. All pairs together represent the final schedule. Before our approach assigns free timeslots to specific transmissions, all transmissions are ascending sorted accordingly to their deadline. Following, each pair is determined trough the first available slot in the schedule. During the building process, our schedulability test analyzes each transmission's schedulability and decides if the current state leads to a feasible or infeasible schedule (see Section 4.4). The following pseudocode shows our above-described approach at a high-level form.

Algorithm 4.1: Pseudocode of our EDF approach for MASTER.

Data: Γ, a set of transmissions.
M, a flag that determines if a transmission is allowed to fail and how a failed transmission is to handle. It determines the level of strictness of our schedulability test (see Section 4.4).
Result: S, schedule according to EDF.

```
// sorts the set of transmissions such that the
   transmission with the shortest deadline is assigned
   with the highest priority
1  sort Γ ascending to the deadline;
   // iterates over all transmissions of Γ
2  foreach γ  Γ do
       // iterates over all hops of γ
3      foreach γ'  γ do
           // determines the next channel and timeslot for γ'
4          (channel_idx, timeslot_idx) ← next free schedulable channel
            and timeslot;
           // checks if γ' meets its deadline while depending
              on the period and release time of γ
5          if timeslot_idx ≥ γ'.release_time + γ'.deadline then
               // no transmissions are allowed to fail
6              if M == INFEASIBLE_MODE then
7                  return unschedulable schedule;
8              end
               // transmissions are allowed to fail
9              if M == ADVISE_MODE then
10                 inform system about missed deadline;
11             end
12         end
           // saves γ' at the specific channel and timeslot
13         S ← (channel_idx, timeslot_idx, γ');
14     end
15 end
16 return S;
```

4.3.3 Rate-monotonic Scheduling (RMS)

The kind of its priority assignment characterizes a priority-driven algorithm. A priority-driven scheduler is an on-line scheduler that assigns priorities to tasks when released and places them on a run queue in priority order. At each time, the scheduler updates the ready queue and executes the tasks at the queue's head. A scheduling algorithm is static if a task's priority is fixed and cannot change in time. Rate-monotonic Scheduling (RMS) is a priority assignment algorithm used in RTOS with a static-priority scheduling class. These operating systems are generally preemptive and have deterministic guarantees concerning response times. Assignment of the static priorities is done accordingly to the task's cycle duration, so a shorter cycle duration results in a higher task priority. Tasks are independent of each other and are always released at the start of their periods. When applying the sorted, the tasks are ordered in a priority task stack by decreasing rates with some additional rules, e.g., deadlines to break ties. The position in the stack yields the priority of each task. Higher priorities are assigned to higher positions. Consequently, there are as many priority levels as there are tasks (i.e., the stack's height depends directly on the number of tasks). It has been formally proven that if any static-priority scheduling algorithm can meet all the deadlines, it is also Rate-monotonic (RM) schedulable [31].

4.3.4 Design goals: RMS

Our approach of RMS for MASTER entirely differentiates from the traditional design. The traditional design first introduced by Liu and Layland [32] in 1973, uses a stack to determine the priority of the next to be scheduled tasks. This structure is useful if the priority assignment of a task or, in our case, transmissions can be done dynamically. MASTER's scheduling process only supports a static schedule built during compile time. Therefore our approach does not require any updated structure at run-time. We determine the priority for each transmission through the given period and break ties by using the deadline. A transmission with a shorter period with more repetitions during the hyper-period gets a higher priority. A transmission with the same period but an earlier deadline will be preferred. Before our approach can assign free timeslot to a specific transmission, all transmissions are ascending sorted accordingly to their priority. Our approach finds the first available pair of channels and timeslots in the schedule in the same way as EDF does. The following pseudocode shows our above-described approach at a high-level form.

Algorithm 4.2: Pseudocode of our RMS approach for MASTER.

Data: Γ, a set of transmissions.
M, a flag that determines if a transmission is allowed to fail and how a failed transmission is to handle. It determines the level of strictness of our schedulability test (see Section 4.4).
Result: S, schedule accordingly to RMS.

```
// sorts the set of transmissions such that the
   transmission with the shortest period and deadline is
   assigned with the highest priority
1  sort Γ ascending to the period and deadline;
   // iterates over all transmissions of Γ
2  foreach γ  Γ do
       // iterates over all hops of γ
3      foreach γ'  γ do
           // determines the next channel and timeslot for γ'
4          (channel_idx, timeslot_idx) ← next free schedulable channel
            and timeslot;
           // checks if γ' meets its deadline while depending
              on the period and release time of γ
5          if timeslot_idx ≥ γ'.release_time + γ'.deadline then
               // no transmissions are allowed to fail
6              if M == INFEASIBLE_MODE then
7                  return unschedulable schedule;
8              end
               // transmissions are allowed to fail
9              if M == ADVISE_MODE then
10                 inform system about missed deadline;
11             end
12         end
           // saves γ' at the specific channel and timeslot
13         S ← (channel_idx, timeslot_idx, γ');
14     end
15 end
16 return S;
```

4.3.5 Conflict-aware Least Laxity First (C-LLF)

In TSCH networks, transmission conflicts can play a significant role in schedulability even for moderate workloads due to the high degree of conflicts [4]. Different nodes experience different degrees of conflicts on the way through the routing graph caused by interferences. To handle such conflicts, Saifullah et al. [4] introduced Conflict-aware Least Laxity First (C-LLF). It combines Least Laxity First (LLF) as the decision variable and the degree of conflicts associated with a transmission. The principle of LLF distinguishes between the remaining computation-time and the deadline by assigning the highest (dynamic) priority to the smallest laxity task. C-LLF identifies the critical time windows in which too many conflicting transmissions need to be scheduled by determining each released transmission's criticality. It considers the length of the time windows in which the transmission has to be released and the potential conflicts that the transmission may experience in these windows. C-LLF priorities each transmission by giving the highest priority to the transmissions exhibiting lower conflict-aware laxity. Analyses show that this approach significantly outperforms traditional real-time scheduling policies and is highly effective in meeting end-to-end communication deadlines [4] by including each transmission's requirements and incorporating conflicts in scheduling policies.

