University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Implementation of an Energy Efficient LoRa Network

Steiger David
Bassersdorf, Zürich, Switzerland
Student ID: 16-922-015

Supervisor: Eryk Schiller
Date of Submission: September 1, 2021

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Deutsche Zusammenfassung

Im Internet der Dinge (IoT) ermöglichen Niedrigenergie-Weitverkehrsnetze (LPWAN) IoT-Anwendungen eine grosse Netzabdeckung und geringen Energieverbrauch. Long Range (LoRa) Wide Area Network (WAN) ist eine bekannte LPWAN-Technologie, welche von der LoRa-Modulation in der Bitübertragungsschicht Gebrauch macht. Forschungen im Bereich der Netzwerkoptimierung führten Adaptive Data Rate (ADR) Algorithmen ein, welche in LoRaWAN-Netzwerke installiert werden können, um Energieeffizienz, Skalierbarkeit und die Verarbeitungsmenge zu verbessern. Diese These beinhaltet die Implementation und Evaluation von ADR-Algorithmen vom letzten Stand der Technik (SotA). Die Implementation und Evaluation erfolgt in einer Simulationsumgebung namens Network Simulator 3 (ns-3). Die durchgeführten Simulationen gewähren Einblick in LoRaWAN-Netzwerke mit hohem Netzwerkverkehrsaufkommen, was in der Spezifikation, Implementation und Evaluation von neuartigen ADR-Algorithmen resultiert. Die erwähnten ADR-Techniken werden miteinander verglichen, wobei die neu entwickelten ADR-Algorithmen LoRaWAN-Netzwerken eine erheblich höhere Netzwerkskalierung, Verarbeitungsmenge und Ausfallsicherheit ermöglichen.

English Abstract

In the Internet of Things (IoT), Low Power Wide Area Networks (LPWAN) enable IoT applications to cover large areas and operate with low energy consumption. Long Range (LoRa) Wide Area Network (WAN) is a known LPWAN technology that uses LoRa modulation on the physical layer. The performance of LoRa networks, *i.e.,* energy efficiency, network scalability, and throughput, is typically improved with Adaptive Data Rate (ADR) algorithms. This thesis provides implementation and evaluation of State-of-the-Art (SotA) ADR algorithms in the Network Simulator 3 (ns-3) framework. The simulations revealed problems of SoTA ADRs in highly congested networks. The solutions to those problems led to the specification, implementation, and evaluation of novel ADR techniques that further improve the network performance. Several ADR schemes are compared and show that the newly developed ADR algorithms enable substantially higher network scalability, throughput, and reliability in comparison to the SotA techniques.

Acknowledgments

I would like to express my sincere gratitude to my research supervisor, Dr. Eryk Schiller, for guiding me through this thesis by sharing his expert knowledge, giving frequent feedback, and offering me his time. I highly appreciate his support, without which this thesis would not have been possible. I would also like to thank Prof. Dr. Burkhard Stiller for this unique opportunity and his support for doing this thesis at the Communication Systems Group (CSG) of the University of Zürich.

I am extending my heartfelt gratitude to my family and friends, who had a lot of patience with me.

Contents

Chapter 1

Introduction

The Internet of Things (IoT) is an emerging technology for solving societal issues such as digital cities, intelligent transportation, green environment monitoring, and medical care [6]. An IoT application is supported through an underlying network technology that respects the technical requirements of the application. Low Power Wide Area Network (LPWAN) technology offers kilometer-range coverage, limited power consumption, and simplified network architecture [7]. Due to the LPWAN networks, battery-powered devices can operate for years [12] on a single battery charge. For instance, nodes in LPWANs can last for up to 17 years, sending 100 B once a day [16].

Long Range Wide Area Network (LoRaWAN) is a widely adopted LPWAN technology [1]. LoRaWAN defines a communication protocol and system architecture of the network that exploits a robust Chirp Spread Spectrum (CSS) modulation on the physical layer, also referred to as LoRA [7,12]. The LoRaWAN architecture is simple and consists of End Devices (ED), Gateways (GW), and a Network Server (NS) organised in a star-of-stars topology [12, 22]. While LoRaWAN enables the configuration of EDs, numerous Adaptive Data Rate (ADR) algorithms have been introduced that optimize LoRaWAN networks by enabling the NS to dynamically adjust transmission parameters on EDs to appropriately adapt to the current network state. More specifically, the objective of ADR algorithms includes, but is not limited to, the improvement

of network scalability, energy efficiency, and throughput [10].

The rapidly increasing IoT market implies that the scale of future IoT applications will substantially increase [11,21]. As a consequence, the network scalability of LoRaWAN networks needs to be improved for the LoRaWAN technology to stay competitive in the IoT domain. However, this is a non-trivial problem because LoRaWAN protocol only exploits the CSS modulation technique on the physical layer [7] and as an ALOHA-like protocol does not make use of Listen Before Talk (LBT) or Carrier Sense Multiple Access (CSMA) mechanisms [2].

This thesis investigates how reliably a LoRaWAN can operate in highly congested situations. The evaluations performed in this thesis result from simulations that are conducted in a discrete-event simulation environment called Network Simulator 3 (ns-3) [18]. The ns-3 framework enables realistic LoRaWAN simulations through the integration of LoRaWAN modules [14,27]. Another subject that needs further investigation is the extent to which classical ADR algorithms improve the network scalability of LoRaWAN networks. During the constant evaluation of simulations in this thesis, novel ADR algorithms are specified, implemented, and evaluated which are designed to improve the reliability and scalability of LoRaWAN networks.

1.1 Motivation

Since the LoRaWAN technology offers low energy consumption, high communication range, and cost-effective devices [12], LoRaWAN is a competitive technology in the IoT network domain. As more and more businesses will operate with IoT applications [11,21], more devices will be connected to the IoT. As a consequence, the technical requirements of IoT networks will become more demanding and more difficult to fulfill. LoRaWAN technology has to stay competitive by the optimization research performed to improve the scalability of LoRaWAN networks.

1.2 Description of Work

In this thesis, multiple ADR algorithms are studied which optimize LoRaWAN networks. Since energy efficiency of LoRaWAN networks is a metric that can be optimized in various ways, other metrics that are related to energy efficiency are put into focus as well. For instance, improving the reliability of LoRaWAN networks positively impacts the energy efficiency due to the fact that less retransmissions must be performed by EDs.

Many ADR algorithms have been introduced in the LoRaWAN community. This thesis implements one of the most recent State-of-the-Art (SotA) ADR algorithms [3]. The algorithm is ported towards ns-3 as initially, it was only implemented in a custom-made simulator by its authors. The simulations of this algorithm yield important results, which provide relevant feedback that allows for the specification of new ADR algorithms in this work. Those algorithms are then again implemented and evaluated in ns-3. Such an iterative process considering the specification, implementation, evaluation, and feedback is executed multiple times until newly optimized ADR algorithms are implemented. Finally, they improve the overall reliability, scalability, throughput, and collision rates in the LoRaWAN.

1.3 Thesis Outline

The thesis is divided into the following chapters. Chapter 2 provides descriptions of related work, alternative LPWAN technologies, and the simulation framework used in this thesis. Chapter 3 lays out the specifications of LoRaWAN and ADR algorithms. Chapter 4 provides descriptions of the implementation of ADR algorithms. Chapter 5 presents and discusses the results of the conducted simulations of the implemented algorithms. Chapter 6 contains a summary as well as the conclusion of this thesis' findings. Furthermore, an outlook on future work is given.

Chapter 2

Related Work & Technologies

As many ADR algorithms are developed by the LoRaWAN community, an overview is needed that allows the existing ADR solutions to be organized. Kufakunesu, Hancke, and Abu-Mahfouz (2020) compared numerous ADR algorithms in a survey, in which they reveal the strengths and drawbacks of different approaches [10]. Furthermore, they provide an overview of techniques that prove to be useful in ADR algorithms [10]. This chapter contains several examples of related work and sheds light on alternative LPWAN technologies and simulation frameworks.

2.1 Related Work

C. Moy (2019) implements a novel learning algorithm (IoTligent) that operates on the ED-side [17]. He models the spectrum access issue with a Multi-Armed Bandit (MAB) problem [17]. He uses reinforcement learning, a form of machine learning, to find the best configuration of transmission parameters. The NS "rewards" successful transmissions with an acknowledgment (ACK) which informs nodes that the previously used configuration enabled a successful transmission. A positive characteristic of this approach is that very low processing and memory overhead is added to LoRaWAN devices in the network [17]. However, the utilization of downlink transmissions (ACKs) for every successful transmission is questionable

because the performance of networks with an ALOHA-like protocol quickly degrades as the network load increases [29]. Nevertheless, his evaluation demonstrates that EDs improve with IoTligent (i.e. are more likely to use a channel that results in more successful transmissions).

Cuomo et al. (2020) specify a machine learning technique using classification and regression trees in a theoretical model [5]. Nodes with a similar behavior within the network are clustered together with a k-means algorithm [5], such that the machine learning toolbox can be used for groups of nodes with a similar behavior. The NS using this machine learning technique can predict future inter-arrival times (in this thesis referred to as TxTime) of nodes. This might be particularly useful if the inter-arrival times vary. By using this technique, high traffic periods could be detected and even avoided. However, they never implemented their model but their work suggests that their algorithm may find usage in future ADR schemes.

Cuomo et al. (2017) specified and implemented an ADR algorithm named EXPLoRa-AT [4], which balances the usage of Spreading Factors (SF) by taking Time on Air (ToA) into consideration. They assume a single gateway with one channel and perfect orthogonality of LoRa signals. They apply an ordered-waterfalling approach to balance out ToA in the channel. Their evaluation shows that the number of packet collisions is reduced in a network using EXPLoRa-AT, ultimately making the network more reliable. While their heuristic algorithm causes low processing overhead, their approach (arguably) does not address the margin of successful communication. For instance, in a LoRaWAN network with a low network load, the ADR mechanism will balance out the SF distribution, although there might be no need for it. However, when the system encounters high network loads, EXPLoRa-AT substantially reduces the collision rate and possibly increases throughput by doing so.

Coutaud et al. (2020) specified and implemented an ADR solution [3] that optimizes SFs and the Number of Transmissions (Nb_{trans}) for nodes. By being aware of lost packets, the NS can estimate the nodes' signal conditions. This estimation allows the NS to configure the nodes' radio parameters in a way, such that the nodes send packets successfully while transmitting with minimal effort. However, their suggested algorithm implies the assumption that packet

loss only occurs due to weak signals between EDs and GWs. This assumption is careless due to the fact that a LoRaWAN network with a high network load mainly loses packets due to collisions and not due to weak signals (i.e. "bad network connection"). Their ADR algorithm might even invoke destructive behavior in the network when a collision scenario is present (i.e. when packet losses mainly occur due to collisions). Nonetheless, the evaluation of their network emulations demonstrates that their algorithm is a significant improvement over the LoRaWAN ADR algorithm suggested by TheThingsNetwork (TTN) [3]. The Data Error Rate (DER), the loss ratio between EDs and NS, falls below the value of 1%, which indicates that the simulated networks achieved high reliability.

2.2 Introduction to LPWAN

Low Power Wide Area Network (LPWAN) technologies address long-range communication, low power consumption, and cost-effectiveness [22]. Long-range communication in LPWAN technology is often achieved by using Sub-GHz bands and special modulation schemes, such as narrowband and spread spectrum [22]. Having a star topology and duty cycling reduces network complexity and allows nodes to turn off when they are neither transmitting nor receiving [22]. Due to reduced hardware complexity and simplified architecture [22], the cost and power consumption of LPWAN devices is minimized [12, 24]. All the mentioned characteristics let LPWAN technologies compete within domains, where short-range wireless technologies and cellular networks are already well-established [22].

2.3 Alternative LPWAN Technologies

Besides LoRaWAN, other LPWAN technologies exist as noteworthy competitors. Each offers a set of diverse advantages, which makes them unique in their own way. This section will present a short list of alternative LPWAN technologies with a short description of each:

- **SIGFOX** is a well-known LPWAN technology, where EDs connect to base stations using a Binary Phase Shift Keying (BPSK) modulation in an ultra-narrow (100Hz) Sub-GHz band carrier [22]. By using ultra-narrow bands (UNB), their systems achieve very low noise levels, high receiver sensitivity, ultra-low power consumption, and cost-effectiveness [22]. However, these benefits come at the expense of a maximum uplink throughput of 100 bps [22]. Like LoRaWAN, SIGFOX uses an ALOHA-like protocol, which simplifies communication, but greatly limits the scalability of the network [29].

