LORAWAN VULNERABILITY ANALYSIS

(In)validation of possible vulnerabilities in the LoRaWAN protocol specification.

E. van Es - 851180815 - 2 March 2018

Open Universiteit
www.ou.nl

LORAWAN VULNERABILITY ANALYSIS

(In)validation of possible vulnerabilities in the LoRaWAN protocol specification.

by

E. van Es

in partial fulfillment of requirements for the degree of

Master of Science

Graduation Assignment at the Open University of the Netherlands,
Faculty of Management, Science and Technology

Master's Programme in Computer Science

Student number:	851180815
Course code:	IM990C
Thesis committee:	dr. ir. H.P.E. Vranken (Chairman & Supervisor)
	dr. A.J. Hommersom (Supervisor)

Open Universiteit
www.ou.nl

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In today's world, technology is moving forward at a faster pace than ever. The Internet has expanded enormously, and still many new devices are being added every day. This expansion is creating many new opportunities, but it also comes with a price. Malicious use can lead to significant (financial) damages. The security aspect of IT is becoming increasingly important year-after-year.

LoRaWAN is a network protocol designed by the LoRa Alliance. This protocol is intended for wireless, battery-operated nodes in a regional, national, or global network, and is mainly used for Internet-of-Things and Wireless Sensor Network applications. Three versions of the specification are published so far: v1.0 in October 2015, v1.0.2 in July 2016, and v1.1 in October 2017. This research study is mainly based on v1.0.2. Since the protocol is quite recent, there is not much (scientific) information available in respect to LoRaWAN security.

To study LoRaWAN security, the following research question is defined: *Can evidence be found that (in)validates vulnerabilities in the LoRaWAN protocol specification?* To answer this question, a research method is created that includes two major research activities: 1) analysis of selected attacks applicable to the LoRaWAN specification, and 2) modeling and validation of possible findings.

In a first analysis, selected attacks have been analyzed against the LoRaWAN v1.0.2 specification. This analysis resulted in three vulnerabilities that have been selected for a more formal analysis: 1) an attack on network beaconing frames to manipulate downlink routing windows (beaconing), 2) an attack on end-device uplink network frames to manipulate the network server downlink routing table (downlink-routing), and 3) an attack on a Join-Accept message during an Over-the-Air-Activation (end-device activation).

During a second analysis, Colored Petri Net models are constructed for each vulnerability, and simulation is used for a more formal validation. For all three vulnerabilities, simulation is supporting the findings, which result into a specific Denial-of-Service of the end-device.

This research study has resulted in three vulnerabilities, of which one is based on an earlier finding (end-device activation), and two (beaconing, downlink-routing) are completely new.

1 INTRODUCTION

This thesis describes the results of a research study on the security of Long Range Wide Area Network (LoRaWAN), a recent Internet-of-Things (IoT) network protocol. The document describes the details and impact of several network attacks on the LoRaWAN protocol specification. In this chapter the context of the research is explained.

Section 1.1 of this chapter explains the importance of IT security. In section 1.2, a brief introduction to LoRaWAN is given. Section 1.3 describes the objectives of this research study. In section 1.4, an overview of related (academic) literature is presented. Section 1.5 shows the problem statement on which this research study is based. Section 1.6 describes the defined research questions. In section 1.7, the document outline is provided.

1.1 THE IMPORTANCE OF IT SECURITY

IT security is still getting more important every year. In the 70s, modern day hackers tried to misuse the emerging technologies of the telecommunications industry. The 80s marked the era of widespread use of personal computers and increased networking, which lead to increased computer and network hacking. The 90s brought the Internet to consumers, which increased the attack surface for hackers even more. After the year 2000, the Internet became widespread and the amount of services on the Internet has grown immense. So did the number of people having access to the Internet. Hacking changed from (heavily) misuse to criminal activities with high financial gains. In recent years the Internet has grown even more because of the enormous amount of devices connected to it.

At least once a year many organizations and governments release reports of studies that show the results of their analysis regarding IT security [28][9][8][10][24]. Every year, the majority of these reports (if not all) show increased malicious activities compared to previous years. These increased malicious activities, and the use of more sophisticated attacks, results in higher risks, higher amounts of successful attacks, and an increased impact of successful attacks. On a daily basis we are confronted with news messages about information (Personally Identifiable Information (PII), company secrets, etc.) theft, networks that are compromised, (on-line) services that are sabotaged,

IoT appliances that are maliciously used, and many more IT security related messages.

1.2 LORAWAN

According to the LoRa Alliance[1], "*LoRaWAN™ is a Low Power Wide Area Network (LPWAN) specification intended for wireless battery operated Things in a regional, national or global network. LoRaWAN targets key requirements of Internet of Things such as secure bi-directional communication, mobility and localization services.*". In October 2015, the LoRa Alliance published the first official release (v1.0) of the LoRaWAN specification. In July 2016 v1.0.2 was released, and in October 2017 v1.1 was published. This research study is based on v1.0.2 of the LoRaWAN specification. V1.1 is only studied for assessing several specific findings.

LoRaWAN is mainly used for IoT and Wireless Sensor Network (WSN) applications. Examples of LoRaWAN key characteristics are: low power, long range, localization, both up- and downlink traffic, and security build in from the first version onwards. Commercial- and community based implementations are already available today.

LoRaWAN has an open specification, meaning that an implementation can be done by anyone. While being a quite recent specification, the amount of LoRaWAN based networks is growing. For example The Things network which is implemented globally by a community, KPN which has implemented a national LoRaWAN in the Netherlands, and Proximus with a national LoRaWAN in Belgium. It is expected that LoRaWAN applications will increase[2] because of the low costs and long battery life of end-devices. An increased amount of sensors/actors will enlarge the attack surface of devices that can be misused. Because of these factors, it is very important that security aspects are well covered in the protocol specification (and implementations). If wrong trade-offs are made between power consumption, usability, and security, then this could result in possible security weaknesses.

1.3 OBJECTIVES

In this research study, the security of LoRaWAN is analyzed. There are two main objectives for studying the security of LoRaWAN. First, the outcome may help in improving the security of the LoRaWAN protocol specification. Second, the outcome of this research study contributes to the academic research field of LoRaWAN (security). The amount of

1 https://www.lora-alliance.org/what-is-lora
2 https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts/

available academic information on LoRaWAN security is currently very limited.

1.4 LITERATURE

MRW Labs provides an analysis and guidance around the security of the LoRaWAN protocol [26] (*published Mar-2016*). The main conclusion is that the LoRaWAN protocol can be used securely. However, several things need to be considered so that specific decisions in relation to the actual implementation are well thought of. The main subjects are related to key management (for example key generation, key distribution, and key storage), securing components that are Internet facing, and by looking at security from an end-to-end perspective and not only on a per component basis.

A conflict of interest in the use of keys is identified by Gemalto [11] (*published Dec-2015*). Because security session keys are derived from a master key, there might be a conflict of interest between a network operator and its customer. When a network operator knows the master key, it can also derive the session keys and thus intercept application data.

For his Master thesis, Simone Zulian has analyzed the security threats of LoRaWAN join procedure [41] (*published Oct-2016*). Focus during this thesis has been the generation of random numbers, such as the DevNonce in particular.

Trustpoint[3] (*published Jan-2017*) has also compiled a security overview for LoRaWAN. Their main conclusions on threats and mitigations focus around key management.

Avoine, G., et all [23] (*published Jul-2017*) also looked into the security matters of LoRaWAN. Several (theoretical) vulnerabilities have been identified, such as replay attacks to enforce devices to re-use keys, replay of Join-Accept messages, message harvesting, and network server attacks.

A short, high-level and theoretical, analysis on LoRaWAN security has been performed and published by Aras, E., et all [6] (*published July 2017*). It describes physical aspects, and the more obvious security measures and possible weaknesses such as frame counters, the use of security keys, physical access and jamming.

3 http://www.trustpointinnovation.com/blog/2017/01/17/lorawan-security-overview/

For a MSc in Electrical Engineering, Yang, X. [40] (*published July 2017*) has studied four possible weaknesses (replay attack for ABP, eavesdropping, bit flipping, ACK spoofing) and used experiments to provide additional information.

1.5 PROBLEM STATEMENT

LoRaWAN is a recent and fast-growing Internet-of-Things protocol. Because of the rapid growth of LoRaWAN implementations, and the important applications that LoRaWAN can be used for, severe problems can be expected when vulnerabilities are found in the LoRaWAN specification.

1.6 RESEARCH QUESTION

Based on the problem statement, the following research question is defined:

Can evidence[4] be found that (in)validates vulnerabilities in the LoRaWAN protocol specification?

This research question is split into multiple sub-questions. The first group of sub-question is related to getting to know LoRaWAN. The second group of sub-questions is related to network attacks and the impact on LoRaWAN.

1. What is LoRaWAN? What are the main architectural principles? What security related aspects have been integrated into the LoRaWAN specification?
2. Which known attack methods can be identified that are applicable to LoRaWAN? How can the selected attack methods be applied to LoRaWAN? What is the impact of the selected attack methods on LoRaWAN?

1.7 DOCUMENT OUTLINE

This document is structured according to the following description.

Chapter 2: Research design. A description of the method, phases, scope, and validation used for performing this research study is provided in this chapter.

Chapter 3: The LoRaWAN network protocol. This chapter describes the LoRaWAN specification and the most important factors related to

4 A combination of formal and informal evidence that confirms the existence of selected vulnerabilities.

this research study.

Chapter 4: Relevant attacks. Network attacks that are applicable to LoRaWAN are described in this chapter, including an explanation why the selected attacks are applicable to LoRaWAN.

Chapter 5: Attack analysis. This chapter provides a detailed overview of the impact of selected network attacks on the LoRaWAN specification.

Chapter 6: Attack modeling and validation. This chapter provides additional modeling and validation results for three selected findings from chapter 5.

Chapter 7: Conclusions, discussion, and future work. The final conclusions, based on the outcome of previous chapters, are described in this chapter. Discussions and future work are also included.

Appendix A: Communication scenarios. This Appendix contains a detailed overview of the communication scenarios which are based on the LoRaWAN specification.

Appendix B: Replay scenarios. A detailed description of possible replay scenarios on the LoRaWAN specification is included in this Appendix.

Appendix C: DOS scenarios. Details of possible Denial of Service (DOS) scenarios on the LoRaWAN specification are included in this Appendix.

Appendix D: Petri Nets & CPNTools. This Appendix provides additional details on Colored Petri Nets (CPNs) and CPNTools, which are used for a more formal analysis in this research study.

Appendix E: Modeling/validation details. Additional details for the attack modeling and validation in chapter 6 are provided in this Appendix.

2 RESEARCH DESIGN

To successfully perform a research study, a detailed research design is necessary. This chapter describes the research design used for this research study. The research design includes the method, phases, scope, and validation that are used for performing this research study.

Section 2.1 of this chapter describes the used research method. Section 2.2 describes all defined phases, and the results of each phase. The scope of the research study is described in section 2.3. Finally, section 2.4 explains how the results of this research study have been validated.

2.1 RESEARCH METHOD

The method of Verschuren and Doorewaard[1] is used as a basis for defining a research model for this research study. Figure 2.1 shows the high-level research model that is created using the method of Verschuren and Doorewaard.

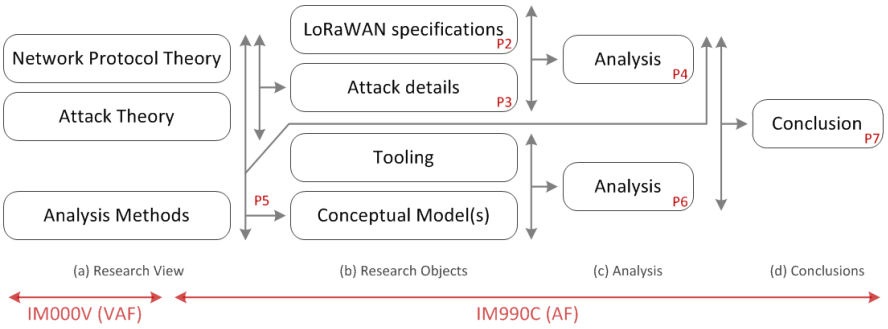

Figure 2.1: Research Model

The defined research model can be split into four sections: a) Research View, b) Research Objects, c) Analysis, and d) Conclusions.
The majority of section a) was part of course IMoooV (Research Preparation). During this course, I have gathered additional knowledge (such as research methods, analysis methods, network protocol theory, and attack theory) and created a research proposal.
The Research Objects in section b) are requirements to perform analysis activities. These requirements have been met by gathering specific documents, and studying additional information.

1 http://www.hetontwerpenvaneenonderzoek.nl

Two different Analysis activities are performed in section c). The first Analysis is based on the LoRaWAN specification, and details of known attack methods, to determine possibilities and impacts. This is a more informal analysis of attack possibilities on the LoRaWAN specification. The second Analysis activity, based on modeling and validation, is the more formal analysis on the LoRaWAN specification. The results of both analysis activities are used to draw an overall Conclusion in section d).

This research model provides a structured way of performing this type of research study. It provides a solid overview of what has been done before starting, what is done during this research study, the correct order, and any prerequisites for the different activities. This model also provided a solid basis for defining different phases in which the research study is split.

2.2 RESEARCH PHASES

The different sections in the research model have resulted into eight different phases that are executed during this research study. Figure 2.1 also includes the different phases, projected onto the used Research Model. Phases one and eight are not included in the research model. Phase one is a preparation phase, and phase eight is for finishing the research study. Both are considered a more overall activity, and therefore not part of the activities defined in the research model. The sections below provide a summary of each phase.

2.2.1 *Preparation*

In the preparation phase, activities are performed such as updating the overall planning according to the latest known information such as detailed activities and start date. Also the creation of the initial thesis document, and documents for time registration and meeting invites/notes is included.

2.2.2 *Analysis of the LoRaWAN specification*

A thorough study of the LoRaWAN specification is performed in this phase. The last version of the specification is retrieved from the LoRa Alliance[2]. After studying the specification, the working of the LoRaWAN protocol is included in the thesis document. Also, security specific aspects of the protocol are studied and summarized in the thesis document. These details are used for the selection of attacks in the succeeding phase. After this phase, the thesis document contains doc-

2 https://www.lora-alliance.org/

umentation about the LoRaWAN specification with a specific focus on the security related aspects of the specification.

2.2.3 *Selection and analysis of attacks*

In this phase, a selection of known attacks, that might be applicable to the LoRaWAN protocol, are documented. This is depending mostly on the results of the preceding phase and attacks studied in the pre-graduation phase. The selected attacks must be studied in detail and well-documented. At the end of this phase, the thesis document includes a detailed overview of the selected attacks.

2.2.4 *Apply attacks to LoRaWAN specification*

This is the most important phase in this research study. In the preceding phase, a selection is made of known attacks that might be applicable to LoRaWAN. In this phase, an analysis is performed to determine the actual possibilities and impacts of the selected attacks on the LoRaWAN specification. Attacks are based on vulnerabilities. Identified vulnerabilities are used in the next phases of this research study. The result of this phase is a detailed description in the thesis document regarding the findings in this phase. Informal methods are used in this phase to document the results. Examples of used informal methods are: natural language, Attack-Defense Tree (ADT)s, and Attack Sequences.

2.2.5 *Preparation for formal analysis*

When applicable attacks are identified in the preceding phase, a selection is made of vulnerabilities that can be (in)validated using formal verification methods. Time is reserved for in-depth knowledge gathering on Colored Petri Net (CPN)s and CPNTools.

2.2.6 *Formally (in)validate vulnerabilities*

In this phase, the selected vulnerabilities are used for formal checking and verification. More formal verification methods are used to prove correctness in respect to the LoRaWAN specification. The construction of a CPN in CPNTools is used for each vulnerability that is found. All CPN models that are constructed, are representing the security related aspect(s) of the LoRaWAN specification and characteristics of the selected vulnerability. As a result of this phase, the thesis document is updated to describe all aspects of the formal analysis process and results.

2.2.7 *Final analysis*

This phase is used to study the outcome of previous analysis phases. The outcome of the different analysis phases is used to determine if it is possible to draw overall conclusions based on this research study. The result of this phase is to have the thesis document completed with all relevant information regarding this research study so that the research questions can be answered, the defined goals are met, and that the thesis document can be finalized in the succeeding phase.

2.2.8 *Finalization and presentation*

Finalizing the thesis document, preparing the presentation, and presenting the results are the major activities for this phase of the research study.

2.3 SCOPE

Because of several constraints during this research study (of which time is the most important one), the scope is limited based on the following statements:

- The research study is limited to the specification only. LoRaWAN specification v1.0.2 is used. Although a newer version (v1.1) is published during this research study, the impact of switching to the new version so late in the research study would be too large. LoRaWAN specification v1.1 is used for brief assessments against the found weaknesses only.
- Since there are many (types of) attacks, only a limited number of known attacks are selected during different phases of this research study. Without limiting the number of attacks, it is difficult to meet the proposed planning.
- This research study is looking at logical aspects of the LoRaWAN specification. Physical aspects such as frequencies and transmitting power are not in scope of this research study.
- Because of possible differences in implementation and the limited time for this study, implementation related activities are not part of this study.

2.4 VALIDATION

The goal of this research study is to find possible weaknesses in the LoRaWAN specification. Weakness are searched for, using a less formal method, by manually studying the specification and compare

specification details against different attacks methods. The results are described in detail by using natural language, Attack Defense Trees, and Attack Sequences. Also a more formal method is used by constructing CPN models. Such models are based on mathematics, and therefore State Space analysis is used to validate the correctness of these CPN models. Simulation is used to verify the functional behavior of the models.

The LoRaWAN specification is written in a natural language. Because of this, interpretation of the specification is a manual process. Also the construction of the CPN models, based on the findings and the LoRaWAN specification, is based on human interpretation and manual activities. This is a disadvantage, and can have an impact on the reliability of the research study results. Therefore, all decisions, assumptions, and details are clearly described in this thesis document in such a way that these can be validated and reproduced by others.

3

THE LORAWAN NETWORK PROTOCOL

This chapter provides an overview of the LoRaWAN network protocol. This overview should provide sufficient detail for the remainder of this document.

Section 3.1 of this chapter provides more details on LoRaWAN. In section 3.2, the security aspects of LoRaWAN are described.

3.1 WHAT IS LORAWAN?

LoRaWAN is a network protocol developed by the LoRa-Alliance[1]. The LoRa Alliance consists of major companies and universities. The protocol is intended for wireless, battery-operated nodes, in a regional, national, or global network. The protocol is based on Semtech's[2] propriety physical communication specification named LoRa. Figure 3.1 shows the LoRaWAN stack. In this research study, the focus will be on the blue LoRaWAN section. The other sections, Application, and LoRa, are out of scope.

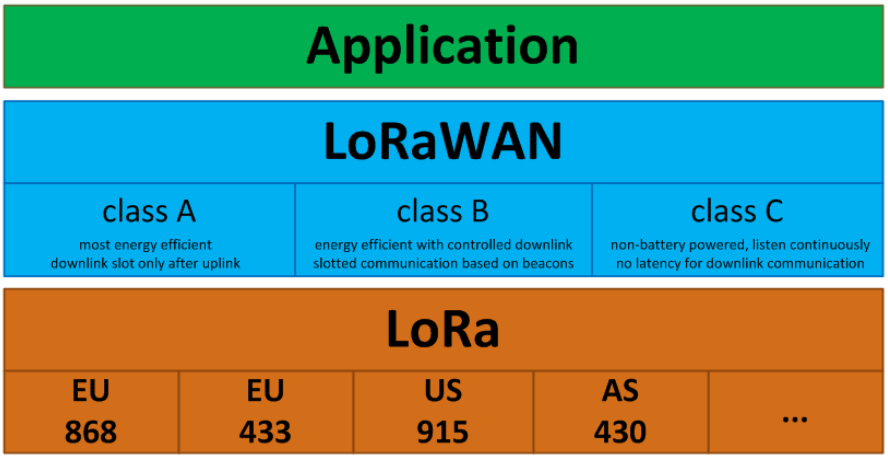

Figure 3.1: LoRaWAN stack

The LoRaWAN details in this document are based on the LoRaWAN specification version 1.0.2 [20], and the LoRaWAN Regional Parameters version 1.0 [21], which have been released in July 2016. In October 2017, version 1.1 of the LoRaWAN specification [22], and version 1.1 of the LoRaWAN Regional Parameters [4] have been released. Because of the limited time in this research study, the new versions of the LoRaWAN

1 https://www.lora-alliance.org/
2 http://semtech.com/

specification have been used to briefly validate the presence of findings in the newer version only. For the findings that have been studied in detail, relevant differences between the LoRaWAN specification versions have been included.

LoRaWAN can be considered as a Low-Power Wide Area Network (LPWAN) protocol that has been designed with very specific properties, which makes it a perfect network protocol for IoT, or WSN, applications. Examples of this are smart cities, object tracking, and environment monitoring. LPWAN is perfectly suited for connecting devices that need to send small amounts of data, over a long-range, while maintaining long battery life [19].

Besides typical LPWAN characteristics (such as low power consumption and long-range communication), LoRaWAN includes several additional characteristics such as secure bi-directional communication, mobility, and localization services. Table 3.1 provides an overview of LoRaWAN characteristics in more detail.

Table 3.1: LoRaWAN characteristics

Characteristic	Details
Long Range	LoRaWAN is part of the LPWAN family. Protocols of this family are known for their Long Range / Low Power characteristics. The physical layer of LoRaWAN uses additional Long Range technologies to increase the range even further.
Low Costs	Currently only Semtech is manufacturing chips that can be used for LoRaWAN implementations. These chips are not very expensive, but it is expected that prices will drop over time.
Low Bandwidth	For LoRaWAN, bandwidth is a trade-off between a long communication range and very low power use. For most IoT applications low bandwidth should not be a problem.
Low Power	LoRaWAN has been designed for very low power use so that it can be used by battery operated end-devices. An end-device should achieve over 10-years of battery life.
Security	LoRaWAN is designed with multiple security features, including a combination of Device-, Session-, and Application cryptographic keys for data encryption, authenticity, and authentication purposes.
Mobility	An end-device can move between gateways. LoRaWAN networks can also be configured to support roaming, so that end-devices can even move outside their home network.

LoRaWAN characteristics – continued

CHARACTERISTIC	DETAILS
Localization	The gateways in a LoRaWAN network can be equipped with GPS sensors. This accurate positioning, combined with exact timing, can be used to determine the position of end-devices.

3.1.1 *Architecture*

A typical architecture for LoRaWAN is presented in figure 3.2. The architecture is based on a star-of-stars network topology. Such a topology is like the network topology that is used for most cellular networks and is relatively simple to implement. A LoRaWAN setup is based on end-devices, gateways, a network server and usually one or more application servers.

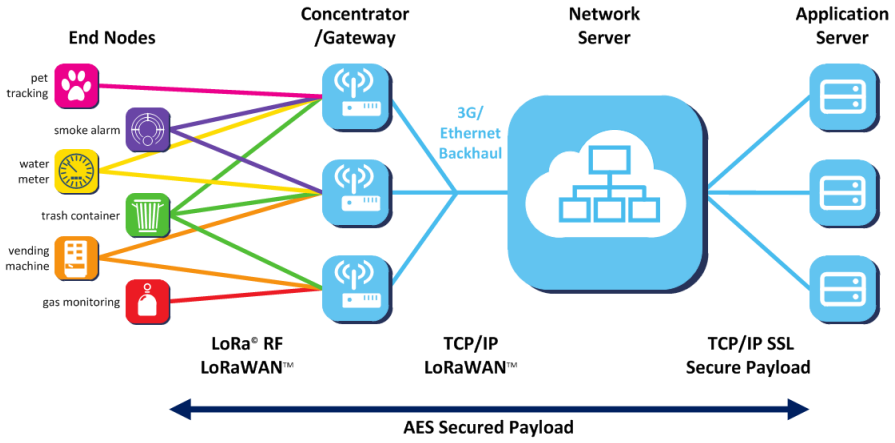

Figure 3.2: LoRaWAN Architecture

In a LoRaWAN architecture, a wireless communication link (single hop wireless LoRa Radio Frequency (RF) communication) is used between end-devices and one or more gateways. A gateway provides a bridging functionality between the wireless LoRa RF network at the front-end, and a TCP/IP (standard IP connection) communication link with the network server at the back-end. The network server processes the data of the end-device so that it can be provided to application servers for further processing. It is also a task of the network server to control the network. An end-device is not associated with a specific gateway. The end-device broadcasts its information (uplink communication), therefore it is typically received and relayed by multiple gateways. The network server will provide de-duplication of network packets. Depending on certain criteria of the last uplink communication, like signal strength, the network server decides and remembers which

network path (gateway) is the best for downlink[3] communication. Although bi-directional communication is possible, it is expected that uplink communication is the predominant type of traffic.

3.1.2 *End-devices*

LoRaWAN supports different types of end-devices (Class A, B, and C), which provide a trade-off between battery lifetime and bi-directional communication options. Device Class A supports the basic LoRaWAN features. Device Classes B and C support optional features, mainly additional downlink communication slots. Devices of Class B and C should also implement all features of the Class A specification. The decision on which Class to use, and when to switch to a different Class, is up to the application.

Device Class A - Bi-directional
Class A is optimized for battery-powered end-devices and is the LoRaWAN device Class that is using the lowest amount of power. All end-devices must support the functions of the Class A device specification and start their initial communication as a Class A end-device. Class A end-devices allow for bi-directional communications where each uplink transmission window is followed by two, fixed-length, downlink receive windows.

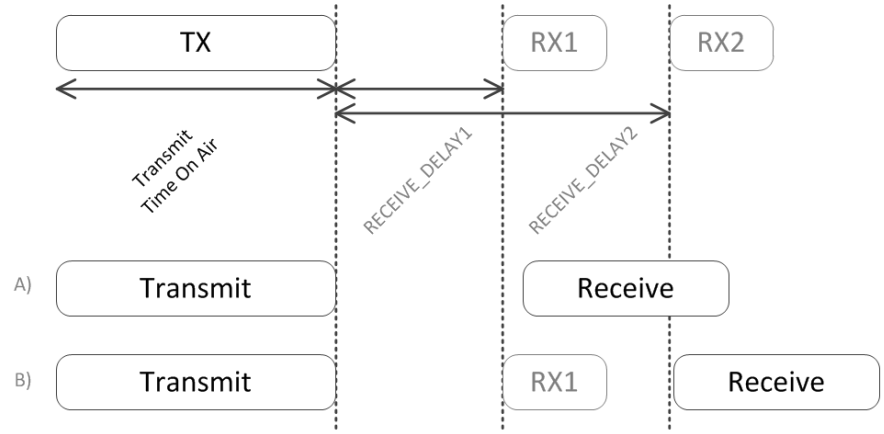

Figure 3.3: Class A - timing of send/receive windows

Figure 3.3 shows the timing of the different windows where the duration of the delays are depending on regional parameters. This figure also includes two examples, A) and B), to show the use of the different windows. The TX window (used for transmission of data) is scheduled by the end-device based on its needs. Then, RX1 starts typically one second after the end of the TX-window. RX2 starts around

3 A message sent by the network server to only one end-device.

two seconds after the TX window (typically one second after the end of RX1). The duration of the receive windows RX1 and RX2 must be at least the time required by the end-device's radio transceiver to effectively detect a downlink. The main difference between RX1 and RX2 is timing and that RX1 uses a frequency and data rate that is based on uplink frequency and data rate, and that RX2 uses a fixed configurable frequency and data rate. If there is no downlink information send in either RX1 or RX2, then the server must wait until the next opportunity, which will be after the next uplink transmission (TX) of the end-device.

Situation A, in figure 3.3, shows when RX1 is used to receive downlink information. In such a case, window RX2 will not be opened by the end-device. Situation B, in figure 3.3, shows when RX1 is not used but RX2 is used to receive downlink information.

Device Class B - Bi-directional with scheduled receive slots
Class B is also optimized for battery-powered end-devices. Compared to Class A, this Class adds a synchronized receive window on the end-device. This additional receive window is opened at calculated time intervals for enabling network server initiated downlink messages. This way the end-device is ready for reception on a predictable time. These additional windows provide a mechanism for applications to contact the end-device at specific times rather than depending on the non-deterministic downlink windows available in the default Class A operation.

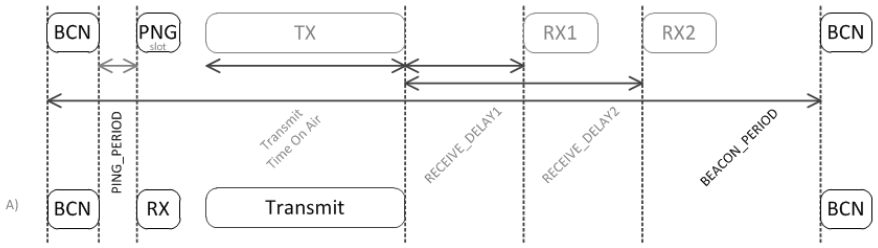

Figure 3.4: Class B - timing of send/receive windows

Figure 3.4 shows the timing of the different windows where the delays are depending on regional parameters and randomization. This figure also includes an example, A), to show the use of the different windows. Compared to Class A, additional windows are added in Class B. Every BEACON_PERIOD interval a beacon is send from the gateways to all end-devices to synchronize timing. When an end-device is in Class B mode, it calculates a random PING_PERIOD after every beacon that is received. This is to make sure the ping slots have different intervals after every received beacon to prevent possible timing issues.

Situation A, in figure 3.4, shows when the ping slot is used by the network server to send information to the end-device, and the next available TX window is used by the end-device to reply information back to the network server.

Device Class C - Bi-directional with maximal receive slots
Class C devices provide maximum send/receive possibilities. These devices listen all the time, except when in transmit mode. Providing continuous communication windows costs more energy compared to the other Classes and therefore this is the Class with the highest power use. Because of this, Class C is usually for end-devices that have sufficient power available (typically not battery-powered devices). The network server can initiate downlink communication at any time as long as the end-device is not already transmitting.

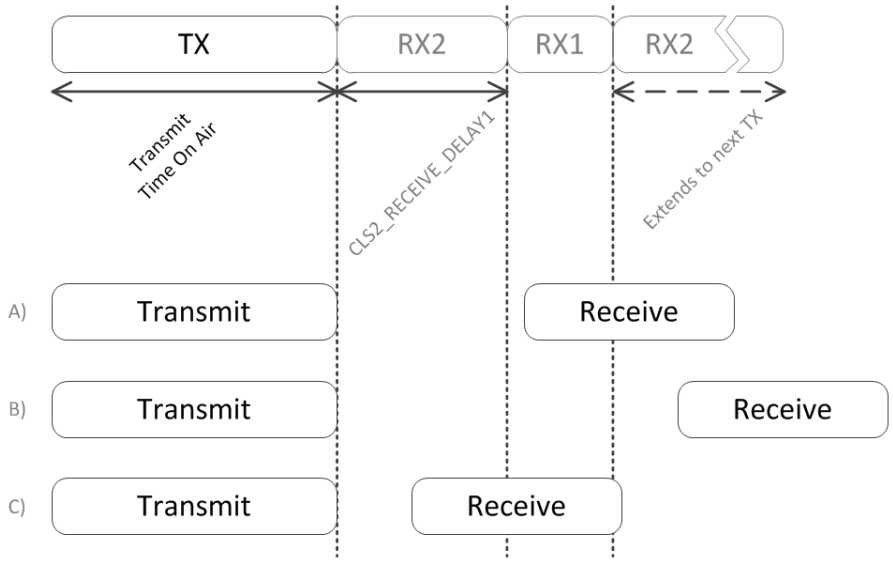

Figure 3.5: Class C - timing of send/receive windows

Figure 3.5 shows the timing of the different windows where the duration of the delay is depending on regional parameters. This figure also includes three examples, A), B), and C), to show the use of the different windows. Communication windows are based on the schedule of Class A end-device. Compared to Class A, there is an additional RX2 window between TX and RX1, and the second (original) RX2 window extends to a next TX window. Class C end-devices cannot implement Class B features.
Situation A, in figure 3.5, shows when RX1 is used to receive downlink information. After receiving information in RX1, RX2 will remain open until the end-device must open TX again. Situation B, in figure 3.5, shows when RX1 is not used but RX2 is used to receive downlink information. Also in this case, RX2 will remain open until the end-

device must open TX again. Situation C, in figure 3.5, shows that the first RX2 windows is used to receive information. By the time the information has been received, the RX1 window has passed, so again RX2 remains open until the end-device must open TX again.

3.1.3 *Gateways*

Gateways are also known as concentrators, or base stations, and bridge the wireless and wired parts of a LoRaWAN network. End-devices broadcast their data on the wireless part of the network which is received by one, or more, gateways. These gateways transfer the information to the network server. The gateways are connected to the network server using a standard IP connection. This IP connection can be setup over any physical layer, such as Wi-Fi, Ethernet, cellular, or any other telecommunications link. Gateways are based on a star-of-stars topology, and spread over a geographical location to provide network coverage in a large geographical area.

LoRaWAN gateways are transparently forwarding network packets between end-devices and a network server. The data, also called payload, of network packets is encrypted and signed, therefore it cannot be read or modified by a gateway. This way the impact of any possible security threats on LoRaWAN gateways are minimized.

3.1.4 *Network server*

The centrally located network server manages the LoRaWAN network. It 'speaks' the LoRaWAN protocol from the gateway side and provides a bridge between applications and the LoRaWAN network. The network server contains intelligence that is responsible for handling (and de-duplication) of uplink data received by the gateway(s), perform security checks, scheduling of downlink data transmissions, keep track of routing-paths to end-devices, etcetera.

3.1.5 *LoRaWAN lifecycle*

When operating a LoRaWAN environment, some crucial steps must be taken before the network is operational and an end-device can be used. Figure 3.6 shows a high-level overview of the steps required. First, a network server must be deployed and configured. Second, one or more gateways must be deployed and configured. From this moment onwards, end-device can be added to the network.

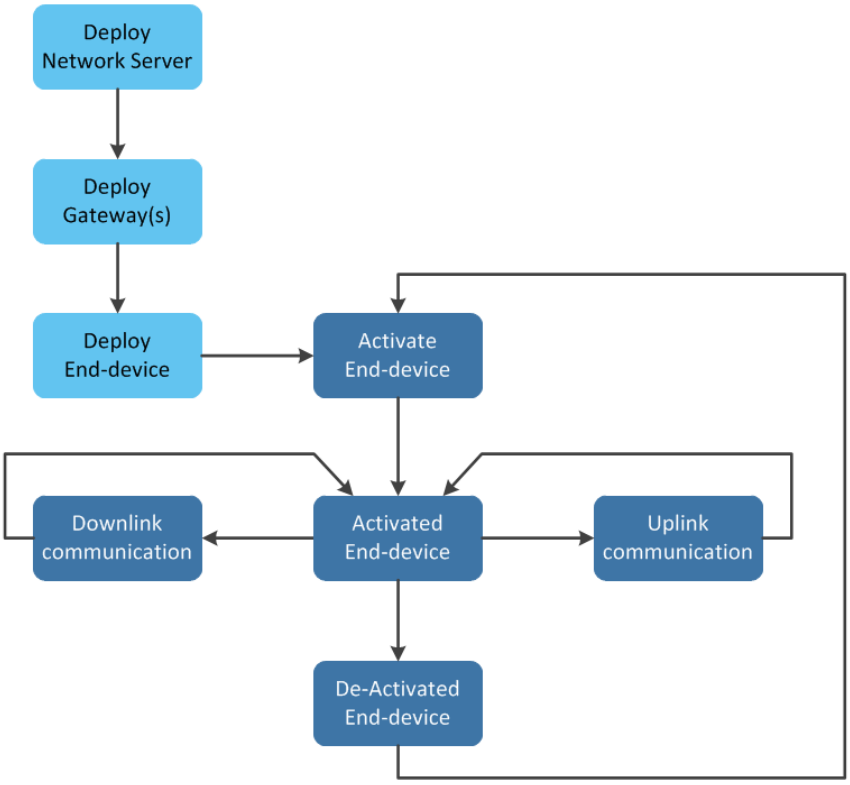

Figure 3.6: LoRaWAN lifecycle

After a deployment of an end-device, the end-device must be activated within a LoRaWAN environment. Once the end-device is activated, it can participate in communications, either uplink, or downlink, depending on intended functionality. Sometimes, end-devices are de-activated. De-Activation means the end-device is not in sync anymore with the network server, in respect to security keys and other configuration settings. This can happen, for example, after a power loss on the end-device. By running the activation procedure again, the end-device is re-activated and can start communication again.

3.1.6 *LoRaWAN Messages*

LoRaWAN uses several message formats (network frames) for different purposes. Appendix A provides a comprehensive overview of the communication scenarios that are available for LoRaWAN.

End-Device activation
Before an end-device can send or receive data , it needs to be activated on the network. There are two procedures to activate an end-device: Activation by Personalization (ABP) (manually entering configuration details), and Over-the-Air-Activation (OTAA) (automatic configuration). For OTAA, two types of messages have been defined in the

LoRaWAN specification: Join-Request (to request the network server to be allowed on the network) & Join-Accept (acceptance from the network server containing end-device configuration details).

Sending data
The most important functionality is the communication of data. Data can be transmitted uplink (end-device towards network server), or downlink (network server towards end-device).

Sending MAC commands
To manage the network, Media Access Control (MAC) commands are required. These command can be used to change speeds, frequencies, request device status, etcetera. MAC commands can be transmitted using a dedicated network frame (encrypted; more secure), or by piggy-backing the command onto an existing data network frame (unencrypted; less secure).

Multi-Cast messages
LoRaWAN also provides the option for downlink communication to multiple end-device simultaneously. If a group of end-devices is assigned the same network address (device address), then the network server can send a network frame to all those end-devices at once.

Beaconing
For Class B operations, a LoRaWAN network uses time synchronization beacons that are generated and broadcasted by gateways. Such a beacon contains a timing reference and, optionally, gateway location information.

3.2 SECURITY ASPECTS OF LORAWAN

IT security is often expressed in terms of Confidentiality, Integrity, and Availability (C-I-A). Especially in networked environments these aspects are very important. Confidentiality means that the content of a message cannot be read by unintended audiences. Only the intended parties are able to read the contents. Integrity guarantees that the content of a message has not been altered between sender and receiver. If the content is modified, then this will be noticed. To be able to send messages between a sender and receiver, a network must be available. If a network is only limited available, it will have a negative impact on the ability to transmit messages. These terms will be referred to in the remainder of this section to explain different security aspects of LoRaWAN.

3.2.1 *AES encryption*

For encryption purposes, LoRaWAN uses the Advanced Encryption Standard (AES), which is a specification for encryption defined by the National Institute of Standards and Technology (NIST) [14]. The AES specification is a symmetric-key algorithm, meaning the same security key is used for both encryption and decryption of data. The LoRaWAN specification uses AES for different purposes. Three generic methods are used for AES operations: aes128_cmac (integrity), aes128_encrypt (confidentiality), and aes128_decrypt (confidentiality). AES can use different key lengths. All three methods use AES with a key-size of 128-bits.

aes128_cmac
This method is used to calculate a hash for a specified input (message). A hash function provides a fixed-size unique string based on input data of arbitrary size. Identical input should always result in the same unique fixed-size string. Different inputs should always result in different fixed-sized outputs. Hashes are mainly used as part of integrity checks. This method is based on RFC4493 [17]. RFC4493 specifies the Cipher-based Message Authentication Code (CMAC), which is a keyed hash function that is based on a symmetric key block cipher (AES in this situation). A CMAC provides data origin authentication and data integrity for the specified message. Table 3.2 shows the specification of the method call: A = aes_cmac(K, M, len).

Table 3.2: aes128_cmac specification

Input	K	[NwkSKey ǀ AppKey]
	M	message to be authenticated
	len	length of the message in octets
Output	A	message authentication code

In the LoRaWAN specification this method is used for:

- Generating a Message Integrity Code (MIC) for a *PHYPayload*
- Generating a MIC for a Join-Request message
- Generating a MIC for a Join-Accept message

aes128_encrypt
The encrypt method is used to create a cipher (encrypted) text based on the encryption key and the input message. The encryption scheme used is based on the generic algorithm described in Institute of Electrical and Electronics Engineers (IEEE) 6 802.15.4/2006 Annex B [IEEE 802154] which is using AES in Counter (CTR) mode by default. Only for Join-Accept messages Electronic Code Book (ECB) mode is used. Both CTR and ECB are modes that are used for confidentiality [36]. Ta-

ble 3.3 shows the specification of the method call: C = aes_encrypt(K, M).

Table 3.3: aes128_encrypt specification

Input	K	[NwkSKey | AppSKey | AppKey | 16 x 0x00]
	M	message to be encrypted
Output	C	cipher (encrypted) text

In the LoRaWAN specification this method is used for:

- *FRMPayload* encryption
- Generation of *NwkSKey* from a Join-Accept message
- Generation of *AppSKey* from a Join-Accept message
- Compute a new pseudo-random offset for slot randomization

aes128_decrypt

The network server uses an AES decrypt(!) operation in ECB mode to encrypt the Join-Accept message so that the end-device can use an AES encrypt(!) operation to decrypt the message. This way an end-device only has to implement AES encrypt but not AES decrypt. Table 3.4 shows the specification of the method call: M = aes_decrypt(K, C).

Table 3.4: aes128_decrypt specification

Input	K	[AppKey]
	C	cipher (encrypted) text
Output	M	original message

In the LoRaWAN specification this method is only used for encrypting a Join-Accept message by the network server. It is not used for any other purpose and therefore not used by any end-device for LoRaWAN purposes.

Possible weaknesses

- The used key lengths for LoRaWAN are 128-bits. Key lengths have an impact on the strength of the encryption (see also chapter 3.2.3).
- AES-ECB mode is used for encryption of the LoRaWAN Join-Accept messages. This ECB encryption mode has the weakness that identical plaintext blocks are encrypted into identical ciphertext blocks (see also chapter 3.2.4).
- AES-CTR mode is used for default encryption in the LoRaWAN specification. A known weakness of this encryption mode is that it is malleable. Malleable means that it is possible to transform

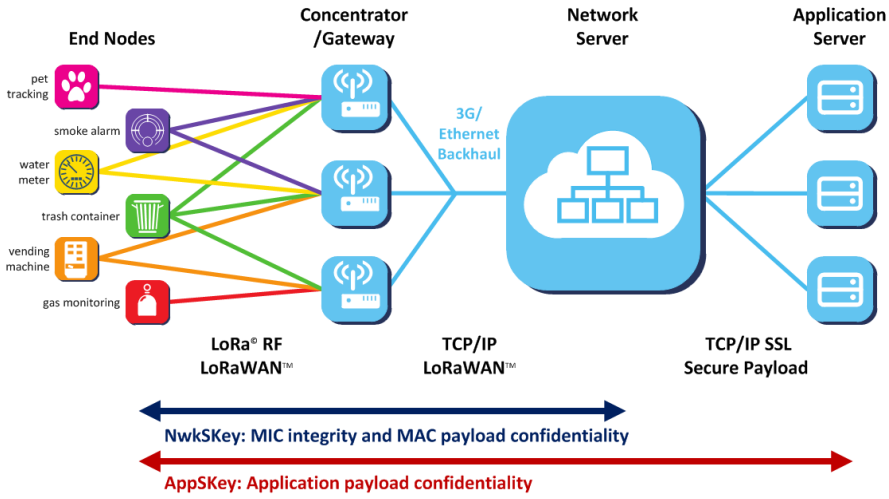

Figure 3.7: LoRaWAN encryption

a ciphertext into another, valid, ciphertext which can be successfully decrypted to a related plaintext [25].

3.2.2 *Data encryption*

Important data that is send over uncontrolled media should be encrypted so that it cannot be read by others. This is called data confidentiality. Application data that is sent over a LoRaWAN network, will be encrypted by default. The data can already be encrypted by the application, however, LoRaWAN has built-in features that always encrypts the data (payload) that must be send over the network. Figure 3.7 shows a high level overview of the LoRaWAN network architecture. In this figure two arrows are included that indicate end-to-end encryption, the blue one on the LoRaWAN layer, and the red one for the application layer.

When looking at the message formats in Appendix A, it may become clear that if there is a payload then it will be present in the *FRMPayload* field in the MAC payload structure. LoRaWAN uses the aes128_ encrypt method to encrypt this field that contains the payload. From version 1.0.1 of the LoRaWAN specification this encryption has become mandatory and cannot be turned off anymore.

The encryption scheme used is based on the generic algorithm described in the IEEE 802.15.4 Annex B [15] using AES with a key length of 128 bits.

Possible weaknesses

- For payload encryption AES-CTR mode is used. See chapter 3.2.1 for more details about the weakness of AES-CTR mode.

- When a payload must be encrypted, the aes128_encrypt method is used to generate a keystream which is XOR-ed with the payload. The output of the aes128_encrypt is not send in the *FRMPayload* field, but the result of the XOR-instruction[4] [26].

3.2.3 *Keys, Addresses and Identifiers*

The LoRaWAN specification includes different keys, addresses and identifiers. These are used for several different purposes, such as confidentiality aspects (for example encryption), and integrity aspects (such as validation and authentication).

Two important keys, *Application Session Key (AppSKey)* and *Network Session Key (NwkSKey)*, are mainly used for security purposes *confidentiality* and *integrity*. These two keys are unique for every end-device and are derived from the *Application Key (AppKey)* during each network (re-)join procedure using the OTAA procedure (see chapter 3.2.4). When the ABP method is used to (re-)join the network, these keys stay the same until they are changed manually.

Application session key (AppSKey)
The AppSKey is a 128-bit application session key which is unique for every end-device. This key is never transmitted over the air but it is shared by the end-device and the application server to encrypt and decrypt the payload field of application-specific messages to ensure data confidentiality. This way there is an end-to-end encryption between the end-device and the application. This is shown by the red arrow in figure 3.7.