4.3.6 Design goals: C-LLF

Our design approach is similar to C-LLF proposed by Saifullah et al. because it is already a scheduler solution for Multi-hop Wireless Networks. It incorporates FTDM by distributing transmissions into channels and timeslots and taking into account possible incompatibilities. Therefore there is no need to search for the channel and timeslot pair as we did in the earlier-described approaches. However, we make several assumptions during the design phase, which possibly can lead to different behaviors. C-LLF works on two sets, the so-called unscheduled transmissions set and the set of released transmissions at timeslot *tms*, where *tms* determines the next release time within the schedule. Thus that C-LLF uses LLF to determine the next transmission. In contrast, all transmissions whose lifetimes intersect with the current transmission lifetime are potential sources of conflict while trying to schedule this transmission. The original paper does not describe which of both sets is necessary to determine this value. We choose the second set as mandatory because it makes much more sense to use all currently starting transmissions and not the whole set. Otherwise, it could be possible that transmission, which would be usually scheduled, cannot be taken because other transmissions are preferred due to their smaller laxity and consequently assigned with a higher priority. Moreover, we also assume that only the source node of every transmission is necessary to compute the laxity and not the destination node. The following pseudocode shows our above-described approach at a high-level form.

Algorithm 4.3: Pseudocode of our C-LLF approach for MASTER.

Data: Γ, a set of transmissions.
M, a flag that determines if a transmission is allowed to fail and how a failed transmission is to handle. It determines the level of strictness of our schedulability test (see Section 4.4).
C, a set of available channels that can be used.
Result: S, schedule accordingly to C-LLF.

```
1  initialization of timeslot tms with the first available timeslot;
   // iterates over all transmissions of Γ
2  while Γ =  do
3      release_set_tms ← set of released transmissions at release time tms;
4      initialization of channel chl with the first available channel;
5      foreach rtrm  release_set_tms do
6          compute the laxity for transmission rtrm;
7      end
       // iterates over all channels in C
8      while chl < C do
9          γ ← transmission with the smallest laxity and shortest
             deadline in release_set_tms;
           // checks if γ meets its deadline while depending on
              its computed laxity
10         if timeslot_idx ≥ γ.release_time + γ.deadline then
               // no transmissions are allowed to fail
11             if M == INFEASIBLE_MODE then
12                 return unschedulable schedule;
13             end
               // transmissions are allowed to fail
14             if M == ADVISE_MODE then
15                 inform system about missed deadline;
16             end
17         end
           // saves γ at the specific channel and timeslot in S
18         S ← (chl, tms, γ);
19         Γ ← Γ \ {γ};
20         release_set_tms ← remove all transmissions conflicting with γ;
21         chl ← chl + 1;
22     end
23     tms ← tms + 1;
24 end
25 return S;
```

4.4 Schedulability analysis

In a TSCH network, conflicting transmissions play a crucial role in creating communication delays and transmissions schedulability [4]. The delays caused by conflicting transmissions can lead to missed deadlines and make the whole schedule infeasible. Therefore schedulability analyses determine whether a set of real-time transmissions (i.e., end-to-end communication between two nodes) can meet their deadlines. However, existing work on schedulability analyses focuses on a worst-case delay analysis, which runs in exponential time [26]. These techniques are less suitable for frequent checking due to changes and failures in the network topology.

Our main design goal is to be flexible during the schedule building phase, which allows us to create a feasible schedule even if some transmissions are infeasible. Our schedulability test cannot alone determine if a set of transmissions is feasible or not. It needs a host algorithm that provides our algorithm with all the necessary information. Therefore our approach is only usable during the schedule build, where all information is available. This specific design allows either the application that builds upon our schedule or the user to leverage the scheduling process and therefore be more suitable for many applications. Our approach's run-time is better but more addicted to the host scheduling algorithm than traditional schedulability analyses. In the base-case, our run-time is linear in the size of transmissions. Our algorithm needs the deadline, release time, and the so-called decision mode to determine the schedulability for the ongoing transmission in the context to the rest of the schedule. The decision mode determines the level of strictness. In other words, it determines the threshold when a schedule is feasible or not. We provide three options: *INFEASIBLE*, *ADVISE*, and *ADJUST* mode. The *INFEASIBEL* mode accepts only feasible transmissions and terminates the whole scheduling process immediately if one transmission is infeasible. This mode is similar to traditional schedulability analyses, which only determine whenever a set of transmissions is feasible or not without any further specifications. Whereas the *ADVISE* mode checks if a transmission is feasible or not, and if it is infeasible, it just informs the system without changing anything on the schedule. Therefore, the resulting schedule should not be used in real industrial applications or verified with one of the other modes, e.g., *ADJUST* mode. This mode should be used more passively. Its primary purpose is to determine if all transmissions are feasible and for debugging. The *ADJUST* mode is a hybrid of the first two modes. It combines the strictness of valid and invalid transmission but without stopping the whole process. Instead, it filters all infeasible transmissions out by also informing the system. This combination of both approaches makes this mode very dynamic in terms of striking a balance. The *ADJUST* mode is useful in case of redundant transmissions or if not all transmissions are needed.

4.5 Hyper-periods in MASTER

MASTER's scheduling process does not consider any periods and, therefore, does not support periodic repetition of transmission within one slotframe cycle (see Section 2.4.2). Our approach changes this behavior by introducing a completely new scheduling process with hyper-periods. Without hyper-periods, transmissions can only be sent once at the beginning of every slotframe cycle, but with it in every free timeslot during the hyper-period. This feature allows being more suitable for IIoT because tasks are often repeated in a specific pattern (see introduction example in Chapter 1). We achieved to be more comparable and extendable with other periodic (non-)deadline scheduling solutions. Hence MASTER can be a platform for much more scheduling solutions, which also increases the suitability. The introduction of hyper-periods changes the design of MASTER in several ways. Periods are necessary to compute the schedule's hyper-period and use scheduling algorithms, e.g., C-LLF. Periods are now mandatory. MASTER only accepts non-relative prime periods to avoid large hyper-periods and thus higher energy consumption. Every end-to-end transmission has its repeating cycle, which is to consider by mapping each transmission to their specific timeslot in TSCH.