- **Ingenu** is a proprietary LPWAN technology that uses a patented Random Phase Multiple Access (RPMA) physical access scheme, which is a variation of Code Division Multiple Access (CDMA) [22]. The usage of RPMA reduces overlapping between transmitted signals and therefore increases the signal-to-interference ratio for each link [22]. RPMA also enables devices to have a high receiver sensitivity and high link budget [22]. As an exception to other LPWAN technologies, Ingenu uses 2.4 GHz ISM bands for which regulations do not impose duty cycles on the network [22]. This enables higher throughput and capacity in comparison to LPWAN technologies using Sub-GHz bands [22].

To further compare the mentioned LPWAN technologies with focus on this thesis' points of interest, the table below (Table 2.1) replicates a subset of the LPWAN comparison table that was created by Raza, Kulkarni, and Sooriyabandara (2017):

	SIGFOX	**LORAWAN**	**INGENU**
Modulation	UNB DBPSK(UL), GFSK(DL)	CSS	RPMA-DSSS(UL), CDMA(DL)
Band	Sub-GHz ISM	Sub-GHz ISM	ISM 2.4GHz
Data Rate	100bps(UL),600bps(DL)	0.3-37.5kbps(LoRa),50 kbps(FSK)	78kbps(UL),19.5kbps(DL)
Range	10km Urban, 50km Rural	5km Urban, 15km Rural	15km Urban
FEC	No	Yes	Yes
MAC	unslotted ALOHA-like	unslotted ALOHA-like	CDMA-like
Topology	star	star of stars	star, tree
ADR	No	Yes	Yes
Payload Length	12B(UL), 8B(DL)	up to 250B	10kB

Table 2.1: Technical Specifications of various LPWAN Technologies [22]

2.4 Simulation Frameworks for LoRaWAN

Communication network research is often constrained by limited resources, such as time and money. While this field of research requires an extensive amount of network measurements for validation, questions arise as to where one gets access to said networks. As a solution to this problem, several simulation frameworks were developed in the past, such as OMNET++ [19] and Network Simulator 3 (ns-3) [18]. Both are based on C++ [26] and facilitate realistic simulations of LoRaWAN networks by either allowing functional extensions of the framework [25] or integration of LoRaWAN modules [14, 27].

Magrin et al. (2017) have developed a LoRaWAN module for ns-3 [14, 15] that allows the simulation of LoRaWAN networks. The module implements class A EDs, GWs, and NS, whereas ADR algorithms can be installed on the NS as a network component. Their precise definition of link models allows the simulation of realistic LoRaWAN networks [26].

Duda and To (2018) created an LoRaWAN module for ns-3 [27], with which they compared the simulation results on a real-world testbed [28] and measured values reported by the work [9] by Haxhibeqiri et al. (2017) [26]. The models are not well documented but it is shown in the publication that the module correctly represents the capturing effect, lowering the packet loss ratio due to collision [26]. However, the capturing effect with orthogonal SFs remains unclear in the simulation [26].

FLoRa (Framework for LoRa) [25] is a simulation framework based on OMNeT++ that also allows users to simulate LoRaWAN networks. FLoRa contains an accurate model of the physical layer that also allows collisions and the capturing effect in simulations [25]. Like the other two previous ns-3 modules, FLoRa also contains implementations of network components such as EDs, GWs, and NS [26]. Additionally, the framework provides statistics of energy consumption in the network [25].

2.5 Evaluation of Simulation Frameworks and Modules for LoRaWAN

This thesis uses ns-3 as the simulation framework because ns-3 has proven to be the most suited framework for similar theses within CSG. As for the ns-3 modules, Surbeck (2019) from CSG evaluated the mentioned modules before [26]. His evaluation shows that the module [14] from Magrin et al. (2017) offers the most beneficial properties, such as great usability, well-written documentation, an acceptable implementation, and an available energy framework [26]. Due to his detailed explanation as to why the module of Magrin et al. (2017) is well suited for his work and the fact that our theses have similar simulation requirements, this thesis makes use of the mentioned LoRaWAN module [14] as well.

Chapter 3

Specifications

The LoRa Alliance proposes a cellular topology with GWs that receive packets from EDs and relay the data to a NS on a TCP connection [16]. LoRaWAN functions in an unlicensed Sub-GHz ISM band (863–870 MHz band in Europe and 902–928 MHz in the USA) [10]. It uses the 125 kHz, 250 kHz, and 500 kHz bandwidth (BW) and transmits payloads of up to 250 Bytes over 5–15 km [10]. A Sub-GHz ISM band normally includes a duty cycle for EDs, which limits the amount of time that an ED is allowed to transmit. The value of the duty cycle varies and is usually set to a value between 0.1% and 10% [13, 14]. However, ns-3 simulations using ISM bands with a duty cycle smaller than 1% require more nodes to successfully study high congestion LoRaWAN networks than they would with a larger duty cycle. If a higher duty cycle is chosen for the network, nodes are allowed to transmit with a relatively high frequency (i.e. TxTime $\in \{9s, 10s, 11s, 12s, 13s\}$), which enables the simulation of high network loads in LoRaWAN networks with fewer EDs. Thus a duty cycle of 10% is used in all simulations in this thesis. The main goal of LoRaWAN is to create a network with low power consumption, long-range transmissions, and low-cost infrastructure [12]. It was mainly designed for sensor networks that use low data rates with relatively high time intervals between packets (e.g. transmit every hour or even days) [4]. Usually, LoRa is referred to as two distinct layers: (a) the physical layer using Chirp Spread Spectrum (CSS) modulation and (b) a MAC layer protocol [4]. The following sections give an overview of the two layers and the LoRaWAN architecture.

3.1 LoRa Physical Layer

Transmissions are spread out on different data rates and frequency channels [13]. By using a CSS modulation, an ED can use up to six different spreading factors $SF \in \{7, 8, 9, 10, 11, 12\}$ or data rates [5], whereas LoRa data rates range from 0.3 kbps to 50 kbps [13]. An ED using a higher SF has a longer communication range, but the ToA increases exponentially with an increasing SF due to a decreased data rate. Additionally, the Code Rate (CR), usually fixed at 4/5 for LoRaWAN [23], is the redundancy implemented by a Forward Error Correction (FEC) mechanism that is used to detect errors and correct them. The relation between bit rate R and SF is given as [4]:

$$R = SF \times \frac{BW}{2^{SF}} \times CR$$

It should be noted that EDs will need to use a higher SF the farther away they are from the GW. However, a transmission having a high ToA is more susceptible to collisions, therefore an increase in SF should be made cautiously. Since SFs are quasi-orthogonal between each other [8], frames sent with different SFs are much less exposed to interference. The interference between LoRa signals is described in the co-channel rejection (dB) matrix (Table 3.1) created by Goursaud and Gorce (2015) and is used for the interference model in the LoRaWAN ns-3 module [14] of this thesis:

Desired | Interferer (dB)	SF7	SF8	SF9	SF10	SF11	SF12
SF7	-6	16	18	19	19	20
SF8	24	-6	20	22	22	22
SF9	27	27	-6	23	25	25
SF10	30	30	30	-6	26	28
SF11	33	33	33	33	-6	29
SF12	36	36	36	36	36	-6

Table 3.1: Cochannel Rejection (dB) for all Combinations of Spreading Factor for the Desired and Interferer User [6]

The rejection coefficients define the values that decide over the destruction of signals due to interference. The desired signal is destroyed by the interfering signal when the negative signal-to-interference ratio is above the rejection coefficient given for a pair of signals [14]. It is

observable that two EDs can transmit simultaneously if none of the frames is received with a power significantly higher [6]. Note that the rejection coefficient increases with increasing SF, which means that distant nodes transmitting with high SF will be able to overcome the simultaneous reception of closer nodes, which will likely be received with higher reception power [6].

3.2 LoRaWAN Architecture

LoRaWAN networks use a long-range star-of-stars topology [22] to preserve battery lifetime while long-range connectivity can be achieved [12]. Mesh networks end up having high complexity and increased energy consumption because nodes both receive and forward information from other nodes [12]. Transmissions from GWs to EDs are referred to as downlink communication and transmissions from EDs to GWs as uplink communication. The LoRaWAN architecture is made up of three main components [4]:

- **End Device (ED)**: The low-power consumption devices that communicate with GWs using LoRa [4]. According to the reception window profile, three classes are distinguished:
  - **Class A**: EDs which enable bi-directional communication [12]. Downlink communication is only possible during two short reception windows, that open sequentially after an uplink transmission [13]. Downlink reception is therefore only possible when a class A node wakes up in order to transmit a frame to GWs. Class A operation is the lowest power ED system for applications that may delay downlink transmissions until an ED sends an uplink transmission [12].
  - **Class B**: On top of the class A functionality, class B EDs give GWs the possibility to schedule uplink-independent reception windows in addition to the two short ones after each uplink transmission [12]. EDs receiving time-synchronized beacons allow the GW to register when EDs will open additional reception windows [12]. Each

additional reception window requires EDs to wake up, which makes class B EDs a less efficient option for IoT applications.

  - **Class C**: Class C EDs have continuously open reception windows, except during uplink transmissions [12]. They represent the ED class with the lowest energy efficiency. Only applications that require EDs to be reachable at all times should consider class C EDs.

- **Gateway (GW)**: GWs are the intermediate devices that forward packets coming from EDs to a NS over an IP backhaul interface allowing high throughput, such as Ethernet or 3G [4]. There can be multiple GWs in a LoRa deployment and the same data packet can be received (and forwarded) by more than one GW [4].

- **Network Server (NS)**: Responsible for deduplicating and decoding the packets sent by the devices and generating packets that should be sent back to the EDs [4]. Usually, ADR algorithms are installed in the NS.

3.3 LoRaWAN Protocol & MAC Commands

The LoRa Alliance has standardized the open-source LoRaWAN protocol that is used above the LoRa physical layer [10]. The LoRaWAN MAC protocol provides the network with the MAC commands that are found in the Table 3.2. These commands provide a wide range of functionalities for the NS, with which the NS can configure radio parameters of EDs [4]. Having access to these functionalities enables the NS to control the network, which ultimately allows the implementation of ADR algorithms [4]. As mentioned before, the LoRaWAN protocol is an ALOHA-like protocol and makes no use of LBT and CSMA mechanisms [2]. ALOHA-like networks are known for their limited scalability, because they rapidly perform worse in terms of reliability when the network load increases [29].

Command	Description
LinkCheck1	has the purpose of validating the connectivity of the device to the network
LinkADR	used to request to the end-device to change data-rate, transmit power, repetition rate or channel
DutyCycle	allows to set the maximum duty-cycle of a device for transmission
RXParamSetup	used to change the reception parameters of the device
DevStatus	used by the network server to reset the status of the device
NewChannel	allows to modify the definition of the radio channel parameters
RXTiming	used to setup the time slots for reception by the device
TXParam	used to change the transmission parameters
DIChannel	allows to create an asymmetric channel by shifting the downlink frequency band with respect to the uplink one (otherwise they have the same band)

Table 3.2: LoRaWAN supported MAC Commands [4]

3.4 Specification of the State-Of-the-Art (SotA) ADR Algorithm

Coutaud et al. [3] specified an ADR algorithm that they call ADR_{opt}. ADR_{opt} dynamically adjusts radio parameters, such that the network gets the most out of the available radio links [3]. More specifically, ADR_{opt} extrapolates a presumable PER (PER_{predic}) for each pair [SF; Nb_{trans}] from the observation on the channel over the previous transmission period [3]. Then ADR_{opt} calculates which pair [SF; Nb_{trans}] will yield the optimal result. This is done by choosing the pair that fits into a PER interval between zero and the upper PER threshold (PER_{max}) that is defined by their used FEC technique. The idea is that the FEC layer still reaches full recovery when the calculated PER_{predic} is below the PER_{max} [3]. If multiple pairs of [SF; Nb_{trans}] result in a value within the PER interval, the pair [SF; Nb_{trans}] with the highest PER_{predic} is chosen. In the work of Couteaud et al. [3] the PER_{max} is set to 0.3, while it is set to 0.6 in this thesis. The reason for this difference is that this thesis' ns-3 simulations use a log-distance path loss

model instead of a model that represents a Rayleigh channel.

The NS registers received frames and saves information about the last 20 received packets. The frame counters of the last 20 received packets reveal lost packets when the sorted frame counter list contains a gap. For instance, in the frame counter list [0, 2, 3, . . . , 19, 20] the frame counter '1' is missing. The NS then concludes that all frames of the packet number '1' were lost. This packet loss detection gives the NS insight into the signal quality ($PER_{current}$) between a node and the GWs that are within the node's reach. If the NS notices that a list of a particular node is missing too many frame counters, the NS will then send a downlink transmission to this specific node. The downlink transmission contains an ADR bit within the header, which informs the ED that the ADR mechanism would like to change the radio parameters of that node. The downlink transmission contains information about the chosen pair [SF; Nb_{trans}] that the nodes will adjust their radio parameters to.