Network session key (NwkSKey)
The NwkSKey is a 128-bit network session key which is unique for every end-device. This key is never transmitted over the air but it is shared by the end-device and the network server to calculate and verify the MIC of each message to ensure data integrity (message signing). It is also used for encryption and decryption of payload field of MAC-only data messages.

Application key (AppKey)
The AppKey is a 128-bit application owner defined root key which is unique for every end-device. This key is never transmitted over the air but it is shared by the end-device and the network server. During a network join using the OTAA procedure, this AppKey is used to generate new AppSKey and NwkSKey session keys specific for that end-device. When this AppKey is compromised this end-device should be considered compromised since important other keys are derived from this

4 https://www.elektormagazine.com/news/lorawan

AppKey. Other end-devices are not impacted since every end-device uses its own AppKey.

End-device address (DevAddr)
The End-device Address (DevAddr) consists of 32-bits and identifies the end-device within the current LoRaWAN. It is based on a 7-bits Network Identifier (NwkID) to distinct between different network operators and a 25-bits Network Address (NwkAddr) which is assigned by the network operator.

End-device identifier (DevEUI)
The End-device Identifier (DevEUI) is a 64-bits MAC address, in the IEEE Extended Unique Identifiers (EUI)-64[5] address space, that uniquely identifies the end-device on a LoRaWAN network during a JOIN request. The chip manufacturer of the modem of an end-device should already have a DevEUI pre-defined in the factory. The idea is almost like that of a MAC address in a TCP/IP Network Interface Card (NIC), to globally and uniquely identify each hardware modem.

Application identifier (AppEUI)
The Application Identifier (AppEUI) is a 64-bits MAC address, in the IEEE EUI-64 address space, that uniquely identifies the application. The network operator should assign an AppEUI from their officially assigned IEEE EUI-64 address space. This address uniquely identifies the entity able to process the Join-Request frame. The AppEUI is stored in the end-device before the activation procedure is executed. The AppEUI is transmitted over the air and is almost comparable to a Wi-Fi SSID.

Possible weaknesses

- Keys are shared because of the symmetric encryption method that is used. This is also known as Pre-Shared Key (PSK).
- Because of the PSK, keys must be exchanged out-of-band. This means that keys must be exchanged using a different medium than the network for which it is intended for.
- Key-generation and Key-derivation are important factors. It should be impossible for others to generate (or derive) valid keys, for example based on already available information.
- Proper key management should be in place. Key management includes the generation of keys, but also storage of keys, distribution of keys, invalidation/expiration of keys, etc.
- Key lengths have an impact on the strength of the encryption. In LoRaWAN keys lengths of 128-bits are used. This is chosen

5 A 64-bit Extended Unique Identifier (EUI-64) is globally unique, and intended to be bound to a hardware device instance or another object that requires unique identification http://standards.ieee.org/develop/regauth/tut/eui.pdf.

because, a) that key length is still considered to be safe, and b) it does not require the performance (power use) that is needed for larger keys. LoRaWAN is mainly developed for low powered devices. Larger key lengths have an impact on performance and power use. Key lengths are a trade-of between performance and security.

- The *AppSKey* is used to encrypt application data between the end-device and the application. However, this key is derived from the *AppKey* and *AppNonce* (a random value or some form of unique ID provided by the network server). Since the *AppKey* is also known by the network server, the network server can generate the *AppSKey* as well, and decrypt application data [11].

3.2.4 *End Device Activation*

Before an end-device can participate in any LoRaWAN network, the end-device must be personalized and activated. There are two methods available to activate an end-device, either by using Over-the-Air-Activation (OTAA) during deployment or after a reset, or by using Activation by Personalization (ABP).

Over-the-Air-Activation (OTAA)
Whenever an end-device is deployed, reset, or otherwise lost its session information, it can use the OTAA join procedure to (re-)establish data exchanges with the network server.

Before the join procedure can be started, the end-device must be personalized with the globally unique DevEUI and AppEUI, and the application owner defined AppKey (see chapter 3.2.3).

After personalization, a Join-Request message is compiled and sent from the end-device to the network server. This Join-Request message is based on the AppEUI, DevEUI, and *DevNonce* which is a random value which is used to prevent replay attacks. The network server will keep track of the current and last Join-Request/Join-Accept states to prevent unintended disconnects and replay attacks. Additionally, a MIC is calculated and included for the Join-Request message before it is sent out. See Appendix A.1.1 for the message format of the Join-Request message. The Join-Request message is not encrypted. For encrypting a payload, the end-device must have a NwkSKey and/or AppSKey which it does not have before it formally has joined the network.

After sending a Join-Request message, the network server will check the MIC, process the request, and respond with a Join-Accept message. The Join-Accept message will only be generated and send, if the end-

device did send a valid Join-Request message, and is allowed to join the network, otherwise no response will be send.

The content of the Join-Accept message contains an *AppNonce* which is a random value or unique ID provided by the network server, a Network Identifier (NetID), a DevAddr, a delay between TX and RX windows (*RxDelay*), and a list of channel frequencies for the network the end-device is joining. The *RxDelay* and the channel frequencies are based on the Regional Parameters. The Join-Accept message is encrypted using the aes128_decrypt(!) method. See Appendix A.1.2 for more details on this.

Once the Join-Accept message is received by the end-device, the first step in processing the Join-Accept message is to decrypt the message using the aes128_encrypt(!) method. Similar to a Join-Request message, a Join-Accept message also contains a MIC. The end-device can also calculate the MIC and compare it with the value in the Join-Accept message. If these two values are the same, then the end-device can assume the Join-Accept message is authentic. Once the message has been decrypted and the MIC has been verified, the DevAddr is extracted and stored, followed by the generation of the NwkSKey and AppSKey keys.

From this moment, the end-device has formally joined the network and can start secure communication within the network until it is reset or has otherwise lost the session information. OTAA makes it also possible, for different LoRaWAN implementations, to provide roaming options for end-devices.

Activation by Personalization (ABP)
Besides the earlier described OTAA procedure it is also possible to use Activation by Personalization (ABP). This procedure by-passes the Join-Request / Join-Accept communication. The values for DevAddr, NwkSKey and AppSKey are directly stored into the end-device without requiring DevEUI, AppEUI, and AppKey. This way the end-device is configured with the required information for use on a specific LoRaWAN network. There is no over-the-air handshake and therefore network roaming is not possible.

Just as with the OTAA procedure, the configured NwkSKey and AppSKey must be unique values. Compromising these keys does not compromise other end-devices. However, if these keys somehow reveal how they are derived or generated, then others might be able to reproduce keys for other end-devices on this specific LoRaWAN network.

After applying the configuration, the end-device has formally joined

the network and can start secure communication within the network until it is reset or has otherwise lost the session information.

Possible weaknesses

- AES-ECB mode is used for encryption of LoRaWAN Join-Accept messages. This ECB encryption mode has the weakness that identical plaintext blocks are encrypted into identical ciphertext blocks [36].
- Chapter 3.2.8 describes a possible issue with LoRaWAN frame re-transmissions. Join-accept messages are typical examples of messages that could result in frame re-transmissions and therefore might have an impact on availability. Especially when many end-devices need to (re-)join the network at the same moment.
- As mentioned in chapter 3.2.3, key management is very important. Especially, when a symmetric encryption method is used.
- Some form of replay attacks of the Join-Request and Join-Accept messages might be possible [41].

3.2.5 *Multicast versus Unicast*

Downlink communication that is send on a (LoRaWAN) network can be either 'unicast' or 'multicast'. Unicast messages are sent from the network server to a single end-device. Multicast messages are sent from the network server to multiple end-devices. Figure 3.8 shows the difference between multicast (red arrows) versus unicast (blue arrow).

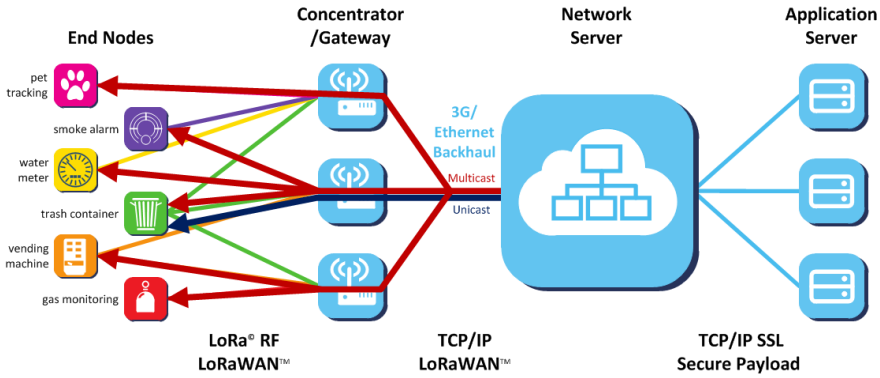

Figure 3.8: LoRaWAN Multicast versus Unicast

To be able to successful address multiple end-devices at once, these end-devices should share the same encryption keys so that all end-devices can decrypt the same message. Because of this encryption key sharing, multicast messages are considered to be less secure (less authentication robustness) and should not contain MAC commands.

Possible weaknesses

- Shared keys are required to decrypt the same message by all relevant end-devices.
- It is not allowed to have MAC commands inside multicast messages because a multicast downlink does not have the same authentication robustness as a unicast frame (for example the shared keys).
- Multicast messages should be of type *Unconfirmed Data Down* (see Appendix A). Multicast message of type *Confirmed Data Down* can have an impact on network performance and availability.

3.2.6 *Frame counters*

A very well-known method of attacks in networked environments are replay attacks. During a replay attack a malicious person captures and records network packets. In a LoRaWAN, packets can be captured and stored as well. It is not possible to read the data without the AppSKey because the data has been encrypted (confidentiality). It is also not possible to change the contents because the integrity is validated using a MIC which is calculated using the NwkSKey. However, it is possible to re-transmit an earlier recorded message on the network. Without proper verification, this re-transmitted message might be processed by the receiver as being valid.

A method to detect and block replay attacks is by using frame counters. Each end-device is keeping track of the number of sent messages (uplink) to the network server (*FCntUp*) and received (downlink) messages from the network server (*FCntDown*). FCntUp is incremented each time the end-device sends a message. FCntDown is tracked by the network server and updated on the end-device each time the network server sends a message to the end-device. Network messages are rejected when it contains a frame counter that is lower than the expected frame counter.

Besides detecting replay attacks, the frame counters are also used for protecting confidentiality and integrity. When the *FRMPayload* field is encrypted, and when a MIC is calculated, frame counters are used as parameters.

Possible weaknesses

- When an end-device has (re-)joined a LoRaWAN using the Activation by Personalization (ABP) method, then resetting the end-device might reset the frame counter at the end-device side only, without resetting the frame counter at the network server side. Such a mismatch in frame counters will prevent a successful communication (rejected frames, failed encryption, and failed

MIC calculations). A (re-)join of the end-device might be required to reset the frame counters at both sides.

3.2.7 *MAC commands*

MAC commands are commands that are used to control the MAC layer in the network. As explained in chapter 3.1.6, there are two methods for sending MAC commands over the network between an end-device, and the network server. One method is using the *FOpts* field and the other method is by using the *FRMPayload* field. When using the first method, the *FOpts* field, the MAC commands are send in clear text over the network.

Possible weaknesses

- MAC commands are sent in cleartext when the *FOpts* field is used. These commands could contain sensitive information, such as information regarding: receive slot timings, channel information and device status.

3.2.8 *Re-transmissions*

Normally, re-transmissions of network frames will happen on a regular basis within a network. If a frame is corrupt or lost, it will usually be send again. This is normal behavior. However, the LoRaWAN specification explicitly warns for uplink frames that require an acknowledgement or an answer by the network- or application server, and that are re-transmitted by the end-device if the answer or acknowledgement is not received. Especially in the situation that such a type of frame can be triggered by an external factor (such as power outages, radio jamming, network outages, etc.). In such an event *a catastrophic, self-persisting, radio network overload situation can be triggered* [20].

Possible weaknesses

- When (answer-/acknowledgement-) frame transmissions can be triggered externally then external factors can have an impact on the performance (and thus availability) of the whole network.

3.2.9 *End-devices*

An end-device that has joined a LoRaWAN network performs integrity checks (MIC calculations) using the *NwkSKey* and provides confidentiality (encryption) using the *AppSKey* for all data communication. If these keys are compromised then others can send and receive data as if it was from the compromised end-device (end-device spoofing). It is important to have a unique key for each end-device. Keys should

not be shared, unless required (such as for Multicast traffic).

Distribution, storing and using different keys on end-devices is an important factor. When someone has physical access to end-devices it becomes relatively easier to get access to these keys using already known attacks vectors such as hacking into the device, side channel attacks, or directly read data from processors and memory chips.

Possible weaknesses

- Key management, especially assignment, distribution and storage of keys.

3.2.10 *Gateways*

On a very high-level, gateways simply perform the forwarding of network packets between end-devices and the network server. Because of the use of integrity checks (MIC calculations) and confidentiality (encryption), the gateway has no access to the actual data that is send over the network. The gateway does not have any of the required encryption keys. In such case, there is little harm if the gateway is compromised.

An exception on this is when MAC commands are send over the LoRaWAN network by using the *FOpts* fields. This method sends MAC commands unencrypted over the network (see Appendix A.3). In this case the MAC commands can be read by everyone who can capture the network traffic. Network capturing can also easily be done on a compromised gateway.

Besides forwarding packets, gateways are also used for downlink timing, and sending network beacons that are used by Class B end-devices. The downlink timing is important because end-devices should receives messages in their receive windows. A network beacon includes a time reference and the Global Positioning System (GPS) coordinates of the gateway. This information is not encrypted and broadcasted by all gateways in a LoRaWAN network. Anyone capturing this network traffic can find out the physical location of gateways (gateway antennas) by looking at the transmitted GPS coordinates.

Possible weaknesses

- The physical location of gateways is broadcasted in cleartext.

3.2.11 *Network Server*

A LoRaWAN network server is the main component of a LoRaWAN networked environment. The network server controls the network (exam-

ples are: accepting and facilitating end-devices and gateways, management of bandwidth all over the network, management of routing between network server and end-devices, and key generation and management).

The Network server contains a lot of (sensitive) information regarding all end-devices, gateways, and application servers. To lower impacts of a compromised network server, data encryption is recommended on the network server. Also, key-generation should be setup in such a way that keys cannot be derived in any way from publicly available information (for example the Device address).

The connectivity between gateways and application servers should be secure. For example, Transport Layer Security (TLS) can be used. TLS is a secure communication protocol which is also used by many web servers (HTTPs).

Possible weaknesses

- A network server contains a lot of keys, addresses and identifiers (see chapter 3.2.3).
- A network server also contains all information required to derive the *NwkSKey* and *AppSKey* for all end-devices.
- The above information is sufficient to see (confidentiality) and modify (integrity) all network messages [11].
- A network server is in many cases (partially) Internet facing, for either the gateway side, application side, or both.

3.2.12 *Application Servers*

The application server hosts the actual IoT application. Such an application processes end-device information and presents the IoT application to end-users. Because this application usually has an Internet facing interface, it is important that the application is well secured. For example by updating software, patching the Operating system, using secure communication based on TLS, and maybe by using client-side certificates for additional authentication.

Possible weaknesses

- Even more than a network server, an application server is almost always Internet facing.
- Besides LoRaWAN related communications, other communications are present as well (for example between end-user and application server, or application server and other online services).
- An application server usually contains additional software packages (web server, data storage, scripting languages, etcetera).

4

RELEVANT ATTACKS

This chapter provides an overview of common (network) attacks that might be applicable to LoRaWAN. From an initial attack selection in the research proposal, a sub-selection is made, based on specific selection criteria. For the selected network attacks, additional analysis is performed, and documented in more detail. The documentation methods used are defined in the research proposal document. These documentation methods include: natural language, Attack-Defense Tree (ADT), and Attack Sequences.

Natural language is used to describe situations, actions, and results. It can be used in any situation and is very flexible. However, it is very difficult to be very precise and exact. It is very difficult to write something in natural language that will be interpreted, the same way, by different persons.

Attack-Defense Trees are evolved from use cases and provide a structural way to represent attacks, including: goals, actions, prerequisites, and countermeasures.

Attack Sequences use the Alice/Bob[1] notation to represent the (order of) communication of network packets.

Section 4.1 in this chapter describes different attacks that are applicable to LoRaWAN. In section 4.2, several criteria are described which are used for attack selection. Section 4.3 describes the selected attacks in more details.

4.1 ATTACK OVERVIEW

The attack overview in table 4.1 shows different examples of (network) attacks that are possibly applicable to LoRaWAN. This overview shows attacks that were defined in the research proposal and are included in the remainder of this research study.

1 https://en.wikipedia.org/wiki/Alice_and_Bob

Table 4.1: Overview of included attacks

Attack	Description
Eavesdropping	Interception of network traffic to capture data. Useful when data is unencrypted or as input for other attacks (such as the replay attack and the Man-in-the-Middle attack).
Replay attack	Capture data and (re)send it when desired. For example, a join request from an end-device towards a gateway, or data packets which are used for modifying connection settings.
Man-in-the-Middle	Try to get in-between an end-device and a gateway to intercept (and modify in real-time) network traffic.
Denial of Service	Sent many network requests so that resources, on specific network nodes, get exhausted. Many different types of network traffic can be used for (Distributed) Denial of Service attacks, including association requests, authentication requests, data packets, etc.
Key cracking	If encryption/decryption keys can be retrieved, encryption can be undone or new, valid, encryptions can be created.

Table 4.2 shows several attack examples from the research proposal that have not been included in the remainder of this research study. The reason for not included these attacks is explained in the description field for every entry.

Table 4.2: Overview of excluded attacks

Attack	Description
Session Hijacking	Can be used for taking over existing sessions. *This technique is also used as an example in the replay and Man-in-the-Middle attacks as well.*
Evil Twin / Rogue AP	Setup a new gateway to impose an existing one, or to which clients will just connect to. This new gateway will act as a starting point for other possible attacks. *LoRaWAN allows setting up new gateways by anyone (by using open source software). The protocol has been defined to make this possible, and to distinct gateway functionality from most of the network traffic by using message signing and encryption. It is not possible to setup an Evil Twin / Rogue AP / gateway which allows access to sensitive data.*

Overview of excluded attacks – continued

ATTACK	DESCRIPTION
Gateway tampering	Try to get an existing gateway modified (physical or by software modifications). Is it possible to change gateway settings remotely? *This research study has limited the scope to the LoRaWAN protocol specification. Physical environments have been excluded from the scope.*
Frame manipulation	Change the order/size of network frame fragments. Manipulating frames can be used as an evading technique to prevent certain types of detection, for example, bypassing checks done by firewalls. Similar techniques are called Frame Injection and Frame Fragmenting. *This subject is mostly implementation specific, but there is a relation with the LoRaWAN protocol specification. However, because of time constrains, this subject has been excluded from this research study.*
MAC spoofing	Impersonate another device by taking over uniquely identifiable addresses. *The LoRaWAN protocol specification does not use MAC addresses like the TCP/IP standard. It does use its own addressing scheme which is partly based on keys. Keys are part of the research study as part of the Key cracking entry.*
Buffer overflow	Enforcing buffer overflows (in frames, packets, data, etc.) trying to impact network functionality. *This research study has limited the scope to the LoRaWAN protocol specification. Buffer overflows are, usually, implementation related.*

4.2 SELECTION CRITERIA AND LIMITATIONS

From the initial list of attacks listed in the research proposal, only several have been selected for further examination in this research study (see table 4.1). The reason for limiting the number of attacks has been driven by scope decisions. Since there are many (types of) attacks, only a limited number of known attacks will be selected during different phases of this research study. Without limiting the number of attacks, it will be difficult to meet the proposed planning. Another reason is that this research study will be looking at logical aspects of the LoRaWAN specification based on the formal specification. Physical and implementation aspects are not in scope of this research study.

4.3 ATTACK DETAILS

The remainder of this chapter provides additional details on the selected attacks.

4.3.1 *Eavesdropping*

One of the most used network attacks is eavesdropping. Eavesdropping in an attack context is like eavesdropping in the real world. It is the action of listening (sniffing), and often recording (capturing), a conversation. During eavesdropping, data remains intact, but its privacy is compromised. This does not mean that the data, which has been captured, can be read or understood.

Eavesdropping is a very common attack and often part of a larger attack scheme. Many (more complicated) attacks are based on captured data. This captured data is used in some form, which is depending on the type of attack.

Attack-Defense Tree
Figure 4.1 shows the ADT for the eavesdropping attack. In a network environment, eavesdropping can be performed on the wired network, or by using a specific device and 'listen' in the air for radio signals that represents wireless network packets. To perform eavesdropping, it is not required to become active on, or a member (having an address) of, the network.

When eavesdropping a wireless network, the received radio signals are transformed into network data packets, and usually stored for later use. Receiving and capturing wireless network packets cannot be prevented. However, a possible counter-measure is to use encryption so that the captured data cannot be read (easily).

For eavesdropping a wired network, physical access to the network is required. Examples are, a wall-outlet in an office environment, or a switch port in a more technical (data center) environment. Once access has been established, listening and capture data is usually easier compared to wireless networks. However, depending on the network setup, and where in the network the physical connection has been made, results may vary.

Nowadays, most networks are fully switched TCP/IP based networks. In switched networks, connected devices only receive broadcast traffic and traffic that is specifically addressed to the device itself. Devices will not receive any traffic that is addressed to other devices. Switches will keep this traffic separated (unlike network hubs). This means that

when using the wall-outlet in an office environment there will not be that much network traffic that can be sniffed. If somehow, access to a switch is possible than there are options for using specific switch ports that do show all network traffic that is passing the switch (these are called *span ports*).

Counter-measures against sniffing wired networks are using switch port security, so that it is not possible to connect non-authorized devices to network wall-outs, and by using encryption so that the captured data cannot be read (easily).

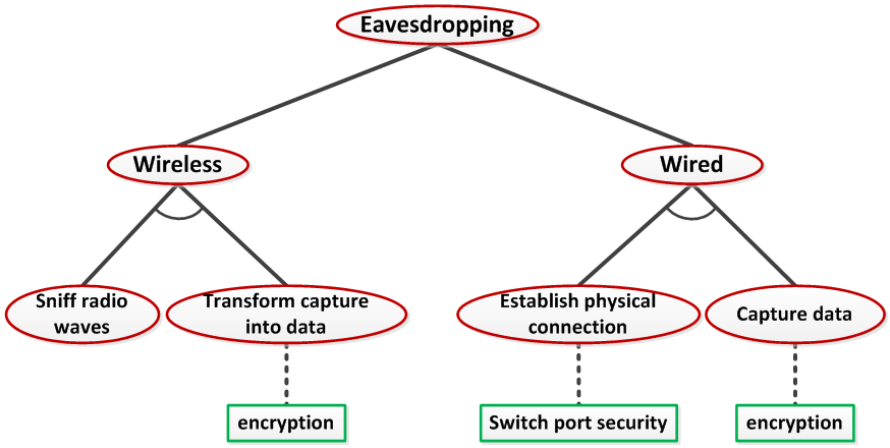

Figure 4.1: Attack-Defense Tree: Eavesdropping

Attack Sequence

An example of eavesdropping is shown in figure 4.2. In this example the first three steps are used to setup the TCP/IP network session. After the session has been established, Alice likes to access something at Bob's side (4). Before this is allowed, Bob asks for proof of authentication (5). In return Alice sends her credentials towards Bob (6). From here on the conversation between Alice and Bob continues (7). Meanwhile, the conversation between Alice and Bob has been sniffed, and possible captured, by Eve. In the case of a passive sniffing action, both Alice and Bob have no way of detecting this action by Eve.

If the conversation between Alice and Bob has been unencrypted, then Eve knows the username and password of Alice. If the conversation has been encrypted, then Eve cannot read username and password, but still has network data that represents the authentication of Alice towards Bob. In this case, there might be other attacks that still could benefit from this data.

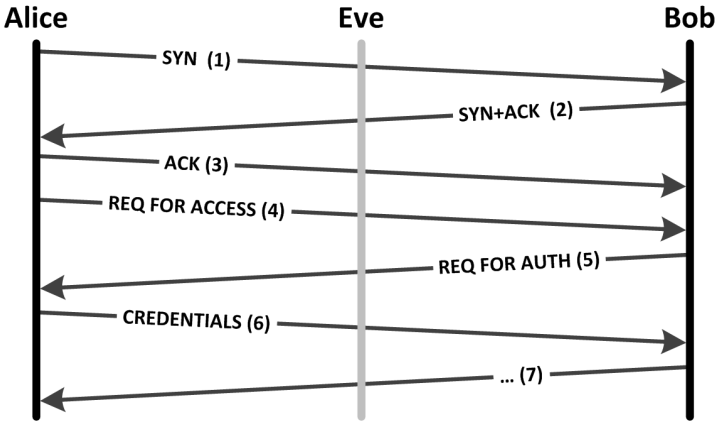

Figure 4.2: Attack Sequence: Eavesdropping

Possible applicability to LoRaWAN
An *eavesdropping* attack can be performed on any component of a LoRaWAN implementation where an adversary can get access to network communication. The easiest component for this attack is the wireless network communication. All communication between end-devices and the Network Server will always pass the wireless network.

If traffic between the Network Server and the gateways, or between the Network Server and Application Servers, needs to be sniffed, then in most cases access to wired networks is required.

Most of the traffic that can be sniffed in a LoRaWAN implementation is encrypted. Additional activities are required to get access to the real transmitted data (if possible at all).

4.3.2 *Replay attack*

A replay attack is one of the attacks that uses information that was captured using eavesdropping. During a replay attack, network packets that have been captured during earlier sniffer sessions, are re-send on the network at a later moment in time.

Even when transmitted data is encrypted, and thus not readable, these kinds of attacks might have a significant impact. However, this is depending on many factors, such as the type of data that was captured, the time when the data is re-send, the setup and configuration of the receiver, the type of network, etc.

A replay attack can be used for many purposes but it is often used for authentication (username/password, sessions, network joins, etc.)

related attacks.

Attack-Defense Tree

Figure 4.3 shows the ADT for a replay attack. The basis of a replay attack is data that has been captured during earlier sniffer sessions. Because of this, eavesdropping is an important part of a replay attack.

The second part of a replay attack is to re-play captured network data (in original or slightly modified form). Replay attacks can be used for very simple situations, such as plaintext credential capturing and re-use, but also for more complicated attacks. An example of a more complicated attack is when captured data is encrypted, but specific authentication information can be extracted for re-use. This re-use could be plaintext (when decryption is possible) but sometimes re-transmitting encrypted data can even result into a successful replay attack.

Possible countermeasures are the use of frame counters on a network level (when re-transmitting frames exactly as they are captured), and/or by implementing correct encryption levels.

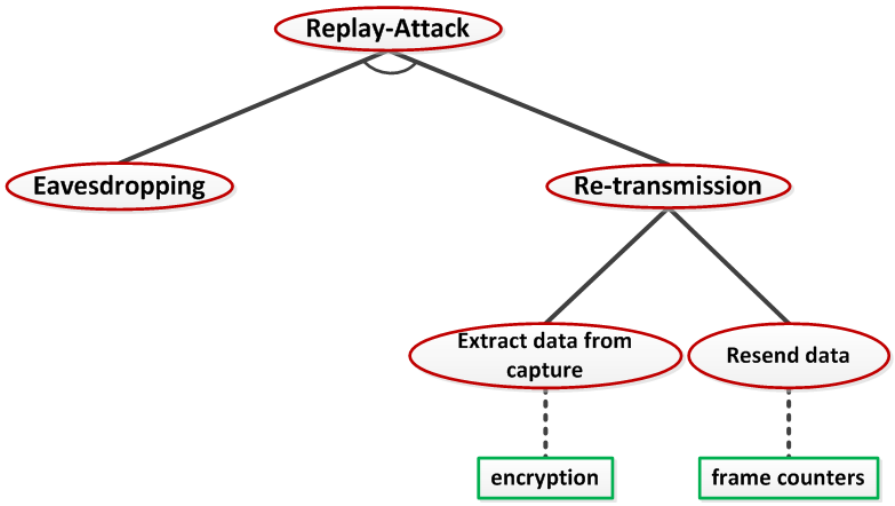

Figure 4.3: Attack-Defense Tree: Replay attack

Attack Sequence

An example of a replay attack is shown in figure 4.4. A replay attack is based on earlier sniffing and capturing sessions. In this example Eve has captured login credentials from Alice during an eavesdropping attack, see figure 4.2. In the case of a passive sniffing action, both Alice and Bob have no way of detecting this action by Eve.

If the conversation between Alice and Bob has been unencrypted, then Eve knows the username and password of Alice. Eve can now

re-use this information. If the conversation has been encrypted, then Eve cannot read username and password, but still has network data that represents the authentication of Alice towards Bob. In certain situations even encrypted authentication data can be re-used. However, this is depending on the setup and configuration of the involved components such as Operating System, network protocol, Application, etc.

During the replay attack, Eve starts a session with Bob (steps one to five) and as soon as credentials must be provided, Eve provides the credentials of Alice, which were captured during the sniffing session (step 6'). From this moment, Eve has access with the identity of Alice.

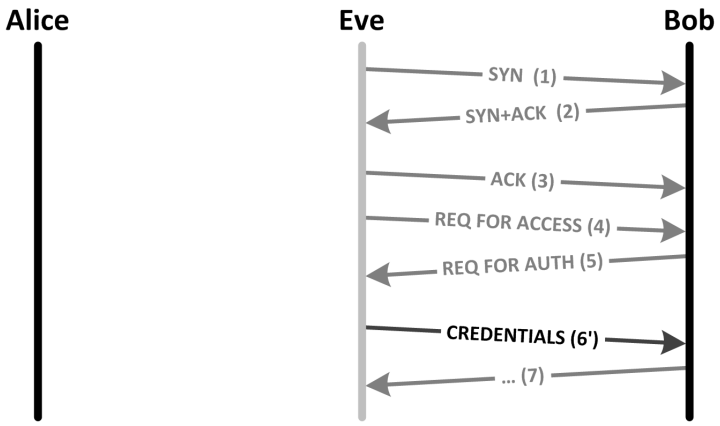

Figure 4.4: Attack Sequence: Replay Attack

Possible applicability to LoRaWAN
A *replay* attack can be performed on any component of a LoRaWAN implementation where an adversary can get access to network communication. The easiest component for this attack is the wireless network communication. All communication between end-devices and the Network Server will always pass the wireless network.

If traffic between the Network Server and the gateways, or between the Network Server and Application Servers, needs to be replayed, then in most cases access to wired networks is required.

Most of the traffic that can be sniffed in a LoRaWAN implementation is encrypted. Additional activities are required to get access to the real transmitted data (if possible at all). A *replay* attack is therefore not easy. However, there is unencrypted traffic, and maybe there are options to replay encrypted data packets.

4.3.3 *Man-in-the-Middle*

In a Man-in-the-Middle (MitM) attack, an adversary places himself into a conversation between two parties and impersonates both parties to gain access to the data that the two parties are sending to each other. It involves an active form of eavesdropping. A man-in-the-middle attack allows an adversary to intercept, send and receive data meant for someone else, without both parties knowing. This attack is a form of session hijacking and is often used in wireless communications.

Attack-Defense Tree
Figure 4.5 shows the ADT for a MitM attack. In a MitM attack an adversary needs to impersonate both the sender and receiver. On behalf of both parties, it needs to perform send- and receive activities so that it can proxy, or relay, traffic between both parties.

The main countermeasures for this kind of attack is the use of encryption for the data that is being send, and signing of messages so that the receiver is sure of the sender's identity.

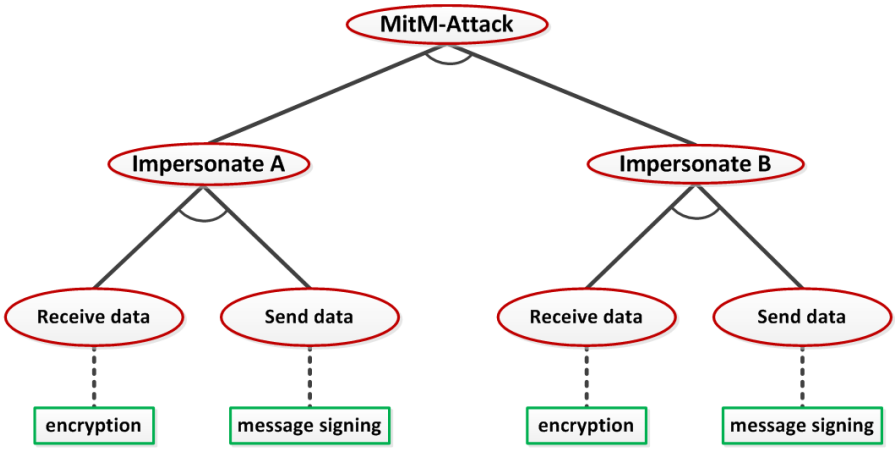

Figure 4.5: Attack-Defense Tree: Man-in-the-Middle attack

Attack Sequence
An example of a MitM attack is shown in figure 4.6. In this example, the communication between Alice and Bob is proxied/relayed by Mallory. All outgoing traffic, from both Alice and Bob, is received by Mallory instead by the intended party. Mallory can view, and modify if required, the data before the data is re-transmitted to the correct receiver.

In the attack sequence, it is clear that the conversation is based on two sessions. A session between Alice and Mallory, and a second ses-

sion between Mallory and Bob. Alice and Bob think they are having a single session between the two of them. It may be clear that when the data is send unencrypted that Mallory can see, and even modify, the data before it is relayed to the intended destination.

During the MitM attack example in figure 4.6, Alice starts a session with Bob (step 1). However, Mallory got in-between the communication, and Alice is now communicating with Mallory while still thinking she is in direct contact with Bob. Mallory relays the data (possibly modified) towards Bob, who thinks the data comes directly from Alice (step 1'). The answer from Bob is send towards Alice (step 2). However, Mallory is again in-between both parties and receives the answer. Mallory relays the answer back to Alice (step 2'). The remainder of the conversation continues in a similar way with Mallory actively relaying the data.

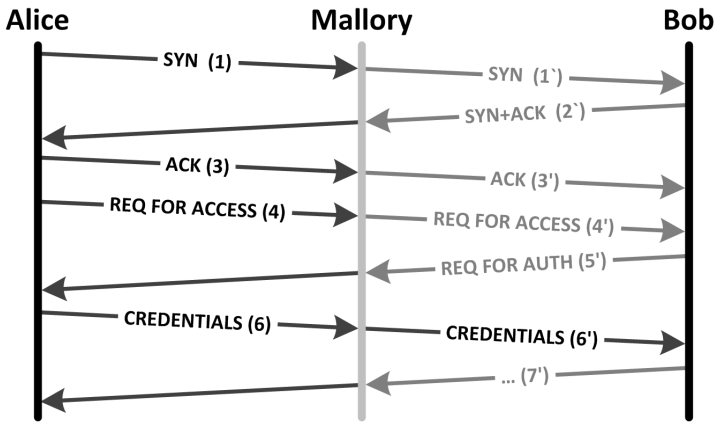

Figure 4.6: Attack Sequence: Man-in-the-Middle attack

Possible applicability to LoRaWAN

A *MitM* attack can be performed on any component of a LoRaWAN implementation where an adversary can get access to network communication. The easiest component for this attack, is the wireless network communication. All communication between end-devices and the Network Server will always pass the wireless network.

If a *MitM* attack between the Network Server and the gateways, or between the Network Server and Application Servers, is required, then in most cases access to wired networks is required.

Most of the traffic that can be sniffed in a LoRaWAN implementation is encrypted. Additional activities are required to get access to the real transmitted data (if possible at all). In the LoRaWAN specification, network frame counters are defined. Encryption and frame counters are successful counter-measures against *MitM* attacks.

4.3.4 *Denial of Service*

A (Distributed) Denial of Service ((D)DOS) attack is one of the more popular attacks nowadays because it can cause great harm with reasonably simple efforts. The goal of a (D)DOS attack is to make sure that the service under attack becomes unavailable (due to overload) for a certain period of time. Usually this is accomplished by exhausting resources in the environment that is providing the service.

During a DOS attack, only one source host is used in the attack. This type of attack is not used very much anymore. These days, infrastructure providing services can handle a lot of traffic, more traffic than can be initiated by a single source host. Because of this, (D)DOS attacks have increased in popularity. In this type of attack, the attack is being initiated by many different hosts instead of a single one. With the rise of botnets, which are virtual networks of hijacked hosts that are under control of malicious actors, such (D)DOS attacks have become easier and more powerful.

(D)DOS attacks are usually all about exhausting resources. There are two major categories for resource exhaustion during (D)DOS attacks:

- Exhaustion of network resources: by sending excessive number of requests (network packets). Sending excessive amounts of network requests can be accomplished by using many source hosts, but also by making use of amplification (amplification is a way for an adversary to magnify the amount of bandwidth they can target at a potential victim; examples are the DNS and ICMP protocols).
- Exhaustion of cpu/memory/disk used by the online application processes.

Attack-Defense Tree

Figure 4.7 shows the ADT for a (D)DOS attack. A (D)DOS attack can either by a Denial of Service (DOS) or a Distributed Denial of Service (DDOS). In case of a DOS only a single host is initiating the attack. Because it is only a single host, blocking all traffic from this host can be very effective countermeasure. For a (D)DOS attack, an adversary needs to have multiple (many) of hosts under control. From all these hosts traffic should be initiated towards the target service so that resources on the target service become exhausted. Countermeasures against a (D)DOS attack are very hard because requests are initiated from an enormous number of hosts. It is impossible to distinguish between valid and invalid traffic. Usually it turns out that resources need to be added so that the target service has more resources than the (D)DOS attack can use. However, this is very difficult and costly.

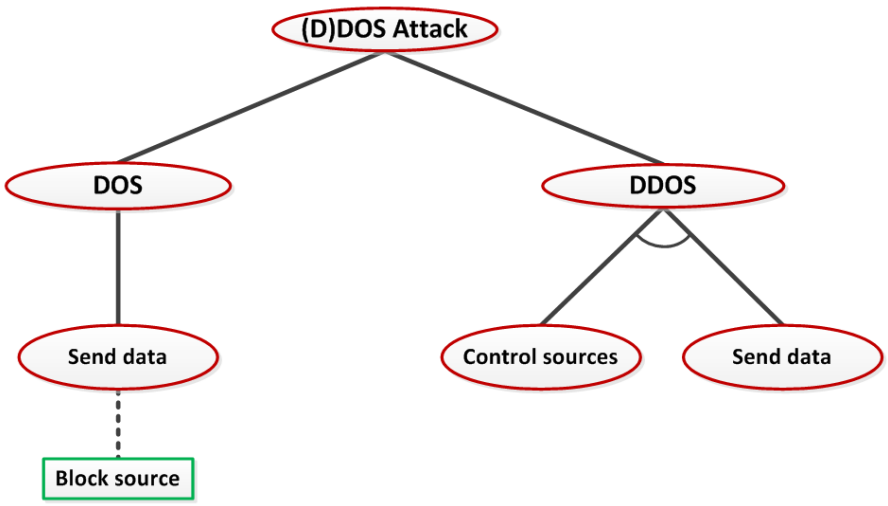

Figure 4.7: Attack-Defense Tree: DDOS attack

Attack Sequence
An example of a (D)DOS attack is shown in figure 4.8. Bob is receiving so many network packets from Eve, that the request from Alice cannot reach Bob anymore.

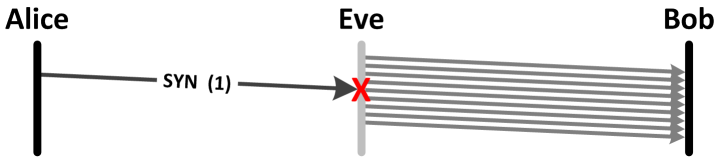

Figure 4.8: Attack Sequence: DDOS Attack

Possible applicability to LoRaWAN
A *(D)DOS* attack can be performed on any component of a LoRaWAN implementation where an adversary can get access to network communication. The most obvious target components within the LoRaWAN communication chain would be the end-devices and the Network Server.

For now, it is difficult to determine which type of (D)DOS would be the most efficient for the individual LoRaWAN components.

As mentioned in chapter 3.2.4, Join-Accept messages might be a possible type of communication that can be used for (D)DOS attacks.

4.3.5 *Key cracking*

Encryption is a very important technique which makes it possible to communicate information in a safer way. Encryption translates data (also known as plaintext) into unreadable data (cipher text). This

translation (encryption) is based on an encryption algorithm which is based on keys to generate unique cipher text. According to Kerckhoff's principle[2], the encryption algorithm is preferred to be public. The used keys should provide the strength of the encryption.

Kerckhoff's principle shows the importance of keys. If a key is compromised then all the encryption based on that key can be decrypted, but also new encryption can be created using the compromised key. A key can be compromised in many ways, such as during key usage, storage of the key, communication of the key, etc. There are many different known attacks to compromise a key. The most common used attacks are explained below:

Ciphertext-only
A *ciphertext-only* attack is an attack where the attacker only has access to (pieces of) ciphertext. The used encryption must be determined based on this ciphertext. This is a very difficult approach to key cracking. Increasing the amount of ciphertext might make the process a bit easier, but nonetheless the attack is very difficult (to impossible) to realize.

Known plaintext
In this attack, an attacker has access to both the plaintext and ciphertext of the same message. The goal is to find a relation between the plaintext and ciphertext which can lead to finding the used encryption key.

Chosen plaintext
When using a *chosen plaintext* attack, an attacker has access to both the plaintext and ciphertext of any message that he can generate himself. The attacker has access to the encryption method and can encrypt any message he wants. The attacker can choose any plaintext which he can encrypt using the available encryption method. Like the *known plaintext* attack, the goal is to investigate the relation between plaintext and ciphertext to find the used encryption key. In this case the attacker has more detail since he can generate new ciphertext messages from any plaintext message he wants.

Chosen ciphertext
This attack is like the *chosen plaintext* attack, however, in this attack the attacker has the ability to use an available decryption method. The attacker can choose to decrypt any ciphertext he has access to. Although the attacker has a way to decrypt ciphertext, the options are less compared to encrypting plaintext in the *chosen plaintext* at-

2 http://www.petitcolas.net/kerckhoffs/

tack.

Side channel attacks
Next to using an attack method that is based on the encryption algorithm to determine the used encryption key, it is also possible to generate and analyze data from an implemented encryption (Implementation Attack).

- *Differential cryptanalysis*: this analysis measures the exact execution times and power use of the cryptographic device that performs the encryption or decryption. In some cases, the results of such an analysis can be used to determine the encryption key and algorithm used.
- *Fault analysis*: the system is forced into an error state so that differences and behavior, compared to a correct situation, can be studied to learn more about the encryption key and algorithm used.
- *Probing attacks* : this is the monitoring of components surrounding the encryption/decryption module to see if these components disclose any information about the used encryption key and algorithm.

Dictionary attack
By using a *dictionary attack*, words and/or phrases are used as a possible decryption key (or as a possible password). These words and phrases are compiled from dictionaries, common word lists, etc.

Related key attack
Keys used for encryption should be kept secret. However, sometimes encryption keys are based on, or derived from, other information. When this other information is known, it might be possible to regenerate existing keys or generate new keys. For example, this is the case when an encryption key of an end-device is based on hardware characteristics (MAC address) of the end-device.

Social engineering
This is a very common attack. Social engineering skills are used to pursue people into providing details regarding passwords, keys, etc. The human factor is probably one of the greatest weaknesses.

Brute force
When using a *brute force* attack, all possible combinations are tried, one by one. E.g. aaa aab aac aad, ... zzz. Key lengths and diversification play a very important role in the success rate of a brute force attack. The longer and more diverse the key, the longer it takes to try all different combinations.

Reverse engineering
By reverse engineering a product, or source code, an attacker might find weaknesses, clues, or other information that could help in breaking the encryption.

Random Number Generators
Many encryption algorithms are highly depending on random numbers. When random numbers are not so random anymore, an attacker might be able to predict values. This behavior can be used in certain cryptographic attacks.

Possible applicability to LoRaWAN
A Key Cracking attack can be targeted on different aspects of LoRaWAN. However, most key cracking attacks are implementation related attacks and therefore out of scope for this research study.

Additional research is required to determine which aspects of the LoRaWAN specification can be targeted for key cracking activities. Areas to research are: brute force feasibility, key relations, and possibly others.

5 ATTACK ANALYSIS

This chapter provides an overview of the results of the attack analysis. For several types of attacks, a detailed study has been performed to determine the impact on the LoRaWAN specification. The studied attacks (methods) are: Eavesdropping, Replay, Denial of Service, Cryptographic, and Man-in-the-Middle.

Section 5.1 in this chapter, describes the possibilities and impact of an eavesdropping attack on the LoRaWAN specification. In section 5.2, the replay attack is described. Section 5.3 describes the DOS attack. In section 5.4, cryptographic attacks are described. Section 5.5 provides more details on the MitM attack.

5.1 EAVESDROPPING ATTACK

An eavesdropping attack is an attack that eavesdrops network traffic to (possibly) get hold of important information. The eavesdropped data can also be captured for later use, for example in other types of attacks. See chapter 4 for more details on the eavesdropping attack.

The LoRaWAN specification can be used to determine what information can be eavesdropped on a LoRaWAN implementation. A structured way to study the different types of information is to determine which types of traffic (network frames) are being sent over the network. The different types of network frames have been categorized into communication scenarios. These communication scenarios have been documented in detail in Appendix A. The results are summarized in the remainder of this section.