5

Implementation

In this chapter, we present the implementation of our design goals described in Chapter 4. We placed our implementation in the third layer (i.e., the OSI network/routing layer). To implemented our design, we had to achieve the following implementation goals:

- Extending MASTER to process and store periods.
- Enabling MASTER to handle hyper-periods.
- Adding sub-flows to each original parent flow, based on the hyper-period.
- Analyzing and deploying the schedule.
- Creating necessary schedule instructions for all participating nodes in the TSCH network.

Our implementation is part of the official repository of the working group *Distributed Systems Research Group*[1] at Kiel University.

5.1 Hyper-periods in MASTER

The basis of our implementation rests on periods. Using periods allows us to determine the hyper-period and defines the size of the schedule. Besides a source and destination, each flow needs a mandatory period that determines the commonness of occurrence within the hyper-period. MASTER needs several options, e.g., the neighbor discovery statistics (required), the scheduling approach, and flows. After parsing and error checking, we determine the hyper-period by calculating the Least Common Multiple (LCM) of all periods. The hyper-period is always a positive non-zero value and should be greater than 20 timeslots to ensure that two flows have enough time to send. Undershooting this threshold can lead to failing transmissions within the TSCH network. We base this value on experimental and heuristic knowledge. It should serve as a recommendation rather than a hard border. If all periods are equal to one, our implementation will use a pseudo-hyper-period and fall back into a version of MASTER, where hyper-periods are not part of the implementation to ensure backward compatibility. However, to conform with our threshold, we will extend the size of the generated schedule to at least 21 timeslots. To ensure

[1] visit: https://github.com/ds-kiel

that the whole schedule's size is not bigger than the hyper-period, we wrap-around each flow if the combination of its period and release time exceeds the schedule. Our wrap-around may lead to an infeasible schedule. To avoid this behavior, we allow setting the level of strictness for schedulability analysis, as described in Section 4.4. We extend the set of flows with several sub-flows. Each sub-flow is a copy of the original parent flow with the same flow number, deadline, and period. Only the specific release time is to adjust for each sub-flow individually. We do this to ensure that we add the correct ration of each parent flow to the schedule and avoid problems with the TSCH network due to the same release times for several flows and MASTER's specifications.

5.2 Deployment of the schedule

Our implementation of EDF, RMS, and C-LLF uses several techniques and data structures to improve the overall run-time, e.g., functional programming pattern. Moreover, our abstraction allows us to reuse our code for EDF almost entirely for RMS. We achieved this by identifying both algorithms similarities, e.g., the general structure going through each flow or determining the next free timeslot. We select the next free schedulable timeslot (pair of channel and timeslot), as mentioned several times in Chapter 4 and shown, e.g., in Algorithm 4.1, by going through each channel and timeslot. Thereby, we intersect the current to schedule flow participants with all already scheduled participants in the schedule. This strategy avoids collisions caused by conflicting nodes. We ensure that no flow has to send and receive at the same timeslot. We also consider the earliest timeslot in which a specific flow is scheduled and choose a particular point in time (timeslot) for each flow. We implemented the rest of each scheduling solution almost in the same way as we described it in each of the corresponding sections in Chapter 4.

5.3 Adaptations in TSCH

MASTER is a high-level centralized scheduler that operates on the routing layer (see Section 2.5). Its main components are written in Python. It is necessary to translate the generated schedule into the host language of TSCH, C. Each scheduled flow and its potential sub-flows have a specific period, which TSCH needs to adjust the sending frequency. We store these sending frequencies into a particular file. TSCH uses this file during run-time to determine the time in which a node has to send. We use each period to determine the sending cycle for the sender node. Thereby we distinguish between platforms on which the generated schedule will run. To be independent of the schedule size, we provide some additional configuration files for bookkeeping, which adapted specific characteristics of each schedule, e.g., the maximal number of links or

slotframes to be used. MASTER includes these files during run-time, which allows us to adjust the behavior of the TSCH network. Since each node has a period that determines the sending pattern, we write all necessary information, e.g., the individual schedule length, a sender, and the timeslots to send into a from the application- or user-defined file. This file contains several more information for MASTER and TSCH, which neglect the rest of this section because they are not part of our implementation. Extending MASTER with hyper-periods requires some changes. Contrary to the old implementation, we now store all timeslots a node has to send in two-dimensional lists. Each node in MASTER uses these lists to determine the next timeslot to send. Both lists are necessary for determining the re-transmissions window in case of failing transmissions. Strategies, e.g., MASTER's Sliding Windows, could not be supported without them.

6
Evaluation

In this chapter, we evaluate the performance of our deadline-based approaches. A comparison with MASTER's Reverse Longest Path First algorithm shows how a non-deadline based scheduler behaves in the same context. Our primary focus was to evaluate our schedulers regarding end-to-end latency, reliability, networking energy consumption, and schedulability. We see these constraints are especially crucial for IIoT networks. We evaluate these scheduling solutions in Cooja, a network simulator, and our 20 nodes testbed at Kiel University.

We start with our evaluation methods, followed by the setup and parameters we used for the evaluation, ending with our evaluation results.

6.1 Evaluation setup

In this section, we describe the setup we used to evaluate the criteria mentioned above. To eliminate any possibility of unwanted extraneous factors (i.e., interference during office time) and be reproducible, we set up a stable testing environment for all of our experiments.

