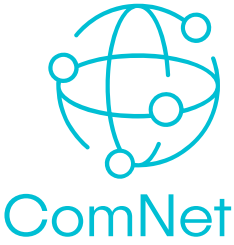

ComNets

TUHH
Institute of
Communication
Networks

PROJECT THESIS

LAURA BECKER

—

Implementation and Evaluation of Age of Information in Status Update Systems using a 6TiSCH Testbed

February 10, 2023

Implementation and Evaluation of Age of Information in Status Update Systems using a 6TiSCH Testbed
Implementierung und Evaluierung von Age of Information in Status Update Systems mittels eines 6TiSCH Testbed

Laura Becker
Matriculation number: 52198
M. Sc. Computer Science

Hamburg University of Technology
Institute of Communication Networks
First examiner: Prof. Dr.-Ing. Timm-Giel

Supervisor: Leonard Fisser, Yevhenii Shudrenko

Hamburg, February 10, 2023

Declaration of Originality

I hereby declare that the work in this thesis was composed and originated by myself and has not been submitted for another degree or diploma at any university or other institute of tertiary education.

I certify that all information sources and literature used are indicated in the text and a list of references is given in the bibliography.

Hamburg, February 10, 2023

Laura Becker

Abstract

This thesis describes the implementation of a testbed to execute and analyze dissemination schedules in multi-source, multi-monitor, and multi-hop Status Update Systems (SUS) with respect to Age of Information (AoI). Applications like environmental monitoring or smart grid require SUS that periodically distribute status information, like sensor data, of each node to all network participants. Because the nodes use the status information in decision making, the freshness of information is a key performance indicator, which can be determined using the performance metric AoI. Dissemination schedules define the broadcasting and forwarding of the status updates and have therefore a direct impact on the AoI. In current literature, optimized dissemination schedules are derived using Minimum Connected Dominating Sets (MCDS). To analyze these theoretical dissemination schedules on physical Information and Communication Technology (ICT) platforms, an IPv6 over the Timeslotted Channel Hopping mode of IEEE 802.15.4 (6TiSCH) testbed is implemented based on the Operating System Contiki NG and executed on OpenMote B boards. The testbed concept supports scheduling of the slotframes and the implementation of the application protocol providing the execution of arbitrary dissemination schedules. Afterwards various setups are evaluated considering different age metrics and the Packet Delivery Ratio (PDR). The setups differ in their configuration of the slotframes, the network topology, and the environment interference. The measurement results prove that the testbed is able to execute dissemination schedules for different topologies using a timeslotted communication with channel hopping. Furthermore, the measurement results are used to derive optimized slotframe configurations depending on the application requirements.

Contents

Acronyms

6LoWPAN	IPv6 over Low Power Wireless Personal Area Networks
6LoWPAN HC	6LoWPAN Header Compression
6LoWPAN ND	6LoWPAN Neighbor Discovery
6LoWPAN RH	6LoWPAN Routing Header
6TiSCH	IPv6 over the Timeslotted Channel Hopping mode of IEEE 802.15.4
6top	6TiSCH Operation Sublayer
AoI	Age of Information
ASN	Absolute Slot Number
CSMA	Carrier Sense Multiple Access
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
EB	Enhanced Beacon
ICT	Information and Communication Technology
IETF	Internet Engieering Task Force
IoT	Internet of Things
IPv6 ND	IPv6 Neighbor Discovery
json	JavaScript Object Notation
MAC	Media Access Control
MCDS	Minimum Connected Dominating Sets
MTU	Maximum Transmission Unit
PAN	Personal Area Network
PDR	Packet Delivery Ratio
QoS	Quality of Service
RPL	Routing Protocol for Low-Power and Lossy Networks
SUS	Status Update Systems

TSCH	Timeslotted Channel Hopping
UDP	User Datagram Protocol
WSN	Wireless Sensor Network

1. Introduction

The demand for distributed control systems is increased by Industry 4.0. Applications like smart grids, environmental monitoring, but also vehicular systems establish distributed control systems often implemented as Status Update Systems (SUS). In SUS the network participants consider the process information, for example sensor data, of the other network nodes in decision making. Therefore, part of SUS are periodic status updates that are disseminated through the network and each participant stores the received information.
To enable the wireless dissemination of information, Wireless Sensor Networks (WSNs) based on the well known IEEE 802.15.4 [1] standard are used which have to fulfill requirements regarding reliability, high throughput, low-power, and low latency. Additionally, in SUS the Age of Information (AoI) is the key performance metric to measure the freshness of information. AoI is the difference between the current timestamp and the timestamp of the status update, which is defined as the timestamp the monitoring node enqueues a packet containing its own status update.
Dissemination schedules determine how information is distributed. In SUS a process is assumed to occur at each node and the node broadcasts the process information to the network. The schedule defines the order in which the nodes broadcast their process updates and which nodes forward these updates. Therefore, the dissemination schedules have a large impact on AoI. In general, scheduling transmission resources optimally with respect to AoI is a NP-hard problem. In current research, theoretical optimal dissemination schedules are derived using Minimum Connected Dominating Sets (MCDS) [2]. These more advanced schedules are based on slotted communication, but current research focuses on unslotted communication. Furthermore, in current research dissemination schedules for SUS are mainly analyzed using simulations which are limited in terms of representativeness in realistic environments.
Therefore, this thesis focuses on the analysis of SUS on physical Information and Communication Technology (ICT) platforms based on slotted communication. IPv6 over the Timeslotted Channel Hopping mode of IEEE 802.15.4 (6TiSCH) is one possible protocol stack including IPv6 communication over a WSN and the Media Access Control (MAC) layer is based on slotted communication involving channel hopping. Timeslotted Channel Hopping (TSCH) enables deterministic transmissions such that TSCH is suitable to fulfill application requirements like a deterministic and small latency. In addition, a high reliability is supported by the channel hopping algorithm.

This thesis describes the implementation of a 6TiSCH testbed that deploys SUS and is able to execute predefined dissemination schedules. The testbed is used to analyze the

performance of theoretical dissemination schedules considering reliability and AoI on physical hardware in a realistic environment. The considered SUS consist of multi-hop network topologies and every node is considered to be a source and sink of information, such that every node has to store the status information of all other network participants and every node sends periodic status updates.

1.1. State of the Art

AoI was first introduced by Kaul et al. [3]. The considered application is a single-hop wireless vehicular network using 802.11 Carrier Sense Multiple Access (CSMA) as MAC layer protocol. Kaul et al. provide an application-layer broadcast rate adaptation algorithm to decrease the probability of packet collisions resulting in reduced AoI [3]. Kaul et al. also consider AoI in vehicular networks in the context of specific multi-hop network settings [4]. Furthermore, the MAC layer is not based on CSMA but on timeslotted communication. Kaul et. al. provide a numerical analysis of AoI based on a round robin schedule [4]. A general transmission policy for multi-hop networks using slotted communication is derived, for example, by Talak et al. [5]. However, this approach requires that the source and monitor node pairs are predefined, such that a single update is only send to a set of nodes. This restriction is relaxed by Farazi et al. [2] providing a dissemination schedule for SUS in which all nodes are both source and monitor of the status information. The derivation of optimal schedules is based on MCDS. Furthermore, fundamental bounds for AoI are determined. The schedules do not consider retransmissions, but among other the work of Farazi et al. [2] is part of the basics that are used to provide a dissemination schedule that also considers retransmissions for a multi-hop, multi-source, and multi-monitor networks [6]. In the described research, AoI is analyzed using analytical computations and/or numerical results, but an evaluation based on a testbed is missing.

Contiki NG is a well known operating system for Internet of Things (IoT) devices, which is available on GitHub, and is adapted for different ICT platforms [7]. Contiki NG is used in current research, for example, Duquennoy et al. evaluate the 6TiSCH implementation of Contiki NG with respect to metrics like Packet Delivery Ratio (PDR) and latency using the FIT IoT-LAB testbed [8], [9]. Furthermore, Ibrahimova et al. use Contiki to analyze 6TiSCH Quality of Service (QoS) routines [10]. Ibrahimova et al. focus on comparing different QoS mechanisms that have an impact on the prioritization of the packets in the TSCH sending queue. The goal of this study is to implement different QoS mechanisms for the 6TiSCH protocol and to analyze their performance from a fairness perspective. Tomasic et al. [11] compare Contiki NG and the operating system OpenWSN with respect to remote health monitoring. The implementation is executed on OpenMote-cc2538 IoT boards, a predecessor of the OpenMote B [12].

To the best of my knowledge, there is no testbed to evaluate AoI in multi-hop SUS based on slotted communication such that there is no possibility to execute theoretical

dissemination schedules based on slotted communication on a testbed. Therefore, this thesis describes a 6TiSCH testbed that executes dissemination schedules to enable the performance evaluation on physical hardware. The testbed implementation of this thesis is based on the operating system Contiki NG and is executed on OpenMote B boards.

1.2. Structure of the Thesis

The focus of this thesis is on the implementation of a 6TiSCH testbed and the evaluation of theoretical optimal dissemination schedules with respect to AoI. In Chapter 2 the fundamental characteristics of 6TiSCH are discussed. Furthermore, AoI and dissemination schedules are described. The testbed setup is explained in Chapter 3 focusing on the MAC layer scheduling and the implementation of the application protocol. In Chapter 4, the measurement setups are shown with respect to the analyzed topologies and testbed configurations. Furthermore, expected results are discussed. Afterwards, the execution of the different dissemination schedules are analyzed based on the introduced performance metrics. In Chapter 6, the main results are summarized.

2. Theoretical Fundamentals

The following chapter focuses on the theoretical fundamentals of AoI and 6TiSCH. The AoI and dissemination schedules are introduced and different age metrics and the performance metric PDR are defined. Afterwards, the main characteristics of the communication stack 6TiSCH are described.

2.1. Age of Information

AoI is the key performance metric in SUS to measure the freshness of information [3]. In SUS, each node requires the status of all other participants such that the nodes periodically broadcast status updates [2]. The age of the information of process H_j at node i at time t is defined as the difference of the current timestamp t and the timestamp of the stored information $\tau_j^{(i)}$ (Eq. 2.1) [2].

$$\Delta_j^{(i)}(t) = t - \tau_j^{(i)} \tag{2.1}$$

The timestamp $\tau_j^{(i)}$ is the timestamp when the packet containing the status update is enqueued by the node monitoring process H_j directly. The age consists of the dissemination time to propagate the update through the network, and the update rate, i.e., how often an update is sent. In a multi-hop network it is required that the network nodes forward the status updates to distribute the information through the network. Hence, the dissemination time depends on the number of hops between sender and receiver as well as on the respective queuing delays. The receiver of a status update only stores the received information if the update decreases the AoI. This process is described by the discrete time model Eq. 2.2 [2].

$$\Delta_j^{(m)}[n+1] = \begin{cases} 1, & m \in N_1(i) \text{ and } i = j \\ \Delta_j^{(i)}[n] + 1, & m \in N_1(i),\ i \neq j \text{ and } \Delta_j^{(i)}[n] < \Delta_j^{(m)}[n] \\ \Delta_j^{(m)}[n] + 1, & \text{otherwise.} \end{cases} \tag{2.2}$$

The discrete time model describes the increasing and decreasing of the age of a process information H_j at node m for the timestamp $n+1$. If a node i sends a status update, three different cases can occur. The update can only be received, if node m is a neighbor of the sending node i. In the first case, node m is a neighbor of node i ($m \in N_1(i)$) and node i sends the status update of its own process H_i ($i = j$). Packets containing the own

status information are referred as *origin packets*. In this model, it is assumed that each transmission is received without retransmissions such that there are no queuing delays. Furthermore, a transmission between direct neighbors is considered to have a delay of one timeslot. Therefore, in the first case the age is initialized with one.
If node i does not send its own update (*forwarding packet*), but the status information of H_j, node m has to check whether the timestamp of the received status information is larger than the timestamp of the stored information. If the received information decreases the age, the information is stored. Therefore, in case two the age of the information stored at node m is the age of the considered process at the sending node i plus one.
The last case considers on the one hand side that node m is not a neighbor of node i, such that the update is not received. On the other hand, the case also includes that the received status update is older then the already stored one, such that the status update has to be ignored. Therefore, the age is simply increased by one [2].
The age vector, representing the AoI of a whole network, stores all combinations of *node id* and *process id*, except if *process* and *node id* are identical. In Eq. 2.3, the age vector of a three-node line network (Fig. 2.1) is shown. The first two entries represent the AoI at node 1. So the first entry is the age of H_2 and the second of H_3. The other tuples consider the age at node 2 and 3 [2].

$$\Delta(t) = [\Delta_2^{(1)}(t), \Delta_3^{(1)}(t), \Delta_1^{(2)}(t), \Delta_3^{(2)}(t), \Delta_1^{(3)}(t), \Delta_2^{(3)}(t)]^T \tag{2.3}$$

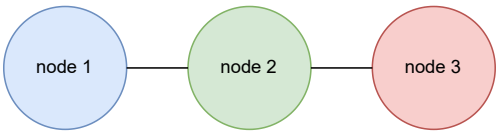

Figure 2.1.: Three-node line network.

2.1.1. Dissemination Schedules

The dissemination schedule determines the distribution of information through the network. In this thesis, it is assumed that the nodes always send broadcast messages. Therefore, the schedule determines at which timeslot a specific sender node distributes which status information. The optimal schedule of the transmission resources with respect to AoI is a known NP-hard problem. In research, MCDS are used to derive optimal schedules [2]. In this thesis, predefined dissemination schedules for slotted communication in multi-hop, multi-source and multi-monitor networks are analyzed which are optimal with respect to AoI.

2.1.2. Example Dissemination Schedule and AoI Graphs in a Line Setup

Table 2.1 illustrates an optimal dissemination schedule for a simple three-node line setup (Fig. 2.1). The schedule is defined by a tuple consisting of the *sender id* and the *process id* of the considered status update. Only the topological neighbors are able to receive the broadcast messages. Furthermore, the initial value of the entries of the age vector are infinite. In the first timeslot node 1 sends its own status update to its direct neighbor node 2. Considering the discrete process (Eq. 2.2) this fits the first case, because node 1 sends its origin packet to its direct neighbor. Therefore, the initial age is one. Afterwards, the status update is forwarded to node 3. Because node 2 is a direct neighbor of node 3, but does not send its own update, the second or third case of Eq. 2.2 has to be considered. Because the status update improves the age of the stored information, the second case is applicable. In the third timeslot node 2 sends its origin packet. Due to the usage of broadcasts it is possible to send the update to all neighbors in a single timeslot such that it is received by node 1 and node 2. The last part of the dissemination schedule determines the sending of the process information of H_3 using node 2 as a forwarding node [2].

Table 2.1.: Dissemination schedule three-node line setup [2].

time	dissemination schedule	sender	process	receiver	age vector $[\Delta_2^{(1)}, \Delta_3^{(1)}, \Delta_1^{(2)}, \Delta_3^{(2)}, \Delta_1^{(3)}, \Delta_2^{(3)}]^T$
1	(1,1)	node 1	H_1	node 2	$[-, -, 1, -, -, -]^T$
2	(2,1)	node 2	H_1	node 3	$[-, -, 2, -, 2, -]^T$
3	(2,2)	node 2	H_2	node 1, 2	$[1, -, 3, -, 3, 1]^T$
4	(3,3)	node 3	H_3	node 2	$[2, -, 4, 1, 4, 2]^T$
5	(3,2)	node 2	H_3	node 1	$[2, 2, 5, 2, 5, 3]^T$

AoI is graphically represented using a line graph. The y axis represents the age of the stored update and the x axis the time measured in timeslots. In Fig. 2.2, the AoI graph at node 1 is shown. The dashed line represents the age due to the dissemination. The time the information is used by the node until it is replaced by the next update is shown by a solid line. The color of the line indicates the considered process and is equal to the color in Fig. 2.1, such that the green line illustrates the age of process H_2. Because node 1 and node 2 are direct neighbors, the dissemination delay is one timeslot. The first update is sent at $t = 2$ and received at $t = 3$. Then the age is increased until $t = 8$. In this example, one dissemination round consists of five slots. Therefore, if the first update is sent at timeslot 2, the next is sent at $t = 7$. A similar shape is described by the red line for the status information of node 3. One difference is, that the dissemination delay is higher compared to the green line due to the fact that node 1 and node 3 are not neighbors. This also implies that the maximum age is larger for process information H_3 compared to H_2 [2].