It follows the pseudo-code of the SotA Server and SotA ADR algorithm. The code lays the foundation of the ns-3 implementation of the SotA ADR specification, which then is evaluated in **Section 5.2**. The pseudo-code is based heavily on the original specification of Couteaud et al. [3]:

Algorithm 1: SotA ADR Algorithm

```
ChHistory(20) // Initialization of the list of the last 20 frames (with unique frame
  counter) received;
while (true) do
    ACK_Req = waitRx();
    if (ACK_Req) then
        TxParameters = executeSotA(ChHistory) // executes the SotA ADR Function;
        changeNodeTxParameters(TxParameters);
    end
end
```

Algorithm 2: SotA ADR Function

INPUT: ChHistory
OUTPUT: [SF_{new}; $Nb_{trans_{new}}$]
PER_{max} = 0.6;
$PER_{current}$ = getPER(ChHistory);
PER_{new} = -inf;
SF_{new} ;
$Nb_{trans_{new}}$;
for $SF \in \{7, 8, 9, 10, 11, 12\}$ **do**
  for $Nb_{trans} \in \{1, 2, 3\}$ **do**
    PER_{predic}[SF; Nb_{trans}] = 1;
    for *GW* $\in$ *receptionGW(ChHistory)* **do**
      PER_{predic}[SF; Nb_{trans}] *= (calculateFER(SF, $PER_{current}$))$^{Nb_{trans}}$;
    end
    if *((PER_{new} < PER_{predic}) && (PER_{predic} < PER_{max}))* **then**
      PER_{new} = PER_{predic}[SF; Nb_{trans}];
      SF_{new} = SF;
      $Nb_{trans_{new}}$ = Nb_{trans};
    end
  end
end
if *((0 < PER_{new}) && (PER_{new} < PER_{max}))* **then**
  return [SF_{new}; $Nb_{trans_{new}}$];
else
  return null;
end

Below are the formulas which are used for the calculation of $PER_{predic<Nb_{trans},SF>}$ (PER at the NS which is extrapolated for each pair [SF; Nb_{trans}]). They were elaborated in the work of Couteaud et al. [3], but some variables have slightly different names in this thesis. First, the $PER_{current}$ is calculated, which leads to the calculation of the $SNR_{correction}$. The NS estimates the current SNR ($\widehat{SNR}$) by subtracting the $SNR_{correction}$ from the maximal SNR that was experienced by all GWs.

$$PER_{current} = \frac{(Number\ of\ packet\ loss)}{(Number\ of\ packet\ loss) + 20} \qquad size_S = \frac{20}{(1 - PER_{current})} * Nb_{trans}$$

$$CDF_{exp}^{-1}(x) = -ln(1-x) \qquad CDF_{exp}(x) = 1 - e^{-x}$$

$$SNR_{correction} = \frac{10 * \log_{10}(CDF_{exp}^{-1}(0.95^{1/size_S})}{2} + \frac{10 * \log_{10}(CDF_{exp}^{-1}(0.05^{1/size_S})}{2}$$

$$\widehat{SNR} = maxSNR_{GW}(ChHistory) - SNR_{correction}$$

Then the estimated SNR is used to calculate the Frame Error Rate (FER) for one transmission at a single GW ($FER_{<GW_i,SF>}$). Considering that multiple frames can be transmitted for one packet, the PER at the GW is $(FER_{<GW_i,SF>})^{Nb_{trans}}$. Finally, the $PER_{<Nb_{trans},SF>}$ at the NS is the multiplication of all PERs from all GWs that are in reach of the transmitting node.

$$SNR_{floor} = (-20) + ((12 - SF) * 2.5)$$

$$FER_{<GW_i,SF>} = CDF_{exp}(10^(SNR_{floor} - \widehat{SNR}))$$

$$PER_{<Nb_{trans},SF>} = \prod_{\forall GW_i} (FER_{<GW_i,SF>})^{Nb_{trans}}$$

3.5 ADR-1 Specification

The evaluation of the SotA algorithm reveals that network congestion must also be considered as a cause of packet loss. The first solution is ADR-1, which adds a collision avoidance feature on top of the SotA algorithm. By saving the timestamps of received frames, the NS can calculate the period between packets (TxTime). The number of received frames (per GW) with the same frame counter defines the minimal Nb_{trans} that was used by a node. Since the NS also registers the used SF for each packet, the NS holds all the information about the radio parameters of a node, as these are crucial for the calculation of the network load contribution.

The NS is now able to calculate the number that represents the current network load. By having a representative number for network load G, the NS can predict the collision probability in the network by using the Collision-Rate-Prediction-Line in Figure 5.4 from the evaluation chapter. Since SFs offer a quasi-orthogonality in transmissions sent with different SFs, six network loads are distinguished by the SFs. The NS ultimately calculates the collision probability of a packet for a given SF.

The idea of ADR-1 is that, on top of the SotA ADR mechanism, the NS detects collision situations and decreases the network load reactively, such that the overall PER decreases (i.e. PSR increases). Decreasing the network traffic is possible without modifying a node's signal quality if TxTime (the period between packets) is included as a modifiable radio parameter. If a node is transmitting within a SF group that has a high collision probability, then the TxTime of the node is increased from TxTime $\in \{9s, 10s, 11s, 12s, 13s\}$ to 50s. In return, the network load decreases because transmissions are sent less often within the network. Then the NS calculates the new radio parameters (SF and Nb_{trans}) of a node with the SotA ADR mechanism. The new SF is only sent to the node if it does not yield a higher collision probability than the old SF. Additionally, the new Nb_{trans} is not sent if the current collision probability is above the probability threshold of 20%. The implementation of the ADR-1 algorithm is evaluated in **Section 5.3**.

Algorithm 3: ADR-1 Algorithm

```
ChHistory(20) // Initialization of the list of the last 20 frames (with unique frame
  counter) received;
while (true) do
    ACK_Req = waitRx();
    if (ACK_Req) then
        if (isCollisionScenario(SF)) then
            increaseTxTime(50s);
            return;
        end
        TxParameters = executeSotA(ChHistory) // executes the SotA ADR Function;
        if (isAllowed(TxParameters)) then
            changeNodeTxParameters(TxParameters);
        end
    end
end
```

3.6 ADR-2 Specification

The evaluation of ADR-1 shows that LoRaWAN networks can be in a state in which parts of the network experience a packet collision rate of nearly 100%. Those networks are not manageable by the NS and a completely new ADR approach needs to be considered because LoRaWAN networks with N$\geq$1200 are (at least partially) in such a state. With the given TxTimes $\in \{9s, 10s, 11s, 12s, 13s\}$, the network is handed to the NS in a 'broken' state for N$\geq$1200. A possible solution is that the initial network load of the network is substantially decreased, such that the network gets into the state of unrecoverability for N much higher than 1200.

Similar to ADR-1, the ADR-2 algorithm builds upon the SotA ADR algorithm as well. However, the idea is that EDs enter a network with a higher TxTime than the TxTime defined by the end-user. More specifically, the initial TxTime of a node is multiplied by α^2 and the NS decreases this TxTime if the predicted packet collision probability is below 20%. Decreasing the TxTime of a node is done by dividing the current TxTime by α for a maximum of two times. The ADR-2 algorithm sort of safely speeds up the network, whereas the ADR-1 algorithm tries to recover the network by slowing it down. The difference between ADR-1 and ADR-2 is also addressed in the evaluation of the ADR-2 algorithm in **Section 5.4**. In this thesis 3 is used for

α but other values will work too if α stays reasonably low ($\alpha \leq 20$). For instance, choosing α = 100 would cause the network to be extremely scalable but it takes roughly 10'000 times longer to activate ADR. The optimal value for α more or less depends on the scale of the network and the requirements of the end-user(s).

Algorithm 4: ADR-2 Algorithm

```
ChHistory(20) // Initialization of the list of the last 20 frames (with unique frame
  counter) received;
numberOfTxTimeDecrease = 0;
while (true) do
    ACK_Req = waitRx();
    if (ACK_Req) then
        if (!isCollisionScenario(SF)) then
            if (numberOfTxTimeDecrease < 2) then
                decreaseTxTime(α));
                numberOfTxTimeDecrease++;
                return;
            end
        end
        TxParameters = executeSotA(ChHistory) // executes the SotA ADR Function;
        if (isAllowed(TxParameters)) then
            changeNodeTxParameters(TxParameters);
        end
    end
end
```

3.7 ADR 3 Specification

In order to further evaluate ADR-2, the ADR-3 algorithm is specified (and evaluated in **Section 5.5**). It needs to be evaluated whether the ADR-2 algorithm benefits from the usage of the SotA ADR mechanism. The ADR-3 algorithm copies the ADR-2 algorithm but removes the usage of the SotA ADR functionality.

Algorithm 5: ADR-3 Algorithm

```
ChHistory(20) // Initialization of the list of the last 20 frames (with unique frame
  counter) received;
numberOfTxTimeDecrease = 0;
while (true) do
    ACK_Req = waitRx();
    if (ACK_Req) then
        if (!isCollisionScenario(SF)) then
            if (numberOfTxTimeDecrease < 2) then
                decreaseTxTime(α));
                numberOfTxTimeDecrease++;
                return;
            end
        end
    end
end
```

Chapter 4

Implementation

The main implementation of ADR algorithms in this thesis can be directly derived from the pseudo-code presented in the specification chapter. However, some functionalities need further explanation. More specifically, two mechanisms are required for the calculation of the number of packet losses and the measurement of the network load within the network. As a part of these implementations, this thesis provides two tools written in C++ that fulfill the mentioned functions.

4.1 Packet Loss Tool

As mentioned in the specifications, the NS needs to keep track of unique frame counters, such that the NS can register packet losses. Assuming exactly three GWs and the number of nodes never exceeds 2000, the NS initializes a 3-dimensional integer array (counterHistory) with all elements equal to zero. Moreover, another global array (LostHistory) is initialized that keeps track of the calculated number of packet losses per GW from counterHistory:

```
int counterHistory[2000][20][3] = {0};
int LostHistory[2000][3] = {0};
```

Every time the NS receives a frame which was forwarded by a GW, a callback is executed that causes a network-controller-component (this is where ADR is installed) to receive a frame copy and related information (status of the GW). For instance, a frame arrives at the NS and thus a copy of the frame is sent to the network-controller-component. The network-controller-component removes the header from the frame copy and inserts the frame counter into the counterHistory for a given node (nodeIndex) and GW (gwIndex). The counterHistory array can not contain duplicates and must be sorted in increasing order. Thus the insertion and packet loss calculation is done as follows:

```
int counter = extractFrameCounter(header);

for (int index=0; index<19;index++)
{
    // Find the smallest number in the array
    if (counterHistory[nodeIndex][index][gwIndex] >
        counterHistory[nodeIndex][index+1][gwIndex]){

        // Disable insertion of duplicates
        if(counterHistory[nodeIndex][index][gwIndex]==counter ||
            counterHistory[nodeIndex][index+1][gwIndex]==counter){
                break;
        }
        // Calculate the number of packets that were lost between the
        // last 20 received packets
        LostHistory[nodeIndex][gwIndex]= counter -
            counterHistory[nodeNumber][index+1][gwIndex]- 20;
        // Save the new frame counter
        counterHistory[nodeNumber][index+1][gwIndex] = counter;
        break;
    }

    // If the smallest number is the at index==0, then the code above
    // will never be executed. In such a case, the code below is reached
    // with index==18, since the next element is never smaller. It remains
    //  to be checked whether the last element in the array is no duplicate.
    if (index==18 && counterHistory[nodeIndex][19][gwIndex] != counter){

        if(counterHistory[nodeIndex][0][gwIndex] != 0){

            // Calculate the number of packets that were lost between the
            // last 20 received packets
            LostHistory[nodeIndex][gwIndex] =
                counter-counterHistory[nodeIndex][0][gwIndex] - 20;
        }
        // Save the new frame counter
        counterHistory[nodeIndex][0][gwIndex] = counter;
        break;
    }
}
```

Note that the number of lost packets is negative before 20 packets are successfully received for a given node and GW. This does not matter to the ADR algorithm because GWs will only be included in the calculation of PER_{predic} if they already forwarded at least 20 packets to the NS. $PER_{current}$ can then be calculated at any time within the simulation:

```
int numberOfLostPackets = LostHistory[nodeIndex][gwIndex];
double currentPER = numberOfLostPackets / (numberOfLostPackets + 20);
```