The communication scenarios that are defined are:

- End-device activation (Join-Request & Join-Accept)
- Sending data
- Sending MAC commands (Dedicated & Piggy-backed)
- Multi-Cast messages
- Beaconing

The overviews in Appendix A shows the different network frames per communication scenario. The field values in these network frames are based on an actual implementation to represent the network frames as accurate as possible.

Table 5.1 lists all network frame fields that can be captured. These network frames are communicated between end-devices and the network server (via the gateways).

Table 5.1: Overview of data that can be eavesdropped

Data field	Remark
AppEUI	Application Identifier. A global application ID in the IEEE EUI64 address space that uniquely identifies the entity able to process the Join-Request network frame. The AppEUI is, before end-device activation, communicated out-of-band, so that the end-device can use this value as a parameter in the Over-the-Air-Activation (OTAA) procedure. The out-of-band communication is not used for secrecy but to provide the owner of the end-device with a parameter that is required for the join procedure. This value is, combined with DevEUI and DevNonce, used by the network server to generate a valid Join-Response message. Scenario: *Join-Request*.
DevEUI	Global end-device Identifier in the IEEE EUI64 address space that uniquely identifies the end-device. The DevEUI is, before end-device activation, communicated out-of-band, so that the end-device can use this value as a parameter in the Over-the-Air-Activation (OTAA) procedure. The out-of-band communication is not used for secrecy but to provide the owner of the end-device with a parameter that is required for the join procedure. This value is, combined with AppEUI and DevNonce, used by the network server to generate a valid Join-Response message. Scenario: *Join-Request*.
DevNonce	This is a random value that is, combined with AppEUI and DevEUI, used by the network server to generate a valid Join-Response message. This value is used to prevent replay attacks. The network server remembers the AppEUI, DevEUI, and DevNonce combinations that have been activated. Scenario: *Join-Request*.
<encrypted Join-Accept>	The complete, encrypted, Join-Accept message (which contains AppNonce, NetID, DevAddr, DLSettings, RxDelay, and optionally CFList details). A Join-Accept message is encrypted using the AppKey. Without the AppKey it is not possible to read the Join-Accept message details. Scenario: *Join-Accept*.
DevAddr	End-device address. 32-bit identifier. Bit 0..24: NwkAddr, and 25..31:NwkID. Scenario: *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.

Overview of data that can be eavesdropped – continued

Data field	Remark
NwkAddr	Network address of the end-device; assigned by the (local) network manager. The specification does not state that NwkAddr must be unique. The Things Network, for example, uses a combination of DevAddr (NwkAddr) and NwkSKey (to verify MICs) to uniquely identify an end-device (which is also related to an Application Identifier on the network server). Scenario: *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.
NwkID	Network Identifier to separate addressing of territorially overlapping networks of different operators. It also solves (roaming) issues when end-devices change between different networks. The NwkID is assigned by the LoRa Alliance. Scenario: *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.
ADR	Adaptive Data Rate. When set, the network will control data rate. Can be set / unset by end-device or network on demand. If possible, it should be set to increase battery life of end-devices. Scenario: *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.
ADRACKReq	ADRACKReq (uplink) request ADR acknowledgement to verify receival of uplink frames (verify network connections with higher data rates). ADR acknowledgement request has influence on scheduling of downlink traffic. Scenario: *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.
ACK	Message acknowledgement for a confirmed data message. Can be empty frame or piggyback on data message. ACK needs to be replied on a Confirmed data message. Scenario: *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.
RFU (6x)	MHDR (bit 2..4), DLSettings (bit 7), RxDelay (bit 4..7), CFList (bit 0..7), Fctrl:bit 4 for upload frames, and bit 6 for download frames. (bit 4 shared with Class B/F-pending.) Reserved for Future Use fields should use value 0 (by default). Not specified how implementation should handle non-0 values. Scenario: *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.

Overview of data that can be eavesdropped – continued

Data field	Remark
Class B	Signals the network server that the device has switched to Class B mode and is now ready to receive scheduled downlink pings. (Bit 4 shared with RFU/F-pending.) Scenario: *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.
FPending	Frame pending bit (downlink communication). Indication in downlink communication that the gateway has more data to send. End-device needs to open another receive window asap. (Bit 4 is shared with RFU/Class B) Scenario: *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.
FOptsLen	Frame-options length. Actual length of FOpts field value in the network frame. 0:no FOpts, >0:MAC commands are present in FOpts. Scenario: *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.
FOpts	Frame-options. Can contain unencrypted(!) MAC commands with a max length of 15 octets piggy-backed onto data frames. When used, FPort cannot be 0. Scenario: *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.
FPort	Port field. Must be present if the FRMPayload field is not empty. 0:MAC commands only, 1.223:application specific, and 224:LoRaWAN layer test protocol. Indicated the purpose of the FRMPayload field: MAC commands, application payload, testing, or RFU. Scenario: *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.
<encrypted payload>	The encrypted data (or MAC) payload. The payload is encrypted using the NwkSKey (MAC commands) or AppSKey (app payload). Without these session keys, it is not possible to read the payload details. Scenario: *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.
MType	Message type; distinction between six different message types (000:Join-Request, 001:Join-Accept, 010:Unconfirmed Data Up, 011: Unconfirmed Data Down, 100:Confirmed Data Up, and 101:Confirmed Data Down). There are two additional types listed: 110:Reserved for Future Use, and 111: Proprietary. Implementation should ignore these. Scenario: *Join-Request*, *Join-Accept*, *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.

Overview of data that can be eavesdropped – continued

Data field	Remark
Major	Major version of data message. Shows which Major version message format of the LoRaWAN specification is used. 00:LoRaWAN R1, 01..11:Reserved for Future Use. Scenario: *Join-Request*, *Join-Accept*, *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.
MIC	Message Integrity Code. The MIC is based on the first four bytes of the CMAC value. The CMAC value is based on the NwkSKey, MHDR, FHDR, Fport, FRMPayload, traffic direction, DevAddr, and Frame Counters. Scenario: *Join-Request*, *Join-Accept*, *Sending Data*, *Sending MAC commands*, and *Multi-Cast messages*.

Table 5.2 lists all network frame fields that can be captured. These network frames are communicated between end-devices and gateways (scenario: *Beaconing*).

Table 5.2: Overview of data that can be eavesdropped (2)

Data field	Remark
NetID	Seven Least Significant Bits: NwkID. Seventeen Most Significant Bits: freely chosen by network operator.
Time	Time (seconds since 1/1/1970 00:00) when beacon was send. This is important for opening receive windows by the end-device at the correct intervals.
InfoDesc	Provides details on the next GWSpecific part. It indicates the coordinates for which antenna (0:first, 1:second, 2:third), 3-127:RFU, or 128-255 Reserved for customer network specific broadcasts.
Latitude	Geographical (lat) location of gateway. Reveals physical location of the gateway.
Longitude	Geographical (long) location of gateway. Reveals physical location of the gateway.

Table 5.3 lists important information that **cannot** be captured using the eavesdropping attack. This information is not transmitted over the LoRaWAN network.

Table 5.3: Overview of data that cannot be eavesdropped

Data field	Remark
AppKey	End-device specific root key. Communicated out-of-band between network server and end-device before the end-device is activated on the network. The out-of-band communication is used for secrecy so that this value is never transmitted over the LoRaWAN. This key is used for encryption/decryption of the Join-Accept message and deriving the AppSKey and NwkSKey session keys specific for that end-device to encrypt/decrypt and verify network communication and application data.
AppNonce	Random and Unique value provided by the network server to derive session keys (NwkSKey & AppSKey).
AppSKey	Used for encryption of the application payload. Derived from AppKey, AppNonce, NetID, and DevNonce.
NwkSKey	Used for encryption of MAC commands and MIC calculations. Derived from AppKey, AppNonce, NetID, and DevNonce.
CMAC	Cipher-based Message Authentication Code (CMAC) used to determine the integrity of the message. When the CMAC and data used to calculate the CMAC is available, a brute force method could be used to determine encryption keys. However, only the first four bytes of the CMAC are included in network frames in the MIC field value.

Conclusion

Based on studying eavesdropping on LoRaWAN, it has become clear that a lot of data can be captured from the wireless part of a LoRaWAN network. However, the most important data is protected by using encryption (the Join-Accept message, FRMPayload, and MIC field values).

The data that is available when using an eavesdropping attack can provide insights into a LoRaWAN implementation. Examples are:

- the NwkID shows which network is being used (more details about assigned NwkIDs are available on the LoRa Alliance website)
- the Class B field, which shows if an end-device is using scheduled receive slots (class-B operations)
- settings about Adaptive Data Rate (ADR) can be seen, which gives indications about speeds, transmission quality, and availability

- MAC commands, when piggy-backed onto a data network frame, will show networks commands between end-devices and the network server

All these examples do not reveal any sensitive information, but can provide insight into the setup, or configuration, of a LoRaWAN implementation, which can possibly be used in other types of attacks.

A small side-note must be made for proprietary messages. LoRaWAN allows the use of proprietary message types (MType value 111). The use of these message is fully depending on the agreements made between end-devices and network server. The built-in security measures of LoRaWAN might not be applicable to these types of messages.

Eavesdropping on the TCP/IP network connections between gateways and the network server is more difficult. The recommendation is to have the LoRaWAN traffic tunneled over a Transport Layer Security (TLS) encrypted connection.

Two important follow-up attacks are the replay attack, which can be used to (re-)transmit (modified) captured data, and a brute-force attack, which might be used to brute-force MIC values, and the NwkSKey or AppKey values. The brute-force attack is a very complicated and time-consuming attack. More details about these attacks are provided in chapters 5.2 and 5.4.

5.2 REPLAY ATTACK

A replay attack is an attack that re-transmits (modified) data which has been captured during an earlier eavesdropping attack. See chapter 4 for more details on the replay attack.

The LoRaWAN specification can be used to determine what information can be re-transmitted onto a LoRaWAN implementation, and what the possible impact might be for some of those re-transmits. It is not possible to determine all impacts since this is often depending on implementation details.

Like the eavesdropping attack study, the communication scenarios from Appendix A are used to study possible replay attacks. Possible replay scenarios based on the communication scenarios have been documented in detail in Appendix B. The results of the replay scenarios are summarized in the remainder of this section.

End-device activation

The end-device activation process is based on a Join-Request network message from end-device to Network server, and a Join-Accept net-

work message from network server to end-device.

The Join-Request message uses a network frame that includes a MIC, but it does not apply any encryption. Figure 5.1 shows an Attack-Defense Tree (ADT) for three possible replay attacks for the Join-Request message. Before replay attacks can be performed, an eavesdropping attack is required to capture a valid and existing Join-Request message.

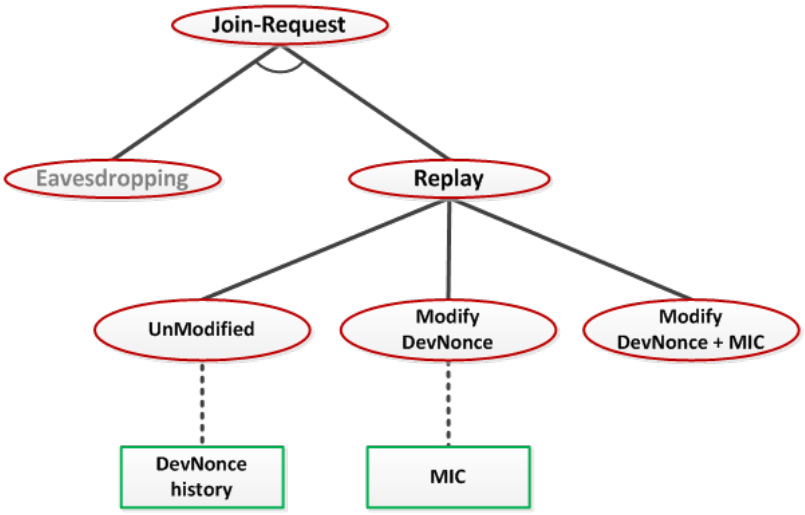

Figure 5.1: Join-Request: Replay scenarios

The first replay attack is performed by re-transmitting the Join-Request network frame without any modifications. This attempt will fail because the AppEUI+DevEUI+DevNonce combination is already known as an active device by the network server. However, if the DevNonce is not known by the network server (for example, because it is a very old message), then the Join-Request will be processed by the network server, potentially disrupting the current connectivity of the end-device.

The second replay attack is performed by first modifying the DevNonce field value before re-transmitting the network frame. This attempt will fail because the MIC verification will fail. The MIC is based on field values such as DevNonce, which has been changed in this second replay attack.

The third replay attack is performed by first modifying the DevNonce and MIC field values before re-transmitting the network frame. This attempt will be successful. However, creating a valid MIC is not possible with current technology. See Appendix B and chapter 5.4 for more details.

The Join-Accept message is almost entirely encrypted. Without breaking the encryption, the only replay-attack can be performed by re-transmitting the Join-Accept frame without any modifications. The LoRaWAN specification does not include details on the behavior of an end-device when a Join-Accept message is received for a second time. If the end-device accepts a previous Join-Accept message, then there might be an issue if the end-device starts using the details from this previous Join-Accept message (such the AppNonce field) while the network server might use a newer AppNonce value. If this is the case then the end-device is not able to communicate with the network server anymore until it performs a new OTAA.

Sending data

There are several ways for sending data, uplink traffic, downlink traffic, confirmed, unconfirmed, and multi-cast messages. All these different types of communication use the same network frame. In some cases, there are small differences in used control fields. Because of the identical network frame, all situations are represented by the same Sending Data scenarios.

Figure 5.2 shows an ADT for four possible replay attacks for Sending Data. Before replay attacks can be performed, an eavesdropping attack is required to capture a valid and existing Sending Data message.

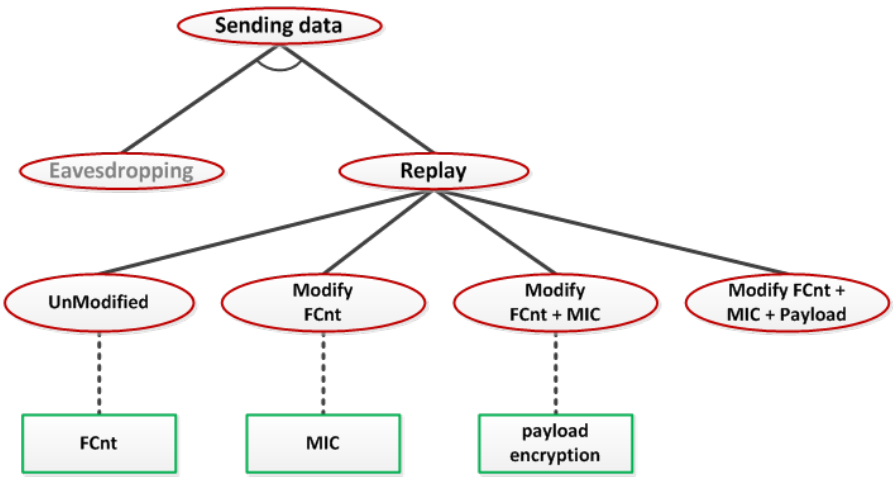

Figure 5.2: Sending data: Replay scenarios

The first replay attack is performed by re-transmitting the Sending Data network frame without any modifications. This attempt will fail because the frame counters will prevent processing of the re-transmitted network frame.

The second replay attack is performed by first modifying the frame counter (FCnt) field value before re-transmitting the network frame.

This attempt will fail because the MIC verification will fail. The MIC is based on field values such as FCnt, which has been changed in this second replay attack.

The third replay attack is performed by first modifying the FCnt and MIC field values before re-transmitting the network frame. This attempt will be successful in re-transmitting the same application data. Transmitting new application data will fail because of the FRMPayload encryption. When modifying field values, such as FCnt, a new MIC must be generated. Generating a new, and valid MIC, is not possible with current technology. See Appendix B and chapter 5.4 for more details.

The fourth replay attack is performed by first modifying the FCnt, MIC and FRMPayload field values before re-transmitting the network frame. This attempt will be successful in re-transmitting new application data. However, it is very difficult to break the encryption that is used for the FRMPayload field value.

Sending MAC commands

MAC commands can be send in two different ways, dedicated (as the FRMPayload field value), or piggy-backed (in the FOpts field value in a Sending Data network frame). Replay attacks on the dedicated method are identical to the Sending Data replay attacks.

Replay attacks on the piggy-backed method are represented in the ADT in figure 5.3. Before replay attacks can be performed, an eavesdropping attack is required to capture a valid and existing MAC message.

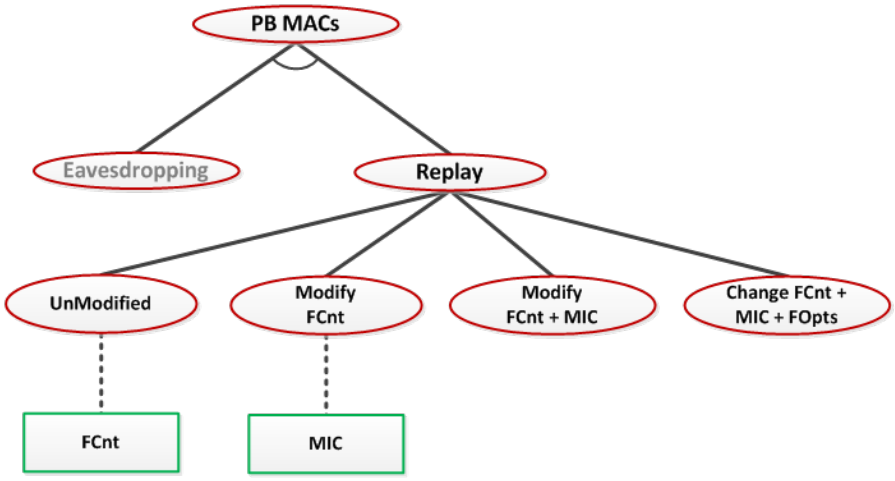

Figure 5.3: Sending MAC (piggy-backed): Replay scenarios

The first replay attack is performed by re-transmitting the Sending Data network frame without any modifications. This attempt will

fail because the frame counters will prevent processing of the re-transmitted network frame.

The second replay attack is performed by first modifying the frame counter (FCnt) field value before re-transmitting the network frame. This attempt will fail because the MIC verification will fail. The MIC is based on field values such as FCnt, which has been changed in this second replay attack.

The third replay attack is performed by first modifying the FCnt and MIC field values before re-transmitting the network frame. This attempt will be successful in re-transmitting the same MAC command. Creating a valid MIC is very difficult. See Appendix B and chapter 5.4 for more details.

The fourth replay attack is performed by first modifying the FCnt, MIC and FOpts field values before re-transmitting the network frame. The FOpts field is not encrypted. This attempt will be successful in re-transmitting new MAC commands.

Beaconing
Besides network traffic between end-devices and the network server, there is also traffic between end-devices and gateways. A beacon is send from gateways to all end-devices on a regular interval for time synchronization features and gateway information. The beacon contains the time when the beacon was generated/sent. This time is used, by the end-device, to calculate the start of the receive slot windows.
Figure 5.4 shows an Attack-Defense Tree (ADT) for three possible replay attacks for the beaconing message. Before replay attacks can be performed, an eavesdropping attack is required to capture a valid and existing Beaconing message.

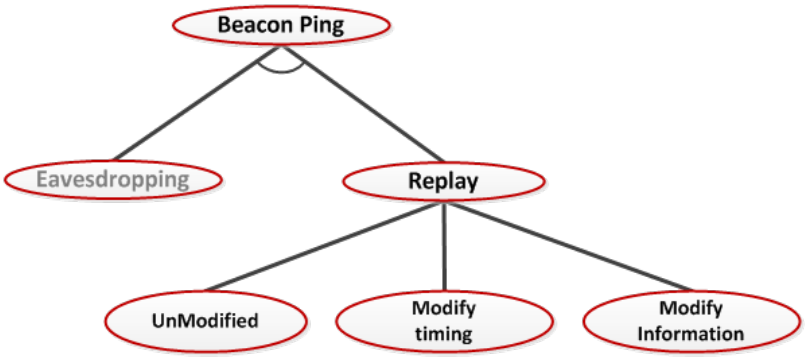

Figure 5.4: Beaconing: Replay scenarios

The first replay attack is performed by re-transmitting the Beaconing network frame without any modifications. The impact of this attack is difficult to determine. Replaying the beacon immediately will make

sure the beacon is received in the beacon receive window. However, it will contain the same value as an earlier beacon which is received the same window. This will lead to the same receive slots and same GPS data. When the beacon is replayed at a later beacon receive window, then it is unknown how the end-device will respond. If the details are processed then this will lead to receives slots in the past, which is most probably not accepted by the end-device.

The second replay attack is performed by first modifying the Time field value before re-transmitting the network frame. This attempt may have an impact on the opening of the receive slot windows by the end-device. When the receive slot window is too much out-of-sync with the real beacon time, the end-device may not be able to receive any information anymore (*beaconing precise timing*, page 54 of the LoRaWAN specification).

The third replay attack is performed by first modifying the Gateway Information field values before re-transmitting the network frame. This attempt may have several impacts. First, the end-device might think it is in reach (or out-of-reach, depending on the change made) of a specific gateway. Depending on the type or function of the end-device this may have an impact on application level. It may also have an impact on downlink routing information. The end-device might use the beacon information to update the network server with its routing details (*uplink on cell change*, page 55 of the LoRaWAN specification).

Conclusion
As with most (wireless) networks, it is possible to capture (eavesdropping) network traffic and re-transmit the captured (and modified) traffic. Although much of the transmitted information is readable, the most important information is not readable (as described in the previous section).

Just as with eavesdropping, a replay attack on the TCP/IP network connections between gateways and the network server is much more difficult. The recommendation is to have the LoRaWAN traffic tunneled over a Transport Layer Security (TLS) encrypted connection which makes eavesdropped very difficult to impossible.

To prevent re-transmissions of captured traffic, LoRaWAN has some solid defenses built-in. First, there is the frame counter that helps in dropping traffic that does not meet the expected counters. Second, there is the MIC field, this is a part of a generated hash value which is based on a combination of keys and field values. Third, application

data is encrypted, using its own key.

Looking at the results of the replay attack study, there are some findings but because most of these depend on brute-forcing key values, it will be very difficult (to impossible) to use these findings.

A similar side note must be made as in the previous chapter. The built-in security measures of LoRaWAN might not be applicable to the use of proprietary message types (MType value 111).

In the current situation, replay attacks are possible with beaconing network frames, and Join-Request & Join-Accept messages.

When additional keys can be broken (which seems to be impossible at this moment), additional replay attacks are possible:

- With NwkSKey: when producing a valid MIC it is possible to replay updated Join-Request and MAC network frames
- With NwkSKey + AppSKey: when producing a valid MIC and access to data encryption it is possible to replay almost any updated network frame, including sending new data

5.3 DENIAL OF SERVICE ATTACK

A (D)DOS attack, is an attack that causes the unavailability of a service. Often this is accomplished by exhausting resources in the environment that is providing the service. However, in this chapter it is shown that there are also other methods for performing (D)DOS attacks on a LoRaWAN. See chapter 4 for more details on the (D)DOS attack. It should be noted that in a LoRaWAN environment DOS attacks are usually not *distributed* DOS attacks, like known from TCP/IP networks. For setting up a distributed attack, control over many end-devices is required. It is nearly impossible to accomplish this within a LoRaWAN environment.

The LoRaWAN specification can be used to determine where possibilities for a (D)DOS attack can be found. It is not possible to determine all impacts since this is often depending on implementation details.

Like the attack studies in the earlier paragraphs, the communication scenarios from Appendix A can be used partially to study possible (D)DOS attacks. Possible (D)DOS scenarios have been documented in detail in Appendix C. The results are summarized in the remainder of this section.

End-device activation
The end-device activation process is based on a Join-Request network

message from end-device to Network server, and a Join-Accept network message from network server to end-device.

For both, a Join-Request, and a Join-Accept message, issues have been found that might have an impact on availability (possible DOS). The possible weaknesses found for a Join-Request, are related to the use of the DevNonce value, the use of different security contexts, and the definition of a retransmission policy for Join-Request messages. A possible weakness in the Join-Accept message is also related to the used security context. These issues have been found during the analysis of the LoRaWAN specification in this research study. During this period, others publications (based on older LoRaWAN specifications) have been found, and new publications have been published, that contain similar findings. References, and a more detailed comparison are provided at the end of this section.

Join-Request
This section will show that by using Join-Request replay attacks, the unavailability of an end-device can be accomplished. Before the attacks are described, first some terms are explained.

Security Context: Most of LoRaWAN security is based on encryption keys. These encryption keys, combined with device details, such as frame counters, appEUI and DevEUI values, form a security context. With every new Join-Request, new encryption keys are generated, resulting into a different security context.

DevNonce: A Join-Request message contains a random number, called DevNonce, which is included to prevent replay attacks. The network server keeps track of an unspecified number of DevNonce values to ignore already processed values. By not specifying the amount of DevNonce values that needs to be tracked, the decision must be made during the implementation. A list that is too short will more easily allow for replay attacks. A list that is too long will increase the chance for a duplicate valid DevNonce (since it is randomly generated). The chance for a duplicate DevNonce is low (size is 2 bytes, thus a chance of 1 in 2^{16}; which is 1 in 65,536), but since it is randomly generated it is a real possibility. The DevNonce is combined with / part of the security context that is been generated based on the Join-Request message.

The ADT in figure 5.5 shows several replay attack scenarios that have been used in this research study while trying to enforce the unavailability of an end-device.

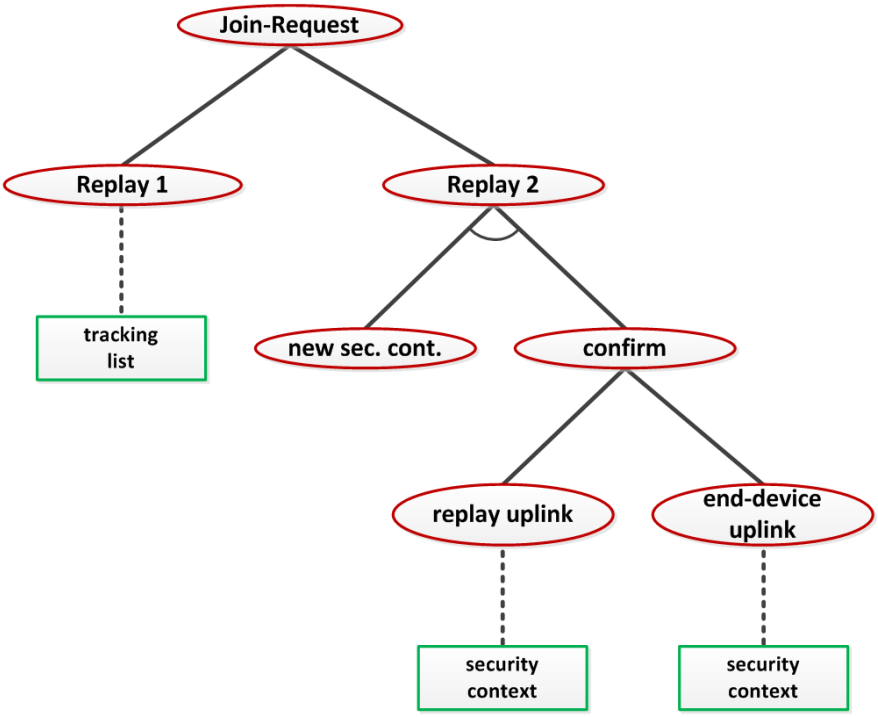

Figure 5.5: Attack Defense Tree: DOS on Join-Request

The first scenario, Replay 1, replays an earlier captured Join-Request message. This attempt will fail because the DevNonce is already on the track list on the network server. When the DevNonce is already on the track list, a Join-Request message will be ignored.

In scenario Replay 2, an earlier captured Join-Request message is replayed. However, in this case the DevNonce is not present on the track list because it is a very old Join-Request and the DevNonce has already been removed from the list. In this case the network server generates a new security context and sends a Join-Accept message to the end-device. This Join-Accept needs to be confirmed by an uplink data frame by the end-device using this new security context. Replaying an earlier captured uplink data frame (*replay uplink*; 2a) will not work because the security context in the past is different compared to the security context that should be used now. The LoRaWAN specification does not include instructions on how an unexpected Join-Accept message should be handled by an end-device (more on that in the Join-Accept section). The end-device can ignore the Join-Accept, then nothing will change and it will be able to continue its communications. When it does not ignore the unexpected Join-Accept, but processes it, it will send an uplink data frame (*end-device uplink*; 2b) to confirm the security context. However, the security context used by the end-device is different to the security context generated on the network server. From now on, communication between end-device

and network server will fail, unless a new OTAA is initiated by the end-device.

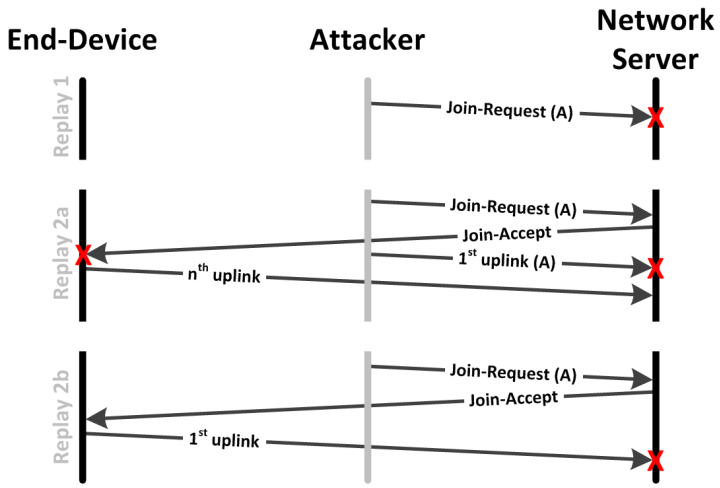

Figure 5.6: Attack Sequence: Join-Request DOS

Figure 5.6 shows the attack sequences for the previous examples. The letters between brackets indicates the session. These letters are related to the sessions shown in figure B.1.

Replay attacks on the Join-Request messages has been studied by Zulian, S. [41] as part of a master thesis focusing on the LoRaWAN join procedure and by Avoine, G., et al [23]. These studies are based on the LoRaWAN specification version 1.0. Since that time, several changes have been made to the LoRaWAN specification, for example the security context and the need to have it confirmed before the old security context is de-activated. This has addressed some of the earlier findings and has made the specification more robust. Therefore, there is no real DOS possibility anymore when replaying Join-Request messages. Only the Replay 2b example, in figure 5.6, might still be an issue since this is not addressed in the specification.

There are two more issues with Join-Requests. First, the LoRaWAN specification does not include details on how the network server should handle an unconfirmed Join-Request. It takes up a place in the tracking list. What happens when a new Join-Request comes in? Is the used DevNonce active on the track list? If multiple unconfirmed messages can be on the track list, then valid entries might be pushed off. If this is implemented in a wrong way, then it is still possible to fill-up the track list with unconfirmed messages so that valid entries (and security contexts) are pushed off, and thus not valid anymore. This would cause a failure in communication between end-device and network server until a new OTAA is initiated by the end-device.

Another issue with Join-Request messages is when there are many concurrent Join-Request messages sent to the network server. Too many Join-Request messages at the same moment can case a DOS on the network server. The LoRaWAN specification, includes details about a retransmission back-off policy. If many end-devices send a Join-Request message at the same time, "*a catastrophic, self-persisting, radio network overload situation can be triggered*", page 37 LoRaWAN specification.

Join-Accept

This section will show that by using replay attacks of the Join-Accept message, the unavailability of an end-device can be accomplished. Similar to the Join-Request section, several attack scenarios have been investigated that might have an impact on the availability of an end-device. The ADT in figure 5.7 shows several replay attack scenarios that have been used trying to enforce the unavailability of an end-device.

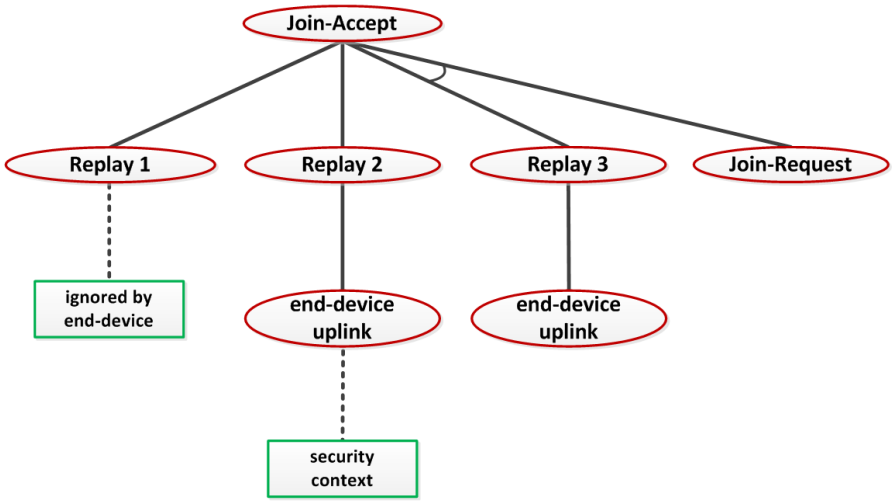

Figure 5.7: Attack Defense Tree: DOS on Join-Accept

The first scenario, Replay 1, replays an earlier captured Join-Accept message. This attempt will most probably fail, because the replayed Join-Accept message will be ignored by the end-device (this is an assumption because the behavior of an unexpected Join-Accept message is not included in the LoRaWAN specification).

In the second scenario, Replay 2, the end-device does accept the replayed Join-Accept message (the opposite of scenario 1). When the end-device processes the Join-Accept, it generates a new security context. However, this is based on its last used DevNonce. The security context on the network server has not changed. Therefore, both security contexts differ, and the end-device will not be able to communi-

cate anymore with the network server until a new OTAA is initiated by the end-device.

Scenario 3, mixes a valid Join-Request with a replayed Join-Accept message. Because the end-device initiated an OTAA and did send out a Join-Request, it is expecting a Join-Accept. When an older Join-Accept is replayed before the end-device receives the authentic Join-Accept message, the replayed version will be processed because there is no way for the end-device to find out it is processing a wrong Join-Accept. When the replayed Join-Accept is processed, the security contexts on both sides will differ, and the end-device will not be able to communicate anymore with the network server until a new OTAA is initiated by the end-device.

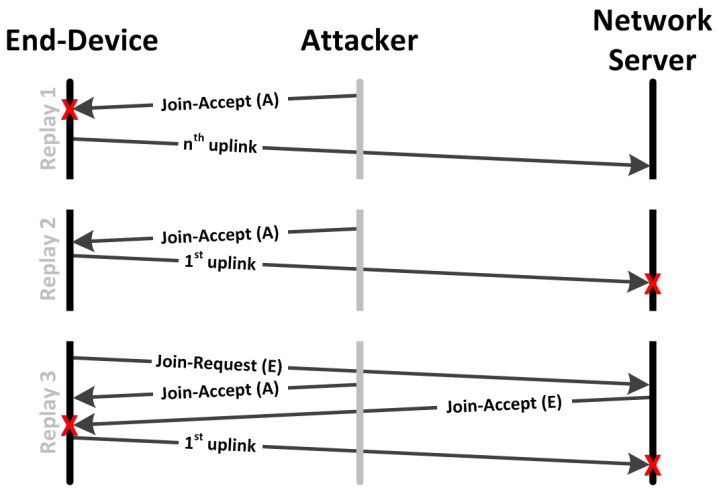

Figure 5.8: Attack Sequence: Join-Accept DOS

Figure 5.8 shows the attack sequences for the previous examples. The letters between brackets indicates the session. These letters are related to the sessions shown in figure B.1.

The study by Zulian, S. [41], also included a similar finding in respect to the mixture of a Join-Accept replay attack with an authentic OTAA. Despite the described changes to the LoRaWAN specification (related to security contexts) this finding is still relevant. Another study, by Avoine, G., et al [23], also included an investigation to mixing up a replayed Join-Accept message into a valid OTAA conversation. This study is also based on the LoRaWAN specification 1.0. Like the findings of Zulian, S., this vulnerability is still present in the current LoRaWAN specification.

Sending MAC commands

One obvious method of disturbing the connectivity between an end-device and the network is to modify reception parameters, like fre-

quencies, bandwidths, slot windows, etc. All these types of parameters can be changed using MAC commands. These commands are communicated by using regular data frames using a dedicated field, or an optional field value. Both have their own specific use, but common is that both are protected by the MIC value. If an attacker wants to use MAC commands to impact availability, the (re-)calculation of the MIC should be addressed first. This is very difficult and more details are available in chapter 5.4.

Beaconing

Besides network traffic between end-devices and the network server, there is also traffic between end-devices and gateways. A beacon is send from gateways to all end-devices on a regular interval for time synchronization features and gateway information. The beacon contains the time when the beacon was generated/sent. This time is used, by the end-device, to calculate the start of the receive slot windows. The security overview of R. Miller [26] includes a remark about manipulating beaconing. However, exact details are not provided.

Manipulating receive slots

Class B end-devices use receive slots in which the end-device is listening for data send by the network server. Such receive slots are based on time values send by gateways in beacons. Figure 5.9 shows a repeating beacon. A single beacon is represented by a beacon period (B_p). A beacon period is split into multiple sections. First, a beacon receive window (B_r), in which the end-device listens for a beacon. Second, a beacon window (B_w), which is a period that is used for scheduling the actual receive slots. And third, a beacon guard (B_g), which is a grace period in which transmissions can be completed before the end-device starts listening again for a beacon in the next B_r. The red and blue colors distinct two different beacon situations. The first blue and red arrow represent a received beacon time (B_t), the remaining arrows (S_x) represent calculated receive slots.

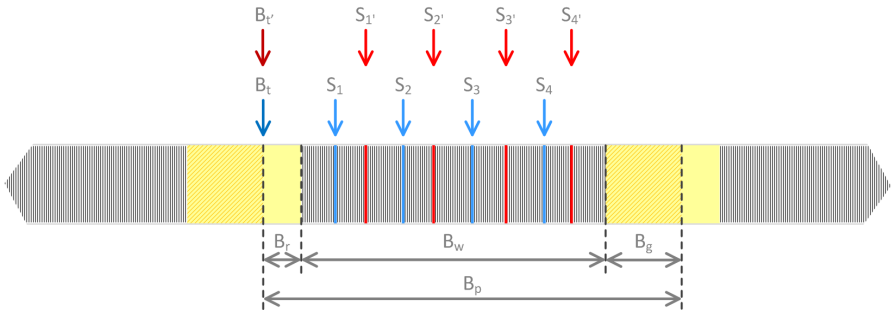

Figure 5.9: Beaconing: manipulating slot calculation

The blue arrows represent the beacon time and calculated receive slots based on an authentic beacon. The exact details of slot calcula-

tions are included in Appendix B.

When an attacker wants to manipulate the calculated receive slots, it should be sufficient to transmit a modified beacon. When this modified beacon is transmitted with a better network signal than the original beacon (for example by a stronger transmitter, or by transmitting it closer to the end-device), then the beacon will be processed by the end-device. When the manipulated time value slightly differs from the authentic beacon, then the calculated receive slots will differ from the originally calculated receive slot. This is represented by the red arrows in figure 5.9. When the end-device uses different receive slots than the network server, it will not be possible to have a successful downlink communication.

Manipulating beacon location information
A beacon also contains the location (GPS) information of the gateway that transmitted the beacon. This information can be used by a Class B, or Class C, end device to determine if it has moved (change from nearest gateway), and that it needs to inform the network server that it now should be reached using a different network path (gateway). When the GPS location is manipulated, an end-device might think it has moved every time it receives a new beacon. Depending on the implementation, this can have an impact on the battery because the end-device must be much more active than intended. This manipulation does not have an impact on the downlink routing itself, only on the frequency of notifications from end-device towards network server to update downlink routing information.

The below sections are not directly related to one of the earlier described communication scenarios. These weakness can be applied to different types of communication scenarios.

Downlink Routing
For downlink traffic, the network server needs to know which gateway to use to address a specific end-device. Based on the last uplink communication of an end-device, the network server updates its downlink routing information. When a way can be found that the uplink traffic of an end-device is re-routed via a different gateway than the network server downlink routing information is updated with the wrong information and downlink traffic might fail for a specific end-device.

There are several methods which can be used for manipulating the downlink traffic. The first method is by using a regular replay attack when a captured network frame is modified and replayed at a later moment in time. However, the presence of a MIC field makes this difficult because the MIC calculation must be broken. Otherwise

the replayed network frame will be rejected because of wrong frame counters. Another method is by using a wormhole attack. In such an attack, a network frame is captured and replayed in such a way that the replayed version reaches the destination before the original transmission does.

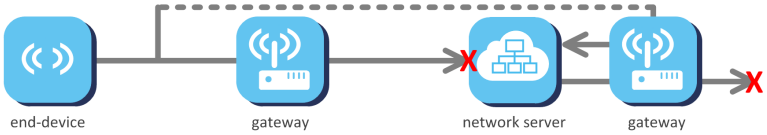

Figure 5.10: Manipulate downlink routing

Figure 5.10 shows how an uplink network frame is captured and replayed via a gateway that is closer to the network server. Key in this attack is that the transmission over the dotted line is much faster than the original transmission. Now that the replayed version reached the network server first, the network server will process the network frame as a legitimate network frame. When at a later moment the authentic network frame reaches the network server, it will be ignored because it already processed a network frame with those frame counters. Since the last successful uplink network frame came via a different gateway, the downlink traffic will now go via that gateway as well. However, that gateway is not able to reach the end-device and therefore downlink traffic will not be successful until the next successful uplink network frame via the correct gateway.

Frame Counters

When an attacker can generate valid network frames for an end-device (see chapter 5.4 for a MIC attack) it might continue uplink traffic when a correct Frame Counter is used. This way the traffic counters at the network server side will increase, while at the original end-device they will not. This method will cause a de-synchronization between the frame counters at the end-device side and the frame counters at the network server side. When this de-synchronization gap becomes too large then the original end-device will not be able to communicate with the network server until a new OTAA is initiated by the end-device.

Resource Exhaustion

The most obvious method to cause a resource exhaustion is to send an excessive amount of network frames. Processing a high amount of network frames might cause an overload of resources, such as the NIC, Central Processing Unit (CPU) or memory. Other methods rely on sending less network frames, but the network frames are specially crafted to possibly cause a CPU or memory overload during processing of the content of the network frame.

Based on the LoRaWAN specification it is difficult to determine if there are resource exhaustion possibilities for LoRaWAN. Purely seen from a specification perspective, and not from an implementation perspective, there are only a few resource limitations. Limitations mentioned in the LoRaWAN specification include duty cycle and dwell time to avoid network congestion. However, these limits are based on local regulations and can be ignored by persons with malicious intend (or even by accident) to negatively impact performance of LoRaWAN implementations.

Collisions
Because wireless networks use a shared media (the air), there is always the possibility for collisions. A collision is when two parties start sending at exactly the same moment. The result is that both transmissions are mixed-up (and thus results into an invalid frame), or that the strongest signal pushes away the weaker signal. Since LoRaWAN does not know a re-transmission mechanism, the network frame will be lost in case of a collision. A form of DOS attack is when a hacker can cause collisions on purpose. Using such an attack, an attacker would be able to corrupt the transmissions of a legitimate end-device (or gateway). Using this attack is difficult, it requires correct timing and signal strengths.

Conclusion
Looking at the results of the DOS attack study on the LoRaWAN specification, there are several findings. Some of them seem to be applicate without difficulties, some of them depend on brute-forcing key values. It will be very difficult (to impossible) to use the findings that depend on a modified MIC. Also, some of the findings that have been raised are based on assumptions because the situations are not included in the LoRaWAN specification.

In the current situation, DOS attacks are possible based on: mixing replayed Join-Accept within an authentic OTAA procedure, too many concurrent Join-Request messages, manipulation of receive slot calculations, manipulation of gateway GPS details, impact downlink-routing, collisions, and network congestion.

When additional keys can be broken (which seems to be impossible at this moment), additional attacks are possible which includes sending of new MAC commands and the modification of frame counters to manipulate communication parameters.

The LoRaWAN specification is not clear in respect to the behavior of end-devices on unexpected Join-Accept messages, and the manipulation of the track list (depending on list length and network server

behavior of unconfirmed Join-Accept messages).

It can be concluded that based on the studied details, there are several realistic options for executing different types of DOS attacks on a LoRaWAN environment.

5.4 CRYPTOGRAPHIC ATTACK

In a LoRaWAN environment, cryptography is used on different moments, and for different purposes. The most important purposes are confidentiality and integrity. Confidentiality is achieved by using encryption of information so that the information is only interpretable by sender and receiver. Integrity is making sure that data has not been changed in-between sender and receiver.

Encryption algorithm
The encryption algorithm in LoRaWAN for both confidentiality and integrity is based on Advanced Encryption Standard (AES) 128. According to the Advanced Encryption Standard [29], the AES algorithm is a symmetric block cipher that can encrypt and decrypt information. The AES algorithm is an open algorithm which means the algorithm itself is publicly available. The secrecy is achieved by the used keys only. Based on the history and its openness, the AES algorithm has been studied numerous times and a formal NIST validation program[1] exists. Because of these arguments, studying the AES algorithm has not been part of this research study. A lot of time, resources and mathematical knowledge is required to be able to make a difference in a (re-)validation of AES.

Brute-force
A very well-known method in breaking encryption is the use of brute-force to find the required cryptographic key(s). A brute-force attack is an attack in which all possible key combinations are tested to successfully perform a cryptographic action. A simple example is when all number combinations (0,000-9,999) for a four-digit pin code are tested. Eventually the right combination will be found. In reality, cryptographic keys are much longer, which increases the number of combinations. The AES algorithm used in the LoRaWAN specification is using cryptographic keys with a length of 128-bits.