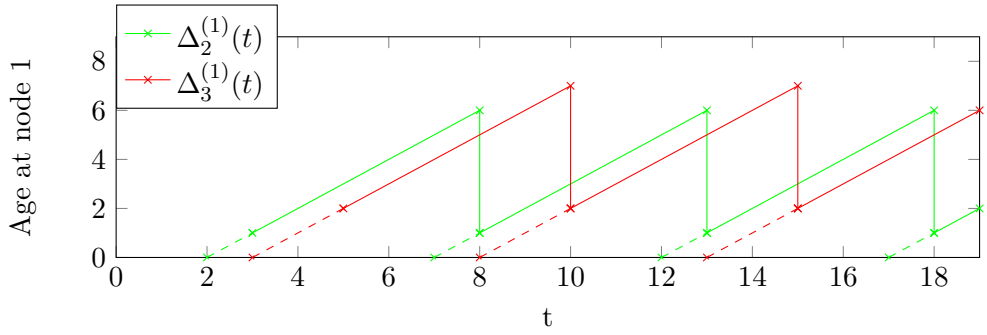

Figure 2.2.: AoI curve at node 1 for a three-node line network [2].

In Fig. 2.3, the AoI at node 2 is shown. The blue line represents the AoI of process H_1 and the red line of H_3. In contrast to Fig. 2.2, the maximum of both lines is equal. The reason for this is, that node 2 is a direct neighbor of all other nodes in the network, such that the dissemination delay is always one.

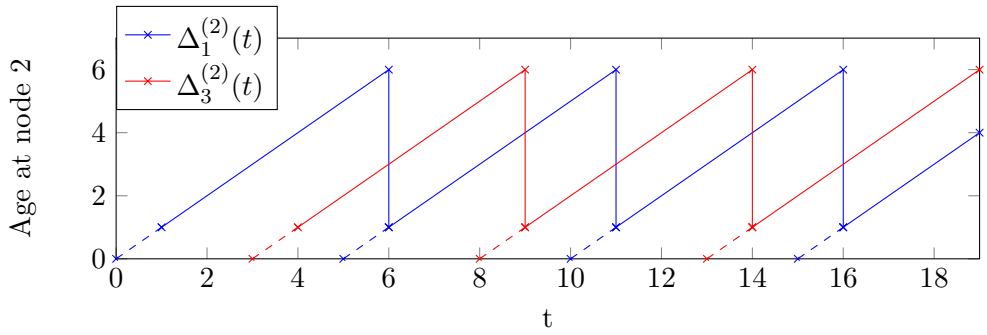

Figure 2.3.: AoI curve at node 2 for a three-node line network [2].

The graph of AoI at node 3 (Fig. 2.4) is very similar to the AoI graph at node 1 because both nodes have only one direct neighbor.

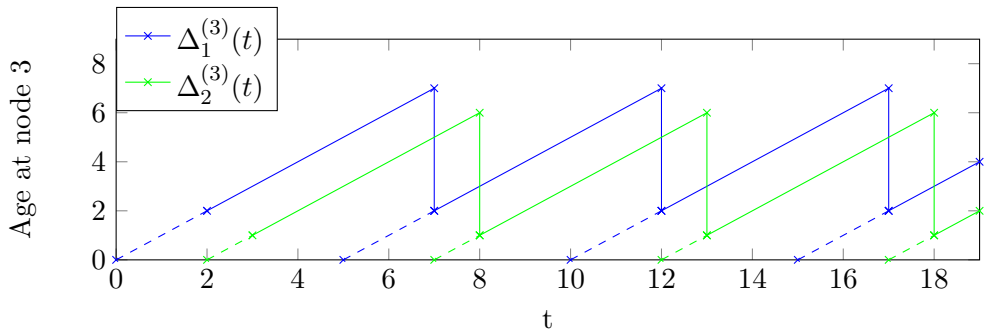

Figure 2.4.: AoI curve at node 3 for a three-node line network [2].

2.2. Age Metrics and Packet Delivery Ratio

To analyze the performance of the dissemination schedule, the AoI is measured. Additionally, the PDR is determined, analyzing the dependency between reliability and AoI as well as the effect of interference. In literature AoI is evaluated using different statistical metrics focusing on peak and average age values. The *instantaneous peak* Δ_{peak} and the *instantaneous average* Δ_{avg} are used to compute the peak and average value for a specific timestamp t over all process updates at all nodes (Eq. 2.4, Eq. 2.5). The variable N is the number of network nodes such that $N(N-1)$ represents the amount of data streams that has to be supported by the SUS.

$$\Delta_{peak}(t) = max\Delta(t) \tag{2.4}$$

$$\Delta_{avg}(t) = \frac{1^T\Delta(t)}{N(N-1)} \tag{2.5}$$

The computation of the instantaneous peak and instantaneous average at every timestamp t results in a graph describing the increase and decrease of peak and average over the whole runtime [2]. The metrics are only defined, if the age of every process status information is defined. This implies that the analyzed node has to receive at least on status update of each process. To analyze the peak and average value over the whole runtime ($[t_0, t_1]$) and not for a single timestamp, the *peak age* and *average age* is used. The *peak age* defines the maximum of the *instantaneous peak* over $[t_0, t_1]$ (Eq. 2.6).

$$\Delta_{peak}(t_0, t_1) = \sup_{t_0 \leq t < t_1} \Delta_{peak}(t) \tag{2.6}$$

The average age describes the integral of the instantaneous average divided by the size of the interval $[t_0, t_1]$ [2] (Eq. 2.7).

$$\Delta_{avg}(t_0, t_1) = \frac{1}{t_1 - t_0} \int_{t_0}^{t_1} \Delta_{avg}(t)dt \tag{2.7}$$

In general, the PDR is defined by the ratio between generated and received packets in the whole network (Eq. 2.8). Therefore, the PDR is an indicator for packet loss [13].

$$PDR = \frac{P_{Received} \cdot 100}{\sum_{i=1}^{N} P_{Generated_i}} \tag{2.8}$$

In this thesis the general formula of the PDR is adapted to analyze the packet loss at every node for every process individually. The testbed sends only broadcast messages. Therefore, to analyze the PDR at each node individually, it has to be measured, which nodes were able to receive a specific broadcast message. Therefore, for a single broadcast message that should be received at x nodes, $P_{Generated}$ is increased by x.

2.3. IPv6 over the Timeslotted Channel Hopping mode of IEEE 802.15.4

6TiSCH is an open protocol stack by the Internet Engieering Task Force (IETF) that provides IPv6 routing over a WSN using TSCH as MAC layer protocol. The stack is provided in detail in Fig. 2.5. The communication schedules based on TSCH enable to fulfill QoS requirements (Sec. 2.3.1). To support distributed scheduling management, the 6TiSCH Operation Sublayer (6top), including the 6P protocol and different scheduling functions, is part of the 6TiSCH stack (Sec. 2.3.2). An adaption layer based on IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN) realizes sending of IPv6 packets over the IEEE 802.15.4 physical layer using header compression and encapsulation of IPv6 routing information in the 6LoWPAN header. Furthermore, 6LoWPAN adapts the neighbor discovery that is used by IPv6 (Sec. 2.3.3). User Datagram Protocol (UDP) is used as the default transport protocol. Another part of the 6TiSCH stack is the Routing Protocol for Low-Power and Lossy Networks (RPL) [14].

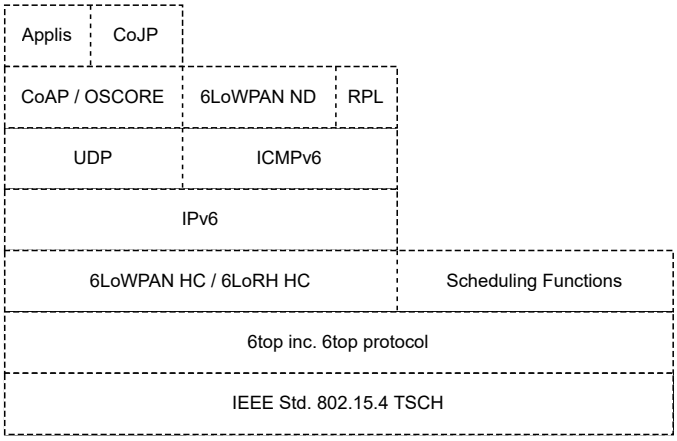

Figure 2.5.: 6TiSCH protocol stack [14].

2.3.1. Time Slotted Channel Hopping

The MAC layer of 6TiSCH consists of the TSCH mode of the IEEE 802.15.4 standard [1]. TSCH defines a slotted MAC layer communication including a channel hopping algorithm. In TSCH networks the time is organized in slotframes and is completely synchronized between network nodes. A slotframe consists of a predefined number of timeslots. In Fig. 2.6 slotframes consisting of five slots are shown. The slotframes are defined locally at every node in the network. Per timeslot it is possible to send a packet and receive the corresponding acknowledgment. An acknowledgment is only requested if the destination of the frame is no broadcast or group address. The definition of the

schedule is based on *cells*, described by the tuple [*slotOffset, channelOffset*]. For example, in Fig. 2.6 the first link between node A and node B is defined by $[0, 2]$. The slot offset corresponds to the used timeslot, such that it is predefined at which timeslot a node is active. Between the active slots, the node is able to go into sleep mode resulting in a reduced power consumption.
The channel offset is used to determine, which physical channel is used from the *macHoppingSequenceList*. The list defines all usable physical channels of the network. In each timeslot the physical channel is determined by Eq. 2.9. The Absolute Slot Number (ASN) represents the current number of slots since network initialization. The slot offset is reset if a new slotframe starts, but the ASN is constantly incremented. Furthermore, Eq. 2.9 depends on the *macHoppingSequenceLength* which is the length of the *macHoppingSequenceList*. Because the physical channel depends on the ASN, a retransmission in a cell with the same channel and slot offset results in a different frequency reducing the effect of interference. The availability of different channel offsets enables to send multiple packets in one timeslot using different channels which is the main advantage compared to the classical Time Division Multiple Access (TDMA) MAC protocol [1], [14].

$$CH = macHoppingSequenceList[(ASN + channelOffset) \mod macHoppingSequenceLength] \quad (2.9)$$

TSCH provides different link options. First the cell can be reserved for one single link (dedicated cell) or can be shared between multiple links (auto cell). In case of a dedicated cell, it has to be defined whether the link is used for transmission (TX) or reception (RX) [1], [14]. In Fig. 2.6, the cell at $[0, 2]$ is a dedicated TX cell at node A. This implies that at node B a RX cell is required at $[0, 2]$ to receive the packet of node A. An example of an auto cell would be at $[2, 3]$.

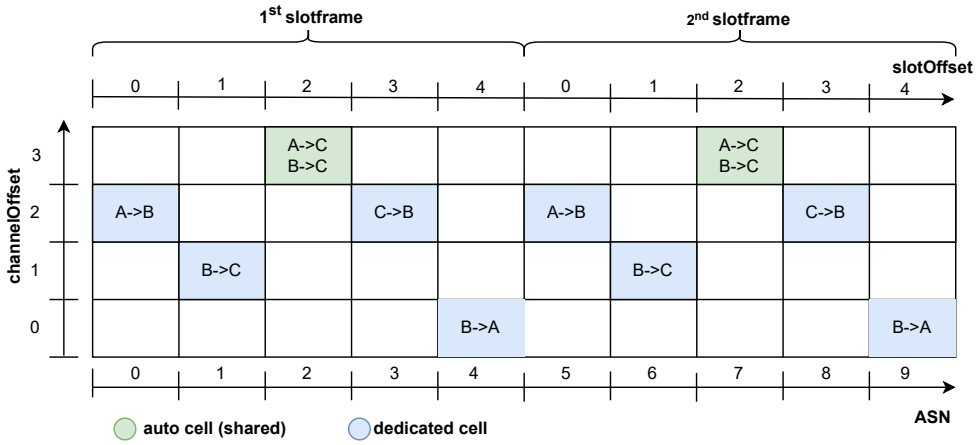

Figure 2.6.: Example TSCH cell schedule.

To create a TSCH Personal Area Network (PAN), the coordinator sends Enhanced Beacons (EBs) containing time synchronization information and specifications regarding channel hopping and timeslots required to join the network. To maintain the time synchronization, a device should synchronize its network time with a time source neighbor. The time drift can be determined by receiving a frame or an acknowledgment containing time correction information from such a neighbor.
The number of retransmissions is limited by the parameter *macMaxFrameRetries*, which has a default value of three. In shared slots collisions are possible such that in IEEE 802.15.4 a TSCH CSMA-CA retransmission algorithm is defined. This retransmission backoff algorithm reduces the probability of repeated collisions [1], [14].

2.3.2. 6P and Scheduling Functions

Due to the TSCH MAC layer, 6TiSCH implies a cell-based scheduling. This schedule can be adapted dynamically by different scheduling functions. These functions manage and control the slotframe configuration. To adapt the slotframe schedule, the 6P protocol is required. The scheduling functions trigger 6P transactions to add, delete, or relocate a cell. Both, scheduling functions and 6P, are part of the 6top [15]. Because every node is able to use 6P to communicate with its neighbors to adapt the schedule, 6TiSCH provides a distributed schedule management. Therefore, each node can improve its performance by adding or deleting cells in case that the provided capacity on the MAC layer is not adequate for its application. Furthermore, relocation of a cell to another slot offset and/or channel offset is used if the probability of collisions is significantly higher on this link compared to the others. In 6top the consistency of the schedules is maintained. Each node defines its schedule locally, but, to enable communication, it is required that the intended receiver has the corresponding reception slot to receive the message. For example, node A wants to send a packet to node B and uses it TX cell $[2, 0]$. In this case, node B requires a RX cell at $[2, 0]$ to receive the packet. This consistency is maintained by 6P [14], [15].

2.3.3. IPv6 over Low Power Wireless Personal Area Network

Sending of IPv6 packets over an IEEE 802.15.4 WSN is the main characteristic of 6TiSCH. But this requires 6LoWPAN as sublayer handling the difference and conflicts between IPv6 and TSCH. 6LoWPAN provides three different capabilities inside the 6TiSCH stack. One issue of sending IPv6 packets over an IEEE 802.15.4 physical layer is due to the provided Maximum Transmission Unit (MTU). IPv6 has a MTU of 1280 bytes, but the size of the effective payload of IEEE 802.15.4 is only 80 bytes which is significantly smaller than 1 MTU in IPv6. Therefore, 6LoWPAN Header Compression (6LoWPAN HC) provides compression of IPv6 and UDP headers [16]. Furthermore, 6LoWPAN is required to support the routing algorithm of 6TiSCH, RPL, in IEEE 802.15.4 packets. To encapsulate RPL artifacts in the IEEE 802.15.4 packets, they are compressed using

6LoWPAN Routing Header (6LoWPAN RH) [17].
IPv6 Neighbor Discovery (IPv6 ND) is based on multicast communication which is not usable in low-power wireless PAN due to the long sleeping periods between the active slots. Therefore, 6LoWPAN Neighbor Discovery (6LoWPAN ND) adapts the classical IPv6 ND to enable the neighbor discovery [14], [18].