6.1.1 Evaluation process

Our evaluation process consists of two stages: applying and evaluating. We run a neighbor discovery to find all TSCH network participants. We use the neighbor discovery data for each scheduling algorithm to build the schedule. We upload, install, and execute all schedules on the testbed for about four hours. We separately evaluate smaller schedules in Cooja. A custom script uses the collected data to determine the end-to-end latency by estimating each sent packet's needed timeslots. The ratio of sent and received packets determines the end-to-end reliability. We measure the networking energy consumption with Contiki-NG's energy monitoring module, Energest.

6.1.2 Environment: Testbed Kiel

We evaluate our approaches in Cooja, a network simulator, and on our 20 node testbed deployed in offices and student lab rooms (see Figure 6.1). It resides on the topmost floor of a university building where the working group stays, spanning 500 m^2. The testbed consists of 20 similar Raspberry Pis placed in different distances and rooms, allowing a realistic mapping to real environments. Our testbed location is exposed to high interference levels, especially during office hours, because it shares the wireless spectrum with Bluetooth and Wi-Fi communications outside of our control. To guarantee the same conditions for all runs, we run the evaluations during night time, when no one is in the building.

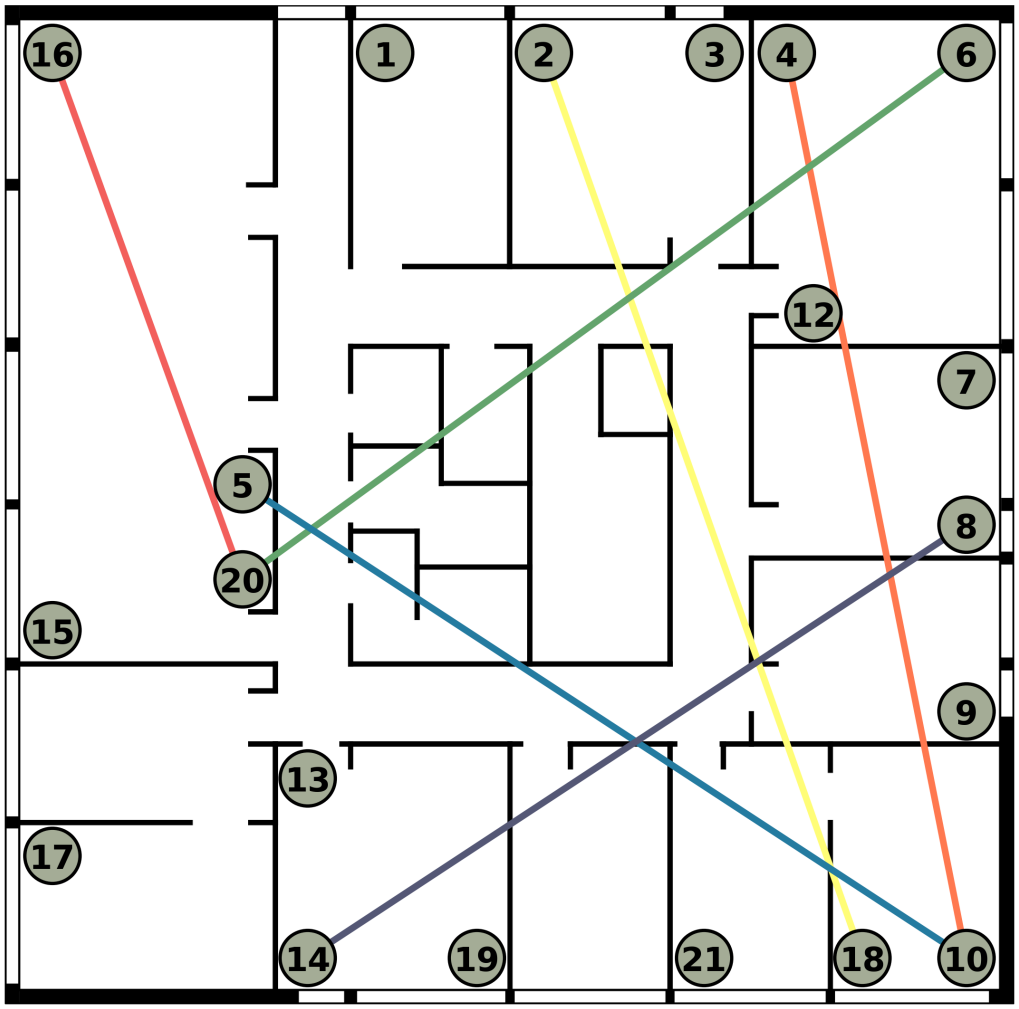


Figure 6.1: Floor plan of our 20 node testbed, spanning 500 m^2 at Kiel University. Each color represents a flow between two nodes.

6.1.3 Target platform: Zolertia Firefly (Zoul)

The platform Zolertia Firefly (Zoul) is a core module developed by Zolertia to target most IoT applications. It provides an affordable and flexible module solution to integrate into most existing products and solutions. Zoul eases the prototyping, enables fast deployment and integration of our firmware into the testbed. Table 6.1 lists the core specifications of our used target platform.

Zolertia Firefly (Zoul) - Specifications	
System on a Chip (SoC) with Microcontroller Unit	TI's CC2538 with a 32 MHz ARM© Cortex©-M3
Random-access Memory (RAM)	up to 32 KB (16 KB with retention)
flash memory	128 KB, 256 KB or 512 KB
wireless communication protocols	ISM 2.4 GHz IEEE 802.15.4 standard & Zigbee compliant

Table 6.1: The core specifications table of Zolertia Firefly (Zoul).