Cipher-based Message Authentication Code (CMAC)
In LoRaWAN, the CMAC is a 128-bits hash that based on a message and a 128-bits key: *CMAC = aes128_cmac(key, message)*

1 https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program

A CMAC brute force attack can be executed by using a *known plaintext* attack. The known plaintext attack is when an attacker knows both the plaintext as well as the cipher text. The CMAC hash has a length of 128-bits. Each bit has two possible values (0 or 1). The total number of possible combinations is 2^{128}, which makes a total of 3.402e+38 combinations which must be tested.

Let's assume a computer that can perform 100,000 key verifications per second. Finding the correct key can take up to 1.078e+26 years. This is calculated by dividing the number of possibilities by the number of keys per second, which results into the number of seconds required to test all combinations. The number of seconds can be converted[2] into years. On average, it will be half of that time (quickest result is the first attempt; slowest result is the last attempt). On average, it will still take 5.391e+25 years. Even if a very large amount of computation power is available it will take many years. For example, distributed.net (computing power differs in time, however at time of checking it was able to verify 771,160,000,000 keys/sec) will still take 6,991,492,428,376,118,500 years on average.

These calculations show that a brute-forced *known plaintext* attack on the CMAC is impossible with today's technologies. What makes things worse, is that an attacker does not have the full cipher text as used in the above example. An attacker is only able to get hold of a partial cipher text, the MIC, which represents the first 32-bits of a CMAC. A brute force attack on a plaintext-MIC combination results in all keys that result into CMAC values with the first four bytes identical to the MIC.

A CMAC is 128-bits (2^{128} combinations), and a MIC is 32-bits (2^{32} combinations). Eliminating every leftmost bit, halves the number of total combinations. Eliminating the left 32-bits of a CMAC results into (1 / 2^{32}) * 2^{128} = 2^{96} combinations.

If every unique CMAC (hash) comes from a unique key (based on the same message), 2^{96} possible keys are found when using a MIC as cipher text. Only by repeating the above process (using different plaintext-MIC combinations) a single key will eventually be found that turns out to be successful for all plaintext-MIC combinations.

A single brute-force attack (with full plaintext and cipher text) is not realistic. Running such attempts multiple times, because of a partial cipher text, is impossible with today's technologies.

2 http://www.convertalot.com/time_converter_calculator.html

Message Integrity Code (MIC)
As seen in the previous section, brute-forcing a 128-bits key turned out to be impossible. What if we can narrow down the brute-forcing to the 32-bits MIC? Unfortunately, a MIC is not a direct result of some formula, it is only a part of a CMAC hash result. This means there is no calculation that can be used that directly results into a possible MIC. The only way of brute-forcing a MIC is by sending modified network frames to a destination. If the destination successfully processed the network frame, then the MIC was a correct one. There are several issues with this. First, a MIC can have 2^{32} possible values, and thus the same number of network frames is required for brute-forcing. Second, only a modified MIC is not sufficient. The frame counter must also be carefully determined so that the destination does not reject the network frame because of an invalid frame counter. Third, it is difficult to detect if a destination has successfully processed a network frame.

Based on the 2^{32} possibilities, a successful brute-force result can be expected after 2,147,483,648 attempts on average. Transmitting two billion network frames (within a reasonable timeframe) to a destination on a LoRaWAN can be considered impossible. Even if this could be successful, for example if a match is found after only a few tries, then the resulting MIC value is only valid for the related network frame. Other network frames require a different MIC value. Using multiple of these brute-force attempts to combine them into a single approach (formula) that would result into a MIC that is valid for multiple frames will not be possible. The main reason is that the MIC is not a result of itself, but only a part of a hash result (CMAC).

Payload encryption
AES cryptography is also used for encrypting the payload in a LoRaWAN network frame. Since an attacker does not have the plaintext details, only a *ciphertext only* attack can be used. A ciphertext only attack is very difficult because the attacker has no idea about the plaintext and thus what to expect. Like a CMAC, the payload encryption is based on 128-bits keys. 1.701e+38 decryption attempts must be performed (on average) to decrypt the ciphertext. Even if one of the attempts results into a successful plaintext, how can this be verified? If the plaintext contains English text then it is relatively easy to detect if decryption went successful. However, usually a payload in a Wireless Sensor Network will contain values. Decryption using a wrong key and a correct key will both result into values. It is nearly impossible to determine which value is the correct plaintext value. Next to time constraints which make brute-forcing impossible, also the lack of plaintext verification makes this attack unrealistic.

Key relations

An important factor in cryptography is the strength of keys. Key lengths play an important role. Previous sections have shown how key lengths play a role in, for example, brute force attacks. The length of a key has a direct relation with the difficulty (duration) of a brute-force attempt. Another aspect in key strength is the relation of keys with other aspects. Examples are keys that are derived from a hardware number, based on date/time, dependency on other keys, etc. In LoRaWAN, session keys (AppsKey and NwkSKey) are derived from a root key (AppKey) combined with other data.

AppKey

The AppKey is the root key specific to an end-device. Before an end-device can be used on a LoRaWAN implementation, the root key is communicated out-of-band so that it is never transmitted over the LoRaWAN. As soon as brute-forcing keys becomes possible, then this AppKey can be brute-forced because it is used for the MIC calculation in a Join-Request.

NwkSKey

The NwkSKey is used to calculate the MIC of regular network frames and encryption of MAC command payloads. As soon as brute-forcing keys becomes possible, then this NwkSKey can be brute-forced because it is used for the MIC calculation in regular network frames.

```
NwkSKey = aes128_encrypt(
  AppKey,
  0x01 | AppNonce | NetID | DevNonce | pad(16)
)
```

When the AppKey root key is compromised, it is still not possible to derive the NwkSKey. Most additional details can be found during eavesdropping, however, the AppNonce value is only transmitted (in encrypted form) in a Join-Accept message.

AppSKey

The AppSKey is used to encrypt application payloads. As soon as brute-forcing keys becomes possible it will still be difficult to derive the AppSKey because of the *ciphertext only* attack, which has been explained in the previous section.

```
AppSKey = aes128_encrypt(
  AppKey,
  0x02 | AppNonce | NetID | DevNonce | pad(16)
)
```

Like the NwkSKey, when the AppKey root key is compromised, it still not possible to derive the AppSKey because of not knowing the

AppNonce value.

Bit flipping
A bit flipping attack is when an attacker is making changes to ciphertext which results into a predictable change of the plaintext. With certain encryption algorithms (like AES in Block Cipher Mode as used in LoRaWAN) it is possible to make changes in a cipher text to modify the resulting plaintext after decryption.

As an example, we take the details of a financial transaction. Bank account 31735746 transfers 10 euros to bank account 35834627. This transaction could be stored as the following data record:

31735746 ; 35834627 ; 10

When transferring this transaction over a network, this record is encrypted. The ciphertext containing this record might look like:

CZ4M8fxH79xIIIbznxmiOxGQ7td8LwTzHFgwBmbqWuB+sQ==

Using a bit flipping attack, an attacker will make a change in the ciphertext which will cause a change in the plaintext after decryption. For example, the original ciphertext is changed into:

*CZ4M8fxH79xIIIbznxmiOxGQ7td8Lw**V**zHFgwBmbqWuB+sQ==*

After decryption, the record might look like:

*31735746 ; 35834627 ; **100***

This example shows how plaintext can be changed by modifying ciphertext, without breaking any encryption. For applying a bit-flipping attack, an attacker does not have to encrypt/decrypt anything. The used cryptographic keys are not required. However, knowledge of the used algorithm is important.

In LoRaWAN a bit-flipping attack should be applied to the FRMPayload field. This is the field that contains encrypted application data. However, because of the MIC it is not possible to change the content of a network frame without breaking integrity. Performing a bit-flipping attack on a LoRaWAN network frame is impossible without generating a valid MIC (which is impossible with today's technologies, see chapter 5.4).

Conclusion
No findings have been found in this cryptographic attack study on the LoRaWAN specification. Brute-forcing cryptographic keys is a theoretic possibility, however, with today's technologies it is not possible

to practically use such attacks. Besides brute-force attacks, also key relations, and bit-flipping attacks have been studied. These also have not lead to any findings. The details are included in this section, because once any brute-force attack (or other method breaking cryptography) is possible, many other attacks can be applied successfully. It is not expected that AES encryption will be broken in the near future. However, flaws are sometimes enough the speed up attacks that have been classified as impossible before.

5.5 MAN IN THE MIDDLE

A MitM attack is an attack in which an attacker places himself in-between two communicating parties. Both parties will not know about the presence of the attacker and both parties think they are still communicating with each other. This way an attacker can get control over the conversation. See chapter 4 for more details on the MitM attack.

In a LoRaWAN environment it is much more difficult to be 'in-the-middle'. Compared to TCP/IP networks, there is no routing and all traffic can be compared with broadcast traffic. Since it is wireless all transmissions are send into the air. Transmissions flow from end-device to one, or more, gateways, to the network servers. And the other way around, from network server, to gateway, to end-device. There is no 'in-the-middle' between end-device and gateway because an end-device transmits into the air, and all gateways in reach will pick up the transmission and forwards it to the network server.

The easiest option to be 'in-the-middle' is to place a malicious gateway. However, that malicious gateway must be registered, and is only 'in-the-middle' if there are no other gateways in reach. When placing such a gateway there is still little control because security for network functionality has been separated from security for application functionality (NwkSKey versus AppSKey). Open community LoRaWAN implementations, such as The Things Network, even encourage people to add gateways[3] to increase geographical coverage. On a malicious gateway, network frames can be intercepted and forwarded to the network server (standard gateway behavior). However, influencing the conversation will be difficult because of the confidentiality (encryption) and integrity (MIC) precautions in the LoRaWAN specification.

Conclusion
In a LoRaWAN environment, a real MitM attack, as known with other network protocols (such TCP/IP), is not possible. The MitM aspects that are possible, are described in the eavesdropping, replay and DOS attack sections.

3 https://www.thethingsnetwork.org/docs/gateways/registration.html

5.6 CONCLUSIONS

After studying eavesdropping, replay, Denial of Service (DOS), cryptographic, and Man-in-the-Middle attacks, it can be concluded that LoRaWAN has several solid security mechanisms build in. Especially confidentiality and integrity are handled very well by the implemented measures that are using AES 128. However, the attack analysis did result into several DOS related findings.

The most important findings from this attack analysis are:

- End-device activation: replay attack of a Join-Accept message, during an authentic OTAA, can lead to a DOS of the end-device.
- Downlink routing: by replaying network frames via a different gateway, a DOS can be caused for Class-B downlink traffic towards the end-device.
- Beaconing: by modifying and replaying gateway beacons, a DOS can be caused for Class-B downlink traffic towards the end-device.

After discovering the finding related to end-device activation, I was also able to find literature that include a similar finding. Both Zulian, S. [41] and Avoine, G., et al [23] have mentioned this probable replay attack in respect of the LoRaWAN v1.0 specification. I have studied this possibility into more detail (in respect to the LoRaWAN v1.0.2 specification) and provided additional details. The finding related to beaconing is also mentioned by R. Miller [26] without providing additional details. For the downlink routing finding no references in other literature was found.

Besides the above listed findings, several findings have been found because of insufficient detail in the LoRaWAN specification. Examples of this are the tracking of security contexts, and how an end-device should handle Join-Accept messages. Since these have not been described in detail, issues could be present, depending on implementation decisions.

6 ATTACK MODELING AND VALIDATION

This chapter provides an overview of the results of the attack modeling and analysis research phase. Studying the impact of Eavesdropping, Replay, DOS, and Cryptographic attacks on the LoRaWAN specification in chapter 5, resulted in several findings that are studied in more detail in this chapter.

The findings that are analyzed in this chapter are: an eavesdropping + replay attack on beaconing network frames which are broadcasted by gateways (Beaconing) to desynchronize downlink ping slots, an eavesdropping + replay attack on end-device uplink network frames to manipulate the network server downlink routing table (Downlink routing), and an eavesdropping + replay attack on the Join-Accept message during an authentic OTAA (End-device activation).

Attack modeling and analysis are performed by constructing a Colored Petri Net (CPN) model in CPNTools for each finding. CPNs are used more often for protocol analysis [31] [39] [30] [16] [7]. CPNTools is one of the most versatile tools for CPNs [34], despite having a steep learning curve. The required models are constructed based on relevant parts of the LoRaWAN specification, a description of the attack steps involved, and defined assumptions and decisions. State Space analysis is used to verify aspects of the CPN model during construction, and simulation is used to validate findings. More details on CPNs and CPNTools can be found in Appendix D. Additional details on these modeling and validation activities are included in chapter E.

Section 6.1 in this chapter describes the modeling and validation results for the beaconing finding. In section 6.2, the results for the downlink routing finding are described. Section 6.3, provides details on the end-device activation finding.

6.1 BEACONING

This section describes the CPN model of the finding related to beaconing in the LoRaWAN specification. This finding shows that it is possible to manipulate Class B ping slot calculations on the end-device by performing a replay attack on the beaconing network frame. Another possible is to modify the location details of the gateway. See also chapter 5.3, Appendix A.5, Appendix B.5, and Appendix C.3 for more details on beaconing. Appendix E.1 contains additional details which

are used for modeling and validation of this finding. This includes Specifications (*BSXX*), Decisions (*BDXX*), and Assumptions (*BAXX*) that are used for decisions in the constructed model and source-code.

Activity steps
Before a model can be constructed, it should be clear what functionality to model, and which steps should be involved. For this particular model, the following steps are defined:

- Gateways create a beaconing frame
- End-device opens beaconing frame receive window
- Gateways broadcast beaconing frame
- Attacker modifies and replays an eavesdropped beacon:
  - just before an original beacon
  - to suppress an original beacon (>transmission output)
  - just after an original beacon
- End-device receives beaconing frame
- End-device processes beaconing frame to calculate Class B receive slots

6.1.1 *Model construction*

In CPNTools, a model is constructed that can be used to study this finding in more detail. Figure 6.1 represents the constructed model on the highest level.

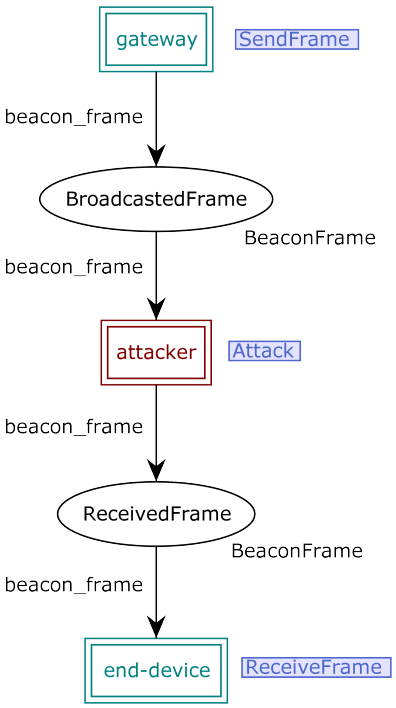

Figure 6.1: Beaconing CPN: top-level

A top-level model is created which includes three of the major components (*BA01*). For the *network server*, a separate model is constructed and described in the remainder of this section. A *gateway* sub-model is constructed that generates new beaconing frames based on the LoRaWAN specification (interval, frame fields, field contents, and timings). An *end-device* sub-model is constructed that calculates receive slots based on the received beaconing frames. An *attacker* sub-model is constructed, that is in-between the gateway and the end-device, and manipulates beaconing frames by replacing authentic beaconing frames with malicious beaconing frames, and by inserting malicious beaconing frames.

Gateway

The gateway section of this CPN model is generating, and broadcasting, beaconing network frames. Figure 6.2 represent the gateway sub-model. These beaconing frames are generated with a specific interval (*BS01*), and contain a time-stamp and gateway location information (*BS02*). The model is designed to transmit a maximum of ten beaconing frames. This is to limit the duration of the simulation.

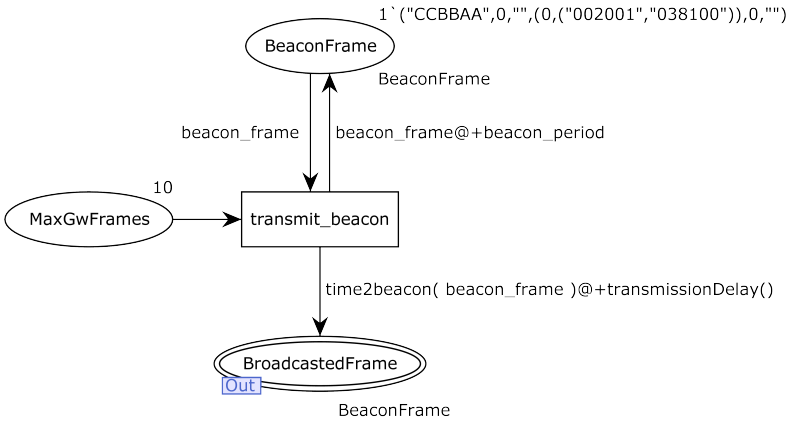

Figure 6.2: Beaconing CPN: gateway

The *transmit_beacon* transition, is consuming a token from *BeaconFrame* and *MaxGWFrames*. *BeaconFrame* contains a token that represents a beaconing frame. Transition *transmit_beacon* only updates the time field value, and a time unit delay is added to the produced token to represent a transmission delay. *MaxGWFrames* serves as a counter which allows for a maximum of ten beaconing frames to be transmitted for this simulation. The token containing the transmitted beaconing frame is produced in the *BroadcastedFrame* output place.

Attacker

An attacker has two methods for manipulating end-devices in respect to beaconing frames. First, an attacker can suppress existing beaconing frames by transmitting malicious beaconing frames with a higher

transmitting power. Second, an attacker can insert malicious beaconing frames before, and after, authentic beaconing frames.

The constructed CPN sub-model (see figure 6.3) replaces regular beaconing frames with malicious beaconing frames at random (*BD04*). Malicious beaconing frames are inserted (before or after regular beaconing frames) from a pre-defined list (*BoD5*) of malicious beaconing frames.

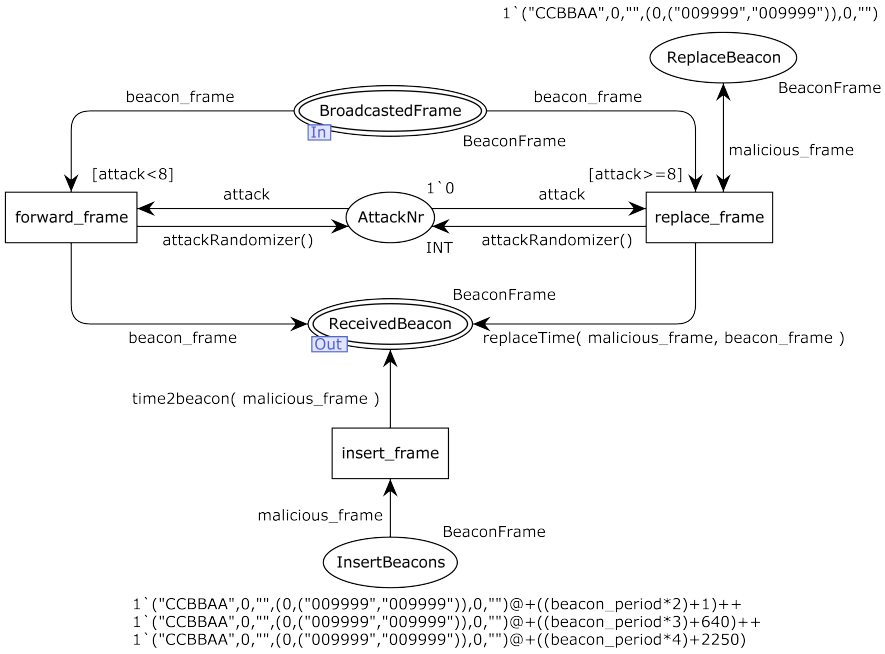

Figure 6.3: Beaconing CPN: attacker

The *BroadcastedFrame* output place from the gateway sub-model is the input place for the attacker sub-model. This sub-model contains a place *AttackNr*, which contains a random number. Based on this random number, the beaconing frame is just forwarded by the *forward_frame* transition, or a beaconing frame is replaced by the *replace_frame* transition. The *forward_frame* transition consumes a token from *BroadcastedFrame* and produces an identical token in the *ReceivedBeacon* place. The *replace_frame* transition consumes a token from *BroadcastedFrame* and *ReplaceBeacon*, and produces a token containing the malicious beaconing frame in the *ReceivedBeacon* place. Both transitions *forward_frame* and *replace_frame* produce a new random number in *AttackNr*. For inserting malicious beaconing frames, the *insert_frame* transition consumes tokens with malicious beaconing frames from *InsertBeacons* and produces a token with these malicious beaconing frames in the *ReceivedBeacon* place.

End-Device

When an end-device processes a beaconing frame, then the beacon

Time field value is used to calculate the required downlink receive slots.

The constructed sub-model (see figure 6.4) first checks if a received beaconing frame is received inside the beaconing frame receive window. If it is not, then the frame is rejected. If the frame is received inside the beaconing frame receive window, then it is checked if this is the first beaconing frame in the beaconing receive window. If it is not the first, then the frame is rejected. If it is the first beaconing frame in this beaconing receive window, then the frame is used to calculate the downlink receive slots.

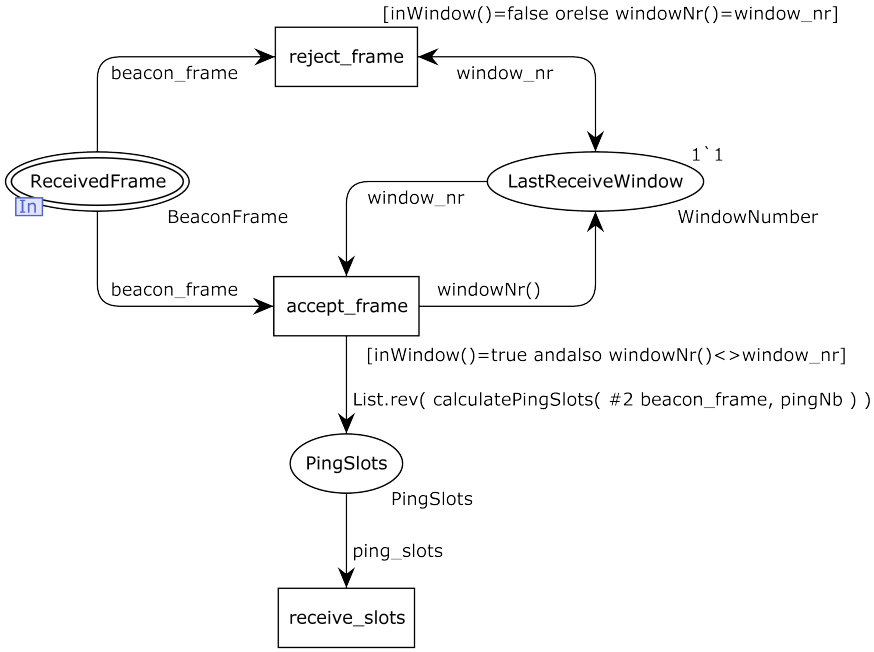

Figure 6.4: Beaconing CPN: end-device

The *ReceivedFrame* output place from the attacker sub-model is the input place for the end-device sub-model. If the token's time unit is outside the beaconing receive window, or a beaconing frame is already processed in the beaconing receive window, then the *reject_frame* transition consumes the token and it is not processed any further. If the token's time unit is inside the beaconing receive window, and no other token is already processed in the same window, then the *accept_frame* transition consumes the token containing the beaconing frame and produces a new token containing a list of calculated ping slots in the *PingSlots* place. The transition *receive_slots* consumes the tokens containing the calculated downlink receive slots.

Network Server

For analysis purposes, an additional CPN model is created for the receive slot calculations at the network server side. By including this

CPN model (see figure 6.5), the slot calculation results of the end-device, and the slot calculation results of the network server, can be compared. Because of malicious beaconing frames on the end-device, the slot calculations on the end-device are not always identical to the slot calculations on the network server. This results in an out-of-sync of the ping slots between end-device and network server, and therefore Class B downlink communication from network server to end-device cannot succeed.

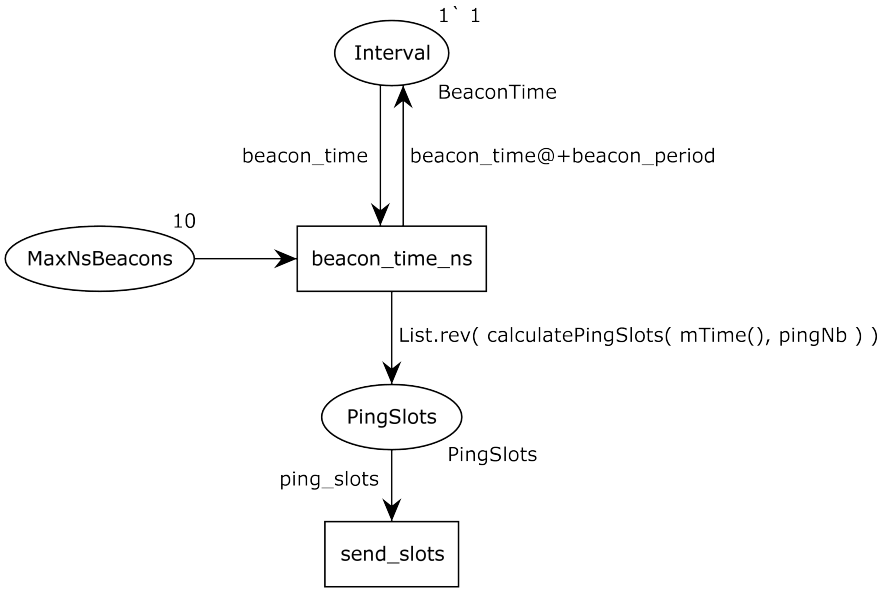

Figure 6.5: Beaconing CPN: network server

The transition *beacon_time_ns* consumes tokens from *Interval*, which is used to determine the interval in-between tokens, and *MaxNsBeacons*, which determines the maximum amount of beacons to process for this simulation. Similar to the end-device, the network server produces a new token containing a list of calculated ping slots in the *PingSlots* place. The transition *send_slots* consumes the tokens containing the calculated ping slots.

6.1.2 *State space analysis*

During construction of the model, state space analysis is used to verify properties of the CPN model. State space analysis is included in CPNTools. The properties that are checked during state space analysis (see chapter D.3.2) can support the correctness of the model. Analyzing the full state space for the Beaconing model has only been possible during the first stages of construction. By extending the model, the state space has grown too large to be analyzed within several days. This behavior is also known as the state space explosion [37]. The partial state space report for the final CPN model has run for over

twelve hours. The report generation took around one hour. Other attempts have been made for a full state space analysis, but the longest run has been canceled after a full week of running.

The properties for this Beaconing model are listed in table E.6. It should be noted that state space analysis only contributes to checking the validity of the CPN model itself. It does not contribute in checking the functional correctness of the model. This research study uses simulation for checking the functional correctness of the model.

Properties in the state space analysis report are based on a partial state space calculation. However, several of the state space properties do support the correctness of the CPN model. The found *boundedness* in the report is in line with expected results. Also, the conclusion in the report regarding a Home Marking is in line with expected results because the model is finite. The report also states that no Dead Transitions are found, and this is correct. In simulation it is shown that all transitions fire at least once. The report also contained many Dead Markings. This was initially not expected. However, this was explained when realizing that the state space is much larger (state space explosion) than the markings shown in simulation. The partial state space analysis did not uncover any incorrectness in the final model.

6.1.3 *Simulation*

Simulation can be used to validate results. In the graphical user interface of CPNtools, a simulation shows how transitions are fired and how tokens are consumed and produced in the CPN model. Because of the defined monitors, all results are written to log files. Besides a visual analysis of the simulation, the log files are used for a more detailed analysis of the simulation results.

The next steps provide a simplified summary of the simulation details and analysis, which will result into a manipulated Class B receive slot calculation on the end-device. Figure 6.6 shows the attack sequence required for this attack. Authentic beaconing frames are transmitted by the gateway. Before an authentic beaconing frame, an attacker transmits a malicious frame (in red) to manipulate the Class B receive slot calculation.

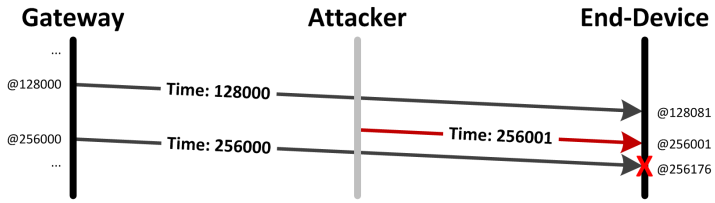

Figure 6.6: Beaconing: simulation attack-sequence

The *gateway* sub-model is constructed to generate beaconing frames based on the LoRaWAN specification. Important fields in a beaconing frame (for this finding) are the beacon time, and the time-unit of the beaconing frame which indicates the order of processing.

Below, two beaconing frames are shown that are used throughout this summary. These beacons are send out by the *gateway* sub-model. The first line represents a beaconing frame that is send out on time-unit 128,081 with a beacon time of 128,000. The second line represents a beaconing frame that is send out on time-unit 256,176 with a beacon time of 256,000.

```
(NetID,Time,CRC,(Antenna,(Lat,Long)),RFU,CRC) @ time-unit
("CCBBAA", 128000,"",(0,("002001","038100")),0,"") @ 128081
("CCBBAA", 256000,"",(0,("002001","038100")),0,"") @ 256176
```

In-between gateway and end-device, an attacker is modeled. This attacker can replace authentic beaconing frames with malicious beaconing frames, or can send out new malicious beaconing frames. In this simulation, the attacker replays a modified beaconing frame just before an authentic one would be send out.

```
("CCBBAA",256001,"",(0,("009999","009999")),0,"") @ 256001
```

The three beaconing frames shown below, represent how they are received by the end-device. The first line is an authentic beaconing frame transmitted by the gateway. The second line shows a malicious beaconing frame that is inserted just before an authentic one. The attacker transmitted a beaconing frame, with a manipulated beacon time and gateway location details (marked in bold), It is send out at time-unit 256,001 so that it will be received just before the authentic beaconing frame that is send at time unit 256,176. The third line shows the authentic beaconing frame that is rejected because the malicious one is received (and processed) first.

```
("CCBBAA", 128000,"",(0,("002001","038100")),0,"") @ 128081
("CCBBAA", 256001,"",(0,("009999","009999")),0,"") @ 256001
("CCBBAA", 256000,"",(0,("002001","038100")),0,"") @ 256176
```

To be able to communicate, both network server and end-device have to be in sync in respect to the ping slots. On both sides, the ping slots are calculated. The outcome should be identical in order to operate with the same ping slots. The two lines below show the calculations performed by the network server. For each beacon time, four ping slots are calculated.

```
Time-unit [  Slot1,  Slot2,  Slot3,  Slot4 ]
128000 >> [ 130379, 161099, 191819, 222539 ]
256000 >> [ 258380, 289100, 319820, 350540 ]
```

On the end-device, the ping slot calculations are performed as well. However, for the second beaconing frame, the beacon time was slightly adjusted by the attacker. The means that a different beaconing time is used in the ping slot calculations, which results in four different ping slots. The result is that in the second beacon period, the ping slots between network server and end-device are out-of-sync. At this moment, communication will fail when the network server initiates a downlink transmission to the end-device within one of those four ping slots. From the moment the end-device receives an authentic beaconing frame, the network server and end-device are back in sync.

```
128000 >> [ 130379, 161099, 191819, 222539 ]
256001 >> [ 258637, 289357, 320077, 350797 ]
```

Figure 6.7 shows the results for the second beacon period. Instead of used correct ping slots on a correct beacon time (blue arrows), the end-devices has calculated wrong ping slots based on a wrong beacon time (red arrows).

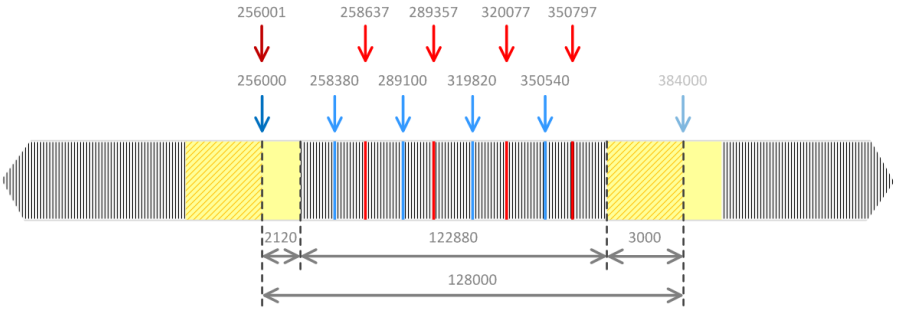

Figure 6.7: Beaconing: manipulated slot calculation

Besides the Time field value in beaconing frames, also the location details of the gateway can be replaced with invalid values. The impact of this is depending on the actual implementation of the end-device. It could lead to battery drain (because of a constant update of the downlink routing table), or it could lead to an unstable application. Analyzing an exact impact is implementation related, and therefore out-of-scope for this research study.

6.1.4 *LoRaWAN v1.1 specification*

On October 17th 2017, a new version (v1.1) [22] of the LoRaWAN specification was published. This specification contains several changes related to beaconing. None of these changes remediates this finding. The related changes are listed in table E.7.

6.2 DOWNLINK ROUTING

This section describes the CPN model of the finding related to downlink routing in the LoRaWAN specification. This finding showed how it is possible to manipulate the downlink routing table, that is maintained by the network server, by using eavesdropping and replay attacks. This attack is resulting into a DOS attack on all downlink traffic from network server to end-device. See also Appendix C.4 for more details on downlink routing. Appendix E.2 contains additional details which are used for modeling and validation of this finding. This includes Specifications (*RSXX*), Decisions (*RDXX*), and Assumptions (*RAXX*) that are used for decisions in the constructed model and source-code.

Activity steps
Before a model can be constructed, it should be known what functionality to model, and which steps must be involved. For this particular model, the following steps are defined:

- An already active end-device sends an uplink network frame (wireless)
- Attacker eavesdrops and replays network frame to other gateway (for the attack to succeed, this must be done using a faster network so that the malicious frame arrives before the authentic frame)
- The replayed uplink network frame is received by one or more gateways that are out-of-reach by the authentic end-device
- The gateway(s) forward the network frame to the network server (TCP/IP)
- The network server updates its downlink routing table based on details of the received uplink network frame
- The authentic uplink network frame is received by one or more gateways
- The authentic uplink network frame is forwarded to the network server
- The authentic uplink network frame is discarded as being a duplicate (de-duplication by network server) because the frame was already received and processed

6.2.1 *Model construction*

In CPNTools, a CPN model is constructed that can be used to study this finding in more detail. Figure 6.8 represents the created model on the highest level. An *end-device* sub-model is constructed that generates new network frames per the LoRaWAN specification (frame fields, field contents, and timings). A *network server* sub-model is constructed that received and processes network frames. An *attacker* sub-model is constructed, that is in-between the end-device and gateways, that replays captured network frames towards a different gateway.

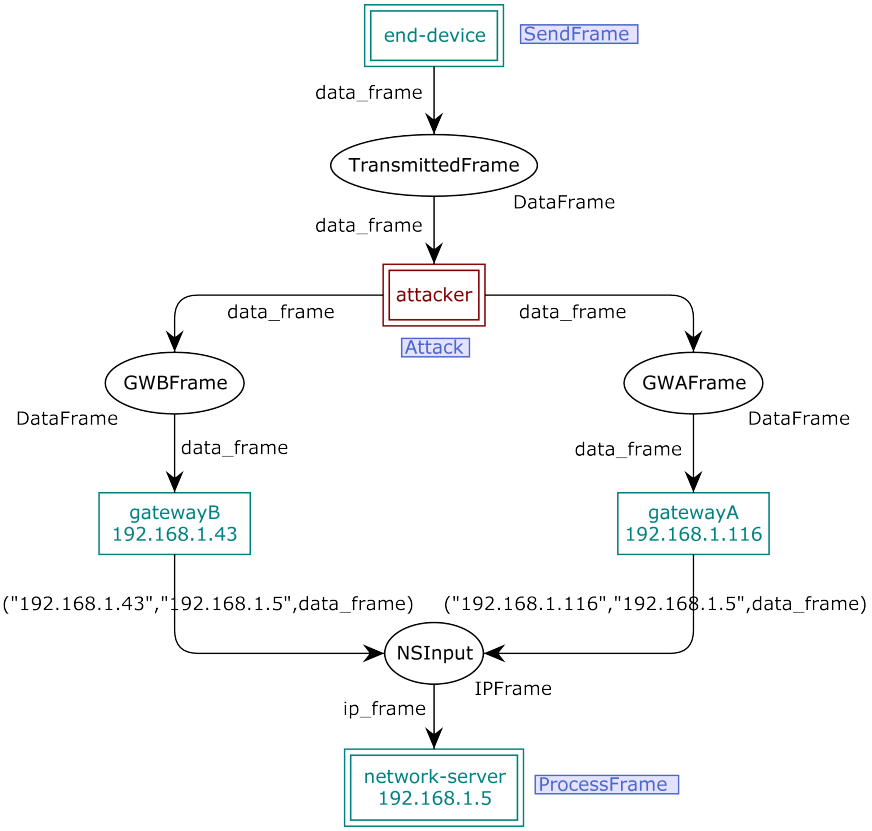

Figure 6.8: Downlink routing CPN: top-level

A top-level page is constructed which includes the major components (*RA01*). An end-device transmits network frames, an attacker eavesdrops and replays network frames, gateways forward network frames, and the network server processes the network frames.

End-Device

The end-device is modeled to transmit a network frame every ten seconds. See figure 6.9 for the sub-model. Thousand network frames are transmitted in total, to limit the simulation duration. The only difference between the transmitted network frames is the frame counter. The frame counter is incremented after every frame transmission. The

transmission has a delay of five to ten milliseconds. This is because the attacker is in-between end-device and gateway, as close to the end-device as possible.

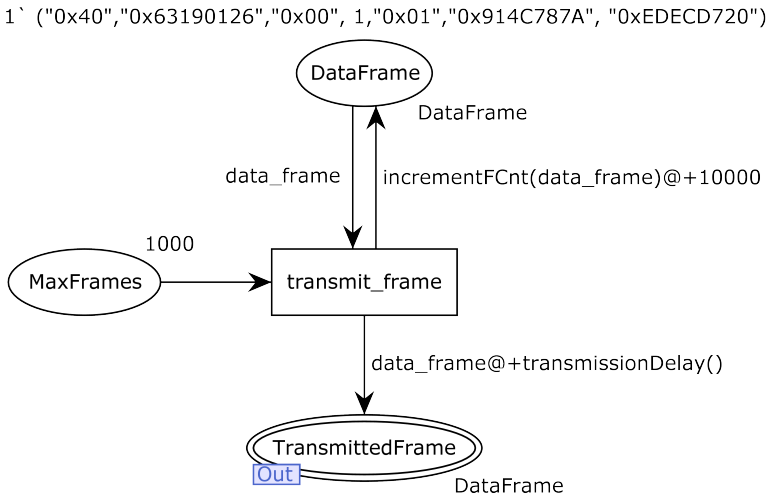

Figure 6.9: Downlink routing CPN: end-device

The *transmit_frame* transition, is consuming a token from *DataFrame* and *MaxFrames*. *DataFrame* contains a token that represents a network frame containing data. *MaxFrames* serves as a counter which allows for a maximum of thousand beaconing frames to be transmitted for this simulation. Transition *transmit_frame* produces a new token in output place *TransmittedFrame* and only adds a time unit delay to the produced token to represent a transmission delay.

Attacker

The attacker is modeled (see figure 6.10) to eavesdrop the end-device network frames as close to the end-device as possible. The closer to the end-device, the more time to replay the packet. A replayed network frame must be received by the network server before the authentic network frame. Practical implementation of this attack is out-of-scope of this study. However, using a fast connection towards a malicious end-device, or gateway, is a real possibility.

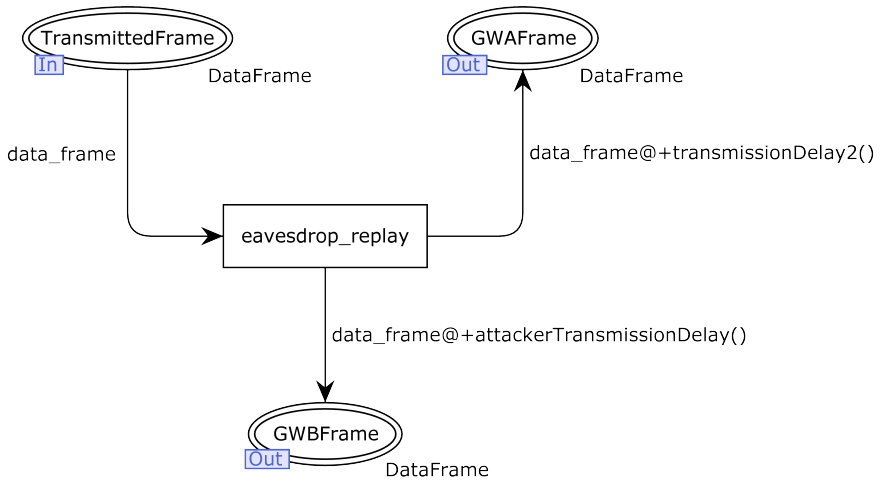

Figure 6.10: Downlink routing CPN: attacker

The *TransmittedFrame* output place from the end-device sub-model is the input place for the attacker sub-model. This sub-model contains a transition *eavesdrop_replay*, which consumes the token containing the data frame, and produces two new tokens. The first new token is produced in output place *GWAFrame*, this represents the authentic data frame that is received by the authentic gateway. The second new token is produced in output place *GWBFrame*, and represents the replayed version of the data frame which is received by a non-authentic gateway. To both tokens, an individual time unit delay is added to represent a transmission delay.

Gateways

There is no sub-model for the gateway functionality. In this CPN model, the gateway functionality is represented as single transitions which only passes the network frame to the network server.

Network Server

The constructed network server sub-model (see figure 6.11) first checks if a received network frame has a correct frame counter. When the frame counter is OK, the frame can be processed further. For this finding, it means that the downlink routing table is updated based on the gateway that forwarded the frame.

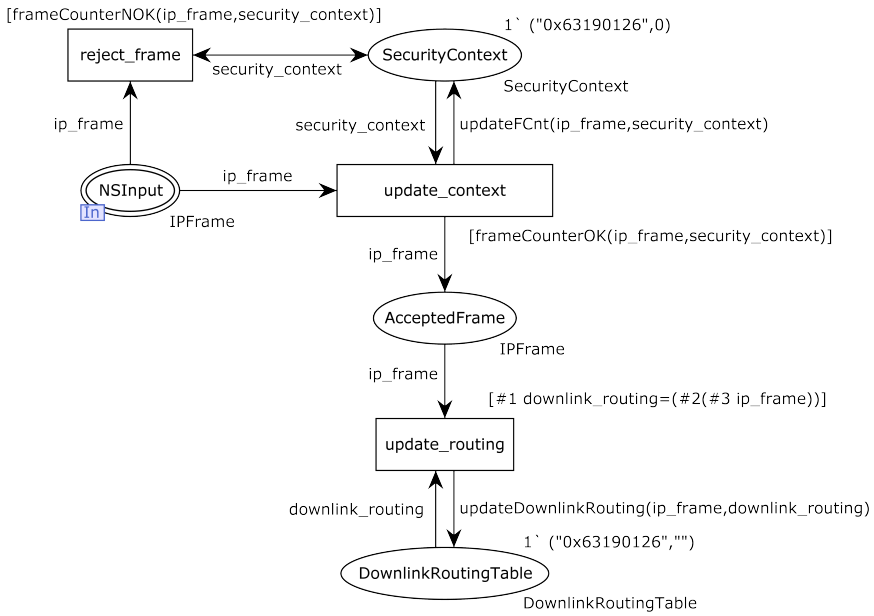

Figure 6.11: Downlink routing CPN: network server

The *NSInput* input place will contain tokens that represent forwarded data frames by both gateways (all input data frames that need to be processed by the network server). A network server maintains a security context for all end-devices, which keeps track of security keys and frame counters (in this simulation only the uplink frame counter is used). When processing an incoming data frame, the *reject_frame* transition verifies the frame counter. If the frame counter is not OK, the token is consumed from *NSInput* and further processing stops. Transition *update_context* also verifies the frame counter, if the frame counter is OK, then the token is consumed from *NSInput*, the frame counter in the security context is updated, and a new token containing the data frame is produced in place *AcceptedFrame*. Besides a security context for all end-devices, also a downlink routing table is maintained by the network server. This table also holds a routing entry for each end-device. Transition *update_routing* consumes the data frame token to update the downlink routing table. The downlink routing table is updated by consuming the token related to the end-device, and producing a new toking related to the end-device, containing the gateway that was used to forward the uplink data frame. The downlink routing table is maintained in the *DownlinkRoutingTable* place.

6.2.2 *State space analysis*

Similar to the previous finding, analyzing the full state space for the Downlink Routing model has only been possible during the first stages of construction. By extending the model, the state space has grown too large to be analyzed within several days. This behavior is

also known as the state space explosion [37]. The partial state space report for the final CPN model has run for over twelve hours. The report generation took around one hour.

The results of the state space analysis for the Downlink Routing model are identical to the state space analysis of the Beaconing model. The partial state space analysis did not uncover any incorrectness in the final model.

6.2.3 *Simulation*

The next steps provide a simplified summary of the simulation details and analysis, which will result into a manipulated downlink routing table. Figure 6.12 shows the attack sequence required for this attack. Authentic network frames are eavesdropped by an attacker and replayed via a different gateway. When the attacker is able to get the replayed network frame processed before the authentic network frame, then the downlink routing table is updated so that downlink traffic towards the end-device is initiated via the wrong gateway.