3. Testbed Setup

This chapter gives an overview about the hardware and the implementation of the testbed. In Sec. 3.1 the technical details of the OpenMote B boards and Contiki NG are provided. Afterwards the implementation of the application protocol is explained. In general, the goal of the application protocol is the execution of the dissemination schedules. The implementation consists of two tasks. The first task considers the MAC layer schedule. The local schedule at each node has to support the execution of one round of the dissemination schedule per slotframe. Therefore, the local slotframes have to include the required reception and transmission cells, described in Sec. 3.2. The second task is that the node has to send the packets containing the defined process update with additional information like the enqueueing timestamp and the *sender id*. Because the slotframe is defined depending on the dissemination schedule such that for each timeslot in the schedule one cell is added, one specific cell is reserved for each broadcast message. Therefore, the application protocol has to ensure that the packets containing the intended update are enqueued just before the timeslot of the reserved cell. This procedure is described in Sec. 3.3.
The input file for generating the TSCH schedule includes the adjacency matrix describing the network topology, the dissemination schedule, and the configuration options in JavaScript Object Notation (json). For a three-node line setup the input file is shown in List. 3.1.

Listing 3.1: Example input file three-node line setup

```
{
        "adjacent_matrix":
        {
                "node1": [0,1,0],
                "node2": [1,0,1],
                "node3": [0,1,0]
        },
        "dissemination_scheduling":
        {
                "ts0": [1,1],
                "ts1": [2,1],
                "ts2": [2,2],
                "ts3": [3,3],
                "ts4": [2,3],
        },
        "redundant_tx_cells": 2,
        "buffer_cells": 2
}
```

First, the connections between the nodes are described by the adjacency matrix. The entry of node 1 defines that it has a connection to node 2 but no connection to itself and

to node 3. The entries of the dissemination schedule describe which sender node has to transmit which process update in a tuple ([*sender id*, *process id*]) for each timeslot (ts0,.., ts4). The entries *redundant_tx_cells* and *buffer_cells* are additional configuration parameters, defined in the Sec. 3.2.3. This input file is read by a code generator, implemented in Python, generating specifications for each node. In the specifications the slotframe definition, timers to enqueue origin packets, and the packet forwarding rules depending on the *process id* of received updates are included. The C based application protocol requires these specifications. This application protocol is implemented in this thesis and is an extension of the example applications of Contiki NG [7]. The generator produces an individualized version of the generic application protocol for each node by storing the generated specifications in variables and constants of the application protocol.

3.1. Testbed Hardware

The testbed uses the OpenMote B board as ICT platform and Contiki NG as operating system. Contiki NG is an operating system for IoT devices and is adapted for different ICT platforms [7]. Different low-power protocols, like IPv6 with 6LoWPAN, RPL, and CoAP are implemented in Contiki, such that the whole 6TiSCH stack is supported. Contiki NG provides a simple adaption of the whole stack, such that it is configurable for each OSI layer which communication protocol is used. In this testbed, a reduced 6TiSCH stack is used containing a simplified implementation of RPL (RPL light). The local schedule at each node should provide only cells to transmit status updates or EB frames such that transmissions triggered by other applications and protocols should be avoided. Therefore, the security features of TSCH are disabled.
The OpenMote B board, developed by OpenMote technologies, runs the 6TiSCH stack containing the implemented application protocol, described in Sec. 3.2 and 3.3 [12]. The board (shown in Fig. 3.1) is based on the wireless micro controller System on Chip CC2538 of Texas Instruments [19] and the AT86RF215 multi-band radio transceiver from Atmel [20]. The OpenMote B boards are connected to an external 2.4 GHz antenna, such that the boards operate on the 2.4 GHz frequency band. Other protocols like Bluetooth and WiFi also transmit on this frequency band such that the coexistence can result in unintended interference [21].

Figure 3.1.: OpenMote B board [12].

3.2. Slotframe Scheduling

The local schedules have to enable the execution of the dissemination schedule. Because the slotframe is defined locally, each node has to adapt its slotframe individually. The described logic of this section is performed by the code generator which calculates a static schedule for each node and stores it in the node specific application protocol.

3.2.1. Contiki NG Timeslotted Channel Hopping Schedule API

Contiki NG provides an TSCH schedule API to adapt the slotframe statically without involving 6P [7]. The API is used to delete the default slotframe, to initialize an empty one, and to add the intended links to the new slotframe afterwards. Contiki NG uses parameters like those shown in Table 3.1 to define a link.

Table 3.1.: Link definition in Contiki NG excluding implementation-specific parameters.

parameter	options
destination MAC address	*
timeslot	*
channel offset	*
link option	TX, RX, shared
link type	normal, advertising only, advertising

The channel offset and timeslot parameters are used to enable the cell definition according to TSCH (Sec. 2.3.1). Furthermore, the link option defines, whether it is a dedicated TX or RX cell, or whether it is a shared (auto) cell. The link type controls which data type can be handled in this cell. In advertising only cells the node only has the opportunity to send EB frames. Advertising cells are able to handle EB and data frames and in normal cells data frames are transmitted.

3.2.2. TX and RX Cells for Dissemination Schedules

A single slotframe should provide the TX and RX cells to perform one round of the dissemination schedule. The concept of the slotframe scheduling of the testbed is visualized in Tbl. 3.2. The dissemination schedule defines the sender node (*sender id*) and the *process id* of the status update for every timeslot. Sender nodes always transmit updates to all topological neighbors. Therefore, the application protocol sends broadcast messages. By definition, a link requires the MAC address of the receiver node. Therefore, instead of a node-specific MAC address, the broadcast address is used.
This testbed executes dissemination schedules which only define one packet transmission per timeslot in the entire network such that it is not required to use multiple channels.

Therefore, each cell gets the same channel offset for simplification (channel offset = 0). To execute the dissemination schedule, RX and TX cells have to be added with the link option normal to enable handling of data frames. The first row of the table describes normal TX cells that has to be added at the sender node in each active timeslot. Therefore, always if the *sender id* of the dissemination schedule is equal to the *node id* the node adds a TX cell to its slotframe. To enable the reception of the status update, the intended receiver nodes have to add a normal RX cell (second row in the table). In a real world setup, the update would automatically be received by the direct neighbors of the sender node only. But due to the short distance in the lab setup, all nodes are in the radio range of each other. Therefore, only if the node is in the neighborhood of the *sender id*, the node adds a RX link in this timeslot (condition a.) in Tbl. 3.2). Furthermore, a node does not have to receive its own status update (condition b.)). Condition c.) ensures that each node only receives one update per process in a slotframe. Therefore, it has to be checked, whether the node already has a RX cell to receive the status update for this *process id*.
To maintain synchronization, sending and receiving of EB frames are used such that advertising only cells are required. In a TSCH network a node gets its initial synchronization by an EB and afterwards tries to maintain the synchronization to a time source neighbor (Sec. 2.3). In Contiki NG the configuration *tsch_autoselect_time_source* determines that a node selects its time source depending on the amount of received EBs. In this testbed the coordinator node has one TX advertising only cell and all other nodes have the corresponding RX cells. Because the other cells have the link option normal, all nodes only receive EB frames from the coordinator, such that this is the time source node. For this single timeslot the topology definition is ignored and the frame is sent as in a star topology. Following the minimal 6TiSCH configuration [14], timeslot zero is reserved for the transmissions of EBs.

Table 3.2.: Slotframe definition depending on dissemination schedule.

link option	link type	cell	condition
normal	TX	[* ,0]	$node\ id = sender\ id$
normal	RX	[* ,0]	a.) $sender\ id \in N(node\ id)$ b.) $node\ id \neq process\ id$ c.) *process id* not already received by another sender
advertising only	TX	[0,0]	coordinator
advertising only	RX	[0,0]	not coordinator

3.2.3. Additional Slotframe Configurations

Besides the link definition, also configurations, defined in the input json file, regarding the general performance of the testbed are considered. There are two different reasons resulting in a packet loss in this testbed.

The first one is due to interference caused by coexistence with other protocols. The usage of broadcast messages implies that acknowledgments are not required such that the sender does not know whether a transmission was successful. To improve the reliability without retransmissions, redundant TX cells are configured, such that the sender has multiple possibilities to send an update. On the application layer these redundant TX cells are used for enqueuing multiple packets at the same point in time. For example, for two redundant TX cells, the node directly enqueues two packets containing the same status update and timestamp.
The second reason for a packet loss is due to enqueuing failures. The logic to determine the local schedule of each node (Sec. 3.2.2) adds for each dissemination step one cell, or in case of using redundant TX cells, the amount of cells considering the configuration parameter. In case a packet is enqueued too late, such that the packet cannot be send in the intended cell, an already reserved cell would be used. This would result in higher AoI values for other processes. To avoid this, a node flushes its queue before it enqueues a new packet resulting in additional packet loss. To reduce the probability of a packet loss from enqueuing failures, the configuration parameter called *buffer cells* is used. This parameter describes the amount of inactive cells between active cells in the slotframe and therefore prolongs the time interval to enqueue a packet on the application layer.

3.2.4. Example Slotframe Schedule

An example schedule for a three-node line setup for the dissemination schedule of Sec. 2.1.2 is shown in Fig. 3.2. The number of redundant TX cells and buffer cells are configured as two, like in List. 3.1. In this scenario, node 1 is the coordinator. Therefore, in the slotframe of node 1 there is an advertising only TX cell at timeslot 0. The other nodes have the corresponding RX cell.
The next link is added at timeslot three due to the configuration of two buffer cells between the active cells. The dissemination schedule determines that node 1 first sends its origin packet, a packet containing an update of its own process. Therefore, in the slotframe of node 1, two TX cells are added to fulfill the configuration of two redundant TX cells. Furthermore, node 2 is a neighbor of node 1, such that it adds RX cells to receive the update of process 1 in this slot. Then node 2 forwards the update of process 1 to node 3. Therefore, node 2 has two normal TX cells and node 3 the corresponding RX cells. The second part of the scheduling describes the dissemination of the status update of process 2. At last node 3 sends it origin packet and node 2 forwards the update to node 1.
To enable that the TSCH channel hopping algorithm cycles through all frequencies of the *macHoppingSequenceList*, the length of the slotframe has to be a prime number [22]. Furthermore, considering this testbed, the length of a slotframe depends on the number of nodes, the topology, the number of buffer cells, and the number of redundant TX cells. Therefore, the length of a slotframe is rounded up to the next prime number after adding all required cells. Considering the example in Fig. 3.2 the last buffer cell

has the slot offset 22. Due to zero indexing of the slot offset, the corresponding slotframe length is 23 such that rounding up is not required.

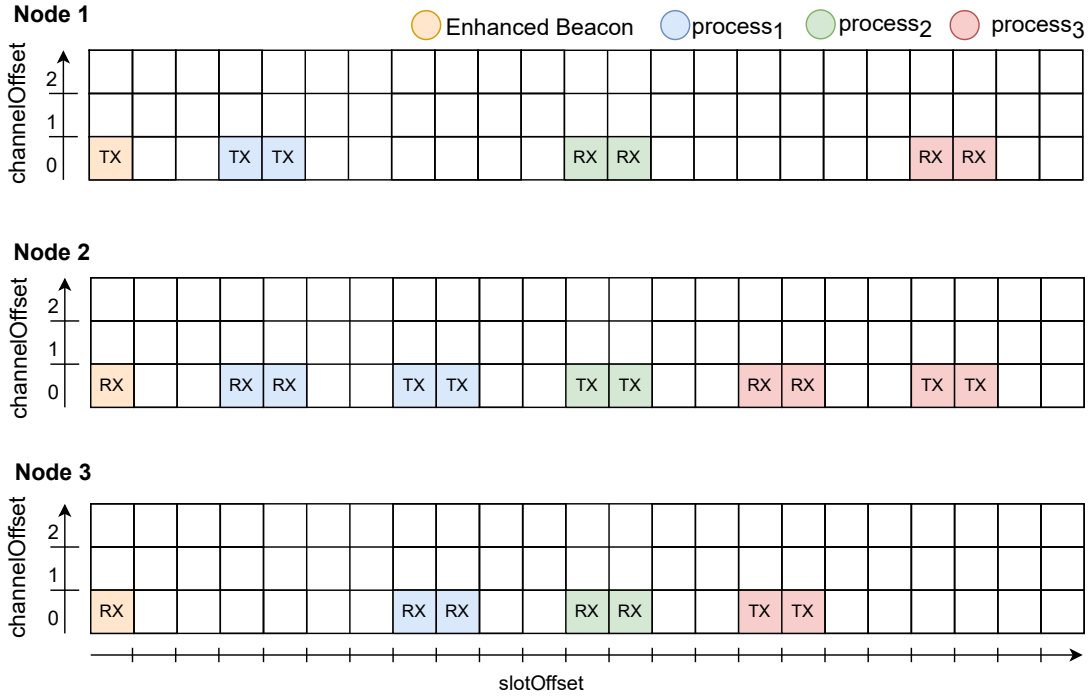

Figure 3.2.: Example slotframes for a three-node line topology with two redundant TX slots and two buffer cells.

3.3. Dissemination of Status Updates

The second task of the application protocol is to enqueue the packet with the correct process information just before the beginning of the intended TX cell.

3.3.1. Forwarding and Enqueuing

Forwarding packets are directly enqueued after the reception of the update, since the dissemination schedules always distribute an update to all nodes before the next process information is considered. Therefore, if a forwarding node receives an update, the next TX cell is reserved for this update. The logic to determine, whether a node has to forward an update, is encapsulated in the code generator. The generator analyzes the dissemination schedule to store an array indicating forwarding tasks in the application

protocol of each node.

The enqueuing of packets containing the own process updates (origin packets) has to be triggered by each node itself. One possibility would be to use the dissemination schedule to get the information which process update should be received immediately before the enqueuing of an origin packet. The disadvantage of this approach is that in case of a node failure the information exchange is stopped completely. Therefore, in this thesis a periodic timer is used to trigger the enqueuing.
Contiki NG provides several timers with different precision. The one with the highest precision is the *rtimer*, but in Contiki Stack only one instance of an rtimer is usable. Because the TSCH implementation requires a rtimer instance, the application layer protocol uses an *etimer* to trigger the origin packets. Etimer are defined by unit ticks. Ticks depend on the interrupts on the hardware. On OpenMote B boards 128 interrupts are triggered per second, such that one tick is equivalent to 7.8125 ms. Because the duration of one slotframe is a multiple of a timeslot length (10 ms), it is not sufficient to set a constant periodic timer in ticks. Therefore, the timer has to be corrected to avoid that the timer expires at varying timeslots in the slotframe.
Furthermore, it has to be ensured that the origin packet is enqueued immediately before the intended TX cell to minimize the AoI. The code generator determines for each node a lookup table, defining the time in ticks from each timeslot to the slot offset when the origin packet should be enqueued. When a valid update is received the timer is set to the corresponding number of ticks of the lookup table. Additionally, after enqueuing an origin packet the timer is set to the number of ticks that describes the length of a slotframe as precise as possible. This is required to ensure that the origin packet is also sent if no other packet is received in a slotframe.