4.2 Network Load Measurement Tool

ADR-1, ADR-2, and ADR-3 provide collision avoidance features that require the measurement of the network load within the LoRaWAN network. In this thesis, the network load G is measured in Erlangs (Erl). The network load measured for a given simulation time can be calculated as follows:

$$G = \frac{(Number\ of\ transmitted\ frames) * (ToA\ /\ transmission)}{(Simulation\ time)}\ Erl$$

However, the NS must measure the network load at any point in time (i.e. independent of the simulation time). By rewriting the formula, the simulation time can be removed from the formula.

$$(Number\ of\ transmitted\ frames) = \frac{(Simulation\ time) * (Number\ of\ nodes)}{(TxTime/transmission)}$$

$$G = \frac{\frac{\cancel{(Simulation\ time)} * (Number\ of\ nodes)}{(TxTime/\cancel{transmission})} * (ToA\ /\ \cancel{transmission})}{\cancel{(Simulation\ time)}}\ Erl$$

$$= \frac{(Number\ of\ nodes) * (ToA)}{(TxTime)}\ Erl$$

Finally, the following equation describes the network load contribution by one node at any point in time. If Nb_{trans} is a modifiable radio parameter of nodes, then the NS just multiplies the network load contribution of a node by Nb_{trans}.

$$\frac{G}{(Number\ of\ nodes)} = \frac{(ToA)}{(TxTime)}\ \ Erl$$

Since the network loads are separated by SFs, the ToA can take six different values depending on the used SF. These ToA values are initialized in a double array (TimeOnAirTable). TxTime is calculated as the time difference between the last two subsequently received packets. As the last relevant parameter, Nb_{trans} values are saved in the NS as well.

```
// The time on air values for all SFs in seconds
// The ToA value of SF 7 would be 0.055552s
double TimeOnAirTable[6] =
    {0.055552,0.100864,0.185344,0.33792,0.643072,1.220608};

// txNodeFrequency[nodeIndex][0] current TxTime
// txNodeFrequency[nodeIndex][1] last timestamp in seconds
// txNodeFrequency[nodeIndex][2] last frame counter
double txNodeFrequency[2000][3];

int NbHistory[2000]; // initially all elements are '1'
```

```
int counter = extractFrameCounter(header);

if (counter==0){
    txNodeFrequency[nodeIndex][0]=0;
    txNodeFrequency[nodeIndex][1]=timestamp.GetSeconds();
    txNodeFrequency[nodeIndex][2]=counter;
}
else if(counter>txNodeFrequency[nodeIndex][2]){
    txNodeFrequency[nodeIndex][0] =
        (int)((timestamp.GetSeconds() - txNodeFrequency[nodeIndex][1]) /
        (frameCounter-txNodeFrequency[nodeIndex][2])+0.5);

    txNodeFrequency[nodeIndex][1]=timestamp.GetSeconds();
    txNodeFrequency[nodeIndex][2]=counter;
}
```

Finally, the network load contribution of a node can be calculated with the current TxTime, ToA, and SF at any time in the simulation:

```
    double networkLoadContribution =
        NbHistory[nodeIndex]*TimeOnAirTable[SF-7] /
        txNodeFrequency[nodeIndex][0];
```

Chapter 5

Evaluation

This chapter demonstrates general LoRaWAN network behavior, the performance of implemented ADR specifications, and the explanation of critical events of simulations that led to new specifications. Note that most figures demonstrate the performance of a network by showing the Packet Success Rate (PSR) which is directly related to the Packet Error Rate (PER). The relation between these expressions can be described by the following equation:

$$PSR = 1 - PER$$

To validate the evaluations in this thesis, every data point in a figure represents the average of four measurement values. Some figures also indicate the standard deviation as y-error bars, such that results can be compared more effectively.

5.1 Simulation Settings and Environment

The simulations were run on three virtual machines provided by the University of Zürich. The chosen operating system for the VMs is Ubuntu 20.04 LTS because Linux is generally recommended within the ns-3 community.

The simulation settings are split into three different simulation profiles. All profiles use the constant simulation parameters defined in Table 5.1. However, they differ in the usage from the simulation parameters in Table 5.2.

(Constant) Simulation Parameter	**Value**
Code Rate	$\frac{4}{5}$
Duty Cycle	10%
Number of Gateways	3
TxTime of EDs	random value of the set $\{9s, 10s, 11s, 12s, 13s\}$
Band Width	125'000 Hz
Uplink Transmission Power	14 dBm
Number of EDs	$N \in \{50, 100, \ldots, 1150, 1200\}$

Table 5.1: Simulation Parameters used in every Simulation

It must also be noted that each node is assigned a SF $\in \{7, 8, 9, 10, 11, 12\}$ and Nb_{trans} equal to 1 at the start of each simulation. The distribution of SFs is done in a realistic setting, in which end-users have no technical know-how of the LoRaWAN technology. Hence it is assumed that end-users initially set the SFs high instead of low, as nodes are more likely to immediately connect to a LoRaWAN network with a higher SF. This is because nodes using a higher SF also have a higher communication range. This is implemented by inserting an offset into the default SF allocation mechanism of the LoRaWAN module [14].

(Variable) Simulation Parameter	**Profile-1**	**Profile-2**	**Profile-3**
Simulation Time	1000s	1000s	3000s
Radius	1m	5000m	5000m
ADR Algorithm	No ADR	No ADR, SotA, ADR-1	ADR-2, ADR-3

Table 5.2: Variable Simulation Parameters used for the different Simulation Profiles

The LoRa channel that is used in the simulations is defined by two different models. The used propagation loss model is a log-distance path loss model. In this thesis, 3.76 is the value of the path loss exponent, whereas the reference distance is equal to 1m and the reference loss is equal to 7.7dB. As for the propagation delay model, the constant-speed propagation delay model is

used for all simulations. Furthermore, all EDs and GWs connected to the network are stationary and placed uniformly on the disc (with constant density) that is defined by the simulation radius. Finally, any GW in the simulations offers a single channel operating at 868 MHz.

5.2 SotA ADR Performance

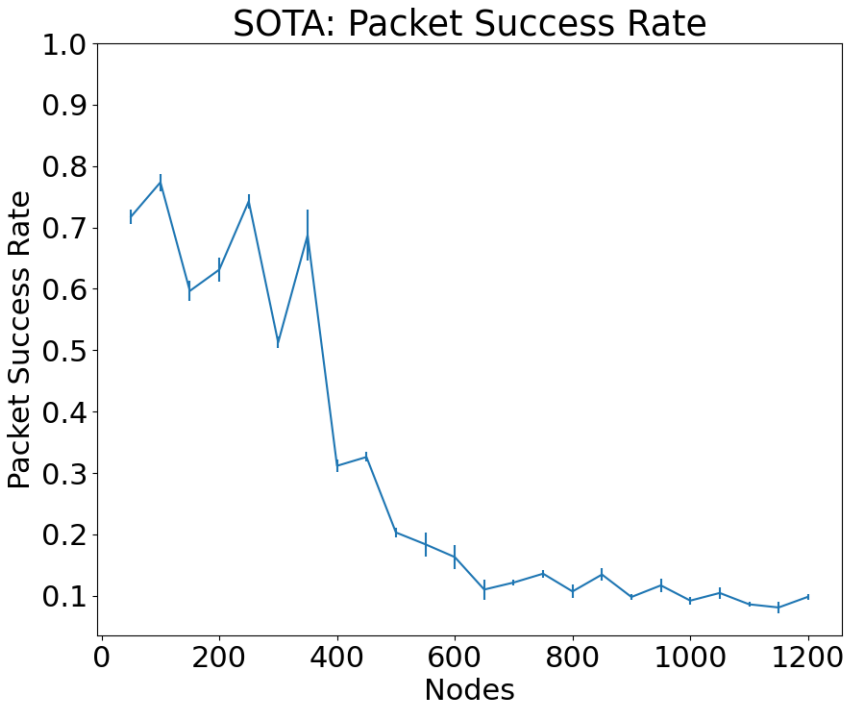

Figure 5.1: Performance of the SotA Algorithm with y-Error Bars

After this thesis' ns-3 implementation of the SotA ADR algorithm specified by Coutaud et al. [3], the performance of the SotA algorithm is elaborated in a Profile-2 simulation. Figure 5.1 shows the achieved PSR in dependency of the number of nodes (N). The standard deviation of the PSR is indicated with y-error bars. Compared to a LoRaWAN network without ADR, the SotA algorithm achieves an improved network performance of a slightly higher PSR (i.e. lower PER) than a default network (No ADR). This can be seen in Figure 5.12, but it must also be noted that the improvement value approximates 0 as the number of EDs in the network increases to a value higher than $\sim$800. This phenomenon is further analyzed in the following subsections.

5.2.1 Network Load Analysis

After observing the SotA algorithm lose value in highly loaded LoRaWAN networks, the basic behaviors of LoRaWAN networks are further investigated in order to understand the observed phenomenon. It is indicated that the number of nodes (N) impacts the performance of LoRaWAN networks in general, because increasing N worsens the default network as well. Thus several Profile-1 simulations without any ADR mechanism are conducted, such that the relation between N and the performance of LoRaWAN networks can be understood. For instance, LoRaWAN networks with EDs which only transmit with SF 7 (i.e. SF-7-only network) are simulated for an increasing number of EDs. Figure 5.2 displays the experienced network load by the NS (not to be mistaken with the actual network load). It can be seen that the network load proportionally increases with the number of nodes connected to the network.

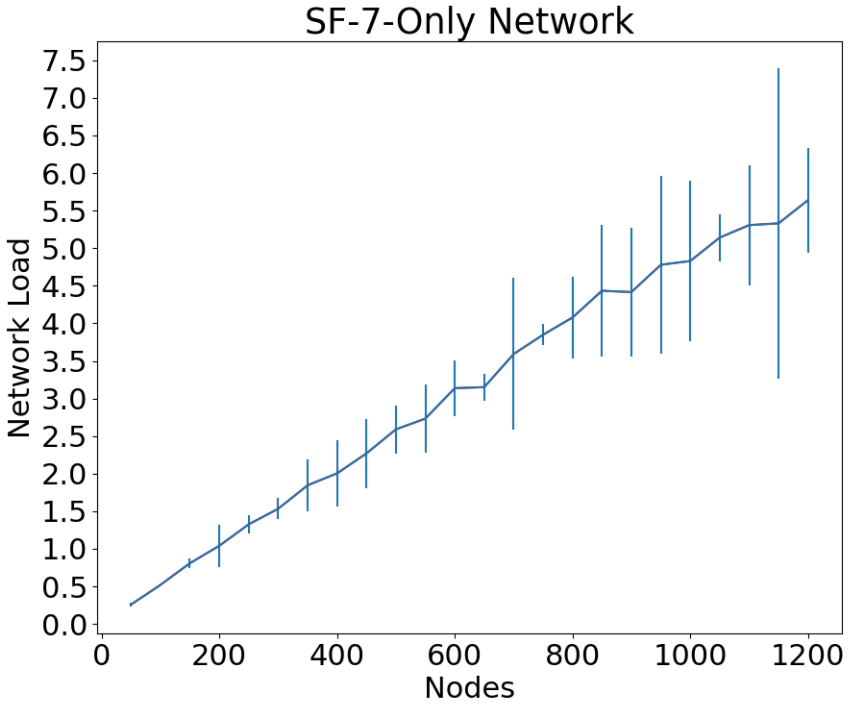

Figure 5.2: The experienced Network Load at the Network Server in a SF-7-only Network

Due to the fact that increasing the SF of transmissions decreases the data rate of transmissions, the ToA of transmissions increases as well. Therefore, the network load increases when the

average SF of EDs increases within the network. This is demonstrated in Figure 5.3. As one compares the two figures (Figure 5.2 and Figure 5.3), it can be seen that the NS in the SF-9-only network initially experiences a much higher network load compared to the NS in the SF-7-only network.