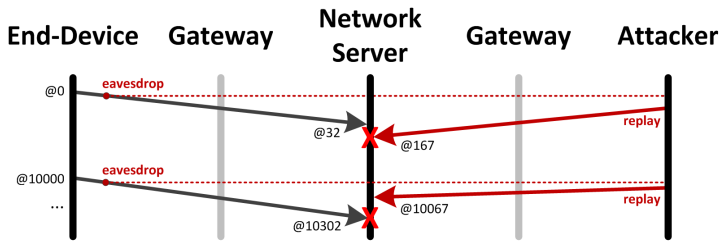

Figure 6.12: Downlink routing: simulation attack-sequence

The *end-device* sub-model is constructed to generate network frames based on the LoRaWAN specification. Important fields in the network frame (for this finding) are the device address (DevAddr), and the frame counter (FCnt).

Below, two network frames are shown that are used throughout this summary. These network frames are send out by the *end-device* sub-model. The first line represents a network frame that is send out on time-unit zero with a frame counter of one. The second line represents a network frame that is send out on time-unit 10,000 (after ten seconds) with a frame counter of two.

```
(MHDR,DevAddr,FCtrl,FCnt,FPort,FRMPayload,...) @ time-unit
("0x40","0x63190126","0x00", 1,"0x01","0x914C787A",...) @ 0
("0x40","0x63190126","0x00", 2,"0x01","0x914C787A",...) @ 10000
```

In-between end-device and gateway, an attacker is modeled. This attacker eavesdrops network frames that are transmitted by the end-

device. The eavesdropped network frames are replayed towards a different gateway using a high-speed network. The two network frames shown below, represent the two network frames at the moment these are eavesdropped by the attacker. All information is identical, the only difference is the time-unit. A small amount of time has passed between transmission and eavesdropping.

```
("0x40","0x63190126","0x00", 1,"0x01","0x914C787A",...) @ 10
("0x40","0x63190126","0x00", 2,"0x01","0x914C787A",...) @ 10007
```

There are two gateways in the CPN model. One gateway (192.168.1.116) forwards the authentic network frames, and one gateway (192.168.1.43) forwards the replayed network frames. The two network frames below are passed on by the gateway used for authentic traffic. The line marked in red is impacted by the replay attack.

```
(GW-IP,NS-IP,(LoRa-frame)) @ time-unit
("192.168.1.116","192.168.1.5",("0x40","0x63..., 1,...)) @ 32
("192.168.1.116","192.168.1.5",("0x40","0x63..., 2,...)) @ 10302
```

The two network frames below are forwarded by the gateway that is used by the attacker.

```
("192.168.1.43","192.168.1.5",("0x40","0x63..., 1,...)) @ 167
("192.168.1.43","192.168.1.5",("0x40","0x63..., 2,...)) @ 10067
```

On the network server, all four network frames are received. Two authentic network frames, and two replayed network frames. The time-unit in the last column shows the arrival time and this decides the order of processing. Based on arrival time, and frame counters, two network frames are rejected (marked red), and two are accepted.

```
("192.168.1.116","192.168.1.5",("0x40","0x63..., 1,...)) @ 32
("192.168.1.43" ,"192.168.1.5",("0x40","0x63..., 1,...)) @ 167
("192.168.1.43" ,"192.168.1.5",("0x40","0x63..., 2,...)) @ 10067
("192.168.1.116","192.168.1.5",("0x40","0x63..., 2,...)) @ 10302
```

When the downlink routing table is updated based on the accepted network frames, then it is clear that one of the frames results into a wrong downlink routing entry. The entry marked red is causing unavailability of the end-device for downlink traffic. This wrong entry remains active until a new authentic network frame is processed by the network server.

```
DevAddr    -> GW-IP         @ time-unit
0x63190126 -> 192.168.1.116 @ 32
0x63190126 -> 192.168.1.43 @ 10067
```

Figure 6.13 shows a graphical representation of the attack. When a replayed network frame is received before an authentic network frame, then the authentic network frame is rejected. When a replayed network frame is processed by the network server, the downlink routing table is pointing to the wrong gateway. This results in unavailability of the end-device, because downlink traffic is forwarded to the wrong gateway for delivery towards the intended end-device.

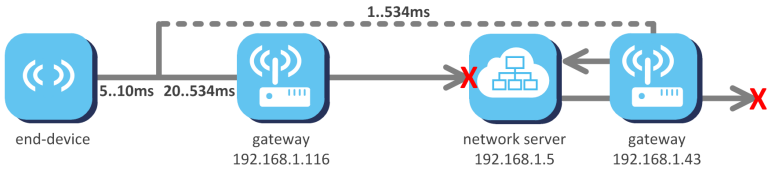

Figure 6.13: Manipulated downlink routing

This simulation has been repeated many times. The results of the first twenty-five runs are represented in figure 6.14. The graph shows the transmission times of the authentic network frames compared to the transmission times of the replayed network frames. Where the red graph is below the blue graph, replayed frames arrive sooner than authentic frames, and receive slot calculations are impacted. The results of the other simulations are all in line with these first twenty-five. Only the first twenty-five results are used in this graph to improve readability.

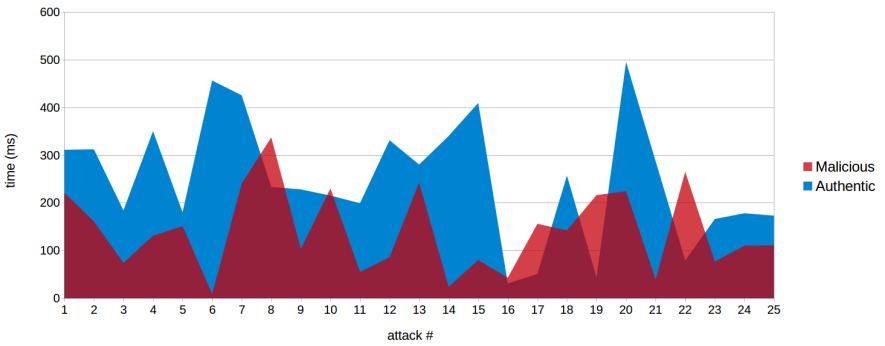

Figure 6.14: Downlink routing - simulation statistics

The key aspect of this finding, is that an attacker is able to get a replayed network frame processed sooner than an authentic network frame. To simulate this, the model contains transmission delays on three arcs (see figure attack-modeling-validation:figure:flow-downlink-routing-wormhole). First a small generic delay between end-device and attacker (5ms..10ms). Second a delay (20ms..534ms) between attacker and authentic gateway. This delay, combined with the first delay, represents the total delay between end-device and authentic gateway (*RD03*). Third, a delay (1ms..534ms) between attacker and malicious gateway. As long as the transmission speed of the attacker

is higher, compared to the LoRaWAN specification, the attack will succeed.

According to the LoRaWAN specification, a network frame of the used size (136 bits) takes at least 25ms to reach the gateway. This means that all malicious network frames that are reaching a different gateway within 0ms..25ms will always arrive before the authentic network frame. If the attacker achieves a speed of 136,000bps (= 0,129 Mb/s) to 9,714bps (= 0,009 Mb/s), then all replayed network frames will arrive before the authentic network frames. Slower speeds will not guarantee a successful attack, however slower transmission speeds can still cause a DOS of Class-B downlink traffic, as long as the malicious network frame arrives before the authentic network frame. Increased attacker transmission speeds result in a higher attack success rate.

6.2.4 *LoRaWAN v1.1 specification*

On October 17th 2017, a new version (v1.1) [22] of the LoRaWAN specification was published. This specification does not contain changes related to this downlink routing finding.

6.3 END-DEVICE ACTIVATION

This section describes the CPN model of the finding related to end-device activation in the LoRaWAN specification. This finding showed that a replay attack can be used to impact availability of the end-device. While initiating a replay attack during an authentic Over-the-Air-Activation (OTAA), it might be possible to get the end-device configured with the wrong configuration settings. See also Appendix C.1 for more details on end-device activation. Appendix E.3 contains additional details which are used for modeling and validation of this finding. This includes Specifications (*BSXX*), Decisions (*BDXX*), and Assumptions (*BAXX*) that are used for decisions in the constructed model and source-code.

Activity steps

Before a model can be constructed, it must be known what functionality to model, and which steps should be involved. For this particular model, the following steps are defined:

- At a certain moment, the end-device must re-join the network, initiates a new OTAA, and sends a Join-Request
- The network server receives the Join-Request and assigns a new security context
- Before the network server can answer, the attacker replays a previous Join-Accept based on an older security context

- The end-device processes the replayed Join-Accept and changes its configuration to a wrong security context
- The network server sends the authentic Join-Accept based on the new security context
- The end-device rejects the authentic Join-Accept because it has already processed a Join-Accept
- The end-device and network server are unable to communicate because both use a different security context

6.3.1 *Model construction*

In CPNTools, a model is constructed that can be used to study this finding in more detail. Figure 6.15 represents the created model on the highest level.

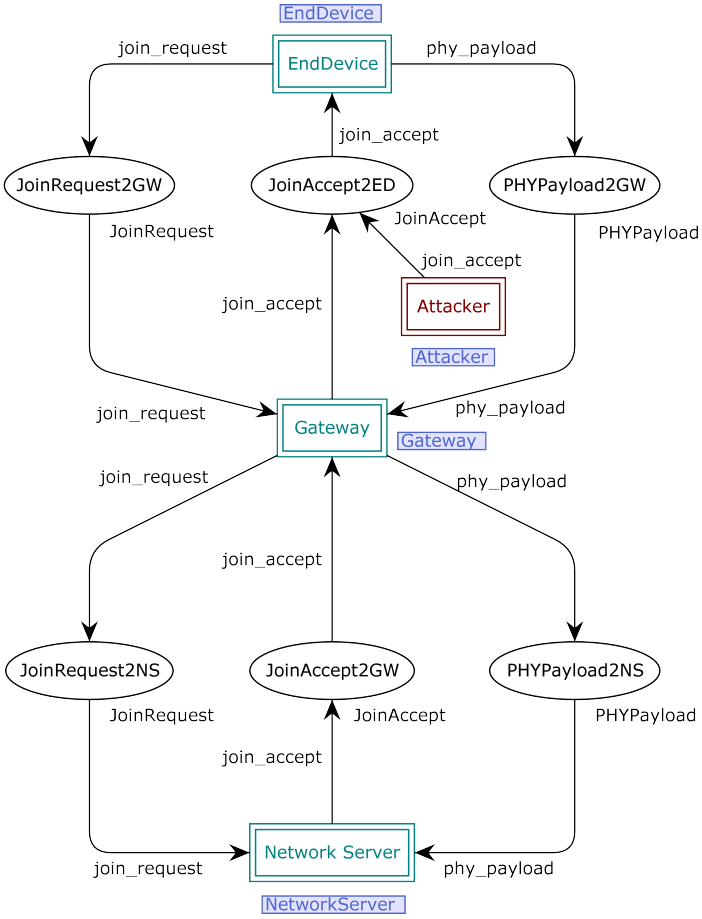

Figure 6.15: End-Device activation CPN: top-level

A top-level model is constructed which includes the major components (*BA01*). An *end-device* sub-model is constructed that joins a network and transmits data. A *gateway* sub-model is constructed that forwards different network frames between end-device and network

server. A *network server* sub-model is constructed that processes the Join-Request and transmitted data. An *attacker* sub-model is constructed, that is in-between the gateway and the end-device, that replays an earlier Join-Accept message at a specific time so that it is accepted and processed by the end-device.

End-device

The end-device is modeled to generate and transmit several Join-Request messages on predefined times. The first Join-Request is transmitted at the start of the simulation run. The end-device keeps track of the Join-Request and will only process a Join-Accept if it has an outstanding Join-Request. When a Join-Accept comes in, the end-device checks if it expects a Join-Accept message. If not, the received Join-Accept is rejected, otherwise the received Join-Accept is processed. Based on the received Join-Accept, a new security context is generated (identical to the security context at the network server). Based on the security context, the end-device transmits a data frame every sixty seconds. The end-device model is shown in figure 6.16.

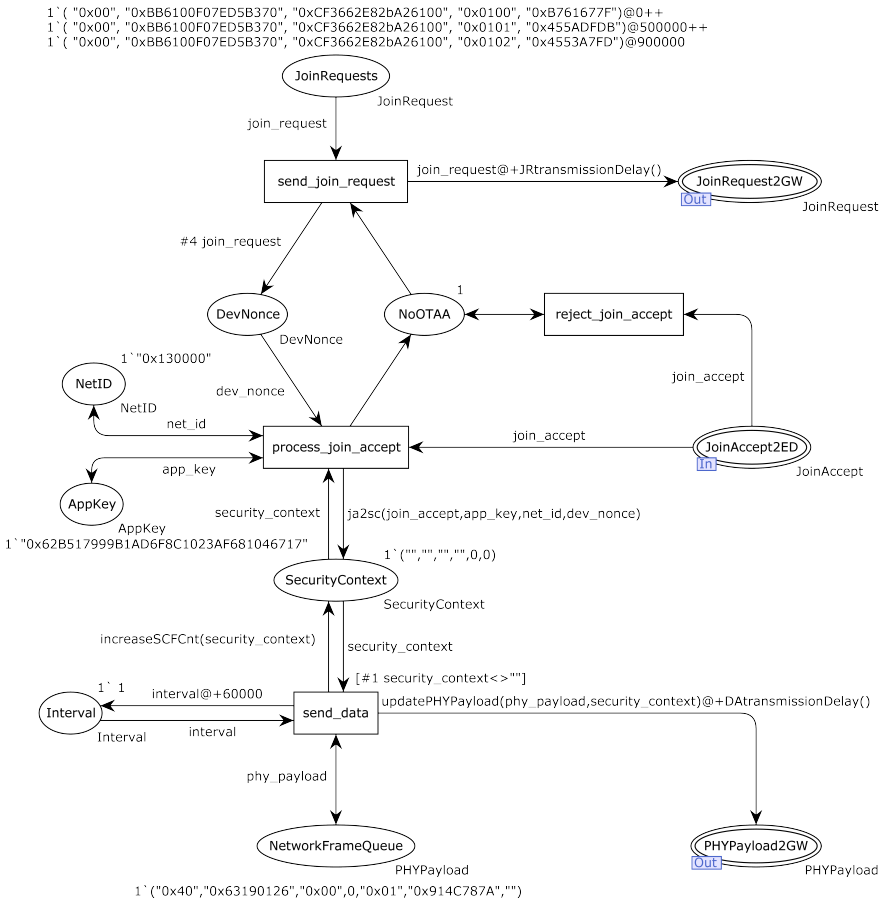

Figure 6.16: End-device activation CPN: end-device

This end-device sub-model has three functionalities: create and transmit Join-Request messages, receive and process Join-Accept messages, and transmit network frames containing data.
The creation and transmission of Join-Request messages is handled by the *send_join_request* transition. This transition consumes tokens containing Join-Request messages from *JoinRequests* and produces tokens contain the Join-Request messages in *JoinRequest2GW*. A time unit delay is added to the produced token to represent a transmission delay. The transition also uses the *NoOTAA* and *DevNonce* places to make sure that only a single OTAA procedure is active, combined with preserving the used DevNonce value for use in processing a Join-Accept message.
Receiving and processing of Join-Accept messages is handled by the *reject_join_accept* and *process_join_accept* transitions. The transition *reject_join_accept* checks if there is an OTAA active. If not, it consumes the token in input place *JoinAccept2ED* and stops further processing. Transition *process_join_accept* also checks if an OTAA is active, if so, then it will generate a new security context. The new security context is based on information from the token containing the Join-Accept message which is consumed from *JoinAccept2ED*, the token containing the DevNonce, and tokens containing the AppKey and NetID. The transition also consumes the old security context, and produces a new security context in *SecurityContext*.
Transmission of network frames containing data is handled by the *send_data* transition. This transition uses a token from *SecurityContext*, and a token from *NetworkFrameQueue* to produce a token containing an uplink data frame in output place *PHYPayload2GW*. The produced token contains an updated frame counter, a calculated MIC value, and a time unit delay is added to represent a transmission delay. The transition also updates the new frame counter in the security context. An *Interval* place is used to limit the transmission of uplink data frames to one frame per sixty seconds.

Gateway
The gateway is modeled to forward different types of network frames between end-device and network server (see figure 6.17). Transmission delays are based on *AD03*, *AD04*, *AD05*, *AD09*, and *AD10*.

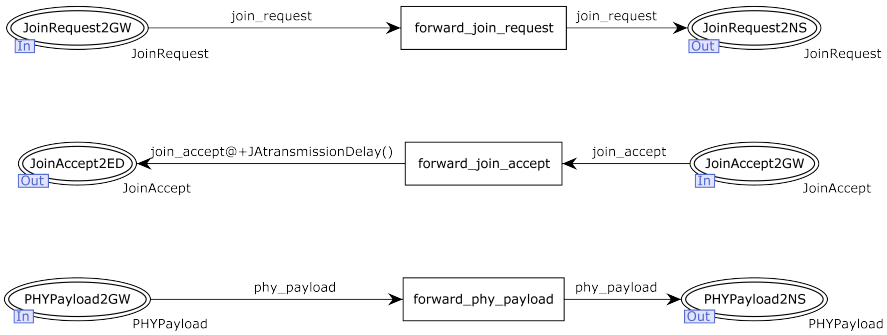

Figure 6.17: End-device activation CPN: gateway

The gateway sub-model uses three transitions to forward tokens between different input- and output places: *forward_join_request*, *forward_join_accept*, and *forward_phy_payload*. Forwarding is performed by consumption and production of tokens. Only when producing a token towards the end-device (JoinAccept2ED), a time unit delay is added to represent a transmission delay (because this is a transmission onto the wireless part of the network).

Network server

The network server is modeled to receive and process Join-Request messages, generate and transmit Join-Accept messages, manage security contexts, and receive data frames. When a Join-Request message arrives, then the network server first checks if the DevNonce is not already used (*AS05*). If it is used before, then the request is rejected. If it is not used before, then the Join-Request is processed. A new, unconfirmed, security context is generated, and a Join-Accept message is generated and transmitted towards the end-device. The first time an uplink data frame is received, the integrity is checked using an unconfirmed security context for that end-device. If this succeeds then the unconfirmed security context is confirmed and the data frame can be processed (*AS04*). Subsequent data frames are validated using the confirmed security context. The network server sub-model is shown in figure 6.18.

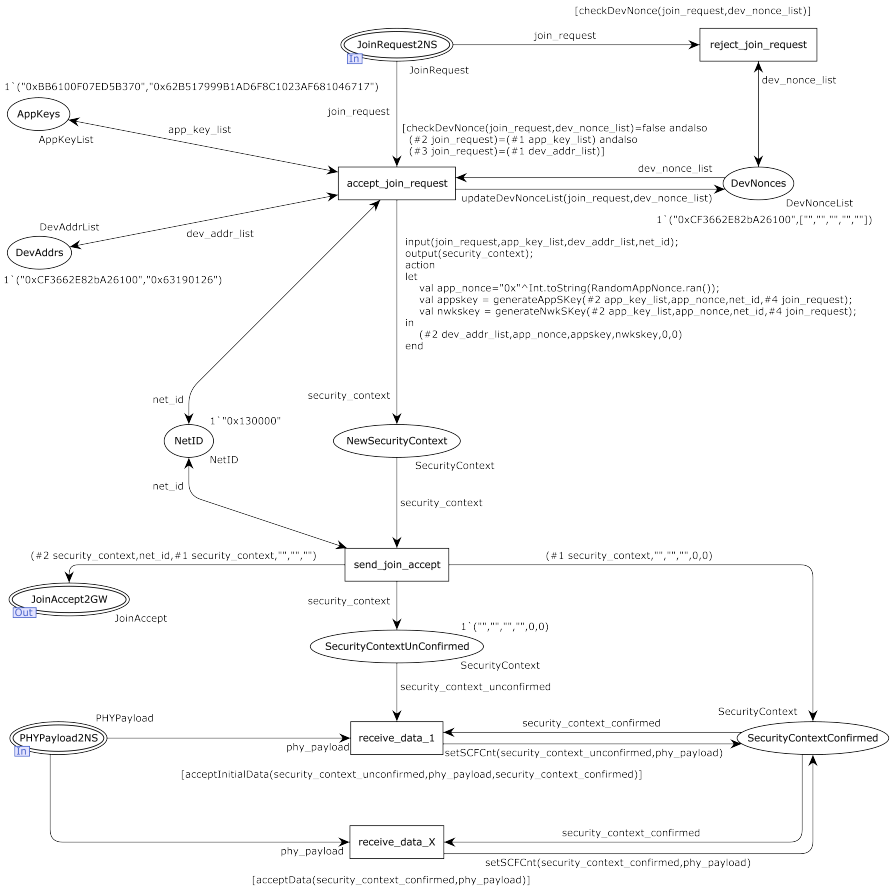

Figure 6.18: End-device activation CPN: network-server

This network server sub-model has several functionalities: receive and process Join-Request messages, generate and transmit Join-Accept messages, managing security contexts, and receive and process network frames containing data.

Receiving and processing Join-Request messages is handled by the *reject_join_request* and *accept_join_request* transitions. Both transitions check if the end-device did not use the DevNonce value before. If it is used before, then *reject_join_request* consumes the incoming token from *JoinRequest2NS* and stops further processing. If the DevNonce is not used before, then *accept_join_request* will consume the token from *JoinRequest2NS*. This transition will also use tokens from other places to gather additional details such as DevAddr, NetID, and AppKey. Based on these details a new security context is generated and added to *NewSecurityContext*.

Generating and transmitting a Join-Accept message is handled by *send_join_accept*. This transition uses the new security context to generate and transmit the Join-Accept message in the outgoing place *JoinAccept2GW*. It also produces tokens in *SecurityContextUnconfirmed* and *SecurityContextConfirmed* so that it can processes future uplink transmissions from the end-device properly.

While receiving and processing data frames, the security contexts are managed as well. These activities are handled by transitions *receive_data_1* and *received_data_X*. Transition *receive_data_1* will be used for an incoming data frame in *PHYPayload2NS* when the security context is still present in *SecurityContextUnconfirmed* (first uplink after joining). If this is the case, the security context token will be consumed from *SecurityContextUnconfirmed* and the token in *SecurityContextConfirmed* will be consumed and a new one produced with updated details (such as the frame counter). When this is not the first uplink data frame for the used security context, then the uplink data frame in *PHYPayload2NS* is handled by transition *received_data_X*. It will update the frame counter for the end-device's security context in *SecurityContextConfirmed*. Validating and checking for a security context is done by calculating the MIC value at the network server side, identical to the calculation performed at the end-device side. When a match is found, the correct security context is present and will be used to process the uplink data frame. If no match is found, incoming data frames are not processed.

Attacker

The attacker is modeled to replay an earlier eavesdropped Join-Accept message at two specific moments in time (@5000, and @500004) (see figure 6.19). The first replayed Join-Accept message will be rejected by the end-device because it does not have a pending Join-Request. The second replayed Join-Accept message will be processed by the end-device, because at that specific moment it is expecting a Join-Accept message. Since there is no way to determine if a Join-Accept message belongs to a specific Join-Request message, the end-device will accept and process this replayed Join-Accept message as long as it is received by the end-device before the authentic Join-Accept message.

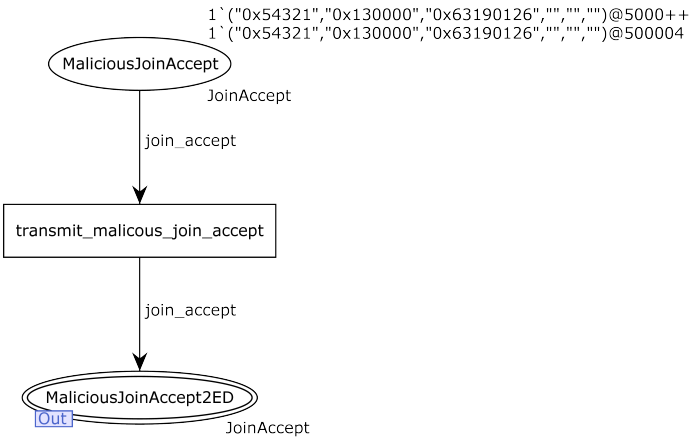

Figure 6.19: End-device activation CPN: attacker

The *transmit_malicious_join_accept* transition consumes tokens containing malicious Join-Accept messages from *MaliciousJoinAccept*, and produces tokens containing the malicious Join-Accept messages in the output place *MaliciousJoinAccept2ED*, which is the *JoinAccept2ED* input place for the end-device.

6.3.2 *State space analysis*

This CPN model leads to the largest state space of all three models. Even running a partial analysis for hundred hours did not result into a state space analysis report in which all transitions have been fired. Besides Dead Markings, this report also contains a Dead Transition (NetworkServer'reject_join_request). However, in simulation it can be seen that this transition, which is marked as dead, is being fired. The (partial) state space analysis report on the final model is less useful compared to the reports of the previous models. The results in the report can be explained, and this partial report did not uncover any additional issues that may indicate an invalid model.

6.3.3 *Simulation*

The following steps provide a simplified summary of the simulation details and analysis which will result into a DOS of the end-device. Figure 6.20 shows the attack sequence required for this attack.

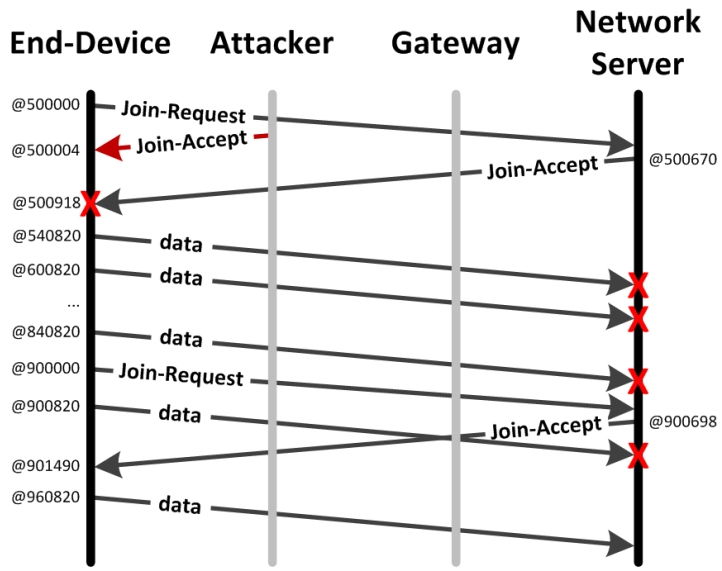

Figure 6.20: End-Device activation: attack-sequence

When an end-device transmits a Join-Request to join the network, an attacker replies with a Join-Accept before the network server can do. From that moment, the end-device does not use the same security context as the network server anymore, and all transmissions will fail

until a next successful join to the network.

End-Device
The end-device in this CPN model transmits Join-Request, processes Join-Accepts, and transmits Data network frames.

The below network frames are the Join-Request messages that are send out by the end-device. The red marked entry is impacted by the actions of the attacker.

```
(MHDR,AppEUI,DevEUI,DevNonce,MIC) @ time-unit
("0x00","0xBB610...","0xCF3662...","0x0101","0x455...") @ 500000
("0x00","0xBB610...","0xCF3662...","0x0102","0x455...") @ 900000
```

The entries below show the received Join-Accept messages by the end-device. The red marked entry is rejected by the end-device because there was no pending Join-Request / OTAA. The Join-Accept @500004 is a malicious Join-Accept which is accepted by the end-device.

```
(AppNonce,NetID,DevAddr,DLSettings,RxDelay,CFList) @ time-unit
("0x54321","0x130000","0x63190126","","","") @ 500004
("0x29229","0x130000","0x63190126","","","") @ 500918
("0x47187","0x130000","0x63190126","","","") @ 901490
```

After accepting a Join-Accept message, the end-device generates a new security context. The below entries are security contexts that are generated during the simulation. The red marked entry, is a security context that is generated based on a malicious Join-Accept message.

```
(DevAddr,AppNonce,AppSKey,NwkSKey,FCntUp,FCntDown) @ time-unit
("0x631...","0x54321","0x023211...","0x0154354...",0,0) @ 500004
("0x631...","0x47187","0x021871...","0x0147147...",0,0) @ 901490
```

During the simulation run, the end-device transmits a network frame every 60 seconds. The red marked entries are rejected by the network server because the network frames are transmitted using a security context that is based on a malicious Join-Accept message.

```
(MHDR,DevAddr,FCtrl,FCnt,FPort,FRMPayload,MIC) @ time-unit
("0x40","0x63...","0x00",8,"0x01","0x91...","0x015...") @ 480820
("0x40","0x63...","0x00",0,"0x01","0x91...","0x015...") @ 540820
("0x40","0x63...","0x00",1,"0x01","0x91...","0x015...") @ 600820
("0x40","0x63...","0x00",2,"0x01","0x91...","0x015...") @ 660820
("0x40","0x63...","0x00",3,"0x01","0x91...","0x015...") @ 720820
("0x40","0x63...","0x00",4,"0x01","0x91...","0x015...") @ 780820
("0x40","0x63...","0x00",5,"0x01","0x91...","0x015...") @ 840820
("0x40","0x63...","0x00",6,"0x01","0x91...","0x015...") @ 900820
("0x40","0x63...","0x00",0,"0x01","0x91...","0x014...") @ 960820
```

Attacker
In this CPN model, an attacker is modeled that replays (transmits) malicious Join-Accept messages.

```
(MHDR,AppEUI,DevEUI,DevNonce,MIC) @ time-unit
("0x54321","0x130000","0x63190126","","","") @ 500004
```

Gateways
In this CPN model, the gateway just provides the functionality for forwarding different types of network frames between end-device and network server.

Network Server
The network server in this CPN model receives and processes Join-Request message, generates and replies Join-Accept message, manages security contexts, and received and processed data network frames.

During the simulation, the network server receives two Join-Request messages from the end-device. All Join-Request messages are accepted, processed, and a Join-Accept message is replied. Based on the processed Join-Accept message, also new security contexts are generated. Below are the Security contexts that are generated for this end-device during the simulation. The red marked security context is never confirmed by the end-device because the end-device is impacted by the malicious Join-Accept, and therefore not able to confirm this particular security context. Notice the frame counter in the 5^{th} column. The red entry does not have any communication, the other one has many received network frames.

```
(DevAddr,AppNonce,AppSKey,NwkSKey,FCntUp,FCntDown) @ time-unit
("0x63190126","0x29229","0x0222913...","0x012922929223...",0,0)
("0x63190126","0x47187","0x0218713...","0x014714717183...",979,0)
```

On the network server, the below network frames are received. The red marked entries are the network frames that are based on a malicious security context on the end-device. This malicious security context is different compared to the security context on the network server, and therefore these network frames are rejected by the network server.

```
(MHDR,DevAddr,FCtrl,FCnt,FPort,FRMPayload,MIC) @ time-unit
("0x40","0x63...","0x00",0,"0x01","0x91...","0x015...") @ 540904
("0x40","0x63...","0x00",1,"0x01","0x91...","0x015...") @ 601197
("0x40","0x63...","0x00",2,"0x01","0x91...","0x015...") @ 660967
("0x40","0x63...","0x00",3,"0x01","0x91...","0x015...") @ 721332
("0x40","0x63...","0x00",4,"0x01","0x91...","0x015...") @ 781319
("0x40","0x63...","0x00",5,"0x01","0x91...","0x015...") @ 841306
("0x40","0x63...","0x00",6,"0x01","0x91...","0x015...") @ 901177
("0x40","0x63...","0x00",0,"0x01","0x91...","0x014...") @ 961059
```

The simulation is in line with expected results from previous analysis. When an attacker is able to insert a malicious Join-Accept before an authentic Join-Accept, then the malicious Join-Accept is processed which leads to a security context on the end-device that differs from the security context on the network server. No communication between end-device and network server is possible until the next successful OTAA.

6.3.4 *LoRaWAN v1.1 specification*

On October 17th 2017, a new version (v1.1) [22] of the LoRaWAN specification was published. This new version of the specification contains significant changes related to end-device activation. The most important changes are listed in table E.18.

The most important change is that joining a network now requires the use of a JoinNonce value in a Join-Accept message (see changes *E+F*). "*The JoinNonce is a device specific counter value (that never repeats itself) provided by the Join Server and used by the end-device to derive the session keys FNwkSIntKey, SNwkSIntKey, NwkSEncKey and AppSKey. JoinNonce is incremented with every Join-Accept message.*" [22] (page 52). By using an incremental value, a replay attack on the Join-Accept message is not possible anymore. However, it must be noted that due to backwards compatibility, this finding may continue to be an issue for the near future.

6.4 CONCLUSIONS

Based on the analysis activities described in chapter 5, three findings are selected for a more detailed analysis. The selected findings are: an eavesdropping + replay attack on beaconing network frames which are broadcasted by gateways to perform a Class-B downlink DOS for all end-devices in reach, an eavesdropping + replay attack on end-device uplink network frames to manipulate the network server downlink routing table to perform a downlink DOS for a specific end-device, and an eavesdropping + replay attack on the Join-Accept message during an authentic OTAA to perform an overall DOS on a specific end-device.
For all findings, a CPN model is constructed in CPNTools, which represent the findings based on a partial LoRaWAN v1.0.2 specification. Only those parts of the specification that are related to these findings are included into the model. Based on references to the LoRaWAN specification, decisions, and assumptions, models have been constructed in the graphical user interface of CPNTools. The models include the required components, such as end-device, gateway, network server, and attacker on the highest level. For each of these a components, a

more detailed sub-model is created to represent the required functionality. These CPN models are constructed manually. Since the LoRaWAN specification is written in natural language, and only parts of the specification are required, it is not possible to convert the specification into CPN models automatically. Because of this natural language, the final models are based on my interpretation of the specification.

During construction of the CPN models, state space analysis is used as a guidance to verify the validity of the models. State space analysis is included in CPNTools and uses the mathematical basis of CPN models to verify certain properties and behavior. Unfortunately all three models suffer from a state space explosion which rendered the final models too big to have a full state space analysis. When the models become large, only partial state space analysis could be performed. However, even a partial state space analysis is useful when used for reasoning about a model's properties (validity).

CPNTools has also built-in options for simulation, which can be used to validate the modeled functionality. Simulation provides visual feedback, and extensive logging options. Both the visual analysis, and the collected log data, are used to validate the selected findings.

Beaconing
For the beaconing finding, a CPN model is constructed that includes a gateway, an end-device and an attacker. The gateway broadcasts beaconing frames with a specific interval. Such a beaconing frame contains a time value that is used for calculating downlink receive slots. Using an eavesdropping attack combined with a replay attack, the attacker is able to insert malicious beaconing frames, or replace authentic beaconing frames with malicious beaconing frames. Malicious beaconing frames can contain a time value that differs from the correct time value, which impacts the calculation of the downlink receive slots. Malicious beaconing frames can also contain invalid location details of the respective gateway.
State space analysis is used during model construction. For the final model, properties are validated using a partial state space analysis. No indication is found that proves the model wrong.
Simulation provided a repeatable visual analysis, and log data that is used to validate the functional correctness of the constructed CPN model. The visual analysis clearly shows the attacker inserting malicious beaconing frames, replacing authentic beaconing frames, the end-device accepting malicious beaconing frames, and the end-device calculating the receive slots using the time value in the malicious beaconing frames. The simulation log data confirmed the visual analysis.

The conclusion for this finding is that using the constructed model, it is possible to manipulate receive slot calculations on the end-device using malicious beaconing frames. This supports the beaconing analysis results from chapter 5.3.

Downlink routing

The CPN model constructed for the downlink routing finding contains an end-device, two gateways, a network servers, and an attacker. The end-device has authentic communications with the network server, using one of the gateways. The attacker eavesdrops authentic network frames, and replays network frames over a faster network towards a second gateway that is out-of-reach for the end-device. When replayed network frames arrive before authentic network frames, the network server will process these (and reject the authentic network frames because of de-duplication), and register the wrong gateway for the end-device in its routing table.
For this model, state space analysis is used during construction, and on the final model. For the final model, only a partial state space analysis is possible. The analyzed properties show no indications that prove the model wrong.
A visual analysis, and collection of log data, is performed using simulation. During simulation it is clearly visualized that the attacker replays all eavesdropped network frames, and when the replayed network frames arrive before the authentic network frames, the network server's downlink routing table is updated with the wrong details causing an unavailable end-device for downlink traffic. The collected log data confirms the visual analysis and shows a wrongly updated downlink routing table.

The constructed model shows that it is possible to modify the network server's downlink routing table by replaying network frames via a wrong gateway. This result supports the downlink routing analysis results from chapter 5.3.

End-device activation

For the end-device activation finding, a CPN model is constructed. This model includes an end-device, a gateway, a network server, and an attacker. The end-device has authentic communications with the network server (Join-Requests, Join-Accepts, and uplink data frames), via the gateway. When an attacker has an eavesdropped Join-Accept message from this end-device, it can be used for a replay attack during a new authentic OTAA of the end-device, causing a DOS. Because an end-device uses details of the Join-Accept message to calculate the security context, it will calculate a wrong security context when a replayed Join-Accept message is used. A wrong security context means the end-device and network server do not use the same security keys

for encryption and message integrity checks.
During construction of the model, state space analysis is used as a guidance. Because the model quickly became larger than the other models, the state explosion problem appeared already soon in the construction process. State space analysis on the final model required a lot more time, and resulted in a less complete state space analysis compared to the other models. Nevertheless, the analyzed properties show no indications that prove the model wrong.
The visual analysis, and log data collection, provided by simulation, clearly shows that when the attacker replays an earlier eavesdropped Join-Accept message, the end-device is using the replayed Join-Accept to generate a new security context. It also shows that all further traffic is rejected by the network server, until the end-device has successfully executed a new OTAA.

The construction and analysis of the CPN model for this finding shows the DOS of the end-device and supports the end-device activation analysis results from chapter 5.3. It must be noted that changes in the newer LoRaWAN v1.1 of the specification prevent replay attacks on the Join-Accept message.

7 CONCLUSIONS, DISCUSSION, AND FUTURE WORK

This chapter concludes this thesis by summarizing the answers to the defined research questions, including the contribution of this research study, explain relations to other work, and by including suggestions for further research.

Section 7.1 of this chapter provides the answers to the defined research (sub-)questions and a final conclusion. In section 7.2, the contributions of this research study are explained. Section 7.3 provides details on possible discussions. In section 7.4, future work is discussed. Section 7.5, provides a reflection on the research study.

7.1 ANSWERS TO RESEARCH QUESTIONS

This section provides the answers to the research questions from chapter 1.6. First, the answers for the sub-questions are provided. Based upon the answers of the sub-questions, the main research question is answered.

What is LoRaWAN?

LoRaWAN is a LPWAN network protocol, developed by the LoRa Alliance. The protocol is intended for wireless, battery-operated nodes in a regional, national, or global network. It is mainly used for IoT and WSN applications. The first protocol specification (v1.0) is published in October 2015. Subsequent specification publications are v1.0.2 in July 2016, and v1.1 in October 2017. Examples of LoRaWAN key characteristics are: low power, long range, localization, both up- and downlink traffic, and security build in from the first version. Commercial- and community based implementations are already available today.

What are the main architectural principles?

LoRaWAN is based on a star-of-stars topology (similar to the mobile phone infrastructure). Major components are: a network server, one or more gateways, and one or more end-devices. LoRaWAN is the link between a physical layer (LoRa) and an application layer. The majority of the traffic is uplink traffic containing small amounts of data. Downlink traffic is also possible. End-device Classes A-C are available, which provide a different trade-off between downlink flexibility and power usage.

What security related aspects have been integrated into the LoRaWAN specification?

The LoRaWAN specification includes measures related to the three pillars of security Confidentiality, Integrity, and Availability. Especially Confidentiality and Integrity are well addressed in the specification. Confidentiality is addressed by applying encryption for data payloads by default. Integrity is addressed by using a Message Integrity Code for every message. Both cryptographic actions are based on different security keys so that there is a distinction between security required for network operations, and security used by the application level. Availability is less addressed. This seems to be a result of low-power / long-range characteristics and the use of a wireless low-bandwidth network. Examples of measures that can be used to improve Availability, are the deployment of multiple gateways in the same geographic region (redundancy), and working with message confirmations. However, using these kinds of measures cannot bring Availability up to the same level as Confidentiality and Integrity.

Which known attack methods can be identified that are applicable to LoRaWAN?

From eleven possible attack methods (Eavesdropping, Replay attack, Man-in-the-Middle (MitM), Denial of Service (DOS), Key cracking, Session Hijacking, Evil Twin / Rogue AP, Gateway tampering, Frame manipulation, MAC spoofing, Buffer overflow), the first five have been selected to be analyzed in more detail in respect to the LoRaWAN specification.
From this analysis, it has become clear that by using combinations of eavesdropping and replay attacks, it is possible to perform DOS attacks on a LoRaWAN. A MitM is not directly possible because of the used network architecture, although the eavesdropping combined with replay attacks do have characteristics of a MitM. A key cracking attack is unfeasible. The used Advanced Encryption Standard (AES) is still a solid cipher, and performing brute force attacks within acceptable time limits is not possible with current technologies.

How can the selected attack methods be applied to LoRaWAN?

Three possible attacks have been found that use an eavesdropping and replay attack. The first attack, a) is to eavesdrop beaconing frames that are broadcasted by gateways, and used by end-device to calculate ping slots for Class-B downlink traffic. By applying this attack, and attacker is able to enforce end-devices to process modified beaconing frames, so that calculations on the end-device are based on wrong data. A second attack, b) is using eavesdropping and replay attacks to manipulate the routing of an uplink network frame, so that the network server registers a wrong routing path for the specific end-device. A third attack, c) is using eavesdropping and a replay attack to inject an older network frame during the end-device registration

process so that the end-device is calculating a security context based on wrong data.

What is the impact of the selected attack methods on LoRaWAN?

All three attacks result into a form of DOS between end-device(s) and network server. Attack a) manipulates timing references of the beaconing frames, which results into a DOS of all end-devices in reach, for all Class B downlink traffic (the processing of a valid beaconing frame will restore communications), b) manipulates the network server downlink routing table which results into a DOS, of a specific end-device, for all downlink traffic (a successful uplink communication of the end-device will restore communications), and c) resulted in a total DOS of a specific end-device (a new, and successfully completed, OTAA will restore communications).

Can evidence be found that (in)validates vulnerabilities in the LoRaWAN protocol specification?

Based on the different analysis activities in this research study three possible attacks are discovered. The three findings that are found, are: a) an eavesdropping + replay attack on beaconing network frames which are broadcasted by gateways to perform a Class-B downlink DOS for all end-devices in reach (Beaconing), b) an eavesdropping + replay attack on end-device uplink network frames to manipulate the network server downlink routing table to perform a downlink DOS for a specific end-device (Downlink routing), and c) an eavesdropping + replay attack on the Join-Accept message during an authentic OTAA to perform an overall DOS on a specific end-device (End-device activation).
By studying the LoRaWAN specification in detail, analyzing different attack methods, constructing and analyzing Colored Petri Net (CPN) models, and by running simulations on the constructed models, different types of evidence are gathered and documented that support the found weaknesses in the LoRaWAN specification. All selected attacks are based on a combination of eavesdropping and replay replaying specific network frames.
This research study has resulted in two new findings (a+b), and a detailed analysis of the application of an earlier v1.0 finding (c) on the LoRaWAN v1.0.2 specification.

7.2 CONTRIBUTIONS

Since there are still not many publications related to vulnerabilities in the LoRaWAN specification, this study adds relevant information to the field of LoRaWAN security. The results of this research study include detailed analysis, formal CPN models, and simulation results related

to three findings in the LoRaWAN v1.0.2 specification. Additionally, for all findings, the results of a brief assessment against the LoRaWAN v1.1 specification are included.

The detailed results of this research study can be used for improving LoRaWAN protocol security by the LoRa Alliance. Furthermore, the results of this research study contribute to knowledge development within the academic field of LoRaWAN security. Other researchers can use these results as input, or reference, for other research studies. The results can also be used by others to be aware of several weaknesses in the LoRaWAN protocol specification, e.g. for implementation designs and considerations.

Because of the lack of information available related to LoRaWAN security, I have started this research study from zero. I have obtained the latest LoRaWAN version available (v.1.0.2), made a selection of attack (methods), and started the first analysis phases in which I tried to apply the selected attacks to the LoRaWAN specification. I did find three vulnerabilities myself: Beaconing, Downlink routing, and End-Device activation.

During the first analysis phase, I found out that the Join-Accept replay in the End-Device activation vulnerability is also described by Zulian, S. [41], for the LoRaWAN specification v1.0, as part of his OTAA analysis (mainly the Join-Request message). I only focused on the Join-Accept replay vulnerability, specifically for the LoRaWAN specifications v1.0.2 and v1.1.

While performing the modeling and validation activities, I found out that Yang, X. [40] has published a thesis (Jul 2017) in which the beaconing vulnerability was described as well. Also, Avoine, G. published a paper (Jun 2017), in which the End-device activation vulnerability is described for LoRaWAN specification v1.0.

7.3 DISCUSSION AND RELATION TO SIMILAR WORK

For this research study, the scope is narrowed to the LoRaWAN (v1.0.2) specification only. This specification is for the large part defined in natural language, and can sometimes be interpreted in different ways. Also, a part of the specification does not provide sufficient detail. The analysis performed in this research study is based on my interpretation of the specification.

The formal CPN models that are constructed, are the result of a manual process. Because the specification is in natural language, it is not possible to translate (parts) of the specification into a formal

CPN model automatically. The models are created after an analysis of known attack methods against the LoRaWAN specification. Both the analysis, and the model construction, are done by the same person. There might be a chance that during model construction the person was biased by the first analysis results.