To calculate the timer values in the lookup table, the amount of slots from the current slot to the intended enqueuing slot is computed and afterwards this time is converted to ticks. The timers are corrected by the UDP callback function, which is triggered by each packet reception. Therefore, it is assumed that the remaining time in the current reception slot is very short. The timer should enable that the packet is approximately enqueued in the first buffer cell, or in case no buffer cells are configured, before the first TX cell.
For example, the first TX cell is at slot ten and the current slot is two. In case three buffer cells are configured, the packet should be enqueued approximately at the beginning of the seventh slot. Therefore, the timer has to be configured to wait for four slots (40 ms) to sleep from the end of the second slot until the end of the sixth slot. Dividing this time by the length of a tick (7.8125 ms) and rounding up to the next integer, the timer has to be set to six ticks. The result is rounded up to ensure, that the packet is enqueued at the end of the sixth slot or in the seventh slot. If the packet is enqueued at the beginning of the sixth slot and in this slot another transmission is performed, the enqueuing could result in deleting the previous packet because enqueuing always implies flushing the queue.

If no buffer cells are used, another rounding procedure is used. Because it is possible that only one TX cell is configured per update, it has to be ensured that the packet is enqueued before the first TX cell such that all TX cells are usable. Considering the example (current slot 2, TX slot of origin packet 10), the packet should be enqueued in the ninth slot. Therefore, the timer has to be equal to seven slots to wait from the end of the second slot until the end of the ninth slot. In this case converting the time to ticks requires rounding down instead of rounding up, to ensure, that the packet is enqueued before the beginning of the tenth slot.

3.3.2. Application Data and Logging

The application protocol produces a log file that is used to store the required data to determine the performance metrics such as PDR and AoI. To support the measurement of PDR, application sequence numbers are included in application messages. In a slotframe, each node enqueues only one update of its process such that the sequence number is increased once per slotframe. But due to slotframes using redundant TX cells, for a single update multiple packets are enqueued containing the same timestamp and sequence number.
To reconstruct the AoI during the whole measurement, it is sufficient to know the initial and peak AoI values. The initial values represent the age directly after receiving the packet and the peak age is the age just before applying a newly received update. TSCH is time synchronized such that the nodes have a shared time source. In Contiki NG the synchronized time is the network uptime defined in ticks [7]. Therefore, part of the application payload is the enqueuing timestamp with respect to the network uptime.
In List. 3.2 an example log file is provided. If a new packet is received, the node calls the UDP callback function, which produces two log file messages. The first one (RX_OLD) represents the peak value of the previous update. In the log message the current timestamp is shown first. Furthermore, the *process id* and sequence number of the packet is provided. The fourth parameter documents the enqueuing timestamp and the last one the peak value of the AoI. The second log message (RX_NEW) additionally provides the *sender id* of the packet. For debugging it is shown whether it is a forwarding or an origin packet (packet_type) and whether it is the first enqueued packet with this sequence number or whether it is one of the redundant packets. In this log message aoi_tsch_ticks represents the initial AoI value. All information required for the log file are transmitted in the application payload.

Both types of log messages are only added to the log file if the received update improves the AoI of the system. In case an update with the same sequence number is received multiple times, the later ones cannot contain newer information because the node of the *process id* produces an update only once per slotframe. If a node receives an update

with a sequence number already received, the packet took a longer route through the network or is one of the redundant packets, such that this packet can be dropped.

Listing 3.2: Example log file five-node line setup

```
1 [INFO: App       ] RX_OLD {timestamp: 3487, process_id: 2, seq_nr: 6,
      timestamp_tsch_ticks: 3357, aoi_tsch_ticks: 130}
2 [INFO: App       ] RX_NEW {timestamp: 3487, sender_id: 2, process_id: 2,
      seq_nr: 7, timestamp_tsch_ticks: 3481, aoi_tsch_ticks: 6, packet_type:
      0, redundancy_nr: 0}
3 [INFO: App       ] RX_OLD {timestamp: 3543, process_id: 4, seq_nr: 7,
      timestamp_tsch_ticks: 3401, aoi_tsch_ticks: 142}
4 [INFO: App       ] RX_NEW {timestamp: 3543, sender_id: 2, process_id: 4,
      seq_nr: 8, timestamp_tsch_ticks: 3524, aoi_tsch_ticks: 19, packet_type:
      1, redundancy_nr: 0}
5 [INFO: App       ] RX_OLD {timestamp: 3569, process_id: 5, seq_nr: 8,
      timestamp_tsch_ticks: 3419, aoi_tsch_ticks: 150}
6 [INFO: App       ] RX_NEW {timestamp: 3569, sender_id: 2, process_id: 5,
      seq_nr: 9, timestamp_tsch_ticks: 3544, aoi_tsch_ticks: 25, packet_type:
      1, redundancy_nr: 0}
7 [INFO: App       ] RX_OLD {timestamp: 3583, process_id: 3, seq_nr: 7,
      timestamp_tsch_ticks: 3445, aoi_tsch_ticks: 138}
8 [INFO: App       ] RX_NEW {timestamp: 3583, sender_id: 2, process_id: 3,
      seq_nr: 8, timestamp_tsch_ticks: 3570, aoi_tsch_ticks: 13, packet_type:
      1, redundancy_nr: 0}
```

3.3.3. Testbed Overview

In Fig 3.3, the flow chart for the application protocol of each node in the testbed is shown. In the starting phase it is checked whether all nodes joined the network. Afterwards each node adapts its local schedule. In the sending phase, first the node sets the timer for the origin packet and then waits until either the timer expires or a packet is received. If the timer is expired, the node enqueues the packet(s) including the status update and additional parameters like the application sequence number in the payload. Before a new update is enqueued the node increases the sequence number and flushes the data output queue. Contiki NG stores the EB frames in a separate queue, such that it is possible to delete the data frames only. Afterwards, it is checked whether the configured amount of status updates, that has to be sent in this measurement round, is reached. In this case, the process is finished. Otherwise, the timer of the origin packet is reset and the node waits again for an event.

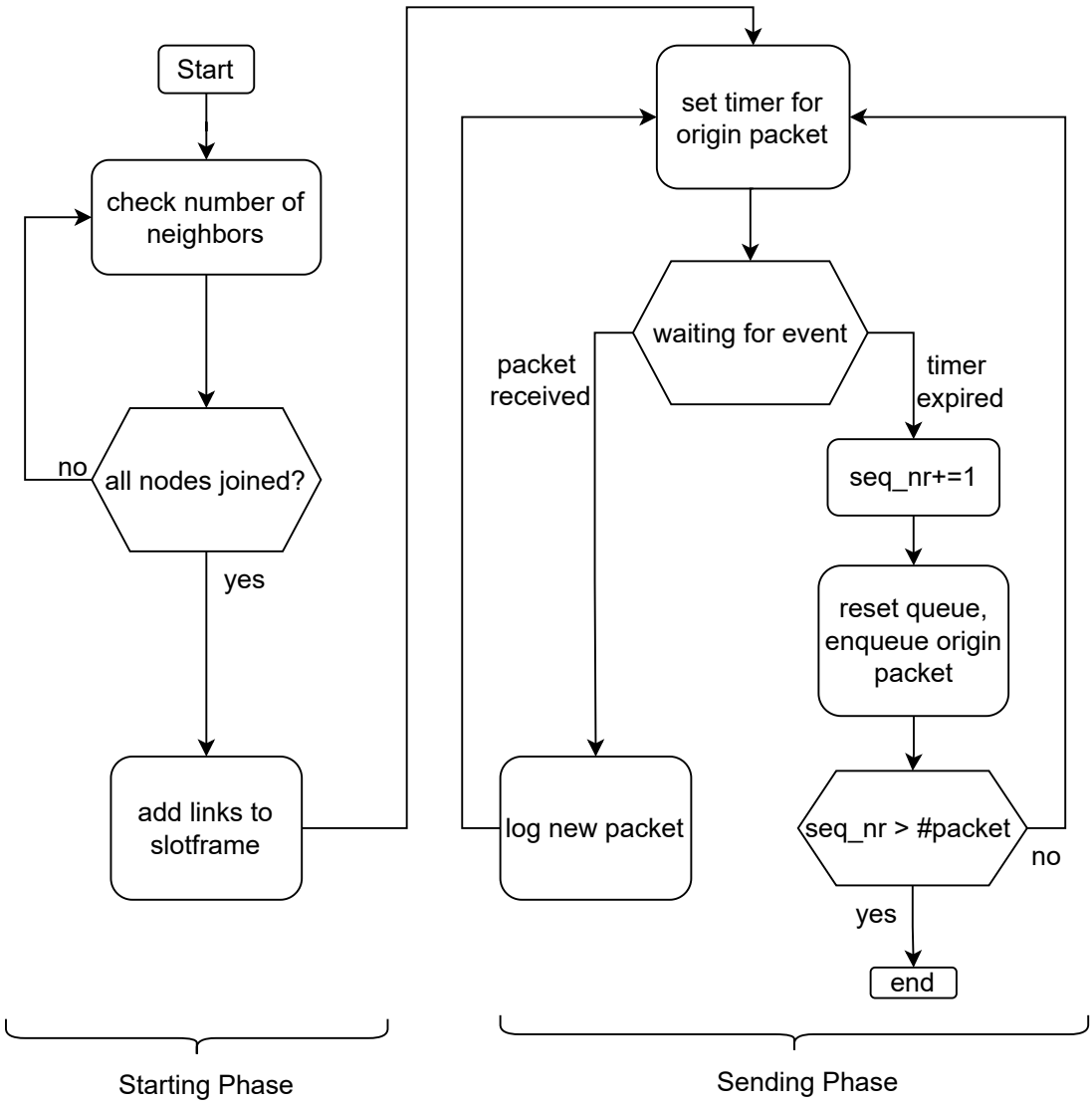

Figure 3.3.: Flow chart of the whole application protocol.

If a new packet is received, the UDP callback function is called which also includes the reset of the timer considering the timer lookup table. The flow chart of the UDP callback function is shown in Fig. 3.4. First the *process id* and the sequence number are read from the application payload. If an update containing this sequence number has already been received, the callback is finished. Otherwise, it is checked whether the packet has to be forwarded. If forwarding is required, the timestamp, stored in the application payload, is not updated and the original timestamp is preserved. The queue is flushed

again and the forwarding packet is enqueued. Afterwards, the timer for the origin packet is corrected and logging is performed.

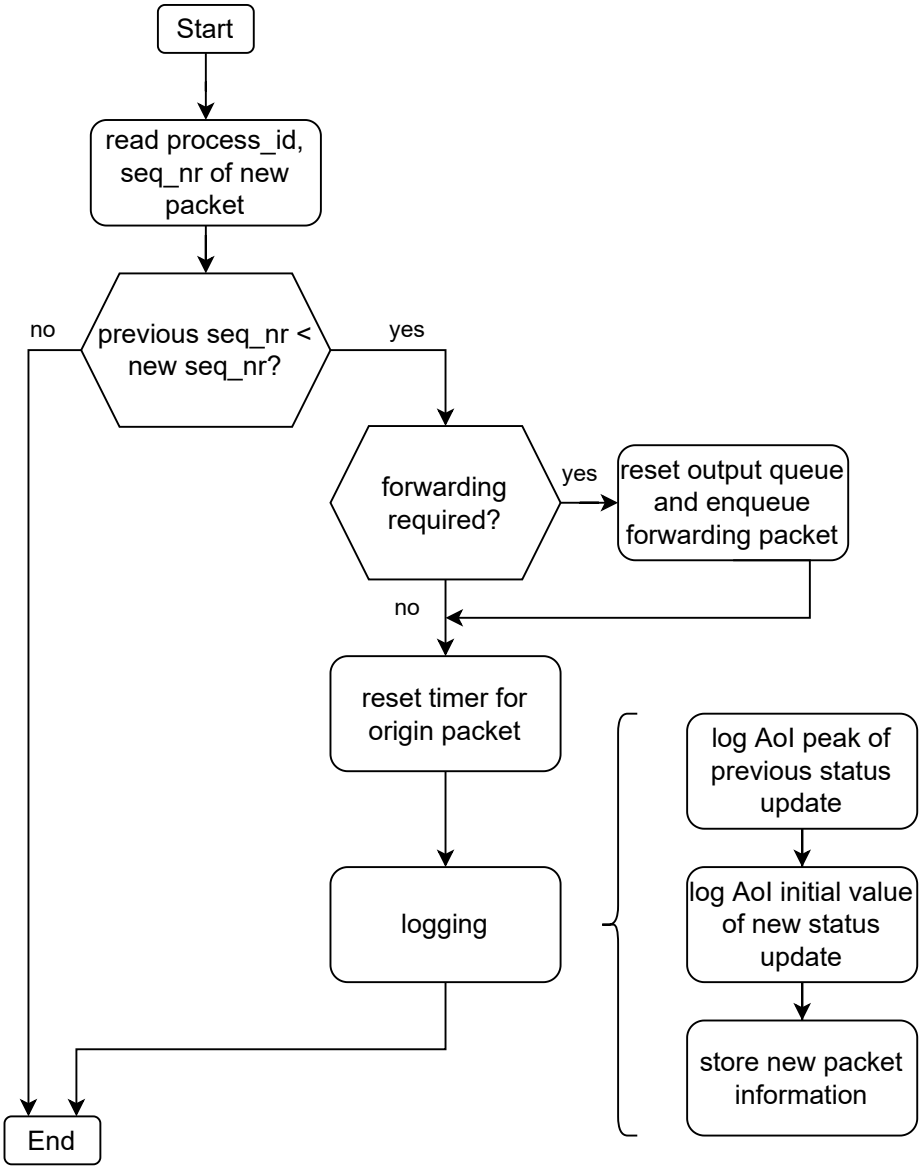

Figure 3.4.: Flow chart of the UDP callback.

4. Measurement Methodology

This chapter describes the setups that are measured and evaluated in this thesis. Afterwards the expectations regarding the measurement results are explained.

4.1. Measurement Setups

An overview about the different measurement setups considering the network topology, slotframe configuration, and measurement metrics is given in Tbl. 4.1. Each setup focuses on a different performance criterion. The configuration parameters, the adjacency matrix describing the topology, and the dissemination schedule are defined in a setup specific configuration file which is used as input for the code generator. The generator individually calculates for each network node the slotframe with the required links and determines a valid slotframe size. Furthermore, the generator provides the individual timer lookup table and determines which processes updates have to be forwarded by the node. The individual generated code is flashed to the corresponding node. This allows to use the testbed for arbitrary topologies, schedules, and configurations.
The first measurement is used to validate that the channel hopping is working and that the transmissions are only performed in intended cells. For validation the Contiki NG TSCH log mode per slot is used [7]. The mode provides detailed information regarding each slot. This information is simplified to only log the channel offset, slot offset, and the physical channel. The other setups analyze the performance regarding PDR and AoI. To evaluate AoI in detail, AoI is measured in ticks (Sec. 3.3.1) and the metrics described in Sec. 2.2 are determined. Except for setup four, the analyzed network topology is a five-node line network. Furthermore, node 1 is always the coordinator of the PAN.
The goal of the second setup is the optimization of the slotframe configuration to minimize AoI. Therefore, the slotframe has to support sending a packet as fast as possible to minimize queuing delays. The usage of buffer cells results in a larger enqueuing interval reducing the probability of enqueuing failures. But it also increases the queuing delay, such the second setup does not use buffer cells. The configurations, that are measured in this setup, differ by the amount of TX cells per update, to analyze which amount of TX cells is enough to compensate packet loss due to interference and enqueuing failures.
The third setup is focused on PDR and energy efficiency. It is expected, that the removal of buffer cells results in additional enqueuing failures such that enqueuing packets too late results in unused TX cells. Therefore, this setup compares configurations with different amounts of buffer cells, to reduce the packet loss due to enqueuing failures. Furthermore, the different amounts of buffer cells are measured with one and two TX cells per update

to minimize the packet loss due to interference.
In the fourth setup, a disrupter node is included in the network resulting in additional interference. This node sends a packet in every even slot. Because the evaluation of the results of setup three should point out, which amount of buffer cells is sufficient to avoid most enqueuing failures, this configuration can be used in setup four. Furthermore, an amount of TX cells between one and four is tested. Analyzing up to four TX cells simulates the standard retransmission procedure of TSCH, which retransmits a packet at most three times.
The fifth setup compares the AoI in a line setup to a circle setup. This setup is mainly used to validate that the testbed is able to handle different network topologies. The optimized configuration of setup two and three are used in this setup.
In the sixth setup a long term PDR measurement is performed to analyze whether a long term execution of the testbed result in timing issues.