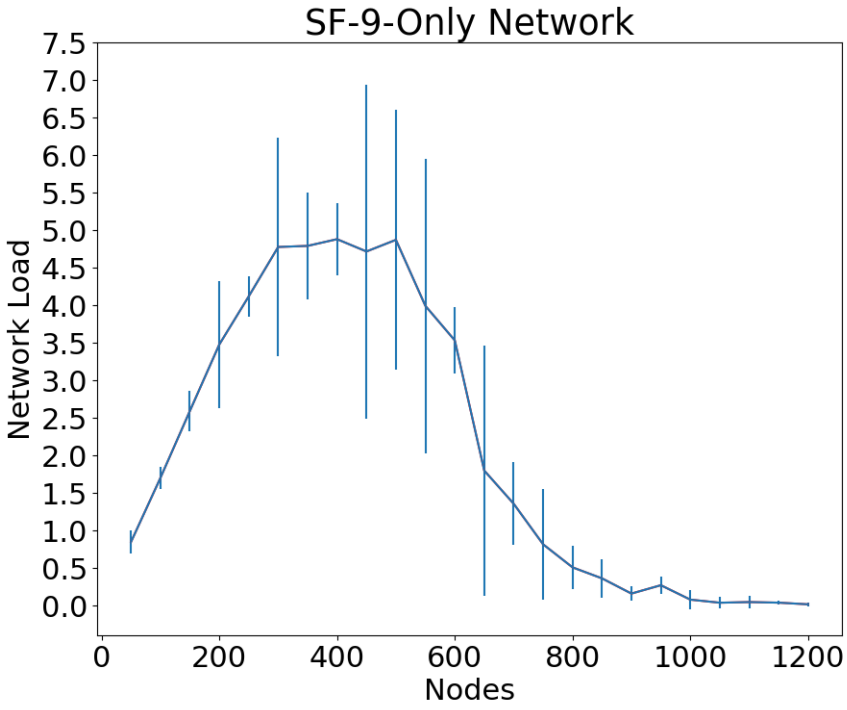

Figure 5.3: The experienced Network Load at the Network Server in a SF-9-only Network

The registered network load decreases at about N = 400 and approximates 0 for N $\geq$ 1000. This can be explained by an increasing number of packet losses in the network. The NS never receives the lost packets and therefore registers a network load lower than the actual network load. Normally, packet loss occurs when the network signals are too weak or when transmitted frames collide with each other. A collision occurs when two separately transmitted frames simultaneously arrive at a GW, thus causing an incorrect reception of those frames. As a consequence, the frames are dropped and the network suffers packet loss. Figure 5.4 displays the results of numerous Profile-1 simulations (SF-7-only, SF-9-only, and SF-11-only networks). More specifically, PSR is shown in dependency of the actual network load. As Profile-1 simulations set the network radius to one meter, the network connection between EDs and GWs

is very strong. Hence it is assumed that packet loss in Profile-1 simulations is only caused by network congestion (i.e. collisions). Thus it can be said that Figure 5.4 shows that an increasing network load significantly worsens the PSR (i.e. PER) of LoRaWAN networks.

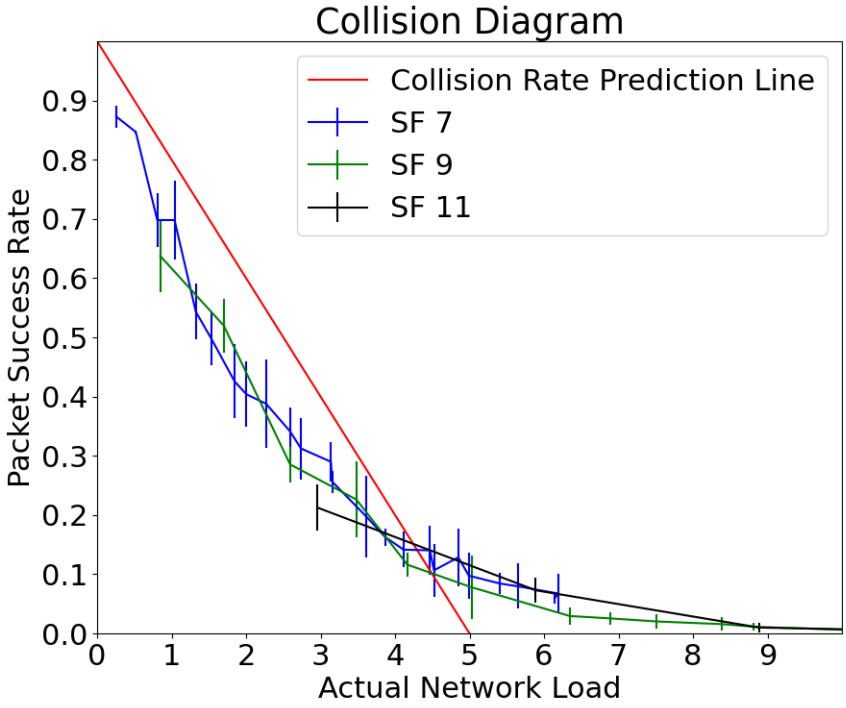

Figure 5.4: Packet Success Rate in Dependency of the actual Network Load

It is observable in Figure 5.4 that LoRaWAN networks fail to reliably operate if the actual network load increases too much. Assuming the end-user requires a PSR of 60%, the actual network load must not exceed $\sim$1 Erl for each SF. As a general trend of the graphs (SF 7, SF 9, and SF 11), a straight line is drawn that represents a rough estimation of the success probability of packets which are currently sent in a LoRaWAN network. The domain of this line's function (C) is $G \in [0,5]$ in Erlangs, whereas the codomain is simply $PSR \in [0,1]$. Network loads that reach values greater than five are simply set to five. The function is defined by the following equation:

$$C(G) = 1 - \frac{G}{5}$$

5.2.2 The Scalability Limit of the SotA ADR Algorithm

The simulations from the previous subsection delivered results that give insight into the problem of network congestion within a LoRaWAN network. Unless the network load is very low (i.e. $G \leq 0.5$ Erl), collisions generally should not be ignored in LoRaWAN networks. This explains why the SotA algorithm fails for highly loaded networks. The work of Coutaud et al. [3] implies the assumption that packet loss only occurs due to weak signals between EDs and GW. Thus the SotA ADR mechanism reacts to packet losses with downlink transmissions which are meant to enhance the signal between an ED and a GW. This is done by either increasing the SF or Nb_{trans}, which - however - also increases the network load G (Figure 5.2 and Figure 5.3). To sum it up, the SotA algorithm fails in collision situations (i.e. highly loaded networks), because the SotA ADR mechanism increases the network load and ultimately further increases the packet collision probability, as this is the main cause for packet loss in a collision scenario.

This degrades the network performance and makes the SotA algorithm questionable. This acquired knowledge enables the specification of ADR-1 that is based on SotA ADR, but additionally considers collisions as a cause of packet loss (see **Section 3.5**).

5.3 ADR-1 Performance

The evaluation of the SotA algorithm shows that the network traffic needs to be considered and that increases in SF or Nb_{trans} need to be supervised. If a collision scenario is detected in ADR-1, the NS first increases TxTime to decrease the network load. Subsequently, the NS executes the SotA ADR sub-algorithm and controls the returned values. More specifically, the NS allows an increase of SFs and Nb_{trans} if the collision probability is below 20%.

Taking a look at the results of the Profile-2 simulations with ADR-1 (Figure 5.12), it is visualized that ADR-1 and the SotA ADR algorithm perform equally well for $N \leq 300$. However, as N increases above that value, ADR-1 yields a worse PSR than the default network. This incident is further analyzed in the following subsection.

5.3.1 The Unrecoverability of LoRaWAN Networks

To explain the observed phenomenon, the behavior of ADR-1 is further analyzed in detail. Figure 5.5 showcases ADR-1 performing in a network with only 100 nodes. Furthermore, Figure 5.5 visualizes the experienced network load of SF 10, 11, and 12 which are effectively reduced by the operations of ADR-1. This is due to the fact that the EDs' TxTime is increased successfully, whereas some EDs reduce their SF to a lower value. Increasing the TxTime of a SF 12 node is effective since the ToA of transmissions increases exponentially with increasing SF. The result is a successful LoRaWAN network.

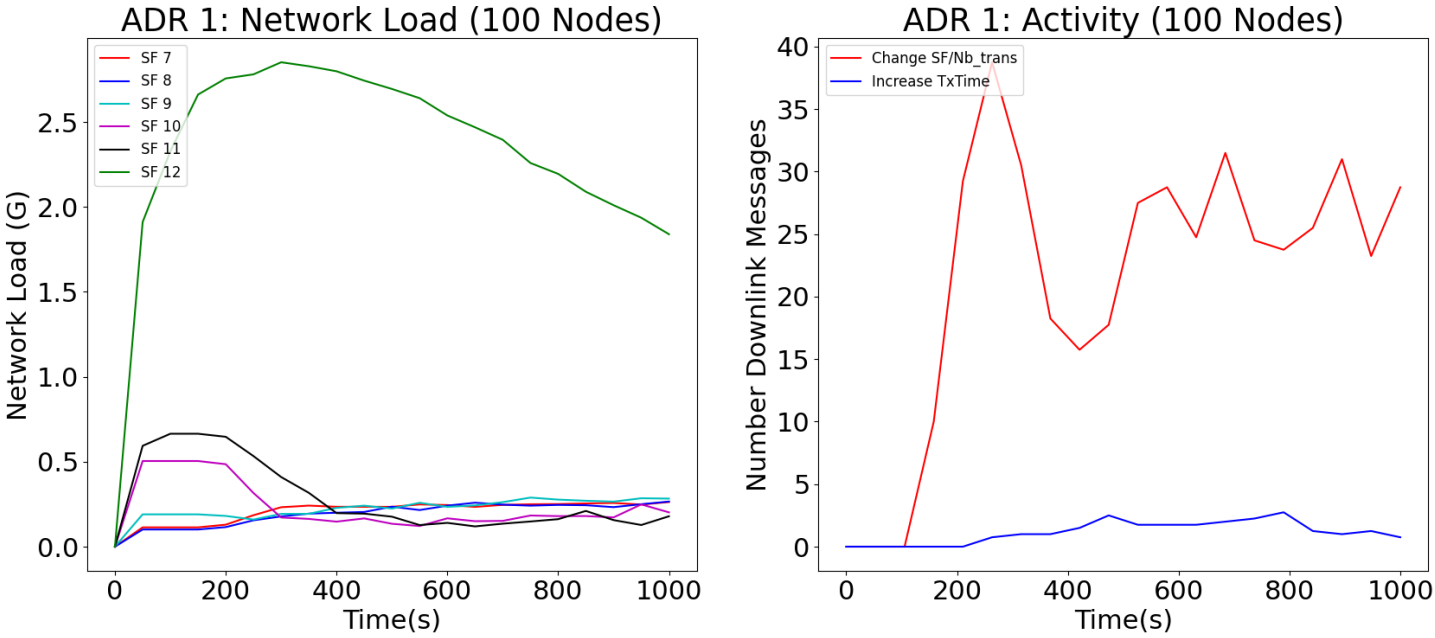

Figure 5.5: ADR-1 Network Activity with N=100

Still, ADR-1 performs poorly for N $\geq$ 300 (Figure 5.12). Figure 5.6 shows the experienced network loads in a highly loaded LoRaWAN network running with ADR-1. Similar to the SF-9-only network (Figure 5.3), the collision rate is so high, such that frames sent by the nodes rarely reach the NS. In Figure 5.6, the experienced network load of SF 12 is 0 Erl while the actual network load is above $\sim$30 Erl. According to the collision diagram (Figure 5.4), the collision probability is predicted to be 100% for a network load above 5 Erl.

Uplink and downlink transmissions sent with SF 12 never reach their destination, whereas ADR-1 does not even activate for nodes that do not transmit 20 packets successfully. The SF 12 group (i.e. the nodes transmitting with SF 12) is principally disconnected from the network. SF groups that operate with an acceptable actual network load below 5 Erl are slowed down by getting their TxTime increased by the NS if the network load for a given SF is above 1 Erl. By slowing down the successful nodes, without slowing down the unreachable nodes, ADR-1 inevitably achieves a lower PSR than the original SotA algorithm, for networks with high congestion.