Because the first LoRaWAN specification is from October 2015 only, not a lot of related work is available. During this research study, there have been a few publications related to LoRaWAN security. The most relevant works are from Zulian, S. [41], Avoine, G. [23], and Yang, X. [40]. They also have been studying LoRaWAN in detail for specific vulnerabilities.

The results of Zulian [41] were published in October 2016, and are based on the LoRaWAN specification v1.0. This research includes an issue related to the DevNonce usage in the OTAA procedure, the possibility of a Join-Request/Join-Accept replay attack, and the Random Number Generator (RNG) of the Semtech SX1772 hardware that is used in most LoRaWAN hardware. Zulian has used a less formal analysis, without describing a research method, to analyze the OTAA, specifically the use of the DevNonce value. In his analysis, he also describes the possibility of replaying a Join-Accept message. The outcome of his analysis, of the Join-Accept message weakness, is similar to mine. The difference is mainly that this thesis contains much more details on the analysis, and the confirmation of the weakness by using a CPN model.

In June 2017, a paper from Avoine [23] is published, based on the LoRaWAN specification v1.0. This paper describes several LoRaWAN security issues based on the LoRaWAN specification v1.0. Main subjects are: enforcement of re-using session keys, replay of Join-Request/Join-Accept messages, and mis-usage of shared root keys. This paper does not describe the used research method, and uses natural language only to describe the possible replay of Join-Accept messages. The conclusion is comparable to the end-device activation vulnerability described in this thesis.

Yang's results [40] were published in July 2017, and are based on the LoRaWAN specification v1.0.2. The research of Yang includes details for a replay attack of the ABP procedure (based on key re-use and frame counter resets), an eavesdropping attack for a cryptographic weakness (several important prerequisites are required to succeed), a bit flipping attack (on the communication between network server and application server), and an ACK spoofing attack (there is no indication which message is confirmed), and a description of the beaconing weakness (similar to the findings in this research study). The document of Yang does not describe the used research method, it does

include a description of several possible vulnerabilities in natural language, and includes results of practical analysis using a working environment. Yang's conclusion on the beaconing weakness is similar to the beaconing conclusion in this research study.

This research study has resulted into three vulnerabilities. Unfortunately, time constraints did not permit a detailed analysis into possible mitigations for these vulnerabilities. The end-device activation vulnerability has been mitigated in the LoRaWAN v1.1 specification by using an incrementing JoinNonce value to prevent replay attacks of Join-Accept messages. For the beaconing and downlink routing vulnerabilities, no mitigation exists at the moment. The beaconing vulnerability may be mitigated by signing the beaconing messages. However, this requires a (shared) key between gateways (the network) and all end-devices. Other generic network control communications could also benefit from such a shared key. The downlink routing vulnerability may be mitigated by using a confirmation-of-change by the end-device. The confirmation request must be send by using both the old and new gateway so that it certainly will reach the end-device.

7.4 FUTURE WORK

In the limited time that was available during this research study, not all aspects of the LoRaWAN specification could be included. Topics that can be considered for future work include Activation by Personalization (ABP), radio level specifications such as frequency hopping, Adaptive Data Rate (ADR), the practical meaning of every single MAC command, multi-cast traffic, and differences between different regions.

In the beaconing vulnerability, it was also shown that besides the timing reference, also the latitude and longitude values of the gateway can be modified. It would be interesting to see which practical use there is for this vulnerability. Can it lead to battery drain because of many updates towards the network server, can applications be misled, etcetera?

Additional analysis is required to determine possible mitigations for the beaconing and downlink routing vulnerabilities in more detail. The end-device activation vulnerability has been mitigated in the LoRaWAN v1.1 specification.

This research study is limited to the LoRaWAN specification only. In case of insufficient detail in the specification, the LoRaWAN reference implementation[1] could be studied to determine which implementa-

1 https://github.com/Lora-net

tion decisions have been made by the LoRa Alliance. This to analyze gaps between insufficient detail in the specification and the reference implementation.

In October 2017, a new release (v1.1) of the LoRaWAN specification was published. This release contains some significant changes in the area of security. This research study only analyzed those parts of v1.1 that are relevant for the vulnerabilities that are found. It would be interesting to see the results of a full analysis of the v1.1 specification (redo this entire research study, but now based on v1.1 instead of v1.0.2).

7.5 REFLECTION

Working on this research study has taken quite a long time with many ups and a few downs. I am satisfied with the created research model, which I have based on the method of Verschuren and Doorewaard (see chapter 2.1). I have used the research model to define phases for this research study, and within the phases I have defined activities and milestones. The phases, activities, and milestones have resulted into an overall planning. There was no need to make major changes to the research model and phases during the research study. Some deviations have been made in respect to defined actions and milestones, based on output of preceding phases. On several occasions, I had to make changes to the overall planning because of the effects of a traffic accident during the preparation phase. Fortunately, I could make up all lost time (and even more) during the last three months of 2017 when my employer allowed me to spend an amount of business hours on this research study.

I have learned many things during this research study, of which the most important are: the application of a research model, planning and structure is even more important for a long running activity, keep your goal in mind and make sure you don't deviate too much (enough-is-enough), keep the correct amount of detail in the thesis document (don't try to include all information and results), and many details on LoRaWAN, which I was not familiar with before starting this research study.

Before the start of this research study it was difficult to define some expectations. The LoRaWAN specification was still fairly recent, and very little other work was available. The expectation was that no weakness would be found because the LoRaWAN specification from the LoRa Alliance is backed by some large corporations and universities. In the area of confidentiality and integrity this turned out to be true. Despite the LoRaWAN being developed by the LoRa Alliance, a

consortium of organizations and Universities, I was able to discover several vulnerabilities in the area of availability. This was an unexpected result, because I did not expect to find any serious vulnerabilities in a product developed by such a consortium.

Two main objectives are defined for this research study. The first one is that it may contribute to the security of the LoRaWAN protocol specification. Now that several vulnerabilities have been found, the results must be shared with the LoRa Alliance. These results may help in improving the LoRaWAN specification standard. The second objective is to contribute to the scientific research field of LoRaWAN security. By publishing the results of this research study, and possibly writing an academic paper, new scientific information on LoRaWAN security will be made available to others.

[5] Raja Waseem Anwar, Majid Bakhtiari, Anazida Zainal, Abdul Hanan Abdullah, and Kashif Naseer Qureshi. "Security issues and attacks in wireless sensor network." In: *World Applied Sciences Journal* 30.10 (2014), pp. 1224–1227. ISSN: 19916426. DOI: 10.5829/idosi.wasj.2014.30.10.334.

[6] Emekcan Aras, Gowri Sankar Ramachandran, Piers Lawrence, and Danny Hughes. "Exploring the security vulnerabilities of LoRa." In: *2017 3rd IEEE International Conference on Cybernetics, CYBCONF 2017 - Proceedings* (2017). DOI: 10.1109/CYBConf.2017.7985777. URL: https://lirias.kuleuven.be/bitstream/123456789/587540/1/camera_ready.pdf.

[7] Ole Martin Dahl and Stephen D Wolthusen. "Modeling and Execution of Complex Attack Scenarios using Interval Timed Colored Petri Nets." In: *Information Assurance* (2006). DOI: 10.1109/IWIA.2006.17. URL: http://ieeexplore.ieee.org/document/1610008/.

[12] Sunil Ghildiyal, Amit Kumar Mishra, Ashish Gupta, and Neha Garg. "Analysis of Denial Of Service (DOS) attacks in Wireless Sensor Networks." In: *International Journal of Research in Engineering and Technology* 03.22 (2014), pp. 140–143. ISSN: 23217308. DOI: 10.15623/ijret.2014.0322030. URL: http://esatjournals.net/ijret/2014v03/i22/IJRET20140322030.pdf.

[14] Simon Heron. "Advanced Encryption Standard (AES)." In: *Network Security* 2009.12 (2009), pp. 8–12. ISSN: 13534858. DOI: 10.1016/S1353-4858(10)70006-4. URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[16] L.M. Kristensen: J.C.A. de Figueiredo. "Using Coloured Petri Nets to Investigate Behavioural and Performance Issues of TCP Protocols." In: *2nd Workshop on Practical Use of Coloured Petri Nets and Design/CPN* (1999), pp. 21–40. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.6691.

[18] Chris Karlof and David Wagner. "Secure routing in wireless sensor networks: Attacks and countermeasures." In: *Ad Hoc Networks* 1.2-3 (2003), pp. 293–315. ISSN: 15708705. DOI: 10.1016/S1570-8705(03)00008-8.

[23] Rescuing Lorawan and Gildas Avoine. "Rescuing LoRaWAN 1.0." In: *N.A.* (2016). URL: https://eprint.iacr.org/2017/651.

[25] David a McGrew. "Counter Mode Security : Analysis and Recommendations." In: *N.A.* (2002). URL: http://cr.yp.to/bib/2002/mcgrew.pdf.

[27] Robert Milner, Mads Tofte, Robert Harper, and David MacQueen. "The Definition of Standard ML (Revised)." In: *N.A.* (1997). URL: http://www.amazon.com/Definition-Standard-ML-Revised/dp/0262631814.

[30] Gonzalez Nieto, Suratose Tritilanunt, Colin Boyd, Ernest Foo, and Juan Manuel Gonz. "Using Coloured Petri Nets to Simulate DoS-resistant protocols." In: *N.A.* (2006). URL: https://eprints.qut.edu.au/23982/.

[31] C Ouyang, B Eng, and M Eng. "Formal Specification and Verification of the Internet Open Trading Protocol using Coloured Petri Nets." In: June (2004). URL: https://link.springer.com/chapter/10.1007/978-3-540-30233-9_1.

[32] Dr. G. Padmavathi and Mrs. D. Shanmugapriya. "A Survey of Attacks, Security Mechanisms and Challenges in Wireless Sensor Networks." In: *International Journal of Computer Science and Information Security (IJCSIS)* 4.1 (2009), pp. 1–9. arXiv: 0909.0576. URL: http://arxiv.org/abs/0909.0576.

[33] Al-Sakib Khan Pathan, Hyung-Woo Lee Lee, and Choong Seon Hong. "Security in wireless sensor networks: issues and challenges." In: *8th International Conference Advanced Communication Technology* 2 (2006), pp. 1043–1048. DOI: 10.1109/ICACT.2006.206151. arXiv: 0712.4169. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1625756.

[34] P.L.P. Petter and S. Sawicki. "A comparison of Petri Net simulation tools." In: *N.A.* (2017). URL: https://www.publicacoeseventos.unijui.edu.br/index.php/salaoconhecimento/article/download/8273/6999.

[35] David R. Raymond and Scott F. Midkiff. "Denial-of-service in wireless sensor networks: Attacks and Defenses." In: *IEEE Pervasive Computing* 7.1 (2008), pp. 74–81. ISSN: 15361268. DOI: 10.1109/MPRV.2008.6. URL: https://www.researchgate.net/profile/David_Raymond3/publication/220299881_Denial-of-Service_in_Wireless_Sensor_Networks_Attacks_and_Defenses/links/54e729200cf2cd2e02912f72.pdf.

[36] Phillip Rogaway. "Evaluation of Some Blockcipher Modes of Operation." In: *N.A.* (2011). URL: http://web.cs.ucdavis.edu/$\sim$erogaway/papers/modes.pdf.

[37] A Valmari. "The state explosion problem." In: *N.A.* (1998). URL: http://www.springerlink.com.ezproxy.elib11.ub.unimaas.nl/index/84354378413L5128.pdf.

[38] Yong Wang, Garhan Attebury, Byrav Ramamurthy, and Nebraska -Lincoln Wang. "A Survey of Security Issues In Wireless Sensor Networks." In: *N.A.* (2006). URL: http://digitalcommons.

unl.edu/csearticles%5Cnhttp://digitalcommons.unl.edu/csearticles/84.

[39] Haiping Xu, Mihir Ayachit, and Abhinay Reddyreddy. "Formal modelling and analysis of XML firewall for service-oriented systems." In: *International Journal of Security and Networks* 3.3 (2008), p. 147. ISSN: 1747-8405. DOI: 10.1504/IJSN.2008.020089. URL: https://www.researchgate.net/profile/Haiping_Xu2/publication/220080769_Formal_modelling_and_analysis_of_XML_firewall_for_service-oriented_systems/links/02bfe51224ff409070000000.pdf.

NON-ACADEMIC REFERENCES

[1] Wil Van Der Aalst. *Analysis of Process Models: Introduction, state space analysis and simulation in CPN Tools*. 2011. URL: http://cpntools.org/_media/book/analysis.pdf%5Cnpapers3://publication/uuid/835F7327-5F4F-4F0D-9B2A-963123E430E8.

[2] Wil Van Aalst. *CPN A concrete language for high-level Petri nets*. 2011. URL: papers3://publication/uuid/2351C816-B7F9-4490-AF53-6CD1BDE1732F.

[3] Wil van der Aalst and Christian Stahl. *Modeling business processes*. London, England: The MIT Press, 2011.

[4] LoRa Alliance. *LoRaWAN™ 1.1 Regional Parameters*. 2011. URL: https://www.lora-alliance.org/.

[8] ENISA. *ENISA Threat Landscape Report 2017 15*. Tech. rep. January. 2018. DOI: 10.2824/967192. URL: https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2017.

[9] Europol. *Internet Organised Crime Threat Assessment (IOCTA) 2017*. Tech. rep. 2017. DOI: 10.2813/55735. URL: https://www.europol.europa.eu/iocta/2017/index.html.

[10] Fortinet Labs. *Threat Landscape 2017*. Tech. rep. 2017. URL: https://www.fortinet.com/content/dam/fortinet/assets/white-papers/Threat-Report-Q3-2017.pdf.

[11] Gemalto. *Low Power, Wide Area Networks security*. 2016. URL: https://docbox.etsi.org/Workshop/2015/201512-M2MWORKSHOP/S04-WirelessTechnoforIoTandSecurityChallenges/GEMALTO-GIRARD.pdf.

[13] Mallikarjun Hangargi. *Denial of Service Attacks in Wireless Networks*. 2014. URL: https://dl.packetstormsecurity.net/papers/wireless/DoS_attacks_in_wireless_networks.pdf.

[15] IEEE Computer Society. *802.15.4-2015 - IEEE Standard for Low-Rate Wireless Personal Area Networks (WPANs)*. Vol. 2015. 2016, pp. 1–709. ISBN: VO -. DOI: 10.1109/IEEESTD.2016.7460875. arXiv: IEEESTD.2016.7460875 [10.1109].

[17] T. Iwata JH. Song, R. Poovendran, J. Lee. *The AES-CMAC Algorithm Status*. 2006. URL: https://tools.ietf.org/pdf/rfc4493.pdf.

[19] LinkLabs. *Low Power, Wide Area Networks*. 2016. URL: http://www.link-labs.com/resources/.

[20] LoRa Alliance. *LoRaWAN ™ Specification V1.0.2*. 2016. URL: https://www.lora-alliance.org/.

[21] LoRa Alliance. *LoRaWAN ™ v1.0 Regional Parameters*. 2016. URL: https://www.lora-alliance.org/.

[22] LoRa Alliance. *LoRaWAN™ 1.1 Specification*. 2017. URL: https://www.lora-alliance.org/.

[24] McAfee Labs. *McAfee Labs Threats Report*. Tech. rep. September. 2017. URL: https://www.mcafee.com/us/resources/reports/rp-quarterly-threats-jun-2017.pdf.

[26] Robert Miller. *LoRa Security - Building a Secure LoRa Solution*. 2015. URL: https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-LoRa-security-guide-1.2-2016-03-22.pdf.

[28] Nationaal Coordinator Terrorismebestijding en Veiligheid. *Cybersecuritybeeld Nederland 2017*. Tech. rep. 2017. URL: https://www.ncsc.nl/binaries/content/documents/ncsc-nl/actueel/cybersecuritybeeld-nederland/cybersecuritybeeld-nederland-2017/1/CSBN2017.pdf.

[29] National Institute of Standards and Technology (NIST). *Announcing the advanced encryption standard (AES)*. 2001. DOI: 10.1016/S1353-4858(10)70006-4. URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[40] Xueying Yang. "LoRaWAN: Vulnerability Analysis and Practical Exploitation." PhD thesis. 2017. URL: https://pdfs.semanticscholar.org/a1e3/9d0f249a1afa2f5ade9d5473b3e64a0e84fe.pdf.

[41] Simone Zulian. "Security threat analysis and countermeasures for LoRaWANTM join procedure." PhD thesis. 2016. URL: http://tesi.cab.unipd.it/53210/1/zulian_simone_tesi.pdf.

ABBREVIATIONS

ABP	Activation by Personalization
ADR	Adaptive Data Rate
ADT	Attack-Defense Tree
AES	Advanced Encryption Standard
AppEUI	Application Identifier
AppKey	Application Key
AppSKey	Application Session Key
CMAC	Cipher-based Message Authentication Code
CPN	Colored Petri Net
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CTR	Counter
DOS	Denial of Service
DDOS	Distributed Denial of Service
(D)DOS	(Distributed) Denial of Service
DevAddr	End-device Address
DevEUI	End-device Identifier
ECB	Electronic Code Book
EUI	Extended Unique Identifiers
GPS	Global Positioning System
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet-of-Things
LPWAN	Low-Power Wide Area Network
LoRaWAN	Long Range Wide Area Network
MAC	Media Access Control
MIC	Message Integrity Code

MitM	Man-in-the-Middle
NetID	Network Identifier
NIC	Network Interface Card
NIST	National Institute of Standards and Technology
NwkAddr	Network Address
NwkID	Network Identifier
NwkSKey	Network Session Key
OTAA	Over-the-Air-Activation
PII	Personally Identifiable Information
PSK	Pre-Shared Key
RF	Radio Frequency
RNG	Random Number Generator
SML	Standard Meta Language
TLS	Transport Layer Security
WSN	Wireless Sensor Network

APPENDIX A - COMMUNICATION SCENARIOS

This Appendix contains different possible communication scenarios that have been identified in a LoRaWAN environment. These scenarios have been worked out into such a detail that network frames have been identified and described on a bit level.

The values used in these scenarios (such as AppKey, AppEUI, DevEUI, DevAddr, AppSKey, and NwkSKey) have been used from an existing[1] LoRaWAN implementation. This is to make sure the studied scenarios are as close to an actual implementation as possible.

The next paragraphs describe different communication scenarios by using a figure that represents the data flow and a table that represents the used network frame. It should be noted that multi-octet field values (except for the MIC field value) are transmitted as Little Endian (bits are ordered from the Least Significant Bit). This difference is shown between the 'Field Values' and the 'Transmitted Data' rows of the network frame table.

A.1 END-DEVICE ACTIVATION

For the Over-the-Air-Activation (OTAA) activation of an end-device two types of network communications are required. First, the request from the end-device to the network server to join the network (Join-Request). Second, when accepted, an accept response from the network server to the end-device (Join-Accept).

A.1.1 *Join-Request*

The Join-Request is a single network frame that is send by the end-device towards the network server (figure A.1).

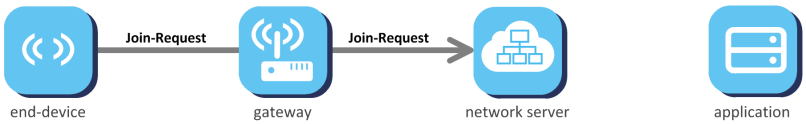

Figure A.1: Communication Flow: Join-Request

The Join-Request frame is shown in figure A.2. Important field values are oxoo specifying a Join-Request frame type in the MType field,

1 https://www.thethingsnetwork.org/

the AppEUI and DevEUI values which are provided by the network operator, a random DevNonce value, and a Message Integrity Code (MIC).

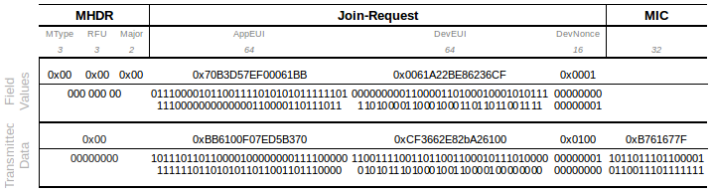

Figure A.2: Network Frame: Join-Request

A.1.2 *Join-Accept*

The Join-Accept is a single network frame that is send by the network server towards the end-device (figure A.3). The Join-Request frame is only send when the end-device is accepted, otherwise no response is send.

Figure A.3: Network Flow: Join-Accept

The Join-Request frame is shown in figure A.4. Important field values are 0x01 specifying a Join-Accept frame type in the MType field, and the remaining values which are provided by the network operator. The CFList is optional. As for almost all frame types, a MIC is included as well. The complete Join-Accept part of the message is encrypted using the AppKey.

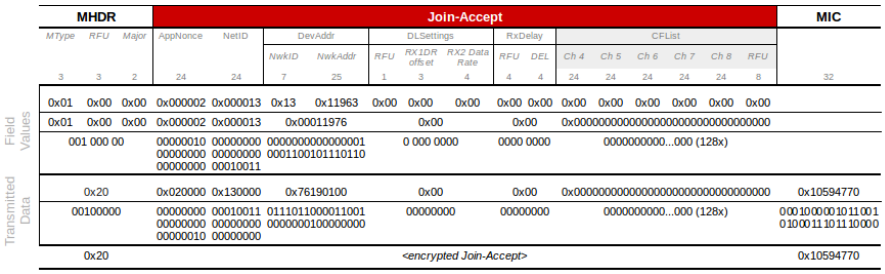

Figure A.4: Network Frame: Join-Accept

A.2 SENDING DATA

Within a LoRaWAN data can be send in different ways: from end-device to application, from application to end-device, confirmed or unconfirmed messages (figure A.5). Despite these differences the used net-

work frame is no different, only some of the field values differ, such as the message type (MType).

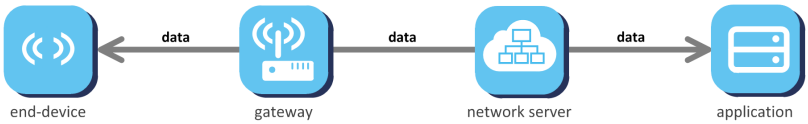

Figure A.5: Network Flow: Sending data

Figure A.6 shows the network frame for sending data. Important field values are the MType (which specifies the frame type), DevAddr which specifies the end-device (as source or destination), Frame Control Options, a frame counter, and the FRMPayload which includes the encrypted payload.

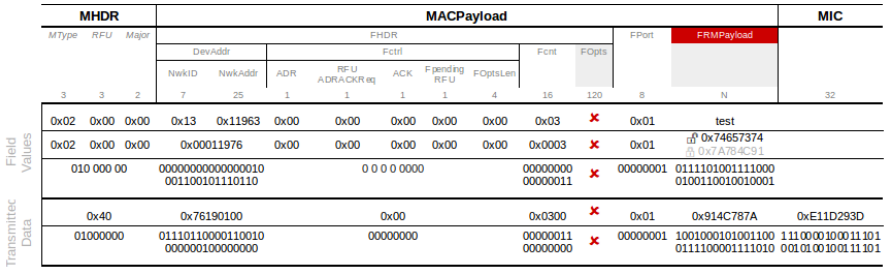

Figure A.6: Network Frame: Sending data

A.3 SENDING MAC COMMANDS

Next to communicating application data over a LoRaWAN, it is also possible to communicate network commands (Media Access Control (MAC) commands). These MAC commands are used for controlling the network, such as speed, frequency, and status. The MAC commands are communicated between end-devices and the gateway / network server. Table A.1 provides an overview of all Class A and Class B MAC commands.

Table A.1: MAC commands

Class	Commands	Remark
A	LinkCheckReq LinkCheckAns	End-device checking the Link connection. The answer provided by the gateway contains signal power so that quality of reception can be determined.
A	LinkADDRReq LinkADDRAns	Request from gateway to end-device to change data rate, transmit power, repetition rate or channel.
A	DutyCycleReq DutyCycleAns	Sets the maximum aggregated transmit duty-cycle of an end-device.

MAC commands – continued

Class	Commands	Remark
A	RXParamSetupReq RXParamSetupAns	Sets the reception slots parameters of an end-device.
A	DevStatsReq DevStatsAns	Requests the status (battery level and its demodulation margin) of the end-device.
A	NewChannelReq NewChannelAns	Creates or modifies the definition of a radio channel.
A	RXTimingSetupReq RXTimingSetupAns	Sets the timing of the of the reception slots of an end-device.
A	TxParamSetupReq TxParamSetupAns	Used by the network server to set the maximum allowed dwell time and Max EIRP of an end-device.
A	DlChannelReq DlChannelAns	Modifies the definition of a downlink RX1 radio channel by shifting the downlink frequency from the uplink frequencies (i.e. creating an asymmetric channel)
B	PingSlotInfoReq PingSlotInfoAns	Used by the end-device to communicate the ping unicast slot data rate and periodicity to the network server.
B	PingSlotChannelReq PingSlotFreqAns	Used by the network server to set the unicast ping channel of an end-device.
B	BeaconTimingReq BeaconTimingAns	Used by end-device to request next beacon timing & channel to network.
B	BeaconFreqReq BeaconFreqAns	Command used by the network server to modify the frequency at which the end-device expects to receive beacon broadcast.

MAC commands can be communicated in two different ways: using a dedicated network frame, or by piggy-backing the MAC command with the next data transmission.

A.3.1 *Dedicated*

Sending dedicated MAC commands is done in a single network frame that is transmitted between end-devices and the network server (figure A.7).

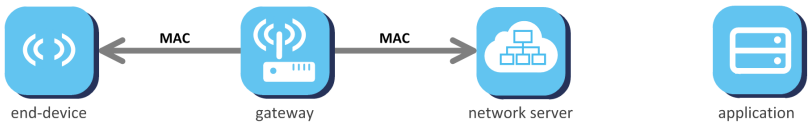

Figure A.7: Network Flow: Sending MAC (dedicated)

The network frame used for sending MAC commands is similar to the network frame for sending data across the network. The MAC commands are stored in the FRMPayload field, which is also used for data transmissions. The value of this field is always encrypted by default! When a dedicated network frame is used for transmitting MAC commands, then the FPort field value should be 0x00.

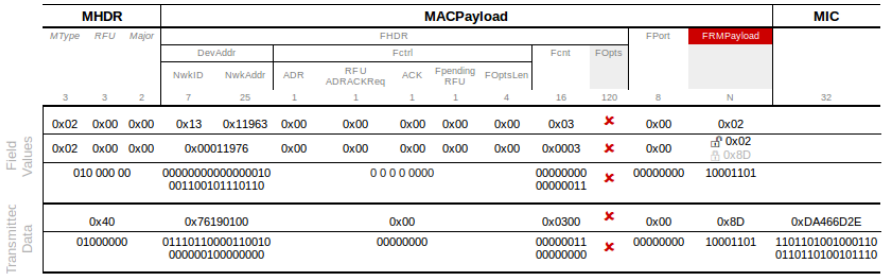

Figure A.8: Network Frame: Sending MAC (dedicated)

A.3.2 *Piggy-backed*

Sending piggy-backed MAC commands is done inside a single network frame that is transmitted between end-devices and the network server / application (figure A.9).

Figure A.9: Network Flow: Sending MAC (piggy-backed)

Instead of using a dedicated network frame for the transmission of MAC commands, it is also possible to include MAC commands into a network frame that is used for a data transfer. The MAC commands are stored in the FOpts field. This field is always unencrypted!

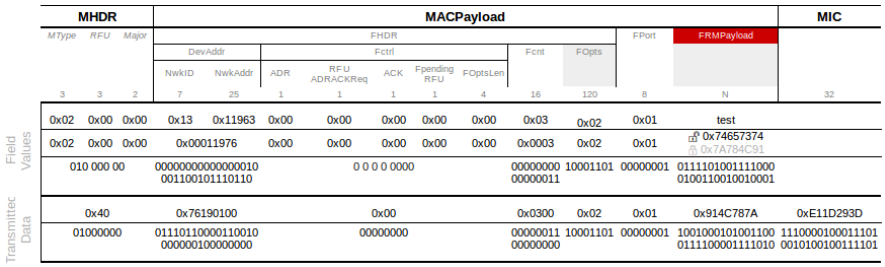

Figure A.10: Network Frame: Sending MAC (piggy-backed)

A.4 MULTI-CAST MESSAGES

The LoRaWAN specification includes multi-cast messages. Multi-cast messages allow the application or network server to send data or MAC commands to multiple end-devices at once (figure A.11). The used network frame is identical to regular transmission of data. All devices of a multi-cast group must share the same DevAddr and associated encryption keys.

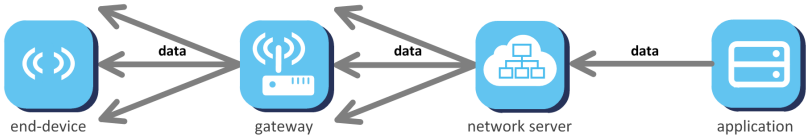

Figure A.11: Network Flow: Sending Data (multi-cast)

The used network frame for multi-cast traffic is identical to that of the network frame of a regular data transmission.

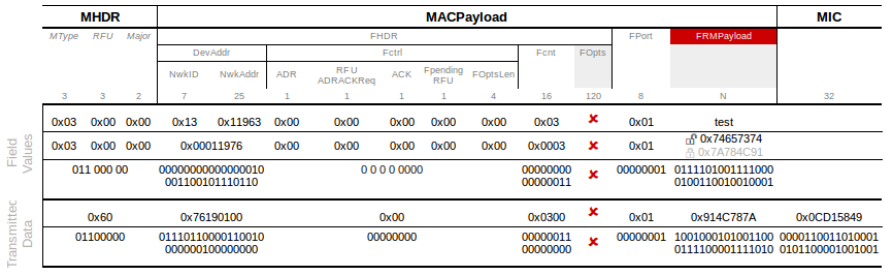

Figure A.12: Network Frame: Sending Data (multi-cast)

A.5 BEACONING

All gateways participate in providing a time-synchronization mechanism by sending beacons at regular fixed intervals configurable per network (figure A.13). By default a beacon is sent every 128s since 1 January 1970 00:00:00 UTC + NwkID + TBeaconDelay. TBeaconDelay is a network specific delay between 0ms and 50ms to allow a slight transmission delay of the gateways (to reduce influence the impact of gateways from other LoRaWAN implementations). This time synchronization is required to synchronize receive slots between end-devices and the network.

Besides timing, the beacon also provides the geographical location of the gateway by including its GPS coordinates. A beacon contains a Time field value which represents the number of seconds since 00:00:00 Coordinated Universal Time (UTC), 1 January 1970. The GwSpecific field value contains the GPS coordinates of the gateway. This gateway location information is for mobile end-devices to detect move-

ment and update the network server downlink routing.

Figure A.13: Network Flow: Beaconing (ping)

For beaconing a dedicated network frame is used. The frame does not provide confidentiality (encryption) or integrity (MIC). The network frame does include a Cyclic Redundancy Check (CRC) for both the timing information and the gateway information.

	NetID	Time	CRC	GWSpecific			RFU	CRC
				InfoDesc	Info			
					Lat	Lng		
	24	32	8 (16)	8	24	24	0 (8)	16
Field Values	0x000013	0x59689120	0xE1	0x00	0x002001	0x038100	✖	0x55DE
	000000000000 0000000010011	01011001011010001001000100100000	11100001	00000000	000000000010 000000000001	000000111000 000100000000	✖	01010101 11011110
Transmittec Data	0x130000	0x20916859	0xE1	0x00	0x012000	0x008103	✖	0xDE55
	000100110000 000000000000	00100000100100011011001	11100001	00000000	000000010010 000000000000	000000001000 000100000011	✖	11011110 01010101

Figure A.14: Network Frame: Beaconing (ping)

B

APPENDIX B - REPLAY SCENARIOS

This Appendix contains detailed Replay Attack scenarios which are based on the communication scenarios which are described in Appendix A.

Many of these replay scenarios include changing the value of the MIC field. Changing the value of the MIC field is very difficult (to impossible) because it requires a brute-force approach. More details on brute-fording the MIC value is described in chapter 5.4.

B.1 END-DEVICE ACTIVATION

For the Over-the-Air-Activation (OTAA) activation of an end-device two types of network communications are required. First, the request from the end-device to the network server to join the network (Join-Request). Second, when accepted, an accept response from the network server to the end-device (Join-Accept).

Figure B.1 shows the different type of sequences during the lifetime of an -end-device. These sequence will be used in the remainder of this chapter.

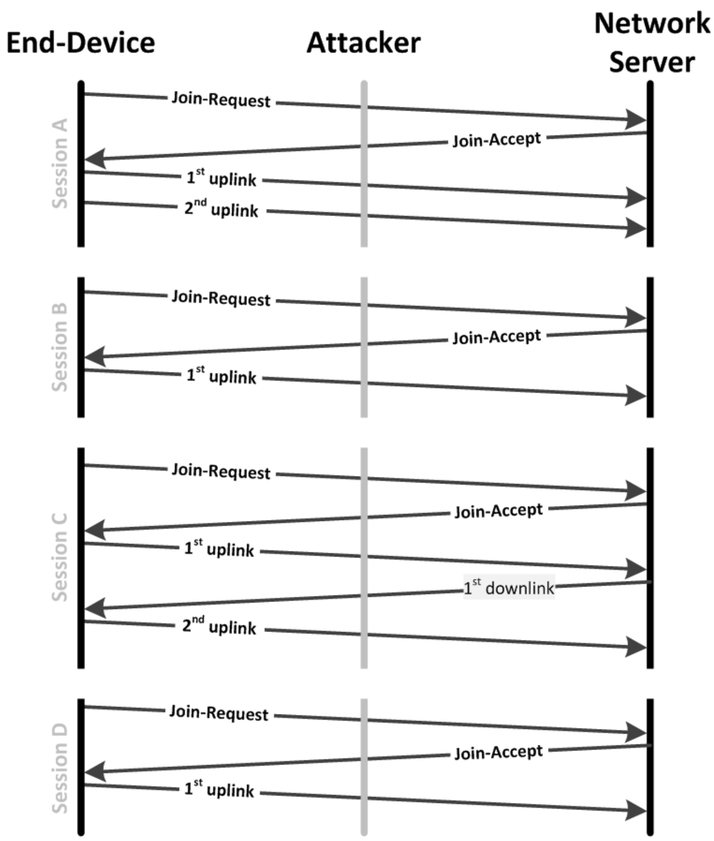

Figure B.1: Join-Request: Replay scenario 1

B.1.1 *Join-Request*

In the first Join-Request replay scenario (figure B.2), the complete network frame is replayed as-is. None of the field values are changed. The network frame is valid and accepted by the network server, however, the Join-Request message will be considered invalid because an end-device with the same details (AppEUI + DevEUI + DevNonce) is already activated on the network.

	MHDR			Join-Request			MIC
	MType 3	RFU 3	Major 2	AppEUI 64	DevEUI 64	DevNonce 16	32
Field Values	0x00	0x00	0x00	0x70B3D57EF00061BB	0x0061A22BE86236CF	0x0001	
	000 000 00			01110000101100111101010101111111101 111000000000000000110000110111011	00000000011000011010001000101011 11010000110001000110110110011111	00000000 00000001	
Transmitted Data	0x00			0xBB6100F07ED5B370	0xCF3662E82bA26100	0x0100	0xB761677F
	00000000			10111011011000010000000011110000 01111110110101011011001101110000	11001111001101100110001011101000 01010111010001001100001000000000	00000001 00000000	1011011101100001 0110011101111111

Figure B.2: Join-Request: Replay scenario 1

To create a Join-Request that represents a re-activation of an existing end-device (figure B.3), the DevNonce must be changed. When an existing end-device sends a new Join-Request message with a different DevNonce value, it can be considered as a new activation request of

the same end-device. This time the network server will discard the network frame because the MIC will fail.

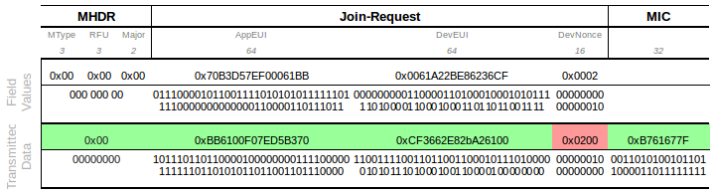

Figure B.3: Join-Request: Replay scenario 2

The third Join-Request replay scenario (figure B.4) contains two modified field values. The DevNonce as described above, and the MIC field value has been changed to make sure that the network frame is accepted by the network server.

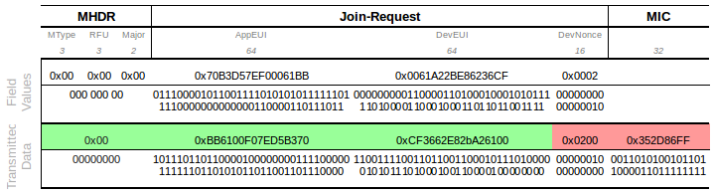

Figure B.4: Join-Request: Replay scenario 3

When a Join-Request can successful be replayed with new DevNonce values, it could lead to a form of Denial of Service for the end-device. At this moment it is not possible to generate a valid MIC field value. The AES algorithm in combination with the 128-bits security keys makes this impossible with todays computing power (see chapter 5.4).

Changing the values of the fields listed in table B.1 does not result into unexpected behavior (based on the LoRaWAN specification).

Table B.1: *Join-Request* fields excluded

Data field	Remark
MType	Must be 00 for a Join-Request message. Changing the value results in rejection of the network frame because it does not match any of the network frames for other message types.
RFU	RFU values should be 00. Impact of changing this value is depending on the LoRaWAN implementation.
Major	Must be 00 for LoRaWAN R1. Impact of changing this value is depending on the LoRaWAN implementation.

Join-Request fields excluded – continued

DATA FIELD	REMARK
AppEUI / DevEUI	This combination identifies a specific end-device. The combination is assigned (and thus known) by the network server before activation. Changing one, or both, values would represent a different end-device (if the combination is known). The network frame would be rejected because of a MIC failure.

B.1.2 *Join-Accept*

In the Join-Accept replay scenario, the complete network frame is replayed as-is (figure B.5). None of the field values are changed. It is not specified how an end-device should respond when a Join-Accept is received again after activation.

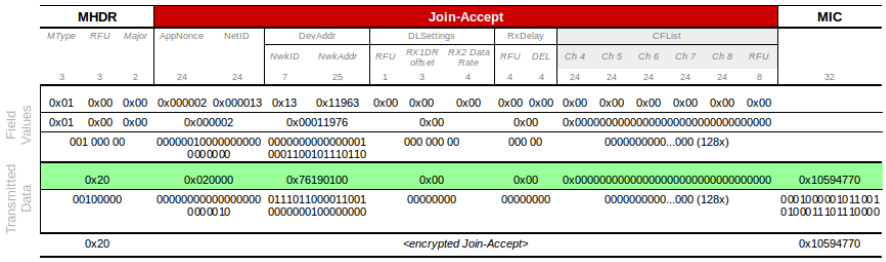

Figure B.5: Join-Accept: Replay scenario 1

It is impossible to change any other values in a Join-Accept message, because besides the MIC, the messages details are encrypted using the AppKey. Both the MIC and encryption use the AES algorithm in combination with the 128-bits security keys.

B.2 SENDING DATA

In the first Sending Data replay scenario (figure B.6), the complete network frame is replayed as-is. None of the field values are changed. The network frame is invalid and will be rejected by the network server because the frame counter (FCnt) field value will not contain the expected counter value. All end-devices and the Network server keep track of FcntUp (uplink) and FCntDown (downlink) frame counters. The sender of a network frame includes the correct counter in the FCnt field value. The use of up- and downlink counters make sure that involved parties know which value to expect when receiving new network frames, and thus preventing replay attacks by rejecting network frames with unexpected FCnt values.

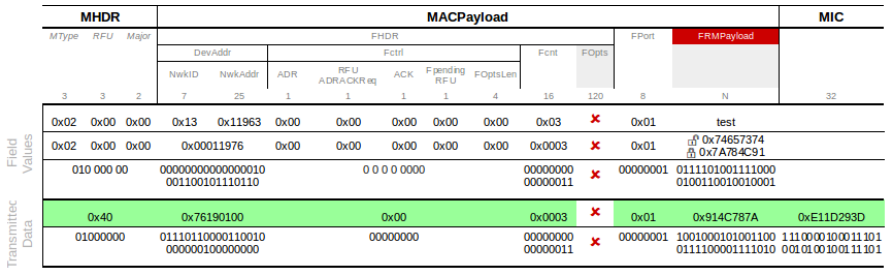

Figure B.6: Sending data: Replay scenario 1

To prevent the network server from discarding the network frame from scenario one, the frame counter (FCnt) must be modified to represent a valid value (figure B.7). A valid value can be determined by eavesdropping more traffic. When the FCnt field value is changed the network server will still discard the network frame because the MIC will fail. The FCnt field value is one of the inputs used for calculating the MIC value.

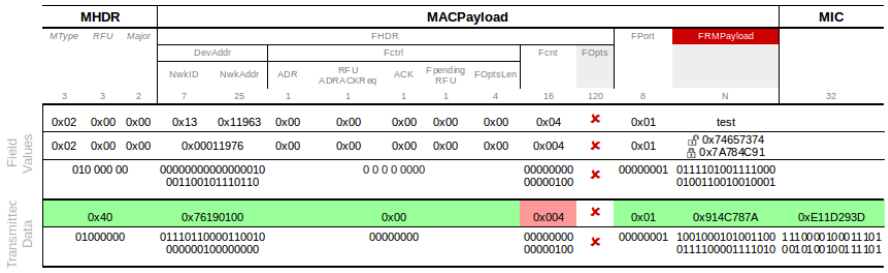

Figure B.7: Sending data: Replay scenario 2

The third Join-Request replay scenario (figure B.8) also contains a modified MIC field value to make sure that the network frame is accepted by the network server. In this case the network frame is successfully replayed and the data is provided to the application by the network server. At this moment it is not possible to generate a valid MIC field value. The AES algorithm in combination with the 128-bits security keys makes this impossible with todays computing power (see chapter 5.4).

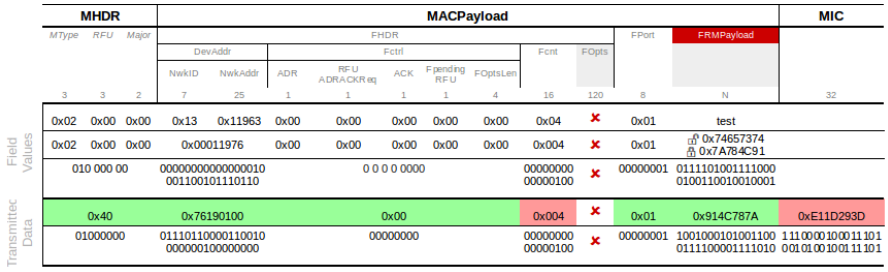

Figure B.8: Sending data: Replay scenario 3

For creating a valid network frame that is accepted by the network server, and that contains new data, it is required that also the FRM-Payload field is changed (figure B.9). However, the value in the FRM-Payload field is encrypted using the AppSKey. Without this key, it is not possible to change the content of the FRMPayload field. Chapter 5.4 contains more details about brute-forcing the AppSKey.

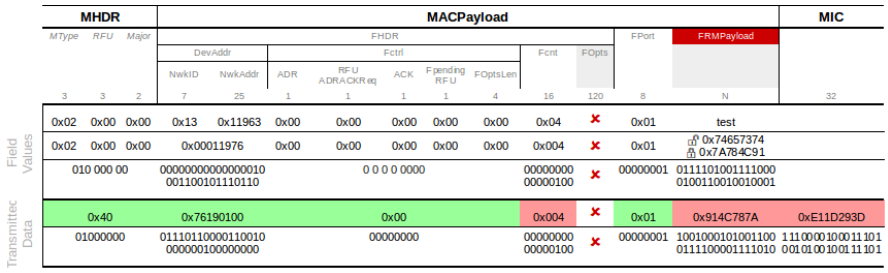

Figure B.9: Sending data: Replay scenario 4

Changing the values of the fields listed in table B.2 does not result into unexpected behavior (based on the LoRaWAN specification).

Table B.2: *Sending Data* fields excluded

Data field	Remark
MType	Must be 00 for a Join-Request message. Changing the value results in rejection of the network frame because it does not match any of the network frames for other message types.
RFU	RFU values should be 00. Impact of changing this value is depending on the LoRaWAN implementation.
Major	Must be 00 for LoRaWAN R1. Impact of changing this value is depending on the LoRaWAN implementation.
DevAddr	Changing the DevAddr field value would change the source/destination (depending on up- / downlink) network address of the end-device in the network frame. The network frame would be rejected by the network server, because the MIC will fail. The DevAddr field value is part of the MIC calculation. Better is to capture a network frame from the specific end-device for replay purposes.
FCtrl	The FCtrl field is composed of multiple values: Adaptive Data Rate (ADR), ACK, FPending, and FOptsLen. ADR requires a request and confirmation in both directions, traffic in both ways need to be replayed. The ACK is used for acknowledging different requests (e.g. confirmed messages and ADR). FPending indicates to an end-device there is more data to come, so that the end-device opens an additional receive window. FOptsLen specifies the length of the MAC commands in the FOpts field (if any).
FOpts	This field value is used for transmitting piggy-backed MAC commands. See chapter B.3.2 for more replay details on piggy-backed MAC commands.

Sending Data fields excluded – continued

Data field	Remark
FPort	The FPort field value is used to specific if there are MAC commands in the FRMPayload field (0), if there is application data in the FRMPayload field (1..223). Field value 224 is for LoRaWAN MAC layer testing purposes. Values 225..255 are reserved for future standardized application extensions. Impact of changing this value is depending on the LoRaWAN implementation.

B.3 SENDING MAC COMMANDS

The LoRaWAN specification describes two methods for transmitting MAC commands between end-devices and network server. The first method is by using a dedicated network frame. In this case, the MAC commands are the actual data payload in the FRMPayload field. The second method is by including (piggy-backing) MAC commands into a network frame that is created for sending application data.