One measurement round consists of five minutes. Because the length of the slotframe depends on the configuration of the buffer cells and redundant TX cells, the amount of packets per round varies. Setup two and three are used to find optimal configurations, that are also used in the other setups. Therefore, these setups are measured five times and the 95% confidence interval is computed. To evaluate the AoI of all measurement rounds, for these setups the following evaluation procedure is used. For each round Δ_{avg} is computed according the definition in Sec. 2.2. Afterwards, the average and the corresponding confidence interval of the five Δ_{avg} values are determined. The Δ_{peak} shown in the measurement results corresponds to the maximum of the Δ_{peak} per measurement round. To analyze the PDR, the PDR of the whole network is computed for each single round. The average PDR of the measurement results is the average of all network PDR values with its confidence intervals.

Table 4.1.: Overview measurement setups considering configuration, topology, and metrics.

setup	focus	topology	TX cells	buffer cells	interference	metrics
#1	Channel Hopping	line	1	4	no	*
#2	AoI	line	1,2,3,4	0	no	AoI, PDR
#3	PDR, Energy	line	1,2	1,2,3,4	no	AoI, PDR
#4	Interference	line	1,2,3,4	optimum of #3	yes	AoI, PDR
#5	Topology	line, circle	optimum of #2,#3		no	AoI, PDR
#6	PDR over 3h	line	optimum of #3		no	AoI, PDR

4.2. Assumptions concerning Measurement Results

In the first setup it is expected that the usage of the physical channel depends on the ASN such that a transmission in the same timeslot is performed on different channels in subsequent slotframes due to the channel hopping algorithm.
In the second setup, increasing the amount of TX cells per update should increase the PDR because an additional TX cell can compensate a packet loss due to interference caused by the coexistence with other protocols. Also in case the packet is enqueued too late for the first TX cell, it can be sent in one of the redundant ones. Adding redundant TX cells could result in different effects on the AoI. In general, the AoI depends on the parameters dissemination delay, update rate and the reliability. The dissemination delay is the delay to propagate an update to all nodes. Hence, the dissemination delay depends on the hop distance, such that the topology has an impact on the AoI. The update rate is determined by the length of one slotframe. A longer slotframe results in a smaller update rate such that, theoretically, additional TX cells result in an increased AoI. But a small PDR due to interference and enqueuing failures results in an AoI peak and increases the average AoI. Therefore, it is expected that an increased amount of TX cells first results in a smaller AoI, because the reliability is improved until enough TX cells are added to compensate the packet loss. Adding then additional TX cells should result in a higher AoI due to a larger slotframe size.
In the third setup, the PDR is improved by a higher amount of TX cells, similar to setup two, but also by additional buffer cells. The buffer cells should reduce the probability of enqueuing failures. Therefore, it can be analyzed how many buffer cells are required to avoid most enqueuing failures such that redundant TX cells are only required to compensate packet loss from interference.
In the setup with additional interference (setup four), a disrupter node transmits a packet in every even timeslot. In a configuration with one TX cell per update this should result in a collision every second timeslot such that the PDR is reduced to 50% approximately. Adding redundant TX cells to the configuration should improve the PDR. Similar to other setups, it is expected to find an optimized configuration which includes enough TX cells to avoid the packet loss and which is small enough to limit the slotframe size.
Comparing the line and circle topology, the probability of enqueuing failures should be approximately identical, such that the PDR values are expected to be similar. The average AoI of the circle network should be smaller compared to the line network, because the network diameter is smaller compared to the line network.

5. Analysis

This chapter analyzes the measurement results regarding AoI and PDR. Each setup is explained in a single section. Afterwards, effects due to the testbed concept on the measurement results are explained. In the last section of this chapter, it is discussed which aspects have to be considered, if the concept of the testbed would be transferred to a realistic network setting outside the lab.

5.1. Setup 1 - Proof of Channel Hopping

In List. 5.1 a part of the log file at node 1 is shown. In this setup a line network is used with one TX cell per update. Analyzing the used timeslots, it is visible that the node always transmits at timeslot zero and 20 (link option tx). Because node 1 is the coordinator, it sends an EB in timeslot zero. At timeslot 20 the node transmits its origin packet. The other active timeslots are used to receive the updates of the other nodes. In Sec. 3.2 it is explained that the channel offset is always zero.

Listing 5.1: Log File of used timeslot, channelOffset and physical channel

```
1  [INFO: TSCH-LOG  ] {timeslot:  0, channel_offset  0, physical_channel 20,
       link_option: tx}
2  [INFO: TSCH-LOG  ] {timeslot:  5, channel_offset  0, physical_channel 15,
       link_option: rx}
3  [INFO: TSCH-LOG  ] {timeslot: 20, channel_offset  0, physical_channel 20,
       link_option: tx}
4  [INFO: TSCH-LOG  ] {timeslot: 50, channel_offset  0, physical_channel 25,
       link_option: rx}
5  [INFO: TSCH-LOG  ] {timeslot: 70, channel_offset  0, physical_channel 25,
       link_option: rx}
6  [INFO: TSCH-LOG  ] {timeslot: 80, channel_offset  0, physical_channel 20,
       link_option: rx}
7  [INFO: TSCH-LOG  ] {timeslot:  0, channel_offset  0, physical_channel 15,
       link_option: tx}
8  [INFO: TSCH-LOG  ] {timeslot:  5, channel_offset  0, physical_channel 25,
       link_option: rx}
9  [INFO: TSCH-LOG  ] {timeslot: 20, channel_offset  0, physical_channel 15,
       link_option: tx}
10 [INFO: TSCH-LOG  ] {timeslot: 50, channel_offset  0, physical_channel 26,
       link_option: rx}
11 [INFO: TSCH-LOG  ] {timeslot: 70, channel_offset  0, physical_channel 26,
       link_option: rx}
12 [INFO: TSCH-LOG  ] {timeslot: 80, channel_offset  0, physical_channel 15,
       link_option: rx}
```

Analyzing the used physical channels, it is visible, that in subsequent slotframes different physical channels are used in the same timeslot. This fulfills the expectations and proves that the nodes use channel hopping. In Contiki NG the default *macHoppingSequenceList* is $[15, 25, 26, 20]$. The log file shows that the next channel of the *macHoppingSequenceList* is always used in subsequent slotframes. For example, in the first slotframe channel 15 is used in timeslot five (line 2) and in the next slotframe (line 8) channel 25. This can be explained considering the definition of the channel hopping, Eq. 2.9. Due to the channel offset of zero, the Eq. 2.9 can be simplified to Eq. 5.1.

$$CH = macHoppingSequenceList[ASN \mod macHoppingSequenceLength] \quad (5.1)$$

In this configuration the slotframe size is 97 slots and the *macHoppingSequenceLength* is equal to four. Comparing two subsequent slotframes, the ASN is always increased by the size of a slotframe. The increase of the ASN by 97 slots, results in an increase of channel index by one ($97 \mod 4 = 1$).
The channel histogram (Fig. 5.1) shows how often each physical channel is used in the whole network. As expected, the channels are used quite uniformly.

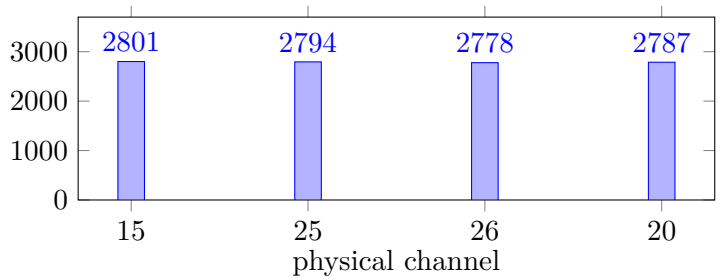

Figure 5.1.: Histogram of physical channels in a five-node line setup.

5.2. Setup 2 - Age of Information Analysis

An overview of the results of setup two is shown in Tbl. 5.1. The slotframe configuration involving one TX cell per update results in a high Δ_{avg} of more than 400 ticks (3.125s). Furthermore, the Δ_{peak} of 6941 ticks is very high, which can be explained by the mean PDR of 56.647%. Adding an additional TX cell results in a significantly improved PDR (99.931%). Also the Δ_{avg} is reduced to 27.91 ticks (218.047 ms), although the slotframe length is increased by 18 slots. The reduced packet loss due to enqueuing failures and interference outweighs the smaller update rate.
The other configurations involving three or four TX cells per update, increases the PDR to 100% but the AoI is also higher due to the smaller update rate. Therefore, with respect to the average AoI the configuration with two TX cells per update performs best. As expected in Sec. 4.2 a configuration reducing packet loss, avoiding AoI peaks, and keeping the slotframe size as small as possible is found.

Table 5.1.: Overview measurement results setup 2.

TX cells	buffer cells	Δ_{avg} [ticks]	$\Delta_{peak}[ticks]$	avg. PDR [%]	slotframe length [slots]
1	0	424.108 ± 202.642	6941	56.647 ± 18.971	19
2		27.910 ± 5.112	104	99.931 ± 0.03	37
3		39.573 ± 5.663	143	99.972 ± 0.02	53
4		52.494 ± 0.013	111	100 ± 0	71

To analyze the reason of the packet loss in the first configuration in detail, Tbl. 5.2 provides the PDR at every node for each process. It is shown, that the origin packets of node 3 and node 4 get lost at these nodes, because the direct neighbors do not receive the corresponding updates. Therefore, the low average PDR in the network is explained by enqueuing failures and not by a forwarding or interference problem.

Table 5.2.: PDR at each node per process in setup 1 using one TX cell, evaluation of a single measurement round.

node	PDR for each process_id PDR [%]				
	process 1	process 2	process 3	process 4	process 5
node 1		82.331	17.099	8.233	97.403
node 2	68.461		17.163	8.233	97.657
node 3	67.764	87.967		8.296	98.1
node 4	67.384	87.587	19.886		99.557
node 5	67.194	87.397	19.696	26.029	

Analyzing the dissemination schedule, included in input file List. A.1, it can be seen that at node 3 and node 4 the number of timeslots to enqueue origin packets is significantly low. The number of slots to enqueue the own update depends on the slot, when the last forwarding task before the enqueuing is performed. In Tbl. 5.3 the amount of inactive slots usable to enqueue the origin packet is shown. The amount of slots is very low at node 3 and node 4, such that this probably increases the risk of enqueuing failures.

Table 5.3.: Amount of inactive slots to enqueue the origin packet at each node.

node	number of inactive slots [slots]
1	18
2	3
3	1
4	0
5	18

5.2.1. Age of Information Analysis for two redundant TX cells

This section analyzes the AoI of the second setup considering a single configuration in more detail. For this, the configuration with two redundant TX cells is used, because it optimizes the average AoI regarding the measurement results of the second setup. The AoI graphs, the average AoI at each node, and the initial and peak AoI values are evaluated.

AoI graphs

In Fig. 5.2 the AoI graphs at node 1 are shown, describing the age of the status information with respect to the network uptime in ticks. It is visible that node 1 receives periodic updates of each process. Furthermore, the typical different levels of peak and initial AoI values are visible due to the different number of hops from node 1 to the other network nodes such that the peak AoI values of process 5 are the largest. This indicates that the packets are forwarded successfully following the dissemination schedule.
Comparing the AoI graph of measurement results to the theoretical ones from Sec. 2.1.2, it is visible, that the period of the measured graph is larger compared to the theoretical period. Theoretically the dissemination schedule requires 17 timeslots to distribute all updates. In the analyzed measurement configuration, a slotframe size of 37 slots is used. The reason for the larger period is the dependency of the slotframe size on the amount of redundant TX cells and the EB cell. In addition, the requirement of TSCH that the slotframe size has to be a prime number could increase the slotframe size. The AoI curves at the other nodes are shown in the Appendix (Fig. A.1, A.2, A.3, A.4).
In Fig. 5.3 the instantaneous average and peak AoI values are shown. The instantaneous peak is approximately between 40 and 60 ticks and the instantaneous average between 25 and 35 ticks. In the considered time interval no unexpected peak is visible in both curves, because no packet gets lost. Otherwise, there would be a peak at least in one of the AoI graphs, resulting in a peak of the instantaneous average and peak AoI.

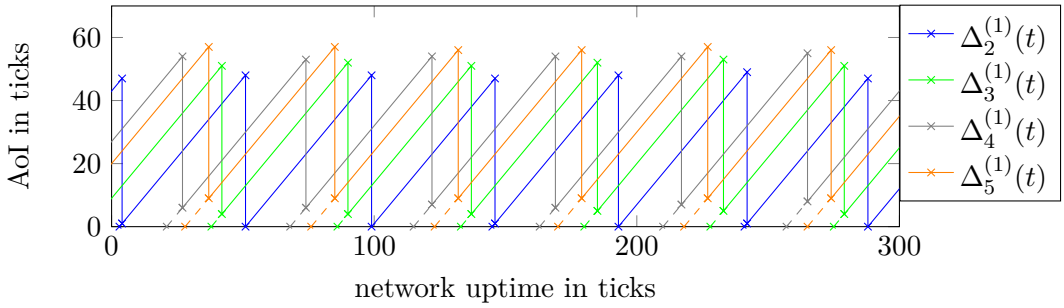

Figure 5.2.: AoI at node 1 in setup 2 using two TX cells.

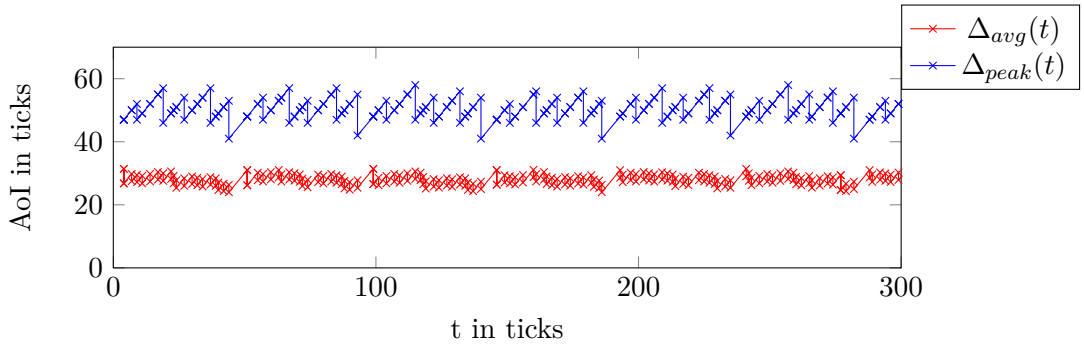

Figure 5.3.: Instantaneous peak and average AoI in setup 2 using two TX cells.