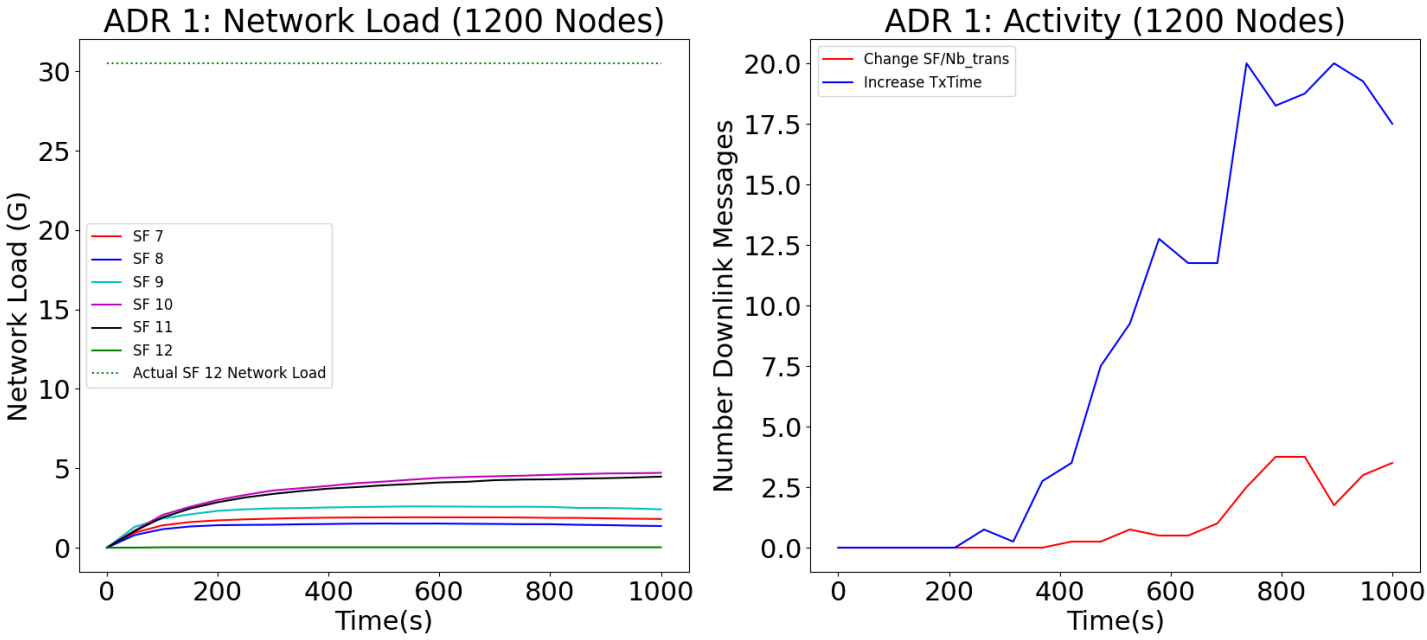

Figure 5.6: ADR-1 Network Activity with N=1200

The evaluation of the ADR-1 simulations reveals that LoRaWAN networks can reach a state in which they are unrecoverable. LoRaWAN networks in this state have such a high N for at least one group of SFs, such that no ADR mechanism can decrease the network load anymore. This means that any reactive ADR mechanism will fail in such a case because they activate only after successfully receiving at least 20 packets.

Additionally, downlink transmissions do not reach the nodes in the first place, if the collision probability is too high. A completely different approach to collision avoidance needs to be taken, leading to this thesis' specification of ADR-2 (see **Section 3.6**).

5.4 ADR-2 Performance

The ADR-2 algorithm takes a entirely different approach than the previously specified algorithm. A LoRaWAN network using the ADR-2 algorithm can detect and prevent collision scenarios much earlier for a given N. The reason for this improvement is that the initial network load is significantly reduced before the network starts running. This is implemented by having EDs multiply their required TxTime by α^2. Note that the value of α is set to 3 in this thesis. Instead of decreasing the transmission frequency (i.e. increasing TxTime) like in ADR-1, the ADR-2 mechanisms decrease the TxTime by α every time the NS allows such a modification (for a maximum of two times). Decreasing the TxTime of nodes is allowed if the predicted collision probability is below the threshold of 20%.

The results are obtained by running Profile-3 simulations for ADR-2. Note that the simulation time (3000s) is three times higher than the simulation time in Profile-1 and Profile-2 (1000s). The reasoning behind this is that EDs take α^2 times longer to successfully transmit the first 20 packets that enable ADR. The achieved PSR (see Figure 5.12) of the ADR-2 algorithm is substantially higher than the previously achieved network performances of ADR-1 and the SotA ADR. Figure 5.7 shows the results of the low-load LoRaWAN network running with ADR-2. After most nodes have successfully transmitted 20 packets, the ADR-2 mechanism starts decreasing the TxTime of EDs, which causes the network load to increase up to 1 Erl. Note that on the right side of Figure 5.7 barely any changes are made for SF and Nb_{trans}, which means that the SotA ADR functionality is rarely used.

The highly loaded LoRaWAN network using ADR-2 (Figure 5.8) also successfully manages EDs. It can be seen for all SFs (except SF 12) that the network load is increased to 1 Erl

as defined by the network. The initial network load of SF 12 is unfortunately already higher than 1 Erl, although the SF 12 nodes transmit α^2 times slower than required by the end-user. The extremely high network load contribution of SF 12 nodes is also demonstrated in Figure 5.6, where the default nodes reach an initial (actual) network load of about 30 Erl. Unlike when ADR-1 is used, the NS in the highly loaded network using ADR-2 is still able to receive packets from SF 12 nodes. For validation's sake, a last ADR algorithm (ADR-3) is specified in **Section 3.7**, which excludes the SotA ADR functionality from ADR-2.

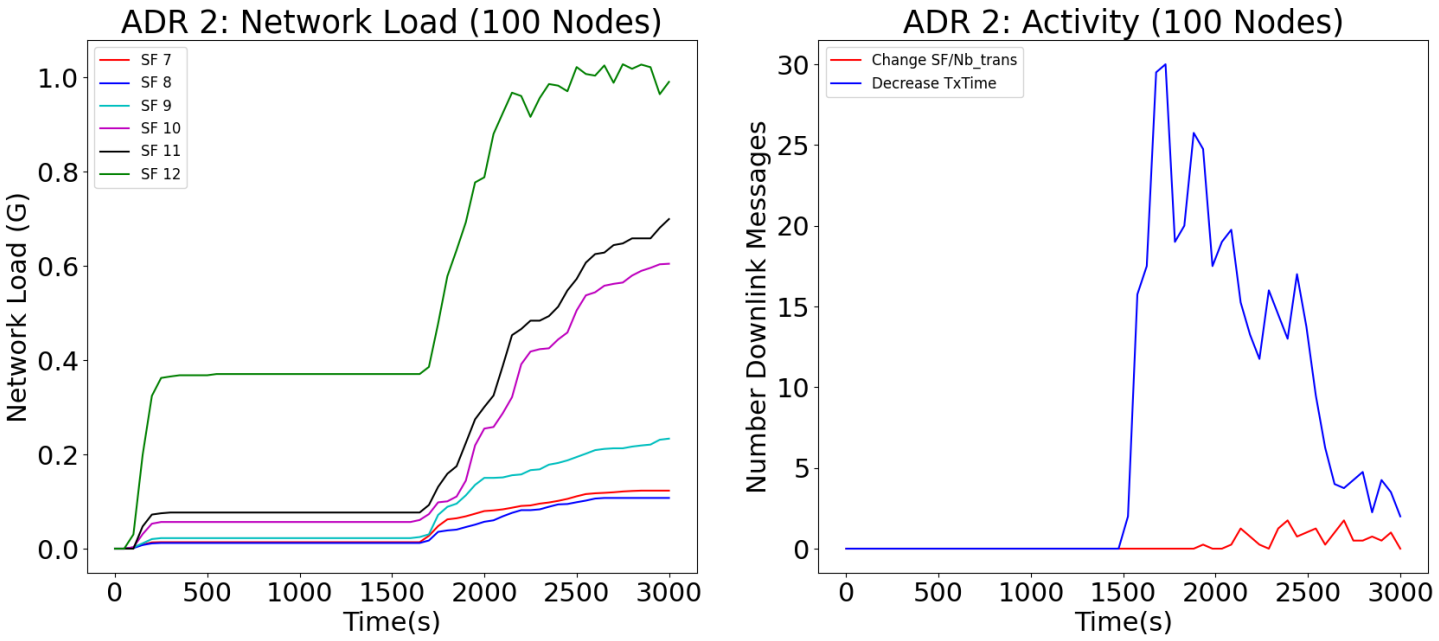

Figure 5.7: ADR-2 Network Activity with N=100

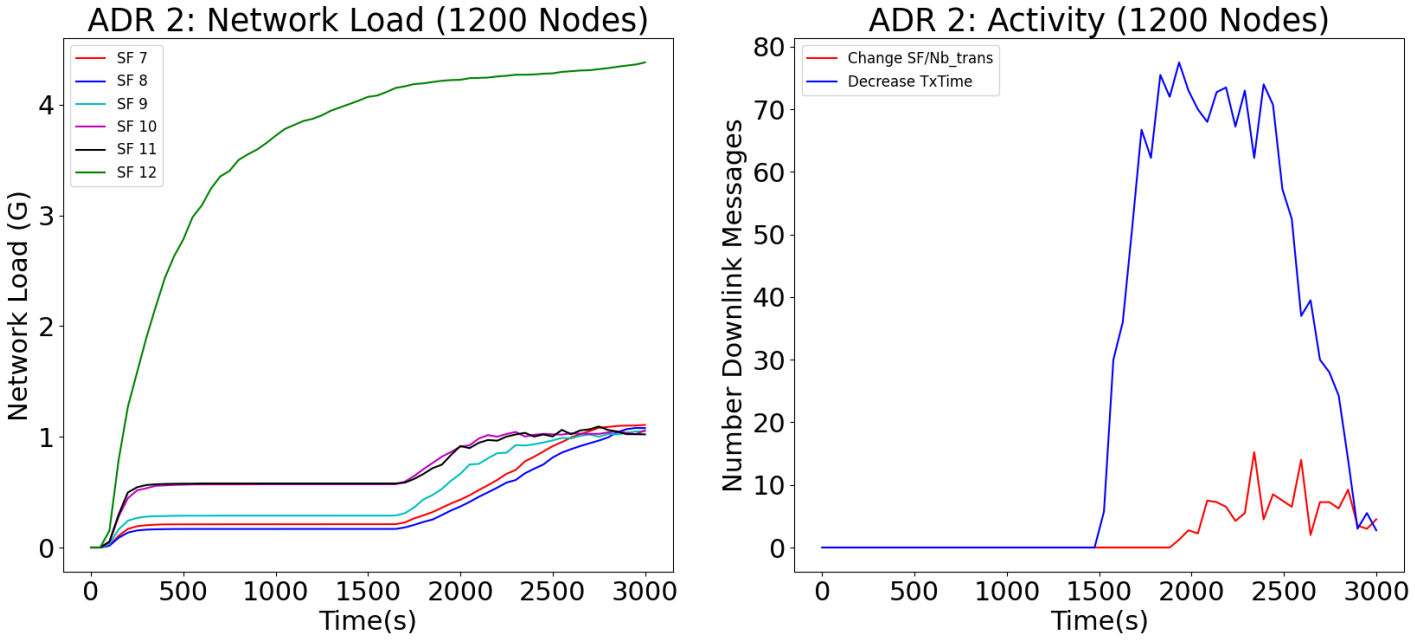

Figure 5.8: ADR-2 Network Activity with N=1200

5.5 ADR-3 Performance

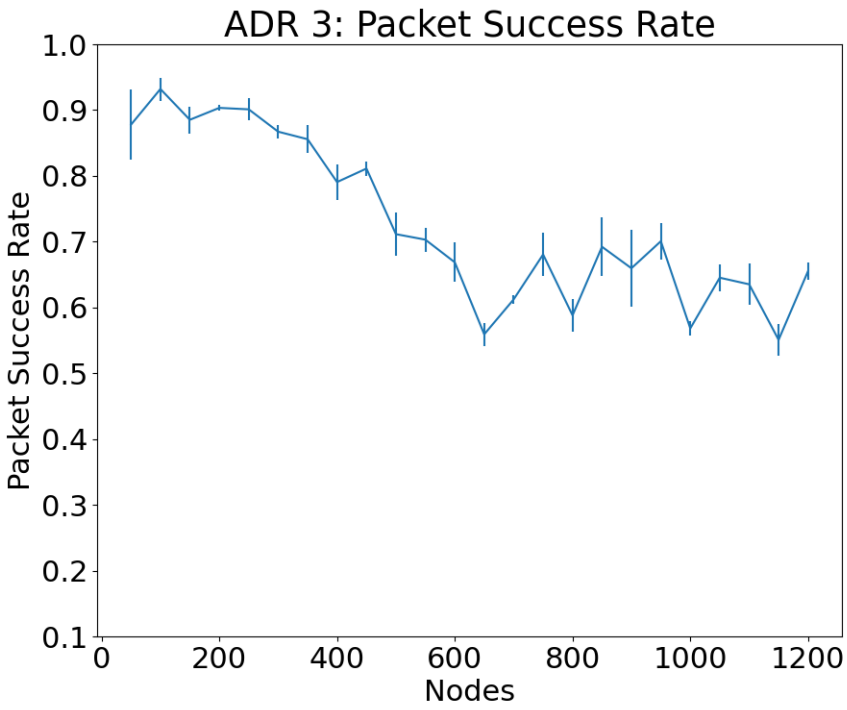

Figure 5.9: Performance of the ADR-3 Algorithm with y-Error Bars

As a simplified version of ADR-2 (without SotA ADR functionality), ADR-3 yields similar results as ADR-2. Comparing the two, the ADR-3 algorithm achieves a slightly higher PSR for any given N. This can be seen in Figure 5.12. The activity of networks using ADR-3 (Figure 5.10 and Figure 5.11) also look similar to ADR-2, but no changes in SF and Nb_{trans} can be seen, as the SotA ADR functionality has been removed.