6.1.4 Evaluation parameters

We use the same set of parameters for all of our experiments. Our experiments use the platforms: Sky[1] and Zoul. To simulate our schedules in Cooja, we use only the marked flows (see Table 6.2) because the target platform Sky has noticeably smaller flash memory than Zoul. Therefore the whole schedule does not fit entirely in it. We adjusted TSCH's hopping sequence, which induces that we are only able to use three channels (i.e., 15, 25, and 26) during TSCH's channel hopping. Using this sequence allows us to use even periods without a drift of timeslots. We used as re-transmission strategy MASTER's Sliding Windows, with a doubled re-transmission rate per link to be comparable with already done evaluations, e.g., "MASTER: Long-Term Stable Routing and Scheduling in Low-Power Wireless Networks" [1] and being more resilient against interference. For each re-transmission, the number of timeslots is doubled. MASTER determines the window's size and the number of re-transmissions depending on each link's ETX-value (see Section 2.5.3). Using the link's ETX-values, MASTER calculates the absolute number of (all) used re-transmission slots of a flow with $n \cdot \sum \text{ETX}_{\text{link}}$, n N. We use in our case: $n = 2$. We use for our experiments six flows, as shown in Figure 6.2. All flows together cover different distances between two nodes using a varying number of relay nodes during a transmission. Thereby, each source node generates a 64 byte randomly generated data payload. The length of the communication slotframes of one second corresponds to 100 timeslots.

[1] visit: https://github.com/contiki-ng/contiki-ng/wiki/Platform-sky

ID	Flow	Hops	Deadline	Period
1.	(2 18)	(2 5), (5 13), (13 18)	34	32
2.	(4 10)	(4 8), (8 10)	66	64
3.	(6 20)	(6 2), (2 1), (1 20)	68	64
4.	(10 5)	(10 21), (21 13), (13 5)	130	128
5.	(14 8)	(14 18), (18 8)	258	256
6.	(16 20)	(16 20) (direct link)	260	256

Table 6.2: Table of flows with their deadline, period, and needed intermediate hops for routing. Bold flow IDs are used in experiments in Cooja.

6.2 Evaluation results

This section summarizes the evaluated results. Our goal is to show the effectiveness of deadline-based scheduling solutions regarding schedulability. We evaluated our created schedules in Cooja, a network simulator, and our 20 nodes testbed at Kiel University. We first evaluate the system's performance in terms of end-to-end latency, reliability, and networking energy consumption. After that, we compare the schedulability by building schedules with more and more randomly generated flows.

6.2.1 End-to-End latency

All experiments show that deadline-based scheduling overall does not significantly increase the end-to-end latency. Indeed Figures 6.2 and 6.3 show that our approaches have a higher average latency than R-LPF. However, this increase can be neglected concerning the schedulability (see Section 6.2.4). Moreover, two flows, (i.e., ID: 4 [(10 5)] and ID: 5 [(14 8)]) experienced an increase in latency during the transmission. This fact is correlated with the used algorithm. A flow that contains several hops is split during the scheduling phase on several channels and timeslots to avoid collisions. I.e., EDF prioritizes flows with a shorter deadline. These flows will be assigned to earlier timeslots with a higher probability of being scheduled in a row. If a flow with a higher deadline is scheduled, the algorithm will distribute all hops on all remaining free timeslots, which results in the worst-case in long gaps between two related hops and an increase of latency. Table 6.3 and 6.4 show this issue. Both tables show that in all cases, where the minimal latency is broadly distributed over the schedule, the average latency increase. The minimal latency determines the earliest timeslot in which a flow can be theoretically completely transmitted from the source to the destination node.

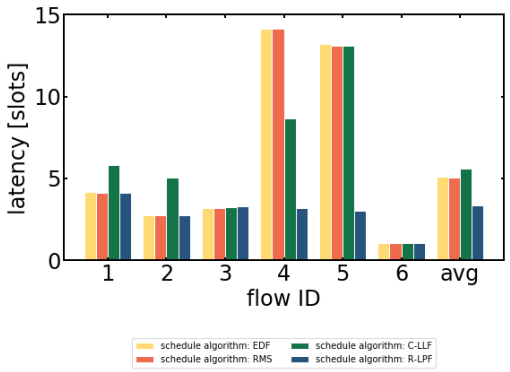


Figure 6.2: Latency of scheduling algorithms per flow executed on our 20 nodes testbed at Kiel University, including both peaks for flow four and five.

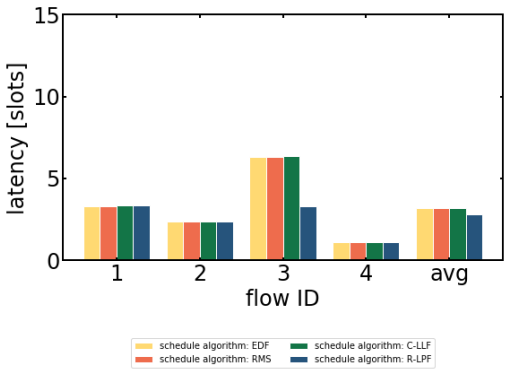


Figure 6.3: Latency of scheduling algorithms per flow executed in Cooja, including the corresponding peak for flow four (here: ID: 3).

Used algorithm	Minimal latency	Average latency
EDF	13	14
RMS	13	14
C-LLF	9	9
R-LPF	3	4

Table 6.3: The latency table of flow four [(10 5)] compares the minimal and average latency.

Used algorithm	Minimal latency	Average latency
EDF	12	13
RMS	12	13
C-LLF	12	13
R-LPF	2	3

Table 6.4: The latency table of flow five [(14 8)] compares the minimal and average latency.

6.2.2 Reliability

Our experiments in Cooja and on the testbed show that a deadline-based generated schedule does not increase the reliability compared to non-deadline-based scheduling algorithms like R-LPF. Figures 6.4 and 6.5 confirm the observations mentioned above. This fact was expected because the primary task of scheduling algorithms is to determine the order in that flows are to release, which does not affect the overall reliability. Independent of using a (non-)deadline-based scheduling algorithm, one flow (i.e., ID: 5 [(14 8)]) experience a reduced PDR compared to all other flows. This experience correlates with the number of hops a flow needs to reach its destination. A source that keeps a direct connection to its destination will never experience packet loss caused by interferences between two relay nodes where one is outside the transmissions range. A flow that's source node cannot reach its destination node through a single link and therefore depends on relay nodes, is more likely interfered by interferences not caused in its own neighborhood.