Average AoI at each node

In Tbl. 5.4 the Δ_{avg} at every node and the average hop distance from this node to all other network participants are represented. A higher hop distance implies a higher forwarding effort to receive the updates resulting in a higher AoI. Therefore, node 3 has the smallest Δ_{avg} with 26.303 ticks and the nodes with the highest average distances, node 1 and node 5, have the largest Δ_{avg} values in the network. The reason, that the AoI at node 5 (29.482 ticks) is higher compared to the AoI at node 1 (28.265 ticks), could be that the update of process 3 is first forwarded by node 2 to node 1 and afterwards by node 4 to node 5, such that the AoI at node 5 is higher for this specific process.

Table 5.4.: Average AoI at each node in setup 2 using two TX cells.

$node_{id}$	Δ_{avg} [ticks]	avg. distance [hops]
1	28.265 ± 0.039	2.5
2	26.824 ± 0.03	2.25
3	26.303 ± 0.042	1.5
4	27.036 ± 0.024	2.25
5	29.482 ± 0.049	2.5

Histograms of initial and peak age

The effect of different hop distances is also shown in the histograms of the initial AoI values. In Fig. 5.4 the histogram at node 1 of the first measurement round is provided.

Packets of the process 2 result in an initial AoI value between -1 and 3 ticks. The negative value could be a result of a network time synchronization misalignment between node 1 and node 2. As explained before, in configurations without buffer cells the packet is enqueued just before the first TX cell. In case the packet is received in the second TX cell, the initial AoI is slightly higher than two timeslots, which is equivalent to three ticks ($3 * 7.8125\ ms = 23.4375\ ms > 20\ ms$).

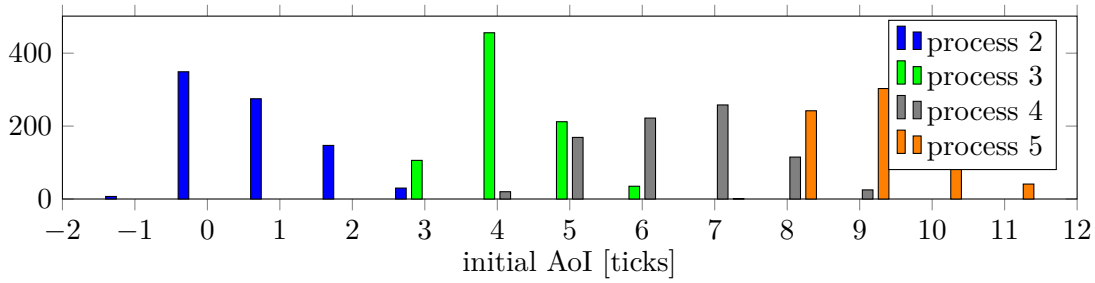

Figure 5.4.: Histogram of initial AoI at node 1 per process in setup 2 using two TX cells, evaluation of a single measurement round.

It is visible that the different hop distances from node 1 to the other nodes result in a grouping with respect to the *process id*. The different groups intersect. For example, there are update of process 2 and of process 3 that have the same initial AoI. If an update of process 2 is successfully sent in the second TX cell, it should have an initial AoI of approximately two timeslots. An update of process 3 should have an initial AoI of three timeslots, if it is received in the first RX cell. Furthermore, the accuracy of the AoI measurement is limited by the precision of the unit ticks and the quality of time synchronization.
Comparing the histogram at node 1 to the histogram at node 2 (Fig. 5.5), it is shown that the initial AoI values of process 1 and process 3 are in the same interval due to the same distance of node 1 and node 3 to node 2. Because the maximum distance at node 2 is smaller, there are no initial AoI values larger than nine ticks. This fits to the analysis of Tbl. 5.4 showing the dependency of the average hop distance and the average AoI.

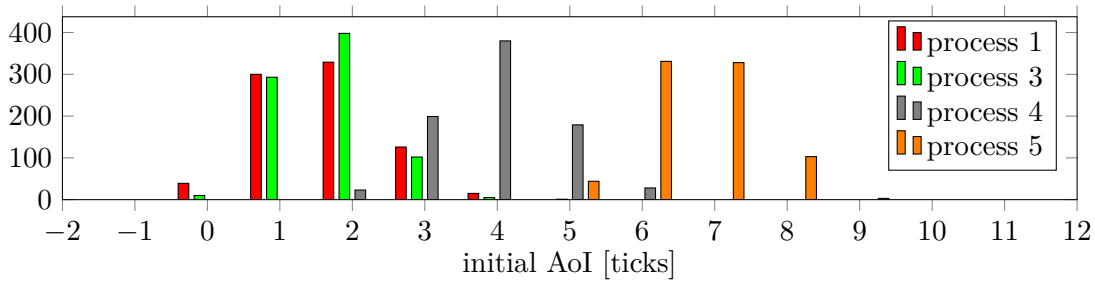

Figure 5.5.: Histogram of initial AoI at node 2 per process in setup 2 using two TX cells, evaluation of a single measurement round.

Although the average hop distances are identical, the histograms of the initial AoI values at node 5 (Fig. 5.6) and node 1 (Fig. 5.4) show a different grouping. Node 2 has a higher hop distance to node 5 than node 3, but the AoI values of their updates are in a similar region (Fig. 5.6). This again shows the influence of the dissemination algorithm, that defines, that the update of node 3 is first forwarded to node 1 and afterwards to node 5.

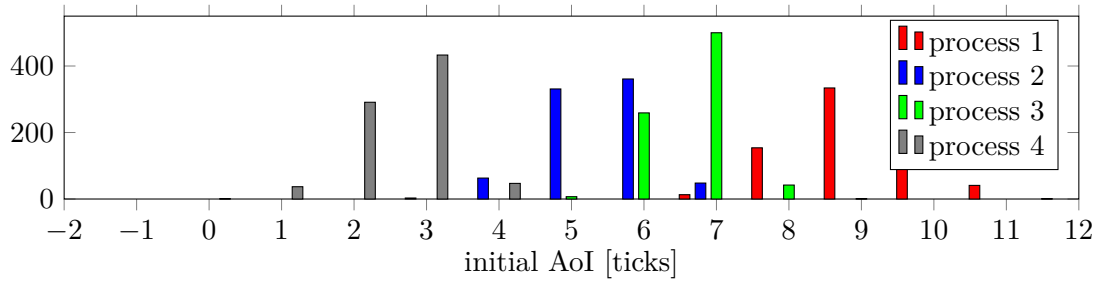

Figure 5.6.: Histogram of initial AoI at node 5 per process in setup 2 using two TX cells, evaluation of a single measurement round.

The histograms of the initial AoI values at the other nodes are shown in the Appendix Fig. A.5, A.6.

In the last histogram (Fig. 5.7), the peak AoI at node 1 is demonstrated. The same grouping as in the histogram of the initial AoI values is shown in the interval of 45 to 60 ticks. The data range can be explained by the update rate. The update rate controls when the next update replaces the old one. In this schedule the slotframe length is 37 slots, equivalent to 47 ticks. Considering the negative AoI values and synchronization effects, this explains the difference of 45 ticks between initial AoI and peak AoI values. In the interval larger than 60 ticks there are only a few, rarely visible, values indicating a packet loss and therefore a peak in the AoI.

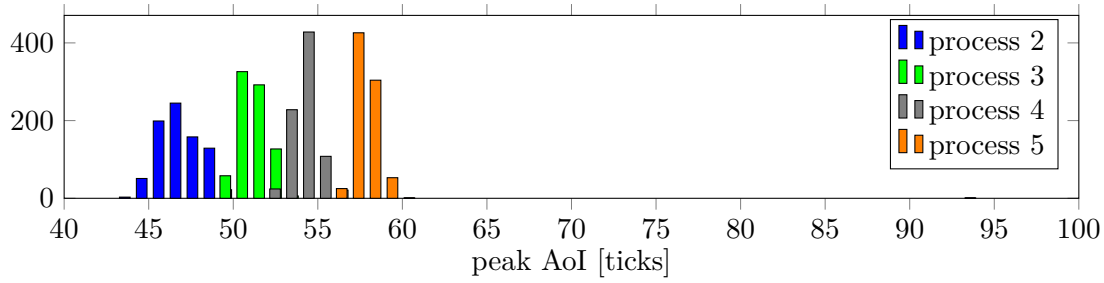

Figure 5.7.: Histogram of peak AoI at node 1 per process in setup 2 using two TX cells, evaluation of a single measurement round.

5.3. Setup 3 - Packet Delivery Ratio Analysis

The goal of this setup is to maximize the PDR with a reduced number of redundant TX cells to improve energy efficiency. Therefore, contrary to setup two, buffer cells are used to decrease the number of packet lost due to enqueuing failures.
In Tbl. 5.5 an overview of the results is provided. Because in this setup multiple parameters are adapted, the configurations are distinguished by config ids. First the

effect of redundant TX cells on the PDR is analyzed. If the number of buffer cells stays the same, but the number of redundant TX cells is increased, the PDR is improved. For example, comparing configuration 3 and 4, the usage of an additional TX cell results in an increase of the average PDR from 97.913% to 100 %. This is the same effect as in the second setup and follows also the expectations of Sec. 4.2.
If the number of TX cells is fixed but the number of buffer cells is increased, the PDR is also improved. For example, comparing configuration 1 and 3, the PDR is increased from 88.679% to 97.913%. This validates that buffer cells reduce the amount of packet loss by avoiding enqueuing failures.
Although the PDR is improved by buffer cells and redundant TX cells, the Δ_{avg} always increases due to the slotframe size. For example, comparing the configuration 3 and 4, the PDR is increased by approximately 2.1% but the Δ_{avg} is increased by approximately 7 ticks. The reason for this is that the usage of a larger slotframe resulting in a smaller update rate outweighs the effect of less packet loss. But additional TX cells per update have a positive effect on the Δ_{peak}. Comparing the configurations with two TX cells to those with one TX cell the peak is always reduced.
Focusing on the PDR, the configurations containing two TX cells maximize the PDR. If multiple TX cells per update should be avoided due to energy efficiency, the configuration with four buffer cells and one TX cell provides the highest reliability. If the AoI is also considered, the setup with two buffer cells and one TX cell is a good compromise between energy efficiency (only one TX cell), reliability (most enqueuing failures are avoided), and delay (smaller slotframe size).

Table 5.5.: Overview measurement results setup 3.

config id	buffer cells	TX cells	avg. PDR [%]	Δ_{avg} [ticks]	Δ_{peak} [ticks]	slotframe size [slots]
1	1	1	88.679 ± 0.889	35.103 ± 0.648	336	37
2		2	99.989 ± 0.015	40.204 ± 0.042	146	53
3	2	1	97.913 ± 0.098	46.854 ± 0.107	468	59
4		2	100	54.144 ± 0.023	113	71
5	3	1	97.797 ± 0.741	58.704 ± 0.859	388	73
6		2	99.941 ± 0.034	68.251 ± 0.063	253	89
7	4	1	98.299 ± 0.170	77.352 ± 0.432	399	97
8		2	99.957 ± 0.053	84.920 ± 3.932	303	107

5.4. Setup 4 - Interference Scenario

This setup simulates the case of high interference in the environment. In Tbl. 5.6 the measurement results are summarized. Because the results of the third setup (Sec. 5.3) show that most enqueuing failures are avoided using two buffer cells, the same amount of buffer cells is configured in this setup. Starting with the configuration containing only one

Table 5.6.: Overview measurement results setup 4.

buffer cells	TX cells	avg. PDR [%]	min PDR [%]	Δ_{avg} [ticks]	Δ_{peak} [ticks]	slotframe size [slots]
2	1	29.191	0			59
	2	99.007	97.4	55.034	202	71
	3	99.453	98.225	68.151	253	89
	4	100	100	79.94	165	107

Table 5.7.: PDR at each node per process in setup 4 using one TX cell and two buffer cells.

node	PDR for each *process id* PDR [%]				
	process 1	process 2	process 3	process 4	process 5
node 1		98.073	0	0	0
node 2	0		96.917	0	0
node 3	0	98.073		0	0
node 4	0	0	96.917		97.688
node 5	0	0	96.146	0	

TX cell per update, the PDR is reduced to approximately 30%. At some nodes no updates of specific processes are received such that the corresponding PDR is zero. Therefore, the average and peak AoI are undefined. Adding one redundant TX cell, the average PDR is significantly increased to 99.007%. Unlike to the first configuration, the minimum PDR considering all nodes and process combinations is 97.4%, which indicates that no update transmission always interferes with the disrupter. This can be explained by the behavior of the disrupter node. The redundant TX cells are in subsequent timeslots. Because the disrupter sends a packet in even timeslots only, at most one of the two TX cells does not interfere with the disrupter. Using more than two TX cells optimizes the PDR up to 100% but results in a higher Δ_{avg}, because the smaller update rate outweighs higher reliability. However, the configuration with four TX cells per update reduces the AoI peak from higher than 200 ticks to 165 ticks. Therefore, the amount of redundant TX cells should be configured depending on the application. In case the average age should be minimized, two redundant TX cells are sufficient in this setup. If AoI peaks should be avoided, additional TX cells provide a higher reliability.

To analyze the low reliability in the configuration with one TX cell, the slotframes of the standard network nodes and of the disrupter node are compared in Fig. 5.8. The reason for the high packet loss is the forwarding characteristic of SUS. If an update gets lost, transmissions forwarding this update cannot be performed. The update of process 2 can be transmitted to node 1 and 3 but the disrupter interferes with the transmission to node 4 such that also node 5 does not get any update. Therefore, the PDR at node 1 and 3 is high (approximately 98%) but at the other nodes no updates are received (Tbl. 5.7). The PDR considering process 1 and 4 is 0% at every node, because the transmissions

to the direct neighbors and the disrupter transmissions are performed in the same slot. Updates of process 5 are only received by the direct neighbor (node 4) because forwarding these updates to node 3 results in collisions. The PDR considering process 3 is 0% at node 1 only because, except the forwarding to node 1, the disrupter transmissions do not interfere with the propagation of these process updates.
Summarizing, the PDR of the first configuration is smaller than the expected 50% because a single packet loss disables forwarding of corresponding updates.