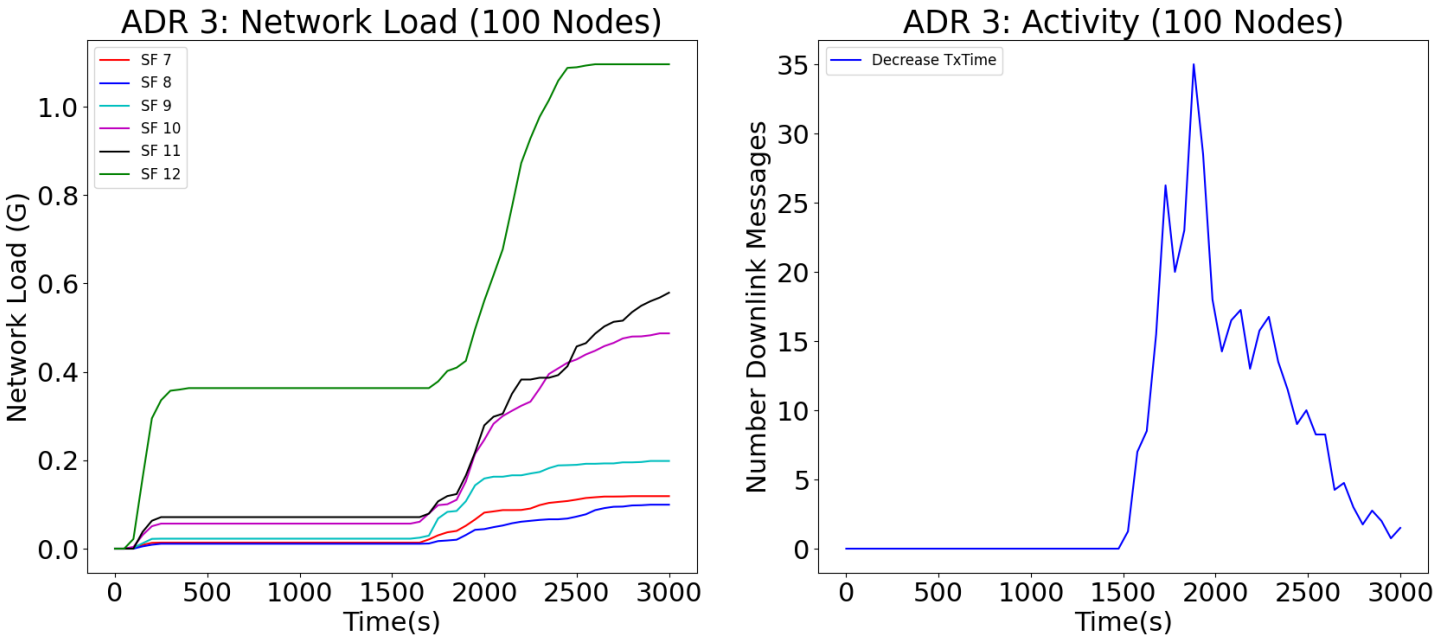

Figure 5.10: ADR-3 Network Activity with N=100

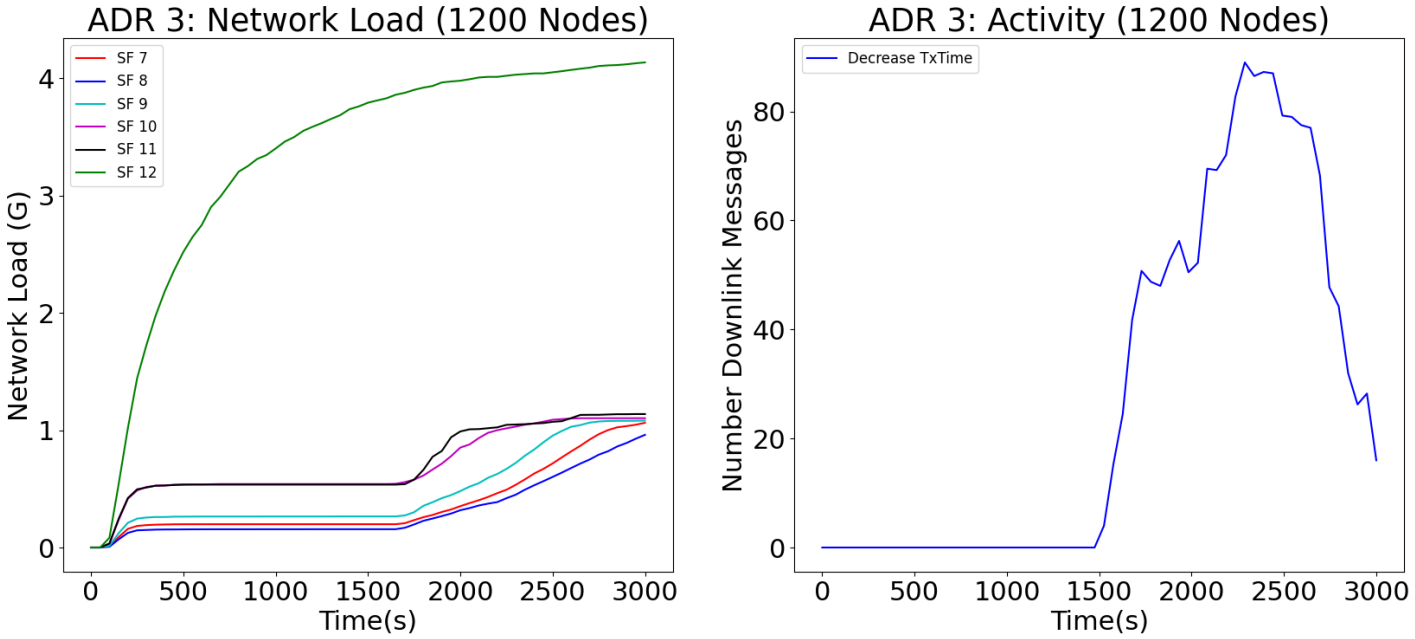

Figure 5.11: ADR-3 Network Activity with N=1200

5.6 Final ADR Comparison

This thesis contains three specifications for three novel ADR algorithms. The implementation of these algorithms and the SotA ADR algorithm yield comparable results. Figure 5.12 displays the PSR of the implementations in dependency of the number of nodes. The figure indicates that the achieved PSR is much higher for ADR-2 and ADR-3, whereas ADR-1 and the SotA ADR only show minor improvements over the default network. Note that the standard deviations shown in Figure 5.1 and Figure 5.9 demonstrate that the performance difference between SotA ADR and ADR-3 is compelling.

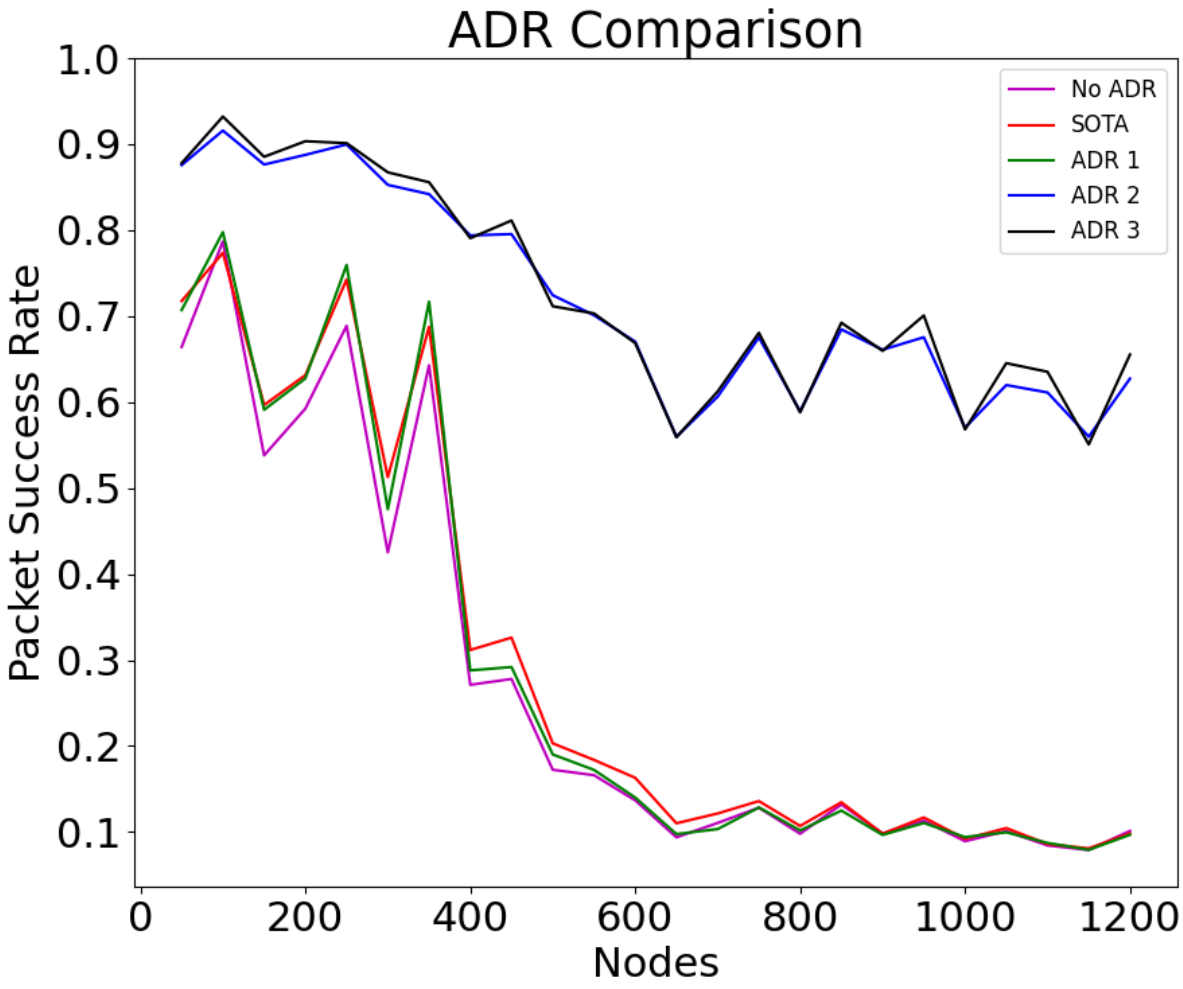

Figure 5.12: ADR Comparison

Chapter 6

Summary and Conclusion

This thesis provides a study of Long Range (LoRa) Wide Area Network (WAN) and Adaptive Data Rate (ADR) algorithms in the ns-3 environment. As the first contribution, a State of the Art (SotA) ADR algorithm [3] is implemented in a LoRaWAN ns-3 module [14], which enables realistic simulations of LoRaWAN networks. It is shown in the simulation that the chosen SotA ADR scheme yields minor incremental improvements, *e.g.*, an increased PSR of around 20% in comparison to regular LoRaWAN networks (i.e., without ADR in uncongested networks). Furthermore, this ADR scheme is not scalable towards larger networks with significant congestion, while it might even provide lower PSR in some situations in comparison to regular LoRa networks. This work experienced a reduced PSR of -3% for a highly congested network using the SotA ADR in comparison to a regular LoRa stack.

The second contribution is the specification, implementation, and evaluation of three novel ADR algorithms, which use different collision avoidance features. The simulations show that LoRaWAN networks finding themselves in highly congested situations can not successfully recover from congestion anymore and, therefore, leave no room for optimization. This is due to the fact that the absolute Packet Success Rate (PSR) is very close to 0 in such a case, and the NS is unable to alleviate the congestion on End Devices (EDs), while the Gateway (GW) is not able to receive from or send to congested EDs. This finding led to the development of ADR-2 and

ADR-3, which require a minimal initial network load before the network starts operating under the control of SotA ADR mechanisms. This way, the NS is still able to control and support a higher number of nodes which would otherwise cause LoRaWAN networks to malfunction.

This thesis demonstrates that ADR-2 and ADR-3 yield substantial improvements, *e.g.*, an increased PSR of around 100% in comparison to regular LoRaWAN networks (i.e., without ADR in uncongested networks). In contrast to the chosen SotA ADR scheme, the newly developed ADR schemes (i.e. ADR-2 and ADR-3) are scalable towards larger networks with high congestion. The simulations show that highly congested networks using either ADR-2 or ADR-3 have a substantially increased PSR of around 500% in comparison to a regular LoRa stack.