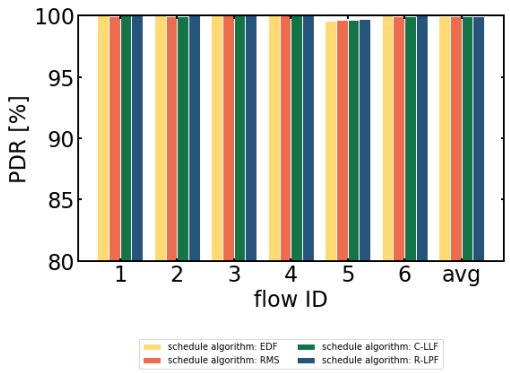


Figure 6.4: This plot shows the reliability measured during our experiments on our 20 nodes testbed at Kiel University. We evaluated each algorithm for four hours during the night.

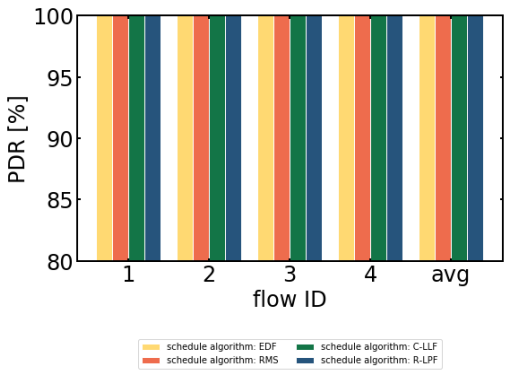


Figure 6.5: This plot shows the reliability measured during our experiments in Cooja. We evaluated each algorithm on a machine with Ubuntu.

6.2.3 Networking energy consumption

We evaluate the networking energy consumption by separating it into the sender, receiver, and relay node energy consumption. To distinguish these groups better, we only schedule one flow. We choose a flow where the source and destination are as far as possible apart from each other on the testbed (i.e., (10 16)). This circumstance should guarantee that we have a significant measurable radio on time and enough relay nodes in between. Nodes that serve as relay nodes have a higher energy consumption than the sender or receiver, as shown in Figures 6.6 and 6.7. This discrepancy may be affected

by the circumstance of using a sliding window approach, which requires that the intermediate nodes start to listen in the medium earlier than they usually would if we used a baseline or slot-based approach. The evidence that not only deadline-based approaches are affected by this issue corroborated our assumption. Finally, we can assume that using deadline-based scheduling algorithms do not affect networking energy consumption significantly.

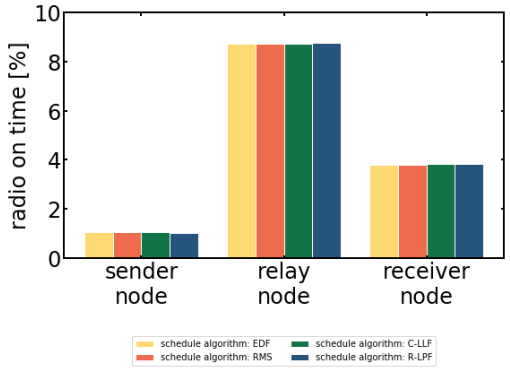


Figure 6.6: The networking energy consumption plot visualized the radio on time for the sender, receiver, and relay nodes - executed on our 20 nodes testbed at Kiel University.

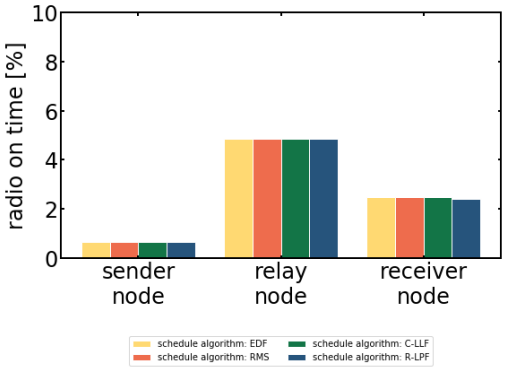


Figure 6.7: The networking energy consumption plot visualized the radio on time for an interference-free environment for the sender, receiver, and relay nodes - executed in Cooja.

6.2.4 Feasibility/Schedulability

We noticed that the more flows there are, the lower the scheduling ability is, as shown in Figure 6.8. Only 55 % of all flows could be scheduled by R-LPF and only EDF could schedule 32 randomly generated flows at once. Moreover, it outperforms all deadline-based scheduling approaches, which results from the circumstance that EDF is optimal in obtaining a solution. If EDF does not find a solution, no other algorithm can find one, as described in Section 4.3.1. C-LLF determines the next flow, which should be released by using the laxity value to identify the critical time windows in which too many conflicting transmissions need to be scheduled. This approach significantly outperforms traditional scheduling policies, e.g., RMS, and is highly effective in meeting deadlines [4]. Therefore it is the second-best algorithm as a result of our experiments. Accordingly to our analysis, the other algorithms (i.e., EDF and C-LLF) outperform RMS, which lies in how it schedules the set of flows. Instead of prioritizing a shorter deadline as EDF it does, RMS prioritizes shorter periods, which lead to infeasible flow constellations where flows miss their deadline even if they usually can hold them. A flow with a longer period, but an overall shorter deadline, can be later included where it may fail. Our

experiments show that the use of deadline constraints outperforms the concept of non-deadline-based scheduling algorithms in terms of schedulability and increases the probability of obtaining a feasible schedule.