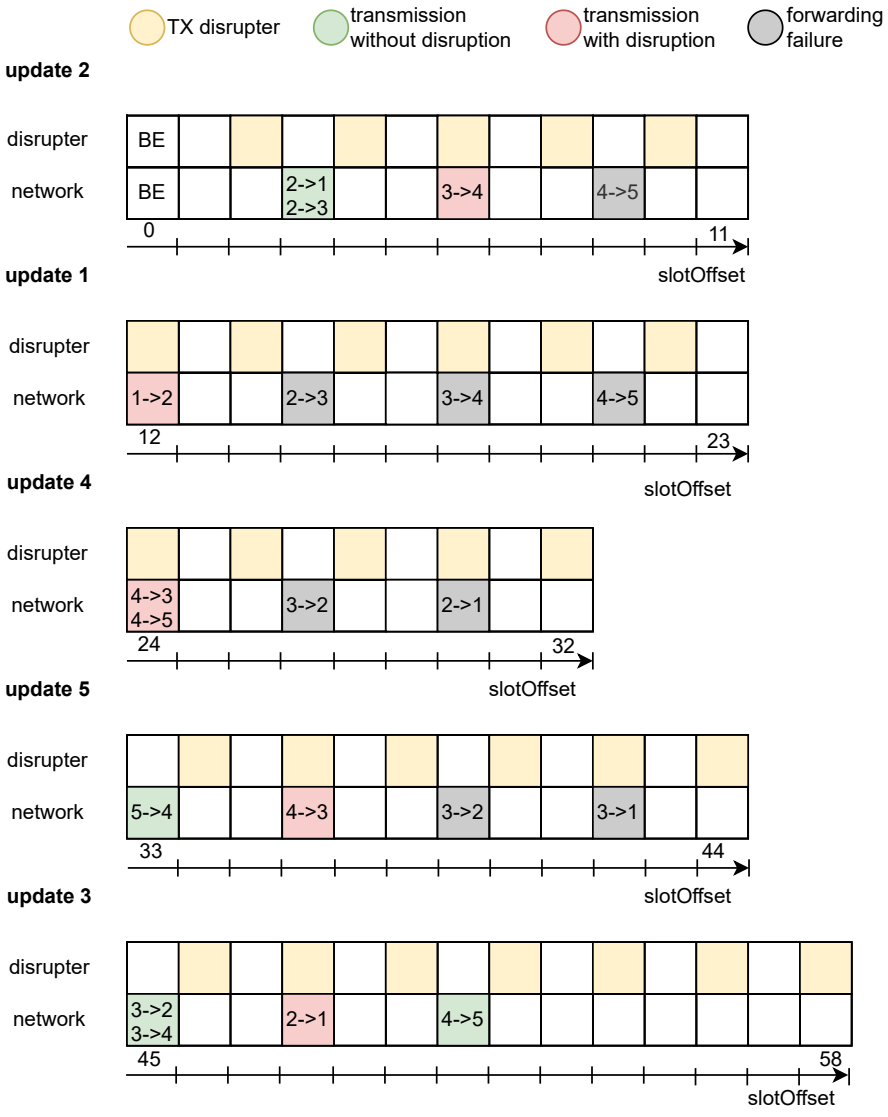

Figure 5.8.: Comparison of the sending and the disrupter schedule for each process (one TX cell, two buffer cells).

5.5. Setup 5 - Circle and Line Topology

In Tbl. 5.8 the measurement results of a line and a circle network (shown in Fig. 5.9, 5.10) are compared in different configurations. The dissemination schedule of the circle network can be found in the input file (Appendix List. A.2).

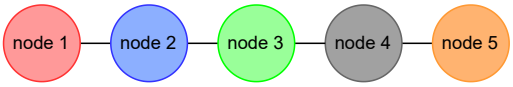

Figure 5.9.: Topology of five-node line network.

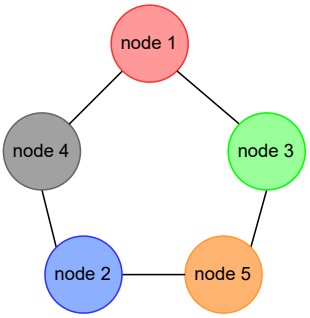

Figure 5.10.: Topology of five-node circle network.

Table 5.8.: Overview measurement results setup 5.

					Δ_{avg} for each *process id*[ticks]				
topo-logy	buffer cells	TX cells	avg. PDR [%]	Δ_{avg} [ticks]	1	2	3	4	5
circle	0	2	99.747	23.497	23.32	23.353	23.464	23.037	23.353
line			99.938	27.9	28.337	26.776	26.277	27.054	29.434
circle	2	1	95.237	43.93	41.816	42.914	41.164	40.445	41.818
line			98.016	46.701	46.245	43.6	42.987	43.922	49.099
circle	4	1	97.029	67.207	65.797	65.528	65.936	64.425	64.222
line			98.467	76.914	77.466	72.315	71.799	73.45	79.338

Starting with the first configuration (zero buffer cells, two TX cells per update), the achieved PDR values of the different topologies are similar, both larger than 99.5%. In a circle network the average hop distance at each node is 1.5 hops such that the Δ_{avg} values at the different nodes are all between 23.32 ticks and 23.464 ticks. The average hop distance at each single node is smaller in a circle network compared to a line network.

Therefore, the general forwarding effort is higher in a line network resulting in a larger slotframe size. Hence, the Δ_{avg} is smaller in a circle network (23.497 ticks) than in a line network (27.9 ticks). The other configurations provide similar results.
The different hop distances of the circle network also influence the histograms of the initial AoI. For the last configuration (four buffer cells, one TX cell), Fig. 5.11 provides the histogram at node 1 of the line network and Fig. 5.12 the histogram of the circle network. In the line network histogram the grouping depending on the hop distance is visible. In the histogram of the circle network, the AoI values of process 3 and process 4 are in the same region, because they are both direct neighbors of node 1. Node 2 and node 5 have both a distance of two hops to node 1, but the corresponding updates have different initial AoI values. This is the result of the dissemination schedule which defines that the update of process 2 is transmitted over the longer path, via node 5 and node 3.

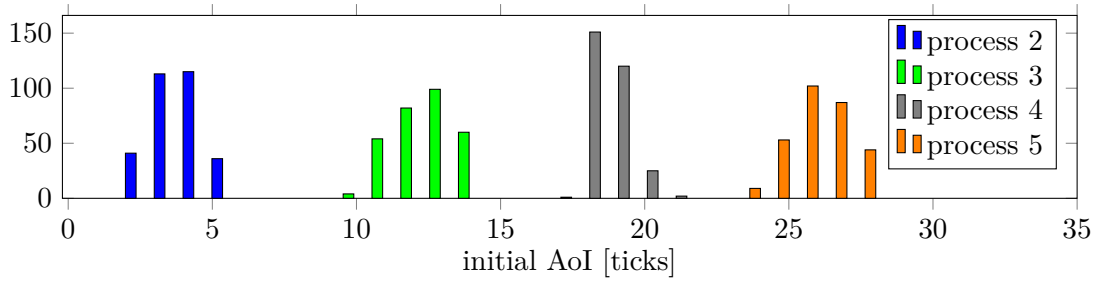

Figure 5.11.: Histogram of initial AoI at node 1 per process in a line network using one TX cells, four buffer cells, evaluation of a single measurement round.

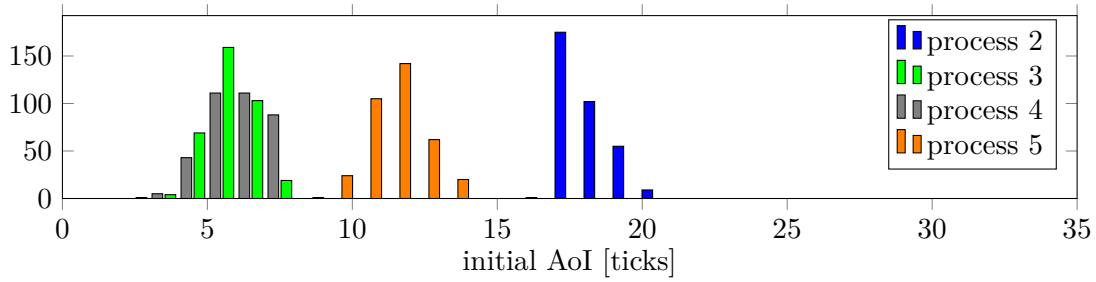

Figure 5.12.: Histogram of initial AoI at node 1 per process in a circle network using one TX cells, four buffer cells, evaluation of a single measurement round.

5.6. Setup 6 - Long Term Packet Delivery Ratio Measurement

The results of the sixth setup are shown in Tbl. 5.9. The goal of this setup is to analyze whether a longer usage of the testbed results in timing issues. Therefore, the measurement

is performed in one round lasting 3h and afterwards the log files are analyzed in six time intervals of 30 min. The PDR minimally fluctuates between 96.668% and 97.586%. The Δ_{avg} values are all in a small interval of 0.802 ticks. Only the Δ_{peak} differs between the intervals. But in general there is no trend visible that would indicate an error in the timing of the testbed or would point to a synchronization issue after a longer runtime.

Table 5.9.: Overview measurement results setup 6.

Time Interval	buffer cells	TX cells	avg. PDR [%]	Δ_{avg} [ticks]	Δ_{peak} [ticks]
1	2	1	97.057	47.435	620
2			97.119	47.367	393
3			96.812	47.619	620
4			96.668	47.745	619
5			97.322	47.127	318
6			97.586	46.943	319

5.7. Effects of Testbed Concept on Performance

In the measurement results a few effects are visible due to the concept of the testbed. In the histogram of the initial AoI values, some age values are negative. Because the computation of the AoI uses the network up time as time source, the negative values have to be a consequence of a synchronization misalignment between the nodes. The histograms show a trend that the initial AoI values are smaller at node 1, which is the coordinator, and slightly higher at the node 5. Therefore, it is assumed that the AoI measurement results have a small inaccuracy due to quality of synchronization.
Furthermore, the logic computing the timer for the origin packet is not optimal for all slots. The timer is always reset in the UDP callback, such that each valid reception results in redefining the timer. The time duration that is used to redefine the timer are stored in a lookup table determining for each timeslot the corresponding time duration. In case a packet is received after the TX slot of the origin packet, the timer has to consider the remaining timeslots in the current slotframe and the number of slots before the intended TX slot in the subsequent slotframe. Due to an implementation error, these timer duration are one timeslot too long. This applies to the entries of the timer lookup table for timeslots after the TX cell for the origin packet. This error should only affect the update that is sent in the slotframe first. Because the timer is always corrected after a reception, all nodes that first receive an update before they send their own update, overwrite the timer with a correct timer duration. In the considered line setup, node 2 sends its updates first, such that these are affected by the error because the risk of enqueuing failures is increased. Considering the PDR depending on the *process id* this is only visible in the measurement results of setup 3 in the configuration with one TX cell and one buffer cell. In this configuration the PDR of process 2 is approximately 61% and

the PDR considering all processes is 88.679%. Fixing the error, would reduce the AoI and increase the PDR of this specific update. In the other configurations with additional buffer cells and/or redundant TX cells, the effect of the error is not visible anymore.
In addition the general quality of the timer definition is limited by the precision of the unit *etimer* ticks. In Sec. 3.3.1 the timer, controlling the enqueuing of the origin packets, is described. The timers are computed by determining the time between the current slot and the slot when the packet should be enqueued in milliseconds. This time is divided by 7.8125 ms to convert it to ticks which may cause rounding errors. In configurations without buffer cells it is not acceptable that a packet is enqueued too late, such that the timer duration is rounded down. In configurations using buffer cells, the goal is to enqueue the packet in the first buffer cell. To avoid that the packet is enqueued before this slot, the timer duration is rounded up. This rounding procedure has a direct impact on the amount of enqueuing failures. It could be possible that more elaborate timer definitions could improve the quality of timing. One possibility would be the usage of another time source. The testbed only considers the slot offset of the last reception as time source. It is assumed that the remaining time is very small in the current timeslot, such that this time difference is ignored. If it is possible to get the remaining time in the current timeslot precisely, the quality of the timer could be improved.
Another aspect is, that the logging of the AoI values requires two log messages per update resulting in an additional effort and therefore in an additional delay on the application layer. This overhead due to the logging is evaluated in a small measurement setup. The measurement uses a small application protocol simulating the logging of packets and measuring the delay to print the log messages in *rtimer* ticks [7]. The rtimer clock does not depend on the network up time but on the clock of the OpenMote board and is able to support a higher precision (1 rtimer tick is equivalent to 0.0305ms). This measurement results in an average delay of 6.93 ms per update. An extract of the log file is shown in the Appendix (List. A.3). Because this logging is not required in real applications, it could be possible that less buffer cells are sufficient to avoid enqueuing failures.

5.8. Considerations using the Testbed outside the Lab

To use the testbed outside the lab in real applications, some aspects have to be considered. An aspect that is not required in a real setup is the enforcement of the topology using the slotframe scheduling. In this testbed, a RX cell is only added at nodes that are defined as neighbors considering the adjacency matrix. In a real environment it would be sufficient to add RX cells at all network nodes, because of the physical distance between the nodes. Furthermore, one aspect of the slotframe schedule, that has to be changed, is the transmission of EB frames. In the testbed the topology is ignored to distribute the EB, such that for this single timeslot, the topology is equivalent to a star with node 1 as the center node.
Another aspect is the flexibility and scalability of the testbed concept. The static

initialization of the slotframe limits the flexibility of the setup, because each change in the network topology requires an adaption of the application protocol of each node. Therefore, the flexibility of the SUS would be improved, if the logic to compute the local schedule is moved to the network node instead of using the code generator. Then each node has the same application protocol, which determines the timing and schedule locally at the node. In addition, using 6P to adapt the schedule instead of a static initialization increases the flexibility of the SUS because 6P is able to adapt the slotframe during runtime. Furthermore, 6P maintains the consistency of the local schedules of neighbored nodes, e.g., if node A adds an additional TX cell, node B will automatically add the corresponding RX cell (Sec. 2.3.2). This can improve the performance of the SUS. For example, additional redundant TX cells can be dynamically added to a node with a lower reliability.
In addition, the scalability of the scheduling is limited. In the current testbed a large amount of network participants would result in many timeslots per slotframe which decreases the update rate significantly. One possibility to reduce the slotframe size is the usage of different channel offsets enabling the transmission of multiple updates per slot. Furthermore, if two transmissions have different destination nodes, it would be possible to send them in parallel. For example, in a five-node line setup, node 1 and node 5 could broadcast their updates in the same timeslot without collisions.
Another idea is to use shared slots instead of dedicated slots. Then the slotframe could be minimized to one shared slot and the dissemination schedule is only used by controlling the sending and forwarding at each node.

6. Conclusion

The 6TiSCH testbed, developed in this thesis, implements multi-source, multi-sink, and multi-hop SUS used to execute and analyze theoretical dissemination schedules on physical ICT platforms. The concept of the testbed includes a code generator providing the logic to analyze the dissemination schedule. As input only the topology, the schedule, and configuration parameters have to be provided. The code generator includes the logic to schedule the slotframes on the MAC layer and to define the sending and forwarding behavior on the application protocol. The output of the generator is the individual implementation of the application protocol which has to be flashed on each node. As ICT platform OpenMote B boards running Contiki NG as operating system are used. Therefore, the testbed supports the execution of arbitrary dissemination schedules in varied topologies, simply by passing the appropriate input file to the generator. In addition, the adaption of configuration parameters improves the performance of the testbed in different setups depending on the application requirements.
To prove the testbed capabilities, different setups are measured and analyzed. It is proven that the testbed performs a timeslotted communication based on channel hopping. Furthermore, the analysis of the AoI graphs shows that the slotframe schedule provides the intended periodic sending and forwarding of status updates as defined by the dissemination schedule. The analysis of different configurations in varied setups illustrates the advantages of the introduced configuration parameters. For the considered setups the thesis provides optimized configurations with respect to the application requirements AoI, PDR, and energy efficiency. The performance comparison of setups with different topologies shows similar results such that it is reasonable that the testbed performance does not significantly depend on the topology.

In general, the measurement results corresponds to the expectations such that the testbed is usable to analyze further topologies, schedules and configurations. In the end, different approaches are discussed to improve the applicability in realistic scenarios. The flexibility of the testbed could be increased by implementing the logic to determine the local schedule and the timing at each node instead of using the code generator. Additionally, replacing the static initialization of the local slotframe by 6P communication would enable a flexible adaption of the schedule. Reducing the slotframe size by parallel transmissions using different channel offsets, could be used to improve the scalability such that the required slotframe size to support large SUS is decreased.