6.1 Future Work

In the scope of this thesis, future work could experimentally verify this thesis' ADR specifications and modify them with additional features (*e.g.*, reducing load through dynamic duty-cycle mechanisms). The implementation of this work is not yet at the running prototype level ready for the development of future Internet-of-Things (IoT) applications on a large scale. Furthermore, future applications may have additional requirements that this work fails to address (e.g., real-time traffic).

Generally, the LoRaWAN technology is very limited by the ALOHA-like Medium Access Control (MAC) protocol. Current LoRaWAN research already tries to integrate various collision avoidance features, such as Listen Before Talk (LBT) [26], Carrier-Sense Multiple Access (CSMA) [28], slotted ALOHA [20], and machine learning techniques [5,17]. However, in order for LoRaWAN to support the number of nodes that are currently connected to cellular networks like in the 3rd Generation Partnership Project (3GPP) 4G/5G, further research is needed in the area of LoRaWAN scalability.

Bibliography

[1] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne, "Understanding the Limits of LoRaWAN," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 34–40, 2017.

[2] A. Augustin, J. Yi, T. Clausen, and W. M. Townsley, "A Study of LoRa: Long Range & Low Power Networks for the Internet of Things," *Sensors*, vol. 16, no. 9, p. 1466, 2016.

[3] U. Coutaud, M. Heusse, and B. Tourancheau, "Adaptive Data Rate for Multiple Gateways LoRaWAN Networks," in *2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2020, pp. 1–6.

[4] F. Cuomo, M. Campo, A. Caponi, G. Bianchi, G. Rossini, and P. Pisani, "EXPLoRa: Extending the Performance of LoRa by Suitable Spreading Factor Allocations," in *2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2017, pp. 1–8.

[5] F. Cuomo, D. Garlisi, A. Martino, and A. Martino, "Predicting LoRaWAN Behavior: How Machine Learning Can Help," *Computers*, vol. 9, no. 3, p. 60, 2020.

[6] C. Goursaud and J.-M. Gorce, "Dedicated Networks for IoT: PHY/MAC State of the Art and Challenges," *EAI endorsed transactions on Internet of Things*, 2015.

[7] M. Guerrero, C. Cano, X. Vilajosana, and P. Thubert, "Towards Dependable IoT via Interface Selection: Predicting Packet Delivery at the End Node in LoRaWAN Networks," *Sensors*, vol. 21, no. 8, p. 2707, 2021.

[8] M. Hanif and H. H. Nguyen, "Frequency-Shift Chirp Spread Spectrum Communications with Index Modulation," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[9] J. Haxhibeqiri, F. Van den Abeele, I. Moerman, and J.Hoebeke, "LoRa Scalability: A Simulation Model Based on Interference Measurements," *Sensors*, vol. 17, no. 6, p. 1193, 2017.

[10] R. Kufakunesu, G. P. Hancke, and A. M. Abu-Mahfouz, "A Survey on Adaptive Data Rate Optimization in LoRaWAN: Recent Solutions and Major Challenges," *Sensors*, vol. 20, no. 18, p. 5044, 2020.

[11] S. K. Lee, M. Bae, and H. Kim, "Future of IoT Networks: A Survey," *Applied Sciences*, vol. 7, no. 10, p. 1072, 2017.

[12] LoRa Alliance, "A Technical Overview of LoRa® and LoRaWAN™," https://lora-alliance.org/resource_hub/what-is-lorawan/, 2015, accessed: 2021-08-30.

[13] ——, "LoRaWAN® L2 1.0.4 Specification," https://lora-alliance.org/resource_hub/lorawan-104-specification-package/, 2020, accessed: 2021-08-30.

[14] D. Magrin, M. Centenaro, and L. Vangelista, "An NS-3 Module for Simulation of LoRaWAN Networks," https://github.com/signetlabdei/lorawan, accessed: 2021-08-30.

[15] ——, "Performance Evaluation of LoRa Networks in a Smart City Scenario," in *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 1–7.

[16] E. Morin, M. Maman, R. Guizzetti, and A. Duda, "Comparison of the Device Lifetime in Wireless Networks for the Internet of Things," *IEEE Access*, vol. 5, pp. 7097–7114, 2017.

[17] C. Moy, "IoTligent: First World-Wide Implementation of Decentralized Spectrum Learning for IoT Wireless Networks," in *2019 URSI Asia-Pacific Radio Science Conference (AP-RASC)*, 2019, pp. 1–4.

[18] NS-3 Consortium, "A Discrete-Event Network Simulator for Internet Systems," https://www.nsnam.org/, accessed: 2021-08-30.

[19] OMNeT++, "What is OMNeT++?" https://omnetpp.org/intro/, accessed: 2021-08-30.

[20] T. Polonelli, D. Brunelli, A. Marzocchi, and L. Benini, "Slotted ALOHA on LoRaWAN-Design, Analysis, and Deployment," *Sensors*, vol. 19, no. 4, p. 838, 2019.

[21] H. Rahimi, A. Zibaeenejad, and A. A. Safavi, "A Novel IoT Architecture Based on 5G-IoT and Next Generation Technologies," in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2018, pp. 81–88.

[22] U. Raza, P. Kulkarni, and M. Sooriyabandara, "Low Power Wide Area Networks: An Overview," *IEEE Communications Surveys Tutorials*, vol. 19, no. 2, pp. 855–873, 2017.

[23] Semtech Corporation, "LoRa® and LoRaWAN®: A Technical Overview," https://lora-developers.semtech.com/library/tech-papers-and-guides/lora-and-lorawan/, accessed: 2021-08-09.

[24] Sigfox, "Sigfox's Ecosystem Delivers the World's First Ultra-low Cost Modules to Fuel the Internet of Things Mass Market Deployment," https://www.prnewswire.com/news-releases/sigfoxs-ecosystem-delivers-the-worlds-first-ultra-low-cost-modules-to-fuel-the-internet-of-things-mass-market-deployment-600382181.html, accessed: 2021-08-10.

[25] M. Slabicki and G. Premsankar, "FLoRa - A Framework for LoRa Simulations," https://flora.aalto.fi/, accessed: 2021-08-30.

[26] T. Surbeck, "Simulation and Efficiency Improvements of the IoT Communication Protocols Used in the Supply Chain Monitoring Systems," 2019.

[27] T. To and A. Duda, "LoRa Module on NS-3," https://github.com/drakkar-lig/lora-ns3-module, accessed: 2021-08-30.

[28] ——, "Simulation of LoRa in NS-3: Improving LoRa Performance with CSMA," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–7.

[29] F. Van den Abeele, J. Haxhibeqiri, I. Moerman, and J. Hoebeke, “Scalability Analysis of Large-Scale LoRaWAN Networks in NS-3,” *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 2186–2198, 2017.

Abbreviations

IoT	Internet of Things
SotA	State of the Art
LPWAN	Low Power Wide Area Network
LoRaWAN	Long Range Wide Area Network
MAC	Medium Access Control
QoS	Quality of Service
SF	Low Power Wide Area Network
Nb_{trans}	The number of transmissions that an end device sends for one packet
TxTime	Transmission Time - The time period between two subsequent packets of a node
P_{tx}	Transmission Power
CR	Code Rate
BW	Bandwidth
PER	Packet Error Rate
PSR	Packet Success Rate
FER	Frame Error Rate
GW	Gateway
ED	End Device
NS	Network Server
N	Number of nodes
CCS	Chirp Spread Spectrum
FEC	Forward Error Correction

LBT	Listen Before Talk
CSMA	Carrier Sense Multiple Access

Glossary

to slow down : to decrease the transmission frequency of end devices

to speed up : to increase the transmission frequency of end devices

to suffer from something : being in a state, in which something degrades one's technical performance

List of Figures

List of Tables

Appendix A

Installation Guidelines

To run simulations with this thesis' source code, the ns-3 framework needs to be installed. Ns-3 can be installed by following the setup instructions on the ns-3 web-site [18]. It is recommended to work with a Linux operating system. Alternatively, Windows10 users can install the Windows Subsystem for Linux (WSL) that works as well.

A.1 Installation ns-3

This thesis uses the latest release of ns-3 (ns-3.34) and it is downloaded using git. Before building ns-3 with the Waf tool, the **NS3_LoRaWAN_SourceCode.zip** (see appendix) needs to be opened. Unzip this zip file and save the content in the directory where ns-3 was cloned in. The file structure around your clone should now look like this:

```
/
└── your_github_repositories
    ├── ns-3-allinone
    │   └── ns-3-dev
    │       ├── scratch
    │       ├── src
    │       └── ...
    └── NS3_LoRaWAN_SourceCode
        ├── lorawan
        ├── scratch
        └── bash-scripts
```

Enter the **your_github_repositories** directory and execute the following commands. They copy the relevant source files of this thesis into the installed ns-3 framework:

```
$ rsync -av NS3_LoRaWAN_SourceCode/lorawan/* ns-3-allinone/ns-3-dev/src/lorawan
$ rsync -av NS3_LoRaWAN_SourceCode/bash-scripts* ns-3-allinone/ns-3-dev
$ rsync -av NS3_LoRaWAN_SourceCode/scratch/* ns-3-allinone/ns-3-dev/scratch
```

A.2 Build with Waf

Now the source code of the complete framework needs to be compiled. Enter into the **ns-3-dev** directory and execute the commands below. These commands configure the Waf tool, enable logging in the previously copied source files, and build ns-3:

```
$ ./waf configure --build-profile=debug --enable-examples --enable-tests
$ export 'NS_LOG=Algorithm1=level_all|prefix_time'
$ export 'NS_LOG=Adr1=level_all|prefix_time'
$ export 'NS_LOG=Adr2=level_all|prefix_time'
$ export 'NS_LOG=Adr3=level_all|prefix_time'
$ export 'NS_LOG=Adr4=level_all|prefix_time'
$ export 'NS_LOG=Adr_Nothing=level_all|prefix_time'
$ ./waf
```

A.3 Simulate LoRaWAN Networks

Every simulation of this thesis can be run with the Algorithm1.cc simulation script that was copied into the scratch directory. From the **ns-3-dev** directory every simulation can be run with this script:

```
$ ./waf --run 'scratch/Algorithm1.cc'
```

Attributes are parameters that can be set to different values before a simulation. There exist multiple attributes and every attribute is set to a default if no value is set.

- **nDevices**: The number of nodes N. N must be divisible by 5. The default is 100.
- **radius**: The simulation radius in meters. The default is 5000.
- **simulationTime**: The simulation time in seconds. The default is 1000.
- **adrType**: An integer that decides which ADR algorithm is installed in the NS. The default is 1.
  - adrType=1 corresponds to the SotA ADR
  - adrType=2 corresponds to the ADR-1
  - adrType=3 corresponds to the ADR-2
  - adrType=4 corresponds to the ADR-3
  - adrType=5 corresponds to no ADR
- **sfDistribution**: This number (1 or 0) decides whether the initial SFs of nodes are set in a realistic scenario or uniformly. The default is 1.
- **homogenousSF**: If sfDistribution equals 0, then all SFs in the network are set to homogenousSF. The default is 7.

Example 1: This command runs a Profile-2 simulation with the SotA ADR algorithm:

```
$ ./waf --run 'scratch/Algorithm1.cc --adrType=1'
```

Example 2: This command runs a Profile-2 simulation with the SotA ADR algorithm and produces a log file of the results:

```
$ ./waf --run 'scratch/Algorithm1.cc --adrType=1' > logs.out 2>&1
```

Example 3: This command runs a Profile-1 simulation of a SF-7-only network with 200 nodes:

```
$ ./waf --run 'scratch/Algorithm1.cc --adrType=5 --sfDistribution=0
    --homogenousSF=7 --nDevices=200 -radius=1'
```

Appendix B

Contents of the CD

- **BScThesis_SteigerDavid.pdf**: The thesis document in PDF format.
- **BScThesis_SteigerDavid.zip**: The Latex source code of this thesis' PDF document.
- **Abstract.txt**: The abstract written in english.
- **Zusfsg.txt**: The abstract written in german.
- **MidtermPresentation.odp**: The intermediate presentation of this thesis.
- **NS3_LoRaWAN_SourceCode.zip**: The ns-3 source code that is required for the simulations in this thesis.
- **SimulationLogs.zip**: The log files that were produced as a result of the simulations.
- **LogParserScripts.zip**: Python parser scripts that were used to extract information from the logged data.
- **SimulationDataSets.zip**: Data sets that were produced from the parser scripts.
- **Figures.zip**: Figures that visually display information gathered in the data sets.