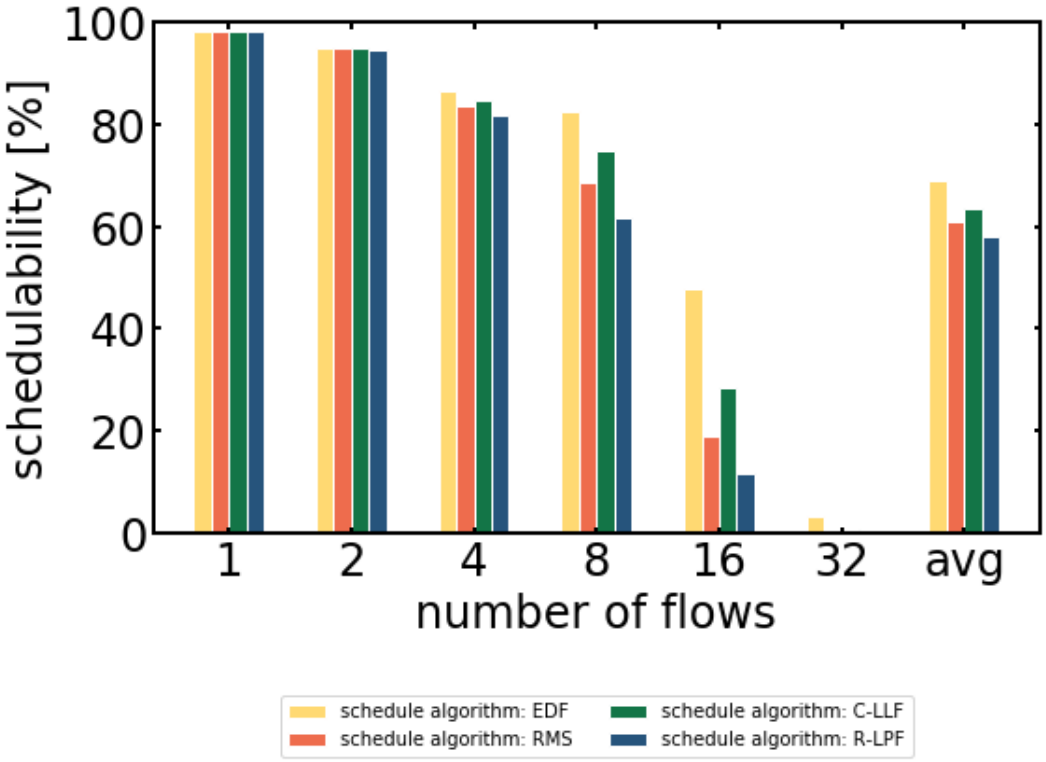


Figure 6.8: The schedulability plot shows the rapid decrease of schedulability. Almost all algorithms failed to schedule 32 flows at once.

7

Conclusion & Further Research

7.1 Conclusion

This thesis introduces and justifies our design ideas and extends MASTER to take into account deadlines and periods. We enabled hyper-periods in MASTER and provided it with a schedulability analysis. Our introduced design shifted MASTER from a single scheduling scheme without hyper-periods to a period-based approach by increasing the overall schedulability. Thereby we utilized MASTER's centralized knowledge about the underlying topology. It enables creating a more adaptable and resilient schedule to interference changes without the need for rescheduling.

We have evaluated our design goals extensively in Cooja, a network simulator, and on our 20 nodes testbed in an environment susceptible to interference. Our primary focus was to evaluate our schedulers regarding end-to-end latency, reliability, networking energy consumption, and schedulability. We have provided a comparison with MASTER's Reverse Longest Path First algorithm shows that deadline-based scheduling outperforms regarding schedulability by a negligent latency increase. The increase in latency can be neglected because it does not differentiate by more than 5 % on average. Overall our design and implementation successfully met our expectations and specifications.

7.2 Further Research

Our schedulability analysis analyzes each schedule during the build-time and tests if all transmissions meet their deadline. This approach may lead to an enormous overhead for larger schedules. For a better modulation in terms of separation of analysis and scheduling, and to increase the performance, we could also use a different schedulability analysis approach. Many approaches verifying the schedulability of a set of transmissions without starting the actual building process. Modekurthy et al. [26] propose a utilization-based approach for schedulability analysis in Wireless Control Systems (WCO), which can be slightly modified and utilized to replace our current approach.

Our approach of hyper-periods has two drawbacks. During the computation of a hyper-period, transmissions that are not adequately modeled can lead to an enormously hyper-period, increasing power consumption. Moreover, the schedule size depends on hyper-period. A higher hyper-period would also result in an increased schedule size. Therefore, we forbid relatively prime numbers to avoid this issue. Our second restriction aims the fact that we cannot use periods that are prime numbers, even if they are small. Blacklisting is highly discouraged and should be prevented. Hence, the next modification should aim at this problem. Ahmad et al. [2] propose a possible solution that would solve the limitation of using only non-relative prime periods. They present a resource-aware approach to minimize the hyper-period of input transmissions based on device profiles [2]. A second approach, which would enable MASTER to build and handle schedules during the run-time, could also eliminate fixed hyper-periods with non-relatively prime periods. We could then create smaller schedules, on-demand, without the need for an overall hyper-period. Both approaches would reduce power consumption, be more suitable for manufacturing usages by not limiting the scheduling process.

Bibliography

[1] O. Harms and O. Landsiedel, "MASTER: Long-Term Stable Routing and Scheduling in Low-Power Wireless Networks", 2020.

[2] S. Ahmad, S. Malik, and I. Ullah et al., "An Adaptive Approach Based on Resource-Awareness Towards Power-Efficient Real-Time Periodic Task Modeling on Embedded IoT Devices", 2018, pp. 1–29.

[3] (2020). 802.15.4-2015 - IEEE Standard for Low-Rate Wireless Networks, [Online]. Available: https://ieeexplore.ieee.org/document/7460875 (visited on 06/12/2020).

[4] A. Saifullah, Y. Xu, C. Lu, and Y. Che, "Real-Time Scheduling for WirelessHART Networks", in *2010 31st IEEE Real-Time Systems Symposium*, 2010, pp. 1–10.