A. Appendix

Listing A.1: Input file for a 5 node line network for setup 2

```
{
        "adjacent_matrix":
        {
                "node1": [0,1,0,0,0],
                "node2": [1,0,1,0,0],
                "node3": [0,1,0,1,0],
                "node4": [0,0,1,0,1],
                "node5": [0,0,0,1,0]
        },
        "dissemination_scheduling":
        {
                "ts0": [2,2],
                "ts1": [3,2],
                "ts2": [4,2],
                "ts3": [1,1],
                "ts4": [2,1],
                "ts5": [3,1],
                "ts6": [4,1],
                "ts7": [4,4],
                "ts8": [3,4],
                "ts9": [2,4],
                "ts10": [5,5],
                "ts11": [4,5],
                "ts12": [3,5],
                "ts13": [2,5],
                "ts14": [3,3],
                "ts15": [2,3],
                "ts16": [4,3]
        },
        "redundant_tx_cells": 1,
        "buffer_cells": 0
}
```

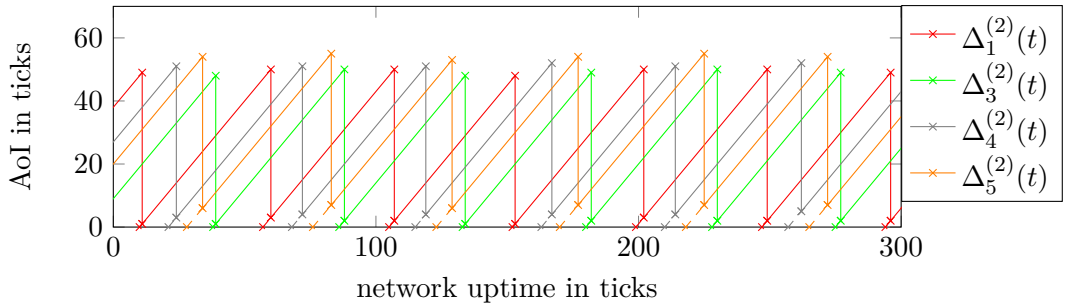

Figure A.1.: AoI at node 2 in setup 2 using two TX cells.

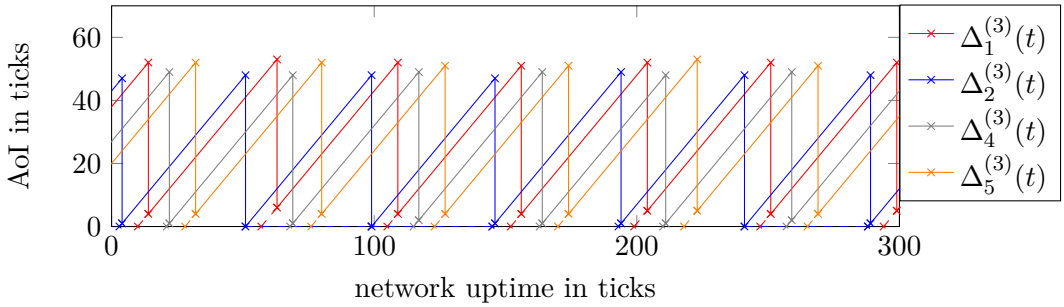

Figure A.2.: AoI at node 3 in setup 2 using two TX cells.

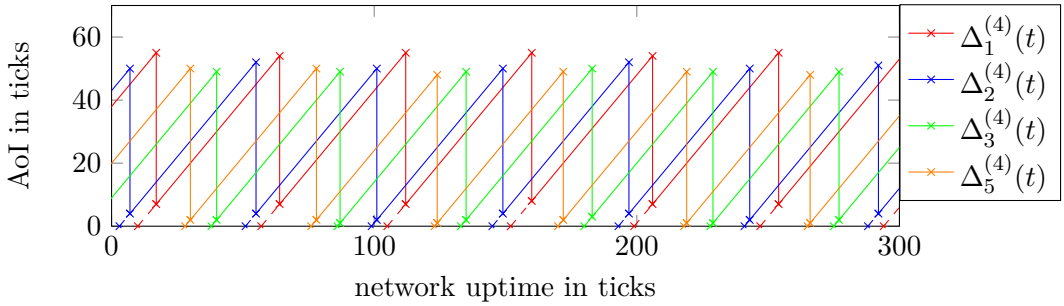

Figure A.3.: AoI at node 4 in setup 2 using two TX cells.

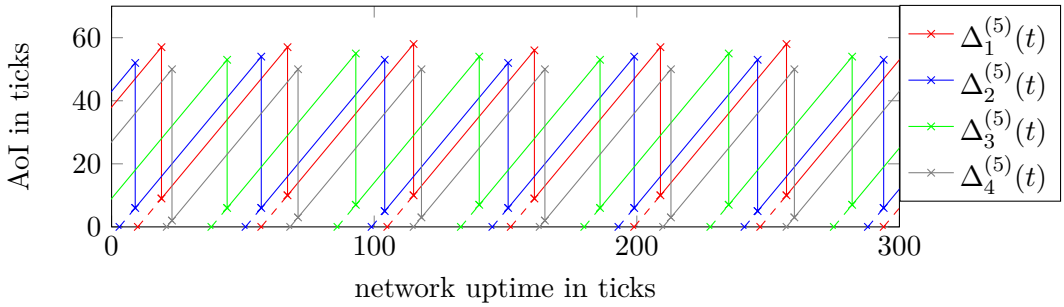

Figure A.4.: AoI at node 5 in setup 2 using two TX cells.

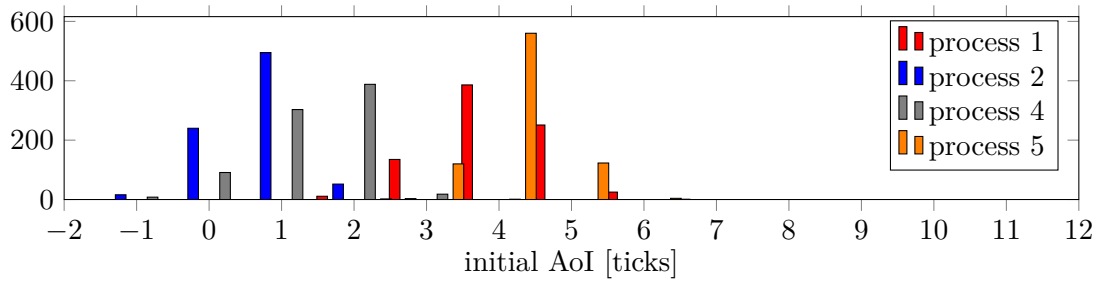

Figure A.5.: Histogram of initial AoI at node 3 per process in setup 2 using two TX cells, evaluation of a single measurement round.

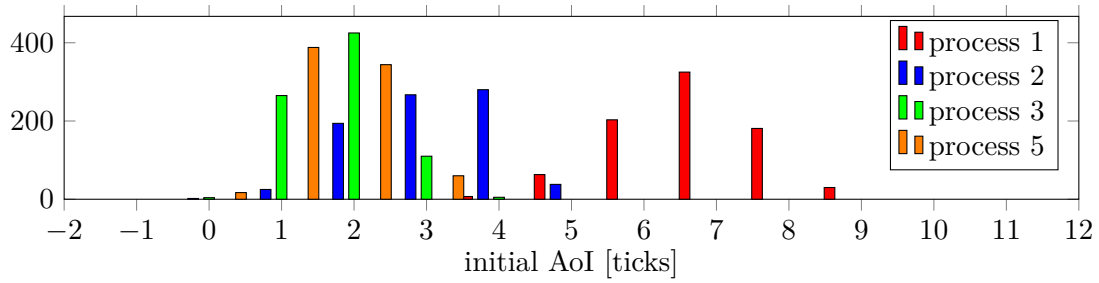

Figure A.6.: Histogram of initial AoI at node 4 per process in setup 2 using two TX cells, evaluation of a single measurement round.

Listing A.2: Input file for a 5 node circle network for setup 5

```
{
        "adjacent_matrix":
        {
                "node1": [0,0,1,1,0],
                "node2": [0,0,0,1,1],
                "node3": [1,0,0,0,1],
                "node4": [1,1,0,0,0],
                "node5": [0,1,1,0,0]
        },
        "dissemination_scheduling":
        {
                "ts0":  [5,5],
                "ts1":  [3,5],
                "ts2":  [1,5],
                "ts3":  [3,3],
                "ts4":  [1,3],
                "ts5":  [4,3],
                "ts6":  [4,4],
                "ts7":  [2,4],
                "ts8":  [5,4],
                "ts9":  [1,1],
                "ts10": [4,1],
                "ts11": [2,1],
                "ts12": [2,2],
                "ts13": [5,2],
```

```
                "ts14": [3,2]
        },
        "redundant_tx_cells": 1,
        "buffer_cells": 2
}
```

Listing A.3: Log File to measure overhead due to logging.

```
[INFO: App       ] 1.log_msg: cnt 1
[INFO: App       ] 2.log_msg: cnt 1
[INFO: App       ] Diff: cnt 1, cc2538_time_diff 163
[INFO: App       ] 1.log_msg: cnt 2
[INFO: App       ] 2.log_msg: cnt 2
[INFO: App       ] Diff: cnt 2, cc2538_time_diff 210
[INFO: App       ] 1.log_msg: cnt 3
[INFO: App       ] 2.log_msg: cnt 3
[INFO: App       ] Diff: cnt 3, cc2538_time_diff 211
[INFO: App       ] 1.log_msg: cnt 4
[INFO: App       ] 2.log_msg: cnt 4
[INFO: App       ] Diff: cnt 4, cc2538_time_diff 211
[INFO: App       ] 1.log_msg: cnt 5
[INFO: App       ] 2.log_msg: cnt 5
[INFO: App       ] Diff: cnt 5, cc2538_time_diff 210
[INFO: App       ] 1.log_msg: cnt 6
[INFO: App       ] 2.log_msg: cnt 6
[INFO: App       ] Diff: cnt 6, cc2538_time_diff 211
[INFO: App       ] 1.log_msg: cnt 7
[INFO: App       ] 2.log_msg: cnt 7
[INFO: App       ] Diff: cnt 7, cc2538_time_diff 210
[INFO: App       ] 1.log_msg: cnt 8
[INFO: App       ] 2.log_msg: cnt 8
[INFO: App       ] Diff: cnt 8, cc2538_time_diff 211
[INFO: App       ] 1.log_msg: cnt 9
[INFO: App       ] 2.log_msg: cnt 9
[INFO: App       ] Diff: cnt 9, cc2538_time_diff 211
[INFO: App       ] 1.log_msg: cnt 10
[INFO: App       ] 2.log_msg: cnt 10
[INFO: App       ] Diff: cnt 10, cc2538_time_diff 216
```

List of Figures

List of Tables

Bibliography

[1] LAN/MAN Standards Committee of the IEEE Computer Society. 'IEEE Standard for Low-Rate Wireless Networks'. In: *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)* (2020), pp. 1–800. DOI: 10.1109/IEEESTD.2020.9144691.

[2] Shahab Farazi, Andrew G. Klein and D. Richard Brown. 'Fundamental bounds on the age of information in multi-hop global status update networks'. In: *Journal of Communications and Networks* 21.3 (2019), pp. 268–279. DOI: 10.1109/JCN.2019.000038.

[3] Sanjit Kaul, Marco Gruteser, Vinuth Rai et al. 'Minimizing age of information in vehicular networks'. In: *2011 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*. 2011, pp. 350–358. DOI: 10.1109/SAHCN.2011.5984917.

[4] Sanjit Kaul, Roy Yates and Marco Gruteser. 'On Piggybacking in Vehicular Networks'. In: *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*. 2011, pp. 1–5. DOI: 10.1109/GLOCOM.2011.6134181.

[5] Rajat Talak, Sertac Karaman and Eytan Modiano. 'Minimizing age-of-information in multi-hop wireless networks'. In: *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. 2017, pp. 486–493. DOI: 10.1109/ALLERTON.2017.8262777.

[6] Shahab Farazi, Andrew G. Klein and D. Richard Brown. 'Age of Information with Unreliable Transmissions in Multi-Source Multi-Hop Status Update Systems'. In: *2019 53rd Asilomar Conference on Signals, Systems, and Computers*. 2019, pp. 2017–2021. DOI: 10.1109/IEEECONF44664.2019.9048736.

[7] George Oikonomou, Simon Duquennoy, Atis Elsts et al. 'The Contiki-NG open source operating system for next generation IoT devices'. In: *SoftwareX* 18 (2022), p. 101089. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2022.101089.

[8] Simon Duquennoy, Atis Elsts, Beshr Al Nahas et al. 'TSCH and 6TiSCH for Contiki: Challenges, Design and Evaluation'. In: *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. 2017, pp. 11–18. DOI: 10.1109/DCOSS.2017.29.

[9] Cedric Adjih, Emmanuel Baccelli, Eric Fleury et al. 'FIT IoT-LAB: A large scale open experimental IoT testbed'. In: *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. 2015, pp. 459–464. DOI: 10.1109/WF-IoT.2015.7389098.

[10] Diliara Ibrahimova and Sedat Gormus. 'QoS for IETF 6TİSCH protocol'. In: *2018 26th Signal Processing and Communications Applications Conference (SIU)*. 2018, pp. 1–4. DOI: 10.1109/SIU.2018.8404187.

[11] I. Tomasic, K. Khosraviani, P. Rosengren et al. 'Enabling IoT based monitoring of patients' environmental parameters: Experiences from using OpenMote with OpenWSN and Contiki-NG'. In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2018, pp. 0330–0334. DOI: 10.23919/MIPRO.2018.8400063.

[12] Openmote. *OPENMOTE B*. URL: https://openmote.com/product/openmote-b (visited on 10th February 2023).

[13] Muhammad Farhan Khan, Emad A. Felemban, Saad Qaisar et al. 'Performance Analysis on Packet Delivery Ratio and End-to-End Delay of Different Network Topologies in Wireless Sensor Networks (WSNs)'. In: *2013 IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks*. 2013, pp. 324–329. DOI: 10.1109/MSN.2013.74.

[14] Pascal Thubert. *An Architecture for IPv6 over the Time-Slotted Channel Hopping Mode of IEEE 802.15.4 (6TiSCH)*. RFC 9030. May 2021. DOI: 10.17487/RFC9030. URL: https://www.rfc-editor.org/info/rfc9030.

[15] T. Watteyne Q. Wang X. Vilajosana. *6TiSCH Operation Sublayer (6top) Protocol (6P)*. Tech. rep. 2070-1721. Internet Engineering Task Force (IETF), November 2018.

[16] Pascal Thubert and Jonathan Hui. *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*. RFC 6282. September 2011. DOI: 10.17487/RFC6282. URL: https://www.rfc-editor.org/info/rfc6282.

[17] Pascal Thubert, Carsten Bormann, Laurent Toutain et al. *IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header*. RFC 8138. April 2017. DOI: 10.17487/RFC8138. URL: https://www.rfc-editor.org/info/rfc8138.

[18] Carsten Bormann, Zach Shelby, Samita Chakrabarti et al. *Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)*. RFC 6775. November 2012. DOI: 10.17487/RFC6775. URL: https://www.rfc-editor.org/info/rfc6775.

[19] Texas Instruments. *TEXAS INSTRUMENTS CC2538*. URL: https://www.ti.com/product/CC2538 (visited on 10th February 2023).

[20] Microchip. *MICROCHIP AT86RF215*. URL: https://www.microchip.com/en-us/product/AT86RF215 (visited on 10th February 2023).

[21] Uros Pesovic, Sladjana Djurasevic, Vanja Lukovic et al. 'Interference classification for IEEE 802.15.4 networks'. In: *2020 International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom)*. 2020, pp. 1–4. DOI: 10.1109/CoBCom49975.2020.9174119.

[22] Thomas Watteyne, Maria Rita Palattella and Luigi Alfredo Grieco. *Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement*. RFC 7554. May 2015. DOI: 10.17487/RFC7554. URL: https://www.rfc-editor.org/info/rfc7554.