B.3.1 *Dedicated*

Transmitting MAC commands using a dedicated network frame is similar to transmitting application data. Transmitting MAC commands in the FRMPayload field means that the MAC commands are encrypted by default as well.

Because the transmission of MAC commands is identical to sending data, the replay scenarios are identical as well. Therefore, the replay scenarios have not been repeated in this section.

B.3.2 *Piggy-backed*

When including (piggy-backing) MAC commands into a network frame used for sending application data, the MAC commands are added to the FOpts field. This field value is never encrypted.

In the first piggy-backed MAC command replay scenario (figure B.10), the complete network frame is replayed as-is. None of the field values are changed. The network frame is invalid and will be rejected by the network server because the frame counter (FCnt) field value will not contain the expected counter value.

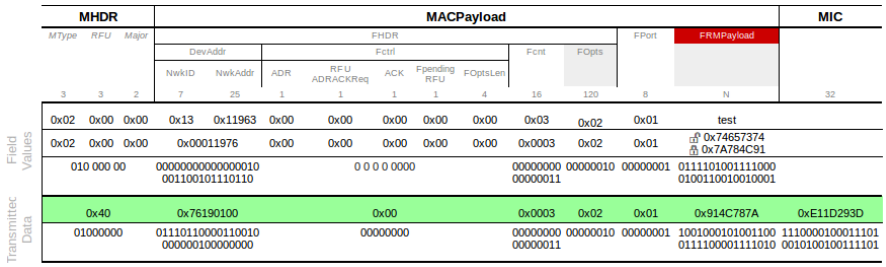

Figure B.10: Sending MAC: Replay scenario 1

To prevent the network server from discarding the network frame from scenario one, the frame counter (FCnt) must be modified to represent a valid value (figure B.11). A valid value can be determined by eavesdropping more traffic. When the FCnt field value is changed the network server will still discard the network frame because the MIC will fail. The FCnt field value is one of the inputs used for calculating the MIC value.

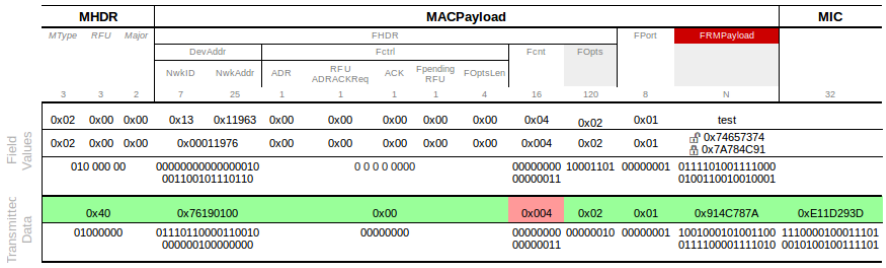

Figure B.11: Sending MAC: Replay scenario 2

The third Join-Request replay scenario (figure B.12) also contains a modified MIC field value to make sure that the network frame is accepted by the network server. In this case the network frame is successfully replayed and the data is provided to the application by the network server. At this moment it is not possible to generate a valid MIC field value. The AES algorithm in combination with the 128-bits security keys makes this impossible with todays computing power (see chapter 5.4).

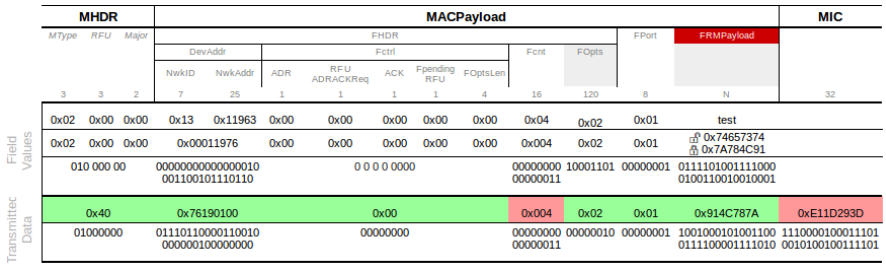

Figure B.12: Sending MAC: Replay scenario 3

When the third scenario can be realized, then it might be possible to transmit new MAC commands and get these accepted (figure B.13).

The FOpts field value contains plaintext MAC commands, so these can be changed as well.

A network frame is processed by different layers in the network stack (like the OSI model). In this scenario, the network frame is valid, the MIC is accepted so the network frame is processed by the MAC layer. In this layer the MAC commands will be processed. Only after that the network frame is being processed by the application layer which will process the re-transmitted application data. The response on processing duplicate application data is depending on the implementation. Even if processing of the application data fails, the MAC layer already finished processing the network frame successfully.

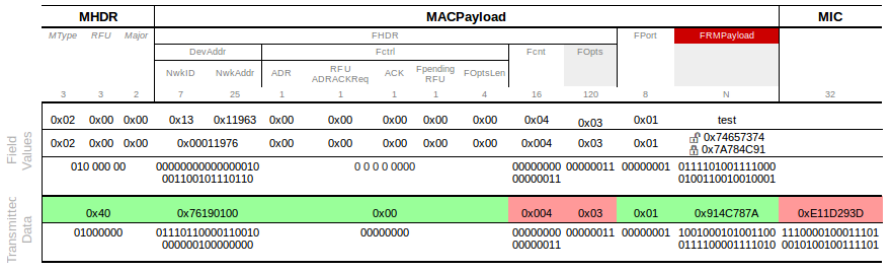

Figure B.13: Sending MAC: Replay scenario 4

Changing the values of other fields is described in the scenarios for Sending Data because the used network frame is identical.

B.4 MULTI-CAST MESSAGES

Transmitting multi-cast messages is like transmitting application data and dedicated MAC commands. The only differences are the used end-device addresses and encryption keys.

Because the transmission of multi-cast messages is identical to sending data, the replay scenarios are identical as well. Therefore, the replay scenarios have not been repeated in this section.

B.5 BEACONING

In the first Beaconing replay scenario (figure B.14), the complete network frame is replayed as-is. None of the field values are changed. The impact of this attack is difficult to determine. Replaying the beacon immediately will make sure the beacon is received in the beacon receive window. However, it will contain the same value as an earlier beacon which is received the same window. This will lead to the same receive slots and same GPS data. When the beacon is replayed at a later beacon receive window, then it is unknown how the end-device will respond. If the details are processed then this will lead to

receives slots in the past, which is most probably not accepted by the end-device.

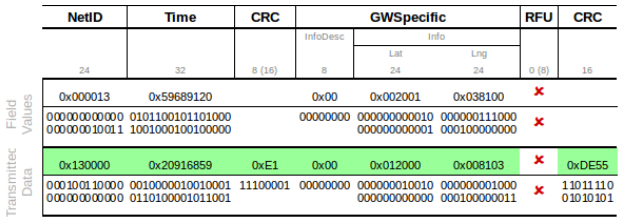

Figure B.14: Beaconing: Replay scenario 1

In the second Beaconing replay scenario (figure B.15), the Time field value is changed. To make sure the network frame is valid, also the first CRC field value must be changed. This first CRC value is based on the Time field value. This replay attack will result in changed receive slots on the end-device, which will prevent the transmitting of data from network server to end-device.

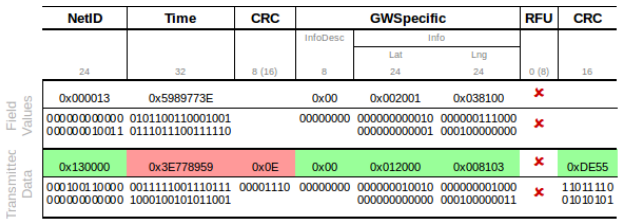

Figure B.15: Beaconing: Replay scenario 2

The third Beaconing replay scenario (figure B.16) includes modified gateway information. To make sure the network frame is valid, also the second CRC field value must be changed. This second CRC value is based on the Gateway information field values. The impact of this attack is depending on implementation details on the end-devices. It could be kept active more than expected (because it needs to send location updates to the network server), or maybe it could have issues with its functionality of the location is wrong.

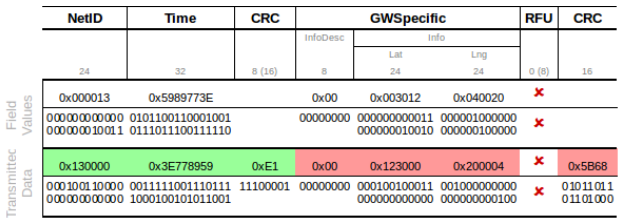

Figure B.16: Beaconing: Replay scenario 3

Changing the values of the fields listed in table B.3 does not result into unexpected behavior (based on the LoRaWAN specification).

Table B.3: *Beaconing* fields excluded

Data field	Remark
RFU	RFU values should be 00 if present. RFU is not available in all regions. Impact of changing this value is depending on the LoRaWAN implementation.

B.6 RELATED ATTACKS

Replay attacks are often part of other, more specific, attacks. Based on papers from Karlof, C., et al. [18], Pathan, A., et al. [33], Wang, Y., et al. [38], and Anwar, R., et al. [5], an overview of related Wireless Sensor Network (WSN) replay attacks has been created. This list, shown in table B.4, provides an overview of the impact of these attacks on the LoRaWAN specification. Many of these related attacks may lead to possible DOS issues which are described in chapter 5.3.

Table B.4: Overview of related replay attacks

Attack	Description and LoRaWAN impact
Spoofed, altered, or replayed routing information	In this attack, network routing information is being targeted. By changing routing details, while communicated between nodes, it might be possible to congest, disturb, or overload the network. LoRaWAN is based on a star-of-stars topology with minimal routing, therefore there is no specific routing traffic which might be targeted. Routing information is present on the network server for downlink traffic. The network server determines the optimal route, and which gateway to use for the downlink traffic. Other attacks, such as the wormhole attack, might have an impact on this routing information.
Collisions	By replaying certain network frames in very large quantities, network collisions may be caused. This can impact the transmissions of network frames by legitimate end-devices.
Exhaustion	By replaying certain network frames in very large quantities, exhaustion of gateways or the network server may be caused.
De-Synchronization	This is the disruption of an existing connection by timing specific replay scenarios such as retransmissions of missed frames. For LoRaWAN, a form of de-synchronization can be caused by replaying timing related beaconing packets receiving slot windows may be opened at the wrong moments.

Overview of related replay attacks – continued

ATTACK	DESCRIPTION AND LoRaWAN IMPACT
Wormhole	The wormhole attack captures network traffic, often routing traffic, and replays this information at network locations where it should not be present. This way, certain network parts can, for example, be misled with wrong routing details. This type of attack cannot be used in LoRaWAN exactly the same way. However, the same principle can be used to replay network traffic from end-devices via other gateways and therefore impact the routing information that the network server uses to determine the best route for downlink traffic. For LoRaWAN it is also possible to modify and replay network frames. The impact on the receiving side is depending on the implementation.
Message corruption	By re-transmitting network frames that are modified, but still corrupt, issues can be caused at the receiving side. For example, by changing network frame field values to incorrect values. Processing corrupt network frames could be more resource intensive for the receiver.
Desynchronization	An attacker disrupts active connections. For example by (re-)playing network frames with incorrect frame counters to desynchronize endpoints. This causes traffic to be dropped or retransmitted. For LoRaWAN the same principle might be used by replaying older network frames, or generating network frames with newer frame counters (requires MIC calculation).

C

APPENDIX C - DOS SCENARIOS

This Appendix contains detailed Denial of Service (DOS) attack scenarios, of which some are based on the communication scenarios which are described in Appendix A.

Some of these DOS scenarios include changing the value of the MIC field. Changing the value of the MIC field is not possible at this moment. The AES algorithm in combination with the 128-bits security keys makes this impossible with todays computing power (see chapter 5.4).

C.1 END-DEVICE ACTIVATION

For the Over-the-Air-Activation (OTAA) activation of an end-device two types of network communications are required. First, the request from the end-device to the network server, to join the network (Join-Request). Second, when the Join-Request is accepted by the network server, an accept response from the network server to the end-device (Join-Accept).

C.1.1 *Join-Request*

A Join-Request message is sent every time an end-device needs to (re-) register itself on a LoRaWAN. Such as Join-Request message is based on the AppEUI, DevEUI, and DevNonce field values (see chapter A.1.1).

DevNonce and Security Contexts
The DevNonce field value in a Join-Request message is to prevent replay attacks by sending-previously captured join-request messages. "*For each end-device, the network server keeps track of a certain number of DevNonce values used by the end-device in the past, and ignores join requests with any of these DevNonce values from that end-device* (page 34 of the LoRaWAN specification)".

The LoRaWAN specification does not include the amount of DevNonce values that must be tracked, by the network server, for each end-device "*the finite list of values tracked by the network server* (page 34 of the LoRaWAN specification)". The amount of tracked DevNonce values provides an indication on how many Join-Requests are required before a replay attack can be performed to DOS attack a specific

end-device. By not specifying the (minimal) amount of security contexts that need to be tracked, it is unclear how vulnerable a certain LoRaWAN implementation is for this type of DOS attack.

Besides tracking DevNonce values, the network server also maintains multiple security contexts. The security keys and frame counters, are called security contexts. During each Join-Request a new security context is created for that particular end-device. The network server maintains both the old and new security context until a successful uplink network frame is received using the new security context.

In summary, an unspecified number of DevNonce values is tracked per end-device for preventing replay attacks, and both (old and new) security contexts are maintained until the new security context has been confirmed by a successful uplink network frame.

In the example attack below, an attacker is using eavesdropping and replay attacks trying to perform a DOS attack on the end-device. By doing so it tries to manipulate the track list that is managed by the network server. The attack is based on captured network frames from the end-device example lifecycle which is shown in figure **??**.

The first step in the attack, is that the end-device sends a Join-Request message to the Network server, using a DevNonce field value of 0x0100.

MHDR			Join-Request			MIC
MType	RFU	Major	AppEUI	DevEUI	DevNonce	
3	*3*	*2*	*64*	*64*	*16*	*32*
0x00	0x00	0x00	0xBB6100F07ED5B370	0xCF3662E82bA26100	0x0100	0xB761677F

Figure C.1: Join-Request DOS: Join-Request 1

This Join-Request message is received and processed by the network server. The network server adds the details and security context to its track list. The security context is confirmed by the first uplink network frame.

#	AppEUI	DevEUI	DevNonce	Keys	Frame counters
1	0xBB6100F07ED5B370	0xCF3662E82bA26100	0x0100	0x37A7...	1 ; 0
2					
3					

Figure C.2: Join-Request DOS: Track List 1

A later moment in time, the end-device needs to re-register itself on the LoRaWAN. It sends a new Join-Request messages with a DevNonce field value of 0x62A9. Once received, processed by the network server,

and confirmed by the end-device (by means of the next uplink network frame), the track list contains multiple DevNonce values and the last confirmed security context (key values and frame counters).

#	AppEUI	DevEUI	DevNonce	Keys	Frame counters
1	0xBB6100F07ED5B370	0xCF3662E82bA26100	0x0100		
2	0xBB6100F07ED5B370	0xCF3662E82bA26100	0x62A9	0x276D...	1 ; 0
3					

Figure C.3: Join-Request DOS: Track List 2

Again, the end-device needs to re-register itself on the LoRaWAN. It sends a new Join-Request messages with a DevNonce field value of 0x12FF. Once received, processed by the network server, and confirmed by the end-device, the track list contains multiple DevNonce values and the last confirmed security context.

#	AppEUI	DevEUI	DevNonce	Keys	Frame counters
1	0xBB6100F07ED5B370	0xCF3662E82bA26100	0x0100		
2	0xBB6100F07ED5B370	0xCF3662E82bA26100	0x62A9		
3	0xBB6100F07ED5B370	0xCF3662E82bA26100	0x12FF	0xA791...	1 ; 0

Figure C.4: Join-Request DOS: Track List 3

Once more, the end-device needs to re-register itself on the LoRaWAN. It sends a new Join-Request messages with a DevNonce field value of 0x838D. Once received, processed by the network server, and confirmed by the end-device, the track list contains multiple DevNonce values and the last confirmed security context. Because the track list only has three positions, the first DevNonce (0x0100) value has been removed from the list.

#	AppEUI	DevEUI	DevNonce	Keys	Frame counters
1	0xBB6100F07ED5B370	0xCF3662E82bA26100	0x62A9		
2	0xBB6100F07ED5B370	0xCF3662E82bA26100	0x12FF		
3	0xBB6100F07ED5B370	0xCF3662E82bA26100	0x838D	0x9F32...	1 ; 0

Figure C.5: Join-Request DOS: Track List 4

Now that the first DevNonce value has been removed from the list, an attacker can replay the older Join-Request message that contained this DevNonce field value. The network frame is valid, the MIC is also valid because it is an original network frame. When the network frame has been received by the network server, the frame is processed as a legitimate new request. The details are added to the track list. At this moment, the end-device is still able to communicate because the attacker has not yet confirmed the new security context by sending an uplink network frame.

#	AppEUI	DevEUI	DevNonce	Keys	Frame counters
1	0xBB6100F07ED5B370	0xCF3662E82bA26100	0x12FF		
2	0xBB6100F07ED5B370	0xCF3662E82bA26100	0x838D	0x9F32...	125 ; 14
3	0xBB6100F07ED5B370	0xCF3662E82bA26100	0x0100	0x1FF9...	0 ; 0

Figure C.6: Join-Request DOS: Track List 5

Even if the attacker could capture the first uplink frame belonging to the initial Join-Request that was captured, the attacker will not be able to use that uplink frame. After replaying the older Join-Request message, a new security context has been generated, which includes new security keys and frame counters. The generation of new security keys means that the captured uplink frame is not valid anymore because it is based on a different set of security keys. Replay 2 attempt, in figure C.7 shows this failure of the replayed first uplink network frame.

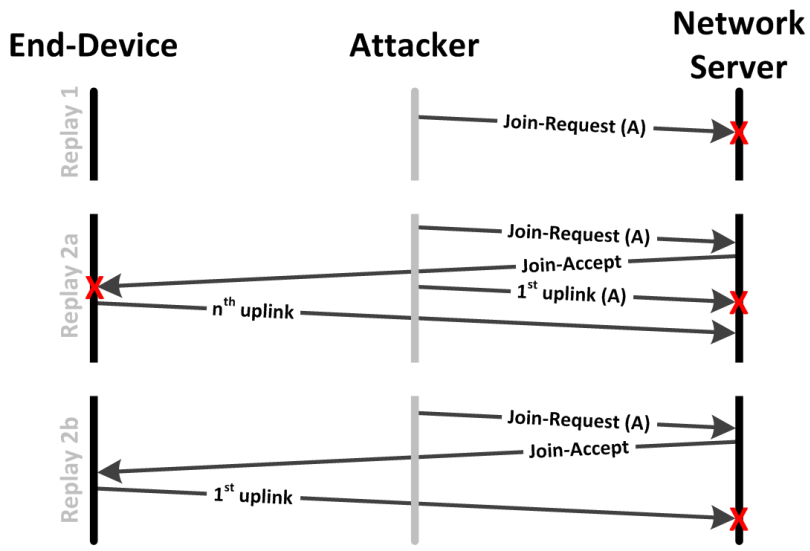

Figure C.7: Attack Sequence: Join-Request DOS

Figure C.7 shows three possible attacks in respect to the Join-Request message. In the first attempt (replay 1) an earlier captured Join-Request is replayed. However, it is ignored by the network server because the used DevNonce is already on the track list for this end-device. The end-device will be able continue communications. The second attempt (replay 2a) also replays an earlier captured Join-Request message. In this case, the DevNonce is not yet on the track list, and the request is processed by the network server. The network server send a Join-Accept to the end-device. The end-device ignores the Join-Accept because it has not initiated a Join-Request. As an alternative the attacker also replays an uplink data frame to confirm the new security context. This fails because the security context in the past is different compared to the newly generated security context. The end-device

will be able continue communications with its present security context. In the third attempt (replay 2b), the end-device accepts the Join-Accept message from the network server and generates an uplink data frame to confirm the new security context. However, the security context on the end-device and the network server differ. The end-device is not able to communicate anymore until a new OTAA is initiated. Unfortunately, the response of the end-device on an unexpected Join-Accept message is not documented in the LoRaWAN specification.

The LoRaWAN specification does also not include a description on how a non-confirmed Join-Request message must be handled. This leads to the following unanswered questions:

- Is the DevNonce still in the track list, taking up a position? Replaying multiple older Join-Request messages might have an impact on the track list and thus on availability of the end-device.
- What happens even another Join-Request comes in? Is the previous unconfirmed Join-Request discarded?
- How does the end-device respond to a Join-Accept message that was not initiated by the end-device? Best is to ignore/drop the Join-Accept message. The current security context still works and the new security context will not be confirmed.

Concurrent requests

When a Join-Request message is not answered with in a certain time, the Join-Request message is retransmitted by the end-device. If this happens after, for example, a power failure, then many end-devices might send a Join-Request message at the same time and the network server might not be able to answer (in time). If the time until the retransmission is not randomly chosen, then all end-device will constantly resend their Join-Request at the same time keeping the network server extremely busy and possibly causing a DDOS (page 37 of the LoRaWAN specification).

If it would be somehow possible to trick all/many end-devices into initiating an OTAA (sending a Join-Request) then this could be considered as a DDOS attack on the LoRaWAN network server. Another possibility is that an attacker can generate many Join-Request messages which might cause a service degradation or service unavailable of the network server due to the processing of all Join-Request messages at the same time. This is shown in figure C.8.

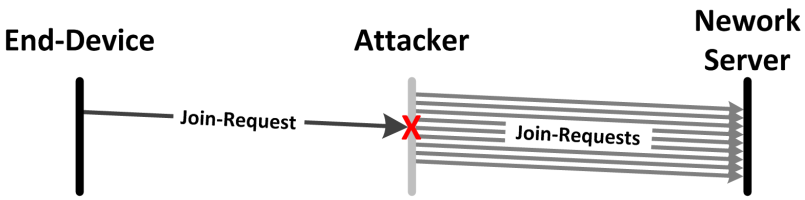

Figure C.8: Attack Sequence: Join-Request DOS

C.1.2 *Join-Accept*

Like the Join-Request replay attack described in the previous section, Join-Accept network frames can also be replayed. This can be used trying to influence the behavior of an end-device and/or the network server. Figure C.9 shows several replay attacks using a previously captured Join-Accept message.

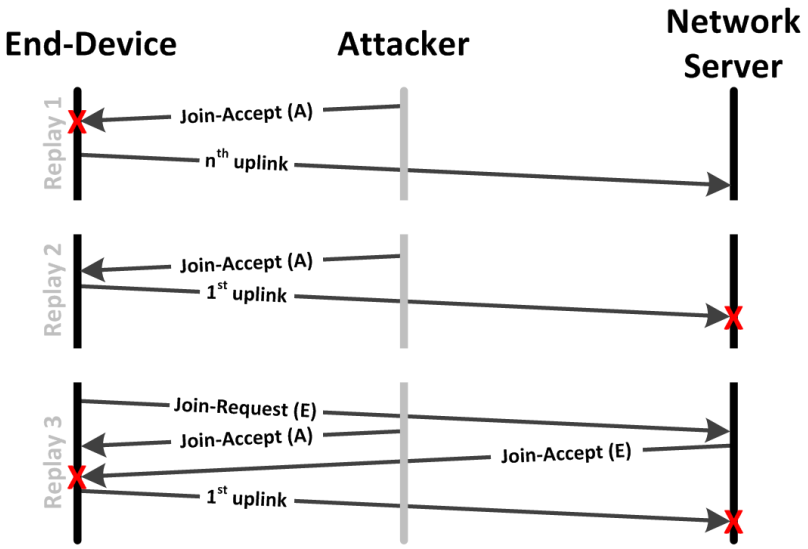

Figure C.9: Attack Sequence: Join-Accept DOS

In the first replay attempt (replay 1), the attacker replays a previously captured Join-Accept message. The LoRaWAN specification does not include details on the expected response of the end-device. Based on the attack descriptions in chapter C.1.1, we should assume the Join-Accept frame is dropped by the end-device. The end-device will continue to communicate with the network server using the existing security context.

If the end-device does not drop the Join-Accept message, but processes it, as shown in the second example (replay 2). Then the end-device cannot communicate with the network server anymore because both use different security contexts.

The third example (replay 3) replays the earlier captured Join-Accept message during an existing OTAA procedure. The attacker makes sure that the replayed Join-Accept messages is received by the end-device before the legitimate Join-Accept message is received by the end-device. Based on the LoRaWAN specification, it seems that this replayed Join-Accept message will be processed by the end-device. First, the end-device is expecting a Join-Accept message. Second, the field values in the Join-Accept message do not have a direct relation with the details in the Join-Request message. The end-device will process the content of the Join-Accept message. However, by doing so it will generate a security context that is based on the wrong information. The result is that both the end-device and network server are working with a different security context and communication is not possible.

C.2 SENDING MAC COMMANDS

The LoRaWAN specification does not include any dedicated MAC network frames. MAC commands can be transmitted using network frames used for sending data (either dedicated or piggy-backed).

A normal replay attack to (re-)play (modified) MAC-commands can only succeed when an attacker is able to generate a valid MIC (see chapter see chapter 5.4) and determine the correct next frame counter (FCnt). In such a case a piggy-backed MAC command can be manipulated in the FOpts field value. In case of a dedicated network frame, the attacker must also be able to manipulate the used encryption in the FRMPayload field value.

When successful replay attacks can be performed for sending MAC commands then it is possible to adjust settings on the client so that they are not in line anymore with settings on the network server or gateway. This would cause the unavailability of an end-device.

C.3 BEACONING

All gateways participate in providing a time-synchronization mechanism by sending beacons at regular fixed intervals configurable per network. This time-synchronization mechanism is used to calculate receive slots in the case of Class B and Class C operations. A beacon also contains location information regarding the gateway that send the beacon. An end-device can use this location information to determine if it has moved to a different gateway.

Manipulating beacons
A beacon is send without any security measures such as integrity checks or encryption. This makes it possible to (re-)play a beacon

with modified field values. If an attacker (re-)plays such a modified beacon with a stronger signal strength than the nearest gateway, then end-devices will process the manipulated beacon instead of the original beacon.

"*All gateways send their beacon at exactly the same point in time (i.e., time-synchronously) so that for network common part there are no visible on-air collisions for a listening end-device even if the end-device simultaneously receives beacons from several gateways. With respect to the gateway specific part, collision occurs but an end-device within the proximity of more than one gateway will still be able to decode the strongest beacon with high probability*¨ [20] (page 53).

Receive slots
The Time field value in a beacon is used to calculate end-device receive slots. Receive slots are time windows in which the (modem) receiver in an end-device is activated so that a network server can initiate a downlink communication. Figure C.10 shows a graphical representation of several beacons. For one beacon, variables are included that are used for receive slot calculations. The receive slots

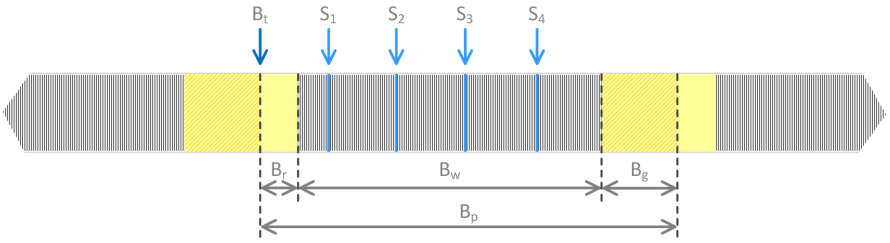

Figure C.10: Beaconing: slot calculation

A single beacon is represented by B_p, which is the BEACON_PERIOD. This period is usually 128 seconds and divided in three sections. First, there is B_r which is the 2.120 second BEACON_RESERVED window. This window is used to receive a beacon. Second, B_w is the BEACON_WINDOW. This windows has a duration of 122.880 seconds and is split into 4096 possible receive slots (each slot is 30ms). Third, a BEACON_GUARD is represented by B_g. This is a waiting period so that existing transmissions can be finished before the receive window of the next beacon must be opened.

The first blue arrow is the received beacon. B_t represents the beacon Time field value. The next four arrows S_1 to S_4, represent calculated receive slots. Table C.1 shows the formulas used for the receive slot calculations, including an example.

Table C.1: Receive slot calculation

VALUE	FORMULA
$Slot_x$	B_t + B_r + (pingOffset + (x-1) * pingPeriod) * 30ms
pingOffset	(rand[0] + rand[1]*256) modulo pingPeriod
rand	aes128_encrypt(Key, B_t | DevAddr | pad16)
pingPeriod	2^{12} / PingNb
pingNb	amount of receive slots to use = [2 .. 128]
rand	aes128_encrypt(0x00000000000000000000000000000000, 0x59689120 | 0x26011963 | 0x0000000000000000) = 297F06C9EA80B17345272CF094725975
pingNb	2^2 = 4 receive slots
pingPeriod	2^{12} / PingNb = 4096 / 4 = 1024
pingOffset	(2 + 9*256) modulo 1024 = 258
S_1	B_t + B_r + (258 + (1-1) * 1024) * 30ms = B_t + B_r + 7740ms
S_2	B_t + B_r + (258 + (2-1) * 1024) * 30ms = B_t + B_r + 38460ms
S_3	B_t + B_r + (258 + (3-1) * 1024) * 30ms = B_t + B_r + 69180ms
S_4	B_t + B_r + (258 + (4-1) * 1024) * 30ms = B_t + B_r + 99900ms

Beacon time manipulation

By manipulating the Time field value in a (re-)played beacon, it is possible to trick an end-device into calculating wrong receive slots. When in B_r a beacon with a higher signal strength is send, that includes a different B_t, the slot calculation will lead to different receive slots. Like figure C.10, figure C.11 shows a BEACON_PERIOD which now includes details for the original beacon (in blue), combined with details for a manipulated beacon (in red).

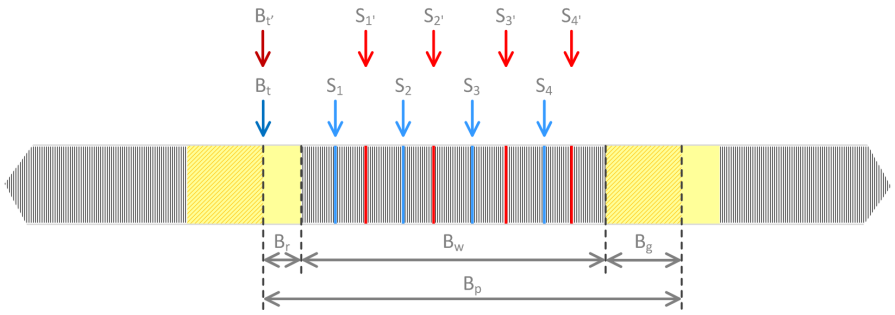

Figure C.11: Beaconing: manipulated slot calculation

The first red arrow is the received beacon. $B_{t'}$ represents the manipulated beacon Time field value. The next four arrows $S_{1'}$ to $S_{4'}$, represent calculated receive slots. Table C.2 shows example receive slot calculations based on $B_{t'}$.

Table C.2: Receive slot calculation

VALUE	FORMULA
rand	aes128_encrypt(0x00000000000000000000000000000000, 0x59689121 | 0x26011963 | 0x0000000000000000) = F3FA45FAAFFF40BFFFB5A56B13A706C3
pingOffset	(15 + 3*256) modulo 1024 = 783
$S_{1'}$	$B_{t'}$ + B_r + (783 + (1-1) * 1024) * 30ms = $B_{t'}$ + B_r + 23490ms
$S_{2'}$	$B_{t'}$ + B_r + (783 + (2-1) * 1024) * 30ms = $B_{t'}$ + B_r + 54210ms
$S_{3'}$	$B_{t'}$ + B_r + (783 + (3-1) * 1024) * 30ms = $B_{t'}$ + B_r + 84930ms
$S_{4'}$	$B_{t'}$ + B_r + (783 + (4-1) * 1024) * 30ms = $B_{t'}$ + B_r + 115650ms

The calculation of receive slots based on a modified Time field value leads to offset receive slots. This means that the (modem) receiver of the end-device will be active on other moments compared to the windows calculated by the network server. The network server uses the same calculation to determine the receive slots of an end-device so that the network server will initiate a downlink connection within the calculated receive slots. If these windows are not identical, there is no downlink communication possible between network server and end-device.

Beacon location manipulation
A beacon also contains a GwSpecific field. This field contains GPS details of the gateway that has send the beacon. An end-device can use this information to determine if it has changed between gateway (being mobile). When an end-device has changed gateway, it must inform the network server so that the downlink routing information can be updated. This is required for Class B and Class C operations.

An end-device has two options for initiating a downlink routing update: "*Systematic periodic update*" and "*Uplink on cell change*". The first option sends a periodic uplink frame (to update downlink routing). The last option initiates an uplink frame (to update downlink routing) when movement has been detected based on gateway location information in a beacon.

When an end-device is using the second option to update downlink routing, changing the GPS coordinates on purpose could trigger an "*Uplink on cell change*". Changing the gateway location in many beacons, might result in excessive power use by the end-device because of all required updates.

C.4 DOWNLINK ROUTING

For downlink communications, the network server keeps track of the best gateway that should be used to address a specific end-device. "*The gateway chosen to initiate this downlink communication is selected by the network server based on the signal quality indicators of the last uplink of the end-device.*" [20] (page 40). Figure C.12 shows the normal flow for this type of communication.

Figure C.12: Downlink routing

For downlink communication, the network server uses the same gateway that was used for the last successful uplink communication.

If it would be possible to replay traffic of a certain end-device via a different gateway, then this might have an impact on downlink routing decisions by the network server. A wrong routing decision might result in using the wrong gateway to reach a specific end-device. In such a situation, the end-device cannot be reached. Figure C.13 shows a possible manipulation of downlink routing by using a wormhole attack. A wormhole attack is a variant of an eavesdropping + replay attack.

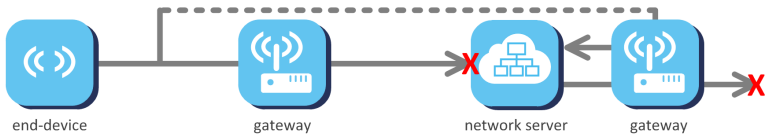

Figure C.13: Manipulate downlink routing - wormhole

In a wormhole attack a network frame is eavesdropped, transported over a faster transmission media, and replayed in a different part of the network. In this example, the uplink network frame is eavesdropped and replayed before the original network frame is received by the network server. Because the replayed network frame is received first, that one is processed by the network server. The network server has no possible to detected that this is not the original network frame. One the original network frame is reaching the network server, it is dropped because it has a wrong frame counter, and it is already processed by the network server.

If the network server wants to initiate a downlink communication, it uses routing information to determine which gateway should be used to reach the end-device. If the last successful uplink communication

from the end-device has come in via a wrong gateway, then the downlink communication will also be send via a wrong gateway.

When the MIC calculation can be attacked successfully (see chapter 5.4), then a regular replay attack can be used to send a successful uplink network frame, via a wrong gateway, to manipulate downlink routing. Figure C.13 shows the flow used for such this type of communication.

Figure C.14: Manipulate downlink routing - replay

C.5 FRAME COUNTERS

When an attacker can generate valid network frames for an end-device (see chapter 5.4 for a MIC attack) it might continue uplink traffic when the correct Frame Counter is used. This way the traffic counters at the network server side will increase, while at the original end-device they will not. For a while it will be OK because of the maximum frame counter gap (MAX_FCNT_GAP). Once the gap is too large, the network frames of the original end-device will be dropped by the network server. Figure C.15 shows an attack sequence that represents a DOS by manipulating frame counters. This type of attack can also be called a de-synchronization attack.

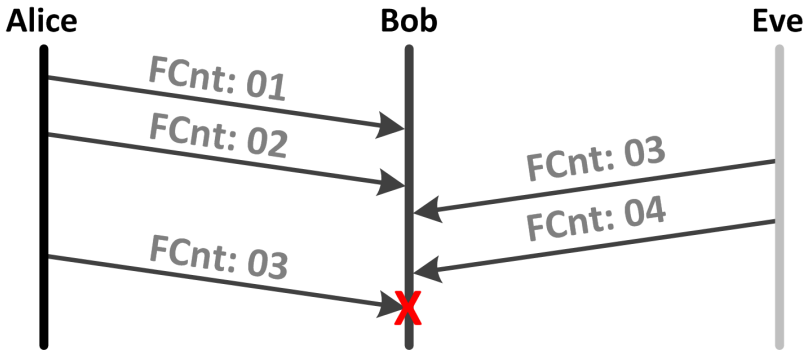

Figure C.15: Attack sequence: Frame counter DOS

C.6 RESOURCE EXHAUSTION

The most obvious method to cause a resource exhaustion is to send an excessive amount of network frames. Processing a high amount of network frames might cause an overload of resources, such as the NIC, CPU or memory. Other methods rely on sending less network frames, but the network frames are specially crafted to possibly cause a CPU

or memory overload during processing of the content of the network frame.

Based on the LoRaWAN specification it is difficult to determine if there are resource exhaustion possibilities for LoRaWAN. Purely seen from a specification perspective, and not from an implementation perspective, there are only a few resource limitations. Limitations mentioned in the LoRaWAN specification include duty cycle and dwell time to avoid network congestion.

Duty Cycle
A duty cycle in the LoRaWAN specification is related to the time that the network is in use by both end-devices and gateways. Figure C.16 shows a total duty cycle of 60%.

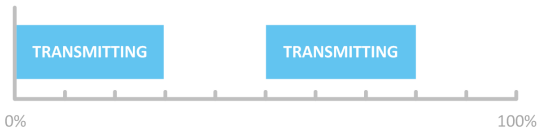

Figure C.16: Duty Cycle

To prevent congestion of the network, the LoRaWAN specification includes maximum duty cycles. These limits are depending on local regulations. For the European bands the duty cycles are regulated by section 4.3.3.2 of the ETSI EN300.220[1] standard. Usually this is 0.1% or 1%. There are some small ranges that have a 10% duty cycle or no restrictions at all. On top of the local regulations, also the network provider will define maximum duty cycles. Figure C.17 shows a maximum duty cycle of 1%.

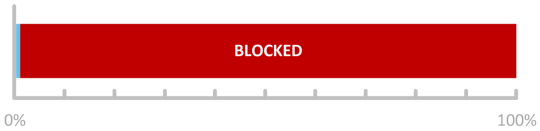

Figure C.17: Duty Cycle Limit

It is a bit more complex than the example since LoRaWAN uses multiple bands (and even sub-bands) and the maximum duty cycle is applied per (sub-)band. This means multiple sub-limits must be aggregated to a limit per band. In some cases, these limits are enforced by the used modem, for example the RN2483[2] modem. In other cases, the developer must enforce these limits in the behavior of the end-device and/or gateway.

1 http://www.etsi.org/deliver/etsi_en/300200_300299/30022002/03.01.01_30/en_30022002v030101v.pdf
2 http://ww1.microchip.com/downloads/en/DeviceDoc/70005219A.pdf

Because these limits need to be enforced by the transmitting device, there is no generic enforcement on a network level. This leaves room for abuse. When a specially crafted end-device uses much more time on the network there is not much that can be done. It can be removed by the network operator for functioning on its network. However, this does not prevent this end-device from keeping the network frequencies busy, and by doing so, performing a way of jamming.

Dwell time

Besides the duty cycle, the LoRaWAN specification also includes a dwell time for specific regions. The dwell time is the maximum transmit duration (of a single uplink transmission). For example, if the maximum allowed dell time is 400ms, then a single uplink transmission must be completed within 400ms. If it would take longer, the transmission must be split in multiple smaller transmissions.

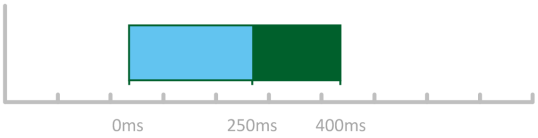

Figure C.18: Dwell Time

Figure C.18 shows a transmission that has a duration of 250ms. It should have been completed in 400ms. In this example, the transmission time is within limits of 400ms which is represented by the green block.

Like the maximum duty cycle, these limits need to be enforced by the transmitting device, there is no generic enforcement on a network level. This limitation can be ignored, or overruled, by a specially crafted end-device to impact availability of the network.

Other factors

Besides the duty cycle and dwell, there are other factors which might have a (big) impact on network availability. Both the local regulations and the network operators have limits to make sure the network does not get congested. However, when multiple LoRaWAN implementations are close together they will impact each other because of the use of a shared medium (the air). Although most LoRaWAN network frames include a network ID, all network components must (partly) receive and read all network frames before it can be decided if a network frame should be processed or dropped because the network frame belongs to a different network.

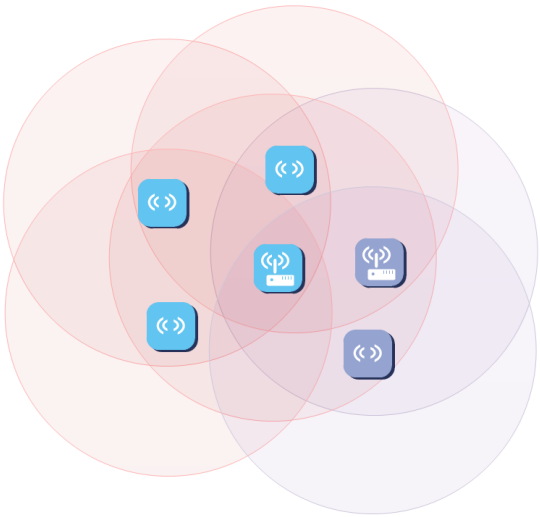

Figure C.19: Impact of multiple LoRaWANs

Figure C.19 shows how two geographically related networks (different colors) can have an impact on each other based on their range. They both use the air as a network medium, and since both follow the LoRaWAN standards, they also both use the same frequencies. Even if a certain LoRaWAN implementation does not have a big scale, it can still be hugely impacted when another, larger scale, LoRaWAN implementation is nearby.

C.7 COLLISIONS

Because wireless networks use a shared media (the air), there is always the possibility for collisions. A collision is when two parties start sending at the same moment. The result is that both transmissions are mixed-up (and thus results into an invalid frame), or that the strongest signal pushes away the weaker signal.

Since LoRaWAN does not know a re-transmission mechanism, the network frame will be lost in case of a collision. A re-transmission mechanism (if required) should be implemented by the application layer. A form of DOS attack is when a hacker can cause collisions on purpose. Using such an attack, an attacker would be able to corrupt the transmissions of a legitimate end-device (or gateway). Using this attack is difficult, it requires correct timing and signal strengths.
Figure C.20 shows an attack sequence that represents a DOS caused by collisions.

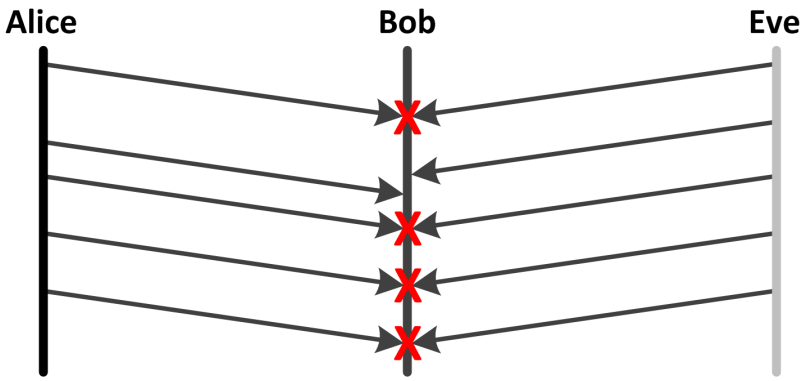

Figure C.20: Attack sequence: Collisions

C.8 TCP/IP ATTACK SURFACE

In a LoRaWAN environment, several components have a TCP/IP connection. Examples of such components are gateways, the network server, and application servers. When these components use the public Internet for their communication then they might be vulnerable to existing TCP/IP (D)DOS attacks.

Besides an TLS (public key encryption) recommendation in LoRaWAN recommendations, there are no details in the LoRaWAN specification regarding the TCP/IP connectivity of LoRaWAN components. There are additional measures that can be taken to improve the security for these connections. However, this is out of scope for this research study.

C.9 RELATED ATTACKS

Based on papers from Raymond, D., et al. [35], Wang, Y., et al. [38], Padmavathi, D., et al. [32], Hangargi, M. [13], Ghildiyal, S., et al. [12], and Anwar, R., et al. [5], an overview of related Wireless Sensor Network (WSN) (D)DOS attacks has been created. The list in table C.3, provides an overview of the impact of these attacks on the LoRaWAN specification.

Table C.3: Overview of related DOS attacks

ATTACK	DESCRIPTION AND LoRaWAN IMPACT
Sinkhole	A sinkhole attack is when (a part of) network is being directed (route) to one specific component. Once all traffic is routed via this component an attacker can have control over the traffic that is passing this component. In a LoRaWAN, a malicious gateway can be considered as a sinkhole for a specific geographical region. For open/community networks this is far more easier compared to closed networks.
Blackhole	In a blackhole attack, all traffic on a sinkhole is being dropped. All connectivity that is using the sinkhole will fail. When a malicious gateway is placed in a LoRaWAN environment, this gateway can be configured to drop all traffic between end-devices and the network server. This will only work if this is the only gateway in reach of the end-device. If the traffic of the end-device is also received by other gateways, then the malicious gateway has no impact on the routing of traffic.
Selective forwarding	During a selective forwarding, specific traffic on a sinkhole is forwarded to the original destination (or to a new destination). When a malicious gateway is placed in a LoRaWAN environment, this gateway can be configured to forward specific traffic only, between end-devices and the network server. If the traffic of the end-device is also received by other gateways, then the malicious gateway has no impact on the routing of traffic.
Denial of Sleep	A denial-of-sleep attack keeps an end-device 'busy' so that it cannot go into a 'sleep' state and therefore consume more power than intended. This is especially harmful for battery-operated devices. The LoRaWAN specification does not include a sleep state for end-devices. However, unusual or unintended behavior could cause an end-device to consume more power. For example, receiving much more network traffic than expected.
Probe request flooding	A probe-request is when a client is probing a Wi-Fi network for (a specific) base stations (SSID) to determine which Wi-Fi networks are available. By sending many of these requests (e.g. thousands of replayed packets with different MAC addresses), Wi-Fi networks (base-stations) could be overwhelmed, resource use may become too high, and the base-station might become unavailable. In the LoRaWAN specification there is no such probe-request.