[5] (2020). Contiki-NG: Home, [Online]. Available: https://github.com/contiki-ng/contiki-ng/wiki (visited on 07/29/2020).

[6] D. De Couto, "High-Throughput Routing for Multi-Hop Wireless Networks", 2004, pp. 15–17.

[7] H. Schulte, A. Badach, and L. Stiegler et al., "TSCH - Time-Slotted Channel Hopping", in *Protokolle und Dienste der Informationstechnologie online*, 2008, pp. 1–19.

[8] D. Chen, M. Nixon, and A. Mok, "Chapter 01 Overview", in *WirelessHART™ - Real-Time Mesh Network for Industrial Automation*, 2010, pp. 3–14.

[9] (2020). HART TECHNOLOGY DETAIL, [Online]. Available: https://fieldcommgroup.org/technologies/hart/hart-technology-detail (visited on 06/12/2020).

[10] T. Lennvall, S. Svensson, and Fredrik Hekland, "A Comparison of WirelessHART and ZigBee for Industrial Applications", in *2008 IEEE International Workshop on Factory Communication Systems*, 2008, pp. 1–3.

[11] E. W. Dijkstra, "A note on two problems in connexion with graphs", in *Numerische Mathematik. 1 (Numer. Math. 1)*, 1959, pp. 269–271.

[12] L. Sha and S. S. Sathaye, “Distributed Real-Time System Design: Theoretical Concepts and Applications”, 1993, pp. 1–3.

[13] F. Zhang and A. Burns, “Schedulability Analysis for Real-Time Systems with EDF Scheduling”, in *IEEE Transactions on Computers Volume: 58, Issue: 9, Sep. 2009*, 2009, pp. 1–2.

[14] A. Toptal and I. Sabuncuoglu, “Distributed scheduling: a review of concepts and applications”, in *International Journal of Production Research Volume: 48, Issue 18, Aug. 2010*, 2009, pp. 5235–5237.

[15] C. Yadati, C. Witteveen, and Y. Zhang et al., “Autonomous Scheduling with Unbounded and Bounded Agents”, in *German Conference on Multiagent System Technologies MATES 2008: Multiagent System Technologies*, 2008, pp. 195–206.

[16] H. Li, “Centralized Scheduler”, in *Communications for Control in Cyber Physical Systems Theory, Design and Applications in Smart Grids*, 2016, pp. 181–217.

[17] O. Beaumont, L. Carter, and J. Ferrante et al., “Centralized versus distributed schedulers for multiple bag-of-task applications”, in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, 2006, pp. 1–2.

[18] S. Bansal and C. Hota, “Priority-Based Job Scheduling in Distributed Systems”, in *International Conference on Information Systems, Technology and Management ICISTM 2009: Information Systems, Technology and Management*, 2009, pp. 110–111.

[19] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, “Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH”, in *SenSys ’15: Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, 2015, pp. 337–350.

[20] J. Shi, M. Sha, and Z. Yang, “DiGS: Distributed Graph Routing and Scheduling for Industrial Wireless Sensor-Actuator Networks”, in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 354–364.

[21] S. Kim, H.-S. Kim, and C. Kim, “ALICE: Autonomous Link-based Cell Scheduling for TSCH”, in *2019 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2019, pp. 121–132.

[22] H. Zhang, P. Soldati, and M. Johansson, “Performance Bounds and Latency-Optimal Scheduling for Convergecast in WirelessHART Networks”, in *IEEE Transactions on Wireless Communications Volume: 12, Issue: 6, June 2013*, 2013, pp. 2688–2696.

[23] Y. Jin, P. Kulkarni, and J. Wilcox et al., "A Centralized Scheduling Algorithm for IEEE 802.15.4e TSCH based Industrial Low Power Wireless Networks", in *IEEE Wireless Communications and Networking Conference (WCNC 2016)*, 2016, pp. 1–6.

[24] A. Darbandi and M. K. Kim, "Path Collision-aware Real-time Link Scheduling for TSCH Wireless Networks", in *KSII Transactions on Internet and Information Systems Volume: 13, Issue: 9, Sep. 2019*, 2019, pp. 4429–4445.

[25] A. Tinka, T. Watteyne, and K. Pister, "A Decentralized Scheduling Algorithm for Time Synchronized Channel Hopping", in *International Conference on Ad Hoc Networks ADHOCNETS 2010: Ad Hoc Networks*, 2010, pp. 201–216.

[26] V. P. Modekurthy, D. Ismail, M. Rahman, and A. Saifullah, "A Utilization-Based Approach for Schedulability Analysis in Wireless Control Systems", in *2018 IEEE International Conference on Industrial Internet (ICII)*, 2018, pp. 1–10.

[27] A. Saifullah, D. Gunatilaka, and P. Tiwari et al., "Schedulability Analysis under Graph Routing in WirelessHART Networks", in *2015 IEEE Real-Time Systems Symposium*, 2015, pp. 165–174.

[28] C. Wu, M. Sha, and D. Gunatilaka et al., "Analysis of EDF scheduling for Wireless Sensor-Actuator Networks", in *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)*, 2014, pp. 31–40.

[29] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "End-to-End Delay Analysis for Fixed Priority Scheduling in WirelessHART Networks", in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2011, pp. 165–174.

[30] J. D. Ullman, "NP-complete scheduling problems", in *Journal of Computer and System Sciences Volume: 10, Issue: 3, June 1975*, 1975, pp. 384–393.

[31] R. Cayssials, J. Orozco, J. Santos, and R. Santos, "Rate Monotonic scheduling of real-time control systemswith the minimum number of priority levels", in *Proceedings of 11th Euromicro Conference on Real-Time Systems. Euromicro RTS'99*, 1999, pp. 1–2.

[32] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", in *Journal of the ACM Volume: 20, Issue: 1, Jan. 1973*, 1973, pp. 46–61.