Overview of related DOS attacks – continued

ATTACK	DESCRIPTION AND LoRaWAN IMPACT
Authentication request flooding	To join a specific Wi-Fi network, an authentication request must be send to the base-station. After authentication, a client can use the specific Wi-Fi network. In the LoRaWAN specification a similar approach is the OTAA procedure. This is a request from a client to the network to join. Sending many Join-Accept messages might case unavailability of a network server.
De-authentication request flooding	To terminate an active session within a wireless network, both the client or the base-station can initiate and send a de-authentication message. By sending many de-authentication messages, the connectivity of a client can be disturbed since it needs to re-authenticated over-and-over again. In the LoRaWAN specification there is no de-authentication / de-association request.

D

APPENDIX D - PETRI NETS & CPNTOOLS

In this research study, Colored Petri Nets (CPN) will be used to model, simulate and analyze attack scenarios. The creation, simulation, and analysis of CPN models will be done in CPNtools[1]. CPNtools is one of many tools, which can be used to model and analyze CPNs. The decision for using CPNs and CPNtools is mainly based on the fact that these have been used in earlier assignments, CPNs are used more often for protocol analysis [31] [39] [30] [16] [7]. CPNTools is one of the most versatile tools for CPNs [34], despite having a steep learning curve.

D.1 PETRI NETS

Petri nets offer a formal, and graphical, notation for representing information systems and business processes in a compact way. It is a mathematical modeling language [3] which is represented as a graph. The graph is based on rectangles that represent transitions (for example events), and circles that represent places (for example conditions). The rectangles and circles are connected by arrows that form pre-, and post conditions for transitions and form the *flow relation*. A Petri net contains one or more tokens, which (in certain conditions) enable transitions. A transition consumes one or more tokens and possibly produces one or more tokens. In graph theory, this is called a bipartite graph. Figure D.1 depicts a Petri net modeling a high-level behavior of a LoRaWAN end-device.

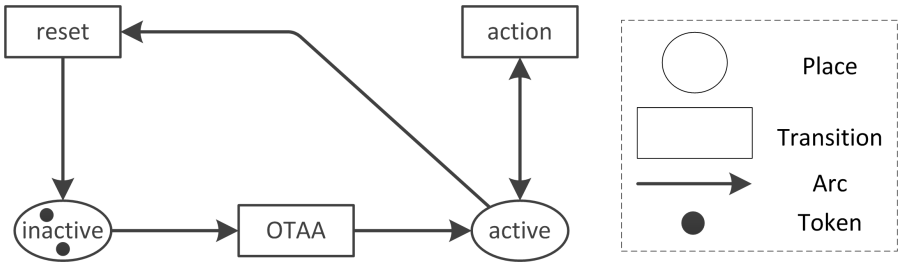

Figure D.1: Petri net example

The tokens in the picture represent end-devices. The status of an end-device can be 'inactive', or 'active'. The statuses are modeled as places (circles). The actions needed to change between status ('OTAA', 'action', and 'reset') are modeled as transitions (rectangles). The arrows show the flow (or sequence). A place may contain tokens, represented

1 http://cpntools.org/

by a black dot. In this example the place 'inactive' has two tokens which indicates two end-devices that are not activated on the network. The transition 'OTAA' is now *enabled* (the pre-condition is met; there is a token in the input place) and ready to *fire* (execute).

Definition: A Petri net is a triple (P, T, F), where:
1. P is a finite set of places,
2. T is a finite set of transitions, and
3. $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation.

For example D.1, this means that the sets are defined as:
P = {'inactive','active'},
T = {'OTAA','action','reset'}, and
F = {('inactive','OTAA'),('OTAA','active'),('active','action'), ('action','active'), ('active','reset'),('reset','inactive')}.

The behavior of a Petri net is defined by the structure of the net, the distribution of the tokens over the places, and the firing of transitions.

D.2 COLORED PETRI NETS

CPNs are an extension of Petri nets [2]. Additionally to regular Petri nets, the properties of time, color, and hierarchy are added. These properties are important to be able to keep the model for complex systems compact, add more expressive power, and the ability for adding explicit model time.

Image that we need to model the previous example for multiple (different) end-devices. A standard CPN can contain multiple tokens, however it is not possible to distinct between the two tokes. When a CPN is used, values can be assigned to tokens and therefore tokens can be identified throughout the model.

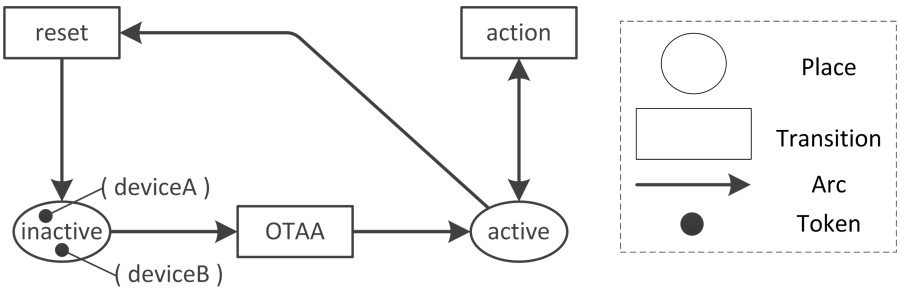

Figure D.2: Colored Petri net example (1)

Figure D.2 depicts an example of a CPN. The model is identical to the model used in a regular Petri net. However, both tokens have been assigned values. This change makes it possible to use the model for

multiple end-devices, and identify the status of the individual end-devices in all states.

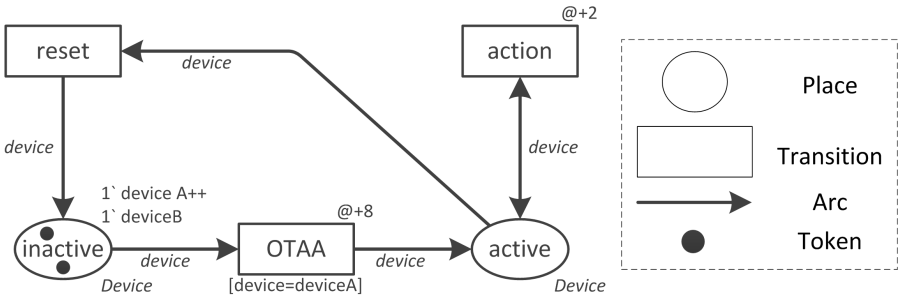

Figure D.3: Colored Petri net example (2)

Figure D.3 shows the complete CPN for the specified example. (*It must be noted that this is a very simple model which does not represent a practical use case. The model can easily be extended to represent different functionality, but it would make the initial example much more difficult.*)

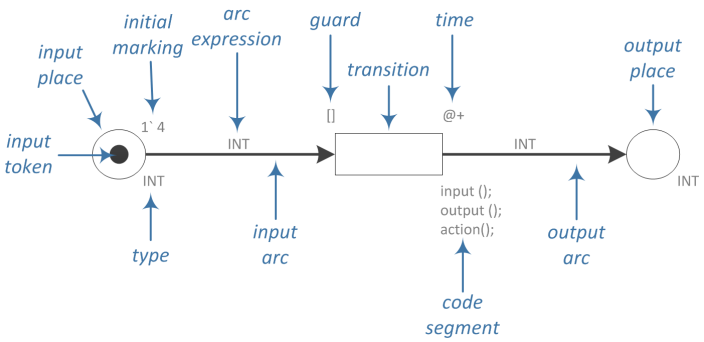

Figure D.4: Colored Petri net constructions

Figure D.4 shows the most important properties related to CPNs. A *place* represents a state in the model and is represented by a circle or ellipse. In a CPN, a place needs to be of a specific *type* (color set). This type must be identical to the type of the *token*(s) that can be in the place. The *initial marking* specifies the tokens that are initially in the place. Time can be added to an initial marking to activate an initial marking at a later moment. A place is connected to a *transition* by an arc. If the arrows points to the transition, then the arc is an *input arc*, otherwise the arc is an *output arc*. An arc can have an *expression*. This expression evaluates into a valid color set (identical to that of the place where the arc is related with). Time can also be added to an arc. A transition can have a guard, which is a Boolean expression, that evaluates the arc inscriptions. A transition can also have a time specified which denotes the amount of time units the transition takes. A transition can also contain a code segment which can perform actions on input and output variables. A transition is connected by an *output arc* with an *output place*.

D.3 CPNTOOLS

"*CPN Tools is a tool for editing, simulating, and analyzing Colored Petri nets*", according to the CPNtools website. "*The tool features incremental syntax checking and code generation, which take place while a net is being constructed. A fast simulator efficiently handles untimed and timed nets. Full and partial state spaces can be generated and analyzed, and a standard state space report contains information, such as boundedness properties and liveness properties.*"

CPNtools uses a graphical interface in which a CPN can be constructed. During the construction of a model, CPNtools also validates syntax of details added to the model (such as language syntax, variable definitions, and type assignments).

D.3.1 *Simulation*

To validate the functional correctness of a model, simulation can be a good method. When a syntactically correct model has been constructed in CPNtools, a simulation can be started to (graphically) execute the firings in the model until the simulation is stopped, or until the model comes to an end state. During simulation, the actual tokens are shown, enabled transitions are highlighted, steps are displayed, and the time units are shown. It is also possible to manually execute steps during a simulation. This can be useful because of non-deterministic decisions during the execution of a Petri net. The transitions that occur during a simulation can be written to a report file.

D.3.2 *State Space Analysis*

A model that has been created should not only work as expected in simulation mode, but also the mathematical properties of the model are import to verify the validity of the model. Validation of a CPN is based on its state space. A Petri net is based on a Transition System and the state space represents all possible states that the model can have. For a very simple model it is easy to determine the state space. See figure D.5 for all states of the model in example D.1. This representation of all states is a reachability graph, especially readable for humans, and can only be used on models that are finite.

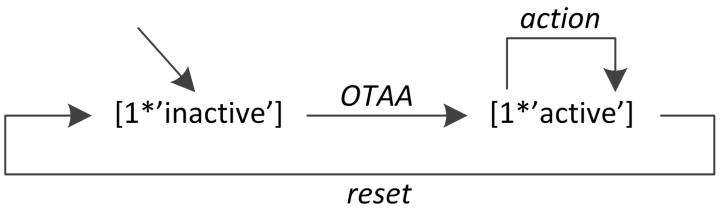

Figure D.5: Petri net reachability graph

The reachability graph shows that the model start with one marking in the place 'inactive'. When the transition 'OTAA' is fired, the state changes to one token in 'active'. After firing transition 'action', there is still one token in 'active'. Finally, when transition 'reset' fires, the model is back in its initial state.

Calculating a state space for a more complex model is much more difficult. This cannot be performed manually. For very larger model, or models that are infinite, only a partial state space is used for analysis purposes. This is because of the state space explosion [37]. The state space will become so big that it is impossible to calculate completely. The analysis of a state space can be used to analyze specific properties of models. See table D.1 for a description of these properties [3] [1] [39].

Table D.1: Overview of State Space properties

Property	Description
Boundedness	A Petri net (N, M_o), is said to be *k-bounded* or simply *bounded* (read: limited capacity) if the number of tokens in each place does not exceed a finite number k for any marking reachable from M_o. A Petri net (N, M_o) is said to be *safe* if k is equal to one. A bounded Petri net always has a finite number of reachable markings. Unbounded places in a Petri net is an indication that the model might be incorrect.
Dead Marking	A marking M_{dead} of a Petri net (N, M_o) is said to be a *dead marking* if, in marking M_{dead}, no transition is enabled in the net.
Terminating	A Petri net (N, M_o), is said to be *terminating* if for any marking M that is reachable from M_o, it is possible to ultimately reach a terminal marking for which no transition is enabled. The terminating property ensures that every run will be finite, and there that are only finitely many runs.

Overview of State Space properties – continued

Property	Description
Liveness	A Petri net (N, M_o), is said to be *live* if for any marking M that is reachable from M_o, it is possible to ultimately fire any transition of the net by progressing some further firing sequence. The liveness property ensures that a transition can always become enabled again. A live Petri net cannot be terminating, and does not have dead transitions.
Home Marking	A marking M_{home} of a Petri net (N, M_o) is said to be a *home marking* if M_{home} can be reached from any reachable marking M_n. The home marking property guarantees that a marking can always be reached again.
Reversibility	A Petri net (N, M_o) is said to be *reversible* if, for each marking M that is reachable from the initial marking M_o, M_o is reachable from M. If a Petri net is reversible then the *initial marking* is a home marking.
Deadlock freedom	A Petri net (N, M_o), is said to be *terminating* if for any marking M that is reachable from M_o, it is not possible to ultimately reach a *dead marking*. At least one transition is enabled at every reachable marking.
Dead Transition	A Petri net has a *dead transition* t, if, and only if t is not enabled at any reachable marking.
Fairness	In a Petri net, transition t is fair when it cannot fire infinitely often while being enabled infinitely often. There are four levels of fairness: *impartial*, *fair*, *just*, and *not fair*.

Analyzing these properties can give a strong indication on the correctness of a model. CPNtools can automatically analyze these properties for a given model.

D.3.3 *Performance Analysis and Monitoring*

It is not always sufficient to know if a model is functioning as expected. Sometimes it may be required to determine if a model works effectively. CPNtools can be used to perform simulation-based performance analysis of a given model. During a simulation run, quantitative information on the system is being collected. This includes details such as queue lengths, response times, throughput times, workloads, etc. To improve effectiveness of performance analysis, time can be added to a CPN model to specify how much time passes between ac-

tivities.

To track specific details of a model in simulation mode, monitors can be attached to different components in a CPN model. For example, a monitor can be attached to a transition, in which the details of the transition are written to a file on disk. There are different types of monitors for which custom code can be written to modify their behavior.

D.3.4 *Standard Meta Language*

Declarations and coding in CPNtools is based on Standard Meta Language (SML)[2]. SML is a general-purpose, modular, and strongly-typed functional programming language [27].

2 http://sml-family.org/

E

APPENDIX E - MODELING/VALIDATION DETAILS

This Appendix provides additional details for the attack modeling and validation that has been performed for this research study (see chapter 6). Details such as decisions, assumptions, references to the LoRaWAN specification, sources-code, functions, log definitions, and assessment details for the LoRaWAN specification v1.1 (released on October 17th 2017).

E.1 BEACONING

E.1.1 *Specifications, Decisions & Assumptions*

This section contains all specification references [20] [21], decisions, and assumptions that are made while constructing the CPN model.

Table E.1: Beaconing specification references

#	Description
BS01	Beacon timing [20] (page 45,55). Beacon_period: 128s, Beacon_reserved: 2.120s, Beacon_guard: 3.000s, Beacon_window: 122.880s
BS02	Network frame details [20] (page 52) [NetID, Time, CRC, GwSpecific, RFU, CRC]
BS03	Ping slot timing and calculations [20] (page 46)
BS04	Used bitrate of 250 bps - 5 kbps [21] (page 8)

Table E.2: Beaconing decisions

#	Description
BD01	Hex values are represented as strings (*BA02*).
BD02	A CPNtools time-unit represents a millisecond. (*BS01*,*BS04*)
BD03	Beaconing frame transmission delay: beaconing frame is 16 bytes (*BS02*). Maximum delay:128bits/250bps=0.512s Minimal delay:128bits/5470bps=0,023s (*BA03*)
BD04	Beaconing frames are replaced with malicious beaconing frames at random.
BD05	Malicious beaconing frames are inserted per a predefined list.
BD06	CRCs will not be calculated and used in simulation (CPNtools limitation) (*BA06*).
BD07	A standard multi-channel LoRaWAN setup is used for beacon transmission. No specific high-speed single-channel configuration.

Beaconing decisions – continued

#	Description
BD08	Class A mode is used for the end-device.
BD09	A standard uniform distribution is used for randomness in transmission delays. This represents the different data rates that are defined in the LoRaWAN specification (*BS04*).

Table E.3: Beaconing assumptions

#	Description
BA01	There are four major components: end-device, gateway, network server, and attacker.
BA02	Representing network frame field values as strings and integers does not have an impact on findings. CPNtools has limited data types and functions.
BA03	When transmitting packets, a delay should be included, based on bit rates and network frame size (*BS04*).
BA04	Only the first beaconing frame, in a beaconing receive-window, will be processed by the end-device. All preceding beaconing frame will be ignored.
BA05	The aes128_encrypt function is not very important for this model, as long as it returns a similar repetitive response for the same input. The response of this function is only used for pseudo-randomness in this model (*BS03*).
BA06	CRCs are not relevant for this finding.

E.1.2 *Declarations*

In CPNtools, declarations must be made, which can be assigned to places, transitions, and arcs in the CPN model. These top-level declarations are based on *BS02*.

```
colset NetID = string;
colset BeaconTime = int timed;
colset CRC = string;
colset InfoDesc = int;
colset Latitude = string;
colset Longitude = string;
colset RFU = int;

colset Info = product Latitude * Longitude;
colset GwSpecific = product InfoDesc * Info;
colset BeaconFrame = product NetID * BeaconTime * CRC *
GwSpecific * RFU * CRC timed;
```

```
var gw_specific : GwSpecific;
var beacon_frame : BeaconFrame;
```

Gateway
The declarations required for the gateway sub-model are partly based on *BS02*. A beacon interval (beacon_period) is specified, functions are created for adding time and location details to a beaconing frame, and declarations are created to include delays in transmitting the beaconing frame (*BD03*).

```
val beacon_period = 128000;

fun mTime() = IntInf.toInt( time() );
fun time2beacon(a:BeaconFrame) =
(#1 a,mTime(),#3 a,#4 a,#5 a,#6 a);
fun location2beacon(a:BeaconFrame, b:GwSpecific) =
(#1 a,#2 a,#3 a,b,#5 a,#6 a);

colset TransmissionDelay = int with 23..512;
fun transmissionDelay() = TransmissionDelay.ran();
```

Attacker
The attacker sub-model declarations are partly based on *BD04* and *BD05*. A random number is required for *BD04*.

```
colset Attack = int with 0..10;
fun attackRandomizer() = Attack.ran();
var attack : Attack;

var malicious_frame : BeaconFrame;

fun replaceTime(a:BeaconFrame,b:BeaconFrame) =
(#1 a, #2 b, #3 a, #4 a, #5 a, #6 a);
```

End-device
Declarations required for the end-device sub-model are partly based on *BS03*, *BA04*, and *BA05*.

```
val beacon_reserved = 2120;
val Key = "00000000000000000";
val DevAddr = "26011963";
val pingNb = 4;
val pingSlots = 4096;
val slotLen = 30;

colset PingSlot = int;
colset PingSlots = list PingSlot;
colset WindowNumber = int;
```

```
var ping_slots : PingSlots;
var window_nr : WindowNumber;

fun inWindow() =
if mTime() mod beacon_period < beacon_reserved
then true else false;
fun windowNr() = mTime() div beacon_period;
fun sum [] = 0 | sum (x::xs) = valOf(Int.fromString (
Char.toString x)) + sum xs
fun aes128_encrypt( k:string, m:string ) : string =
Int.toString( sum( explode (k^m) ) );
fun rand( s:string, p:int ) : int = valOf( Int.fromString (
substring( s, p, 1 ) ) );
fun pingPeriod() = pingSlots div pingNb;
fun pingOffset(bt:BeaconTime) : int = (rand( aes128_encrypt( Key,
Int.toString(bt)^DevAddr ), 0 ) + (rand( aes128_encrypt( Key,
Int.toString(bt)^DevAddr ), 1 )*256) ) mod pingPeriod();
fun pingSlot(bt:BeaconTime,n:int): int = bt + beacon_reserved +
( pingOffset(bt) + (((n-1)*pingPeriod())*slotLen ));
fun calculatePingSlots(bt:BeaconTime,nr:int) = if (nr=0) then []
else pingSlot(bt,nr)::calculatePingSlots(bt,nr-1);
```

Network server

This network server model uses many of the same declarations that are used by the end-device. These are mainly used for ping slot calculations. Additionally, the below declaration is used for the network server model.

```
var beacon_time : BeaconTime;
```

E.1.3 *Functions*

In this CPN model, *functions* are defined. Functions provide additional custom functionality within the model. The functions for this model are listed in table E.4.

Table E.4: Beaconing function(s)

FUNCTION	DESCRIPTION
mTime	Returns the current time-unit as an Integer.
time2beacon	Adds the current time-unit as time field value to a beaconing frame.
location2beacon	Adds the gateway location information to a beaconing frame.
transmissionDelay	Returns a random number, within a specified range, which represents a transmission delay in milliseconds.

Beaconing function(s) – continued

FUNCTION	DESCRIPTION
replaceTime	Replaces the time field value in a malicious beaconing frame with the time field value of a regular beaconing frame.
inWindow	Determines if a beaconing frame was received inside the beacon receive-window.
windowNr	Determines the number of the receive-window, which is used for processing only a single beaconing frame in a beacon receive-window.
sum	Calculates the sum of integers in a string.
aes128_encrypt	Returns a simplified aes128 encrypt result (due to CPNtools limitations).
rand	Returns a specific Integer entry from a provides string with numbers (*BS03*).
pingPeriod	Calculates period of the device receiver wake-up expressed in number of slots (*BS03*).
pingOffset	Randomized offset computed at each beacon period start (*BS03*).
pingSlot	Calculates the n^{th} ping slot for a given beacon time (*BS03*).
calculatePingSlots	Calculates the start times of the ping slots. This calculation is performed on both the end-device and the network server to keep the ping slots in sync. These ping slots will get out-of-sync when the end-device calculations are based on a different beaconTime value.

E.1.4 *Log monitors*

In this CPN model, there are *write-to-file* monitors defined. These types of monitors can be configured to write specific data to a log file when a certain condition is met. A log file has the same name as its monitor name, and is created as a .txt file. The monitors for this model are listed in table E.5.

Table E.5: Beaconing monitor(s)

TRANSITION	MONITOR	DESCRIPTION
BroadcastedFrame	BroadcastedFrame	Log all beaconing frames that are transmitted.
replace_frame	ReplacedFrame	Log all frames that are replaced with a malicious frame.
insert_frame	InsertFrame	Log all frames that are inserted.

Beaconing monitor(s) – continued

TRANSITION	MONITOR	DESCRIPTION
reject_frame	RejectedFrame	Log all frames that are rejected because another beaconing frame has already been processed in the same beacon receive-window, or beaconing frames that are received outside the beaconing receive-window.
accept_frame	AcceptedFrame	Log all frames that are used for ping slot calculations.
receive_slots	ReceiveSlot	Log all calculated ping slots for the end-device.
send_slots	SendWindow	Log all network server calculated ping slots.

E.1.5 *State space properties*

The state space properties for this model are summarized in table E.6.

Table E.6: Beaconing State Space properties

PROPERTY	DESCRIPTION
Boundedness	This CPN model is *k-bounded*, where k is equal to thirteen. The model has been constructed to generate a maximum of ten regular beaconing frames and three additional malicious frames that are inserted. Without the maximum of ten regular beaconing frames, the net would be *unbounded*.
Dead Marking	This CPN model is finite (it is terminating after ten regular beaconing frames), therefore it results into dead markings in which no transition is enabled anymore.
Terminating	This CPN model is finite (it is terminating after ten regular beaconing frames), therefore it will eventually reach a dead marking which makes the net terminating.
Liveness	This CPN model is finite (it is terminating after ten regular beaconing frames), therefore none of the transitions can become enabled over-and-over again. (it is terminating)
Home Marking	This CPN model uses time and therefore it is not possible to return to the initial marking (time passes).
Reversibility	The initial marking of this CPN model is not a home marking, therefore this CPN model is not reversible.

Beaconing State Space properties – continued

PROPERTY	DESCRIPTION
Deadlock freedom	This CPN model has dead markings, and therefore it does not have deadlock freedom. There will be a marking in which no transition is enabled anymore.
Dead Transition	This CPN model does not contain any dead transitions. There is no transition that is not enabled at a certain moment in time. All transitions are enabled at least once.
Fairness	Fairness is not applicable because this CPN model is finite.

E.1.6 *LoRaWAN v1.1 specification*

This new version of the LoRaWAN specification contains several changes related to beaconing. None of these changes have an impact on this finding. The related changes are listed in table E.7.

Table E.7: LoRaWAN v1.1 beaconing changes

#	DESCRIPTION
A	In v1.0.2, **all** gateways **must** participate in providing a time-synchronization mechanisms [20] (page 51). In v1.1, gateways **may** participate in providing a time-synchronization mechanisms [22] (page 80). *Impact on findings: none*
B	In v1.0.2, an end-device can operate in Class-B mode **without out-of-band** information from the network server [20] (page 40). In v1.1, an end-device can **only** operate in Class-B mode when **additional information** is made available from the network server using **out-of-band** communications [22] (page 68). *Impact on findings: none*
C	In v1.0.2, the first field in the beaconing frame (BCNPayload) is the **NetID** (6 bytes) [20] (page 51). In v1.1, the first field in the beaconing frame (BCNPayload) is the **RFU** (6 bytes) [22] (page 80). Note: page 68 still refers to all gateways. *Impact on findings: unclear, seems to make the finding even worse since manipulated beaconing frames are not bound to a single network anymore*
D	In v1.0.2, the Time field value in the beaconing frame (BCNPayload) is based on the number of seconds since **00:00:00 Coordinated Universal Time (UTC), 1 January 1970** [20] (page 52). In v1.1, the Time field value in the beaconing frame (BCNPayload) is based on the number of seconds since **00:00:00, Sunday 6th of January 1980 (start of the GPS epoch) modulo 2^{32}**. [22] (page 80). *Impact on findings: none (end-device and gateway must use same standard)*

LoRaWAN v1.1 beaconing changes – continued

#	Description
E	In v1.0.2, the first CRC field value in the beaconing frame (BCNPayload) is based on the based on **NetID** and Time field values [20] (page 52). In v1.1, the first CRC field value in the beaconing frame (BCNPayload) is based on the based on **RFU** and Time field values [22] (page 80). *Impact on findings: none (end-device and gateway must use same standard)*
F	In v1.0.2, beacon precise timing (chapter 15.4) is based on the Time field value **with** NwkID [20] (page 52). In v1.1, beacon precise timing (chapter 15.4) is based on the Time field value **without** NwkID [22] (page 82). *Impact on findings: none, increased chance for beaconing frame collisions*
G	In v1.0.2, the TBeaconDelay is a network specific delay in the **[0:50] ms** range [20] (page 54). In v1.1, TBeaconDelay is **1.5 mSec +/- 1uSec** delay [22] (page 82). *Impact on findings: none, increased chance for beaconing frame collisions*

E.2 DOWNLINK ROUTING

E.2.1 *Specifications, Decisions & Assumptions*

This section contains all specification references [20] [21], decisions, and assumptions that are made while constructing the CPN model.

Table E.8: Downlink routing specification references

#	Description
RS01	(Uplink) network frame details [20] (page 14) [MType, RFU, Major]
RS02	Used bitrate of 250 bps - 5 kbps [21] (page 8)
RS03	Network downlink route update requirements [20] (page 54)
RS04	Uplink FCnt is incremented by the end-device, and the network server tracks the uplink FCnt. [20] (page 18)
RS05	For each end-device a security context is maintained, which includes frame counters. [20] (page 34)

Table E.9: Downlink routing decisions

#	Description
RD01	Hex values are represented as strings (*RA02*).
RD02	A CPNtools time-unit represents a millisecond. (*RS02*)

Downlink routing decisions – continued

#	Description
RD03	Frame transmission delay: network frame is 17 bytes (*RS02*). Maximum delay:136bits/250bps=0.544s Minimal delay:136bits/5470bps=0,025s (*RA03*)
RD04	Frame transmission delay is split in two parts, a short one (5..10ms) between end-device and attacker, and the remaining delay between attacker and original gateway.
RD05	A standard multi-channel LoRaWAN setup is used for data transmission. No specific high-speed single-channel configuration.
RD06	Similar to the security context (*RS05*), a routing table is maintained from the moment the end-device joined the network.
RD07	A TCP/IP frame is represented in a very simplified notation: (Source-IP, Destination-IP, Data)
RD08	MICs will not be calculated and used in simulation (CPNTools limitation) (*RA06*).
RD09	Transmission delays between gateways and network server are considered to be identical and thus not relevant for this CPN model. The attack is limited to the eavesdropping and replay of network frames, and a high(er)-speed network between attacker and gateway.
RD10	Class A mode is used for the end-device.
RD11	A standard uniform distribution is used for randomness in transmission delays. This represents the different data rates that are defined in the LoRaWAN specification (*RS02*).

Table E.10: Downlink routing assumptions

#	Description
RA01	There are four major components: end-device, gateway, network server, and attacker.
RA02	Representing network frame field values as strings and integers does not have an impact on findings. CPNtools has limited data types and functions.
RA03	When transmitting packets, a delay should be included, based on bit rates and network frame size (*RS02*).
RA04	The attacker is able to setup the required high-speed network link to forward the eavesdropped frames to a malicious end-device or gateway.
RA05	The Network Server performs frame de-duplication.
RA06	Only DevAddr and FCnt network frame field values are relevant for this finding.

E.2.2 *Declarations*

In CPNtools, declarations must be made, which can be assigned to places, transitions, and arcs in the CPN model. These top-level declarations are based on *RS01*.

```
colset MHDR = string;
colset DevAddr = string;
colset FCtrl = string;
colset FCnt = int;
colset FPort = string;
colset FRMPayload = string;
colset MIC = string;
colset IP = string;
colset DataFrame = product MHDR * DevAddr * FCtrl *FCnt * FPort *
FRMPayload * MIC timed;
colset IPFrame = product IP * IP * DataFrame;
var data_frame : DataFrame;
var ip_frame : IPFrame;
```

End-Device
The end-device sub-model declarations are partly based on *RS02*, *RS04*, and *RD03*.

```
colset TransmissionDelay = int with 5..10;
fun transmissionDelay() = TransmissionDelay.ran();
fun incrementFCnt(a:DataFrame) =
(#1 a,#2 a,#3 a,(#4 a)+1,#5 a,#6 a,#7 a);
```

Attacker
The declarations required for the attacker sub-model are partly based on *RS02*, *RD03*, *RD04*, *RD05*, and *RA04*.

```
colset TransmissionDelay2 = int with 20..534;
colset AttackerTransmissionDelay = int with 1..534;
fun transmissionDelay2() = TransmissionDelay2.ran();
fun attackerTransmissionDelay() = AttackerTransmissionDelay.
ran();
```

Network Server
The network server sub-model declarations are partly based on *RS04*, *RS05*, and *RA05*.

```
colset SecurityContext  = product DevAddr * FCnt;
colset DownlinkRoutingTable = product DevAddr * IP;
var security_context : SecurityContext;
var downlink_routing : DownlinkRoutingTable;
fun updateDownlinkRoutingTable(a:IPFrame,b:DownlinkRoutingTable)
= ((#1 b),(#1 a));
```

```
fun updateFCnt(a:IPFrame,b:SecurityContext) =
((#1 b),(#4(#3 a)));
fun frameCounterOK(a:IPFrame,b:SecurityContext) =
if ((#1 b)=(#2 (#3 a)) andalso (#4(#3 a))>(#2 b))
then true else false;
fun frameCounterNOK(a:IPFrame,b:SecurityContext) =
if ((#1 b)=(#2 (#3 a)) andalso (#4(#3 a))<=(#2 b))
then true else false;
```

E.2.3 *Functions*

In this CPN model, *functions* are defined. Functions provide additional custom functionality within the model. The functions for this model are listed in table E.11.

Table E.11: Downlink routing function(s)

FUNCTION	DESCRIPTION
transmissionDelay	Determines a transmission delay value.
incrementFCnt	Increments the frame counter.
transmissionDelay2	Determines a transmission delay value.
attackerTransmissionDelay	Determines an attacker transmission delay value.
updateDownlinkRouting	Determines a transmission delay value.
updateFcnt	Update frame counter in security context.
frameCounterOK	Returns if frame counter in network frame is OK.
framecounterNOK	Returns if frame counter in network frame is not OK.

E.2.4 *Log monitors*

In this CPN model, there are *write-to-file* monitors defined. These types of monitors can be configured to write specific data to a log file when a certain condition is met. A log file has the same name as its monitor name, and is created as a .txt file. The monitors for this model are listed in table E.12.

Table E.12: Downlink routing monitor(s)

TRANSITION	MONITOR	DESCRIPTION
transmit_frame	TransmittedFrame	Log all frames that are transmitted by the end-device.
eavesdrop_replay	EavesdroppedFrame	Log all frames that are eavesdropped by the attacker.

Downlink routing monitor(s) – continued

TRANSITION	MONITOR	DESCRIPTION
reject_frame	RejectedFrame	Log all frames that are rejected (incorrect frame counter).
update_context	AcceptedFrame	Log all frames that are accepted.
update_routing	UpdatedRouting	Log all downlink routing updates.

E.3 END-DEVICE ACTIVATION

E.3.1 *Specifications, Decisions & Assumptions*

This section contains all specification references [20] [21], decisions, and assumptions that are made while constructing the CPN model.

Table E.13: End-device activation specification references

#	DESCRIPTION
AS01	OTAA is used for the join process. [20] (page 33)
AS02	Used bitrate of 250 bps - 5 kbps [21] (page 8)
AS03	For each end-device a security context is maintained, which includes frame counters. [20] (page 34)
AS04	Any time the network server processes a Join-Request and generates a Join-accept frame, it shall maintain both the old security context (keys and counters, if any) and the new one until it receives the first successful uplink frame using the new context, after which the old context can be safely removed. [20] (page 34)
AS05	For each end-device, the network server keeps track of a certain number of DevNonce values used by the end-device in the past, and ignores Join-Requests with any of these DevNonce values from that end-device. [20] (page 34)

Table E.14: End-device activation decisions

#	DESCRIPTION
AD01	Hex values are represented as strings (*AA02*).
AD02	A CPNtools time-unit represents a millisecond. (*AS02*)
AD03	Frame transmission delay: Join-Request frame is 23 bytes (*RS02*). Maximum delay:184bits/250bps=0.736s Minimal delay:184bits/5470bps=0,033s (*RA03*)
AD04	Frame transmission delay: Join-Accept frame is 33 bytes (*RS02*). Maximum delay:264bits/250bps=1.056s Minimal delay:264bits/5470bps=0,048s (*RA03*)

End-device activation decisions – continued

#	Description
AD05	Frame transmission delay: data frame is 17 bytes (*RS02*). Maximum delay:136bits/250bps=0.544s Minimal delay:136bits/5470bps=0,025s (*RA03*)
AD06	Class A mode is used for the end-device.
AD07	In this model, MIC is not used to verify Integrity, it is used to determine the used security context during the first upload after a join. A receiving side calculates the MIC of the received frame to determine both integrity and the security context.
AD08	Before starting the OTAA, the DevEUI, AppEUI, AppKey and NetID are communicated out-of-band.
AD09	A TCP/IP frame is represented in a very simplified notation: (Source-IP, Destination-IP, Data)
AD10	Transmission delays between gateways and network server are considered to be identical and thus not relevant for this CPN model. The attack is limited to the eavesdropping and replay of frames.
AD11	A standard uniform distribution is used for randomness in transmission delays. This represents the different data rates that are defined in the LoRaWAN specification (*AS02*).

Table E.15: End-device activation assumptions

#	Description
AA01	There are four major components: end-device, gateway, network server, and attacker.
AA02	Representing network frame field values as strings and integers does not have an impact on findings. CPNtools has limited data types and functions.
AA03	When transmitting packets, a delay should be included, based on bit rates and network frame size (*RS02*).
AA04	More than one security context are maintained, until confirmation with an uplink (or removed after a time-out). When only a single new security context is maintained, this can be overwritten by a replay attack and making the initial OTAA useless.

E.3.2 *Declarations*

In CPNtools, declarations must be made, which can be assigned to places, transitions, and arcs in the CPN model. These top-level declarations are based on *AS02*.

```
colset MHDR = string;
colset AppEUI = string;
colset DevEUI = string;
colset DevNonce = string;
colset MIC = string;
colset AppNonce = string;
colset NetID = string;
colset DevAddr = string;
colset DLSettings = string;
colset RxDelay = string;
colset CFList = string;
colset FCtrl = string;
colset FCnt = int;
colset FPort = string;
colset FRMPayload = string;
colset JoinRequest = product MHDR * AppEUI * DevEUI * DevNonce *
MIC timed;
colset JoinAccept = product AppNonce * NetID * DevAddr *
DLSettings * RxDelay * CFList timed;
colset PHYPayload = product MHDR * DevAddr * FCtrl * FCnt * FPort
* FRMPayload * MIC timed;

var join_request : JoinRequest;
var join_accept : JoinAccept;
var phy_payload : PHYPayload;
```

End-device

The end-device sub-model declarations are partly based on *AD01*, *AD03*, and *AD07*.

```
colset AppKey = string;
colset AppSKey = string;
colset NwkSKey = string;
colset FCntUp = int;
colset FCntDown = int;
colset AppNonce = string;
colset Interval = int timed;
colset JRTransmissionDelay = int with 33..736;
colset SecurityContext = product DevAddr * AppNonce * AppSKey *
NwkSKey * FCntUp * FCntDown;

var net_id : NetID;
var dev_nonce : DevNonce;
var app_key : AppKey;
var interval : Interval;
var security_context : SecurityContext;

fun JRtransmissionDelay() = JRTransmissionDelay.ran();
fun generateNwkSKey(a:AppKey,b:AppNonce,c:NetID,d:DevNonce) =
"0x01"^substring(b,2,3)^substring(b,2,3)^substring(b,3,3)^
substring(c,3,3)^substring(a,18,3)^substring(b,3,3)^
```

```
substring(b,3,3)^substring(d,2,4)^substring(a,15,2)^
substring(a,8,3);
fun generateAppSKey(a:AppKey,b:AppNonce,c:NetID,d:DevNonce) =
"0x02"^substring(b,4,3)^substring(c,2,3)^substring(b,2,3)^
substring(c,3,3)^substring(a,2,3)^substring(b,3,3)^
substring(b,2,4)^substring(d,2,3)^substring(a,13,3)^
substring(a,18,2);
fun ja2sc(a:JoinAccept,b:AppKey,c:NetID,d:DevNonce) = (#3 a,#1 a,
generateAppSKey(b,#1 a,c,d),generateNwkSKey(b,#1 a,c,d),0,0);
fun increaseSCFCnt(a:SecurityContext) =
(#1 a,#2 a,#3 a,#4 a,(#5 a)+1,#6 a)
fun calculateMIC(a:PHYPayload,b:SecurityContext) = "0x"^
substring((#4 b),2,4)^substring((#6 a),2,4);
fun updatePHYPayload(a:PHYPayload,b:SecurityContext) = (#1 a,
#2 a,#3 a,#5 b,#5 a,#6 a,calculateMIC(a,b))
```

Network server

The declarations for the network server sub-model are partly based on *AS03*, *AA04*, *AS04*, and *AD07*.

```
colset DevNonces = list DevNonce;
colset DevNonceList = product DevEUI * DevNonces;
colset DevAddrList = product DevEUI * DevAddr;
colset AppKeyList = product AppEUI * AppKey;
colset RandomAppNonce = int with 10000..99999;

var dev_nonce_list : DevNonceList;
var dev_addr_list : DevAddrList;
var app_key_list : AppKeyList;
var security_context_unconfirmed : SecurityContext;
var security_context_confirmed : SecurityContext;

fun searchDevNonce(a:DevNonce,b:DevNonces) =
if b=[] then false
else if hd(b) = a then true
else searchDevNonce( a, tl(b) );
fun checkDevNonce(a:JoinRequest, b:DevNonceList) =
if ((#3 a)=(#1 b)) andalso searchDevNonce(#4 a, #2 b)
then true else false;
fun updateDevNonceList(a:JoinRequest,b:DevNonceList) =
( #3 a, tl(#2 b)^^[#4 a] );
fun setSCFCnt(a:SecurityContext,b:PHYPayload) =
(#1 a,#2 a,#3 a,#4 a,(#4 b)+1,#6 a)
fun acceptInitialData(a:SecurityContext,b:PHYPayload,
c:SecurityContext) =
if (#1 a)=(#2 b) andalso (#7 b)=calculateMIC(b,a)
andalso (#1 c)=(#2 b) andalso (#4 b)=0 andalso (#5 c)=0
then true else false;
fun acceptData(a:SecurityContext,b:PHYPayload) =
if (#1 a)=(#2 b) andalso (#4 a)<>"" andalso
(#7 b)=calculateMIC(b,a) andalso (#4 b)=(#5 a)
```

```
andalso (#5 a)<>0
then true else false;
```

E.3.3 *Functions*

In this CPN model, *functions* are defined. Functions provide additional custom functionality within the model. The functions for this model are listed in table E.16.

Table E.16: End-device activation routing function(s)

FUNCTION	DESCRIPTION
JRtransmissionDelay	Determines a transmission delay value for the Join-Request message (*AD03*).
generateNwkSKey	Generates a NwkSKey based on the AppKey and details from the Join-Accept message. Pseudo implementation that implements a repetitive response based on the relevant inputs. Hashing is not available in CPNTools.
generateAppSKey	Generates a AppkSKey based on the AppKey and details from the Join-Accept message. Pseudo implementation that implements a repetitive response based on the relevant inputs. Hashing is not available in CPNTools.
ja2sc	Generate a Security Context entry from the received Join-Accept details.
increaseSCFcnt	Increase the FCntUp frame counter in the security context.
calculateMIC	Calculates MIC based on details from the network frame and security context. Pseudo implementation that implements a repetitive response based on the relevant inputs. Hashing is not available in CPNTools.
updatePHYPayload	Updated PHYPayload network frame with MIC and details from the security context.
searchDevNonce	Check if a value exists in the DevNonce list.
CheckDevNonce	Check if specified DevNonce is already used by specified DevAddr.
updateDevNonceList	Add used DevNonce to DevNonce list to block Join-Request replay attacks (*AS05*).
setSCFCnt	Set the FCntUp frame counter in the confirmed security context.
acceptInitialData	Check if this data uplink transmission is the first after OTAA and that they unconfirmed security context must be processed.
acceptData	Check if this data uplink can be handled with a confirmed security context.

End-device activation function(s) – continued

FUNCTION	DESCRIPTION

E.3.4 *Log monitors*

In this CPN model, there are *write-to-file* monitors defined. These types of monitors can be configured to write specific data to a log file when a certain condition is met. A log file has the same name as its monitor name, and is created as a .txt file. The monitors for this model are listed in table E.17.

Table E.17: End-device activation routing monitor(s)

TRANSITION	MONITOR	DESCRIPTION
send_join_request	TransmittedJoinRequest	Log all transmitted Join-Request messages by the end-device.
process_join_accept	ReceivedJoinAccept	Log all accepted Join-Accept messages by the end-device.
reject_join_accept	RejectedJoinAccept	Log all rejected Join-Accept messages by the end-device.
send_data	TransmittedData	Log all transmitted data frames by the end-device.
reject_join_request	RejectedJoinRequest	Log all Join-Request messages rejected by the network server.
accept_join_request	AcceptedJoinRequest	Log all Join-Request messages accepted by the network server.
send_join_accept	TransmittedJoinAccept	Log all transmitted Join-Accept messages by the network server.
receive_data_1	ReceivedData1	Log all first data uplink transmission after OTAA & confirm security context.
receive_data_X	ReceivedDataX	Log all consecutive data uplink transmissions based on confirmed security context.

E.3.5 *LoRaWAN v1.1 specification*

Several significant changes related to related to end-device activation are included in this new version of the LoRaWAN specification. The most important changes are listed in table E.18.

Table E.18: LoRaWAN v1.1 end-device activation changes

#	Description
A	In v1.0.2, The **AppEUI** is a global application ID in IEEE EUI64 address space that uniquely identifies the entity able to process the JoinReq frame [20] (page 32). In v1.1, The AppEUI is replaced with the **JoinEUI**, which is a global application ID in IEEE EUI64 address space that uniquely identifies the Join Server that is able to assist in the processing of the Join procedure and the session keys derivation [22] (page 47). *Impact on findings: none*
B	In v1.0.2, a Join-Request message contains an **AppEUI**, DevEUI, and DevNonce [20] (page 33). In v1.1, a Join-Request message contains a **JoinEUI**, DevEUI, and DevNonce [22] (page 52). *Impact on findings: none*
C	In v1.0.2, the join procedure consists of two MAC messages exchanged with the server, namely a **Join-request** and a Join-Accept [20] (page 33). In v1.1, join procedure consists of either a **Join or Rejoin-request** and a Join-Accept exchange [22] (page 52). *Impact on findings: none*
D	In v1.0.2, The **NwkSKey** is a network session key specific for the end-device [20] (page 32). In v1.1, the NwkSKey is replaced by a triplet of network session keys: **NwkSEncKey**, **SNwkSIntKey**, and **FNwkSIntKey** [22] (page 49). *Impact on findings: none*
E	In v1.0.2, The **AppNonce** is a random value or some form of unique ID provided by the network server and used by the end-device to derive the two session keys NwkSKey and AppSKey [20] (page 35). In v1.1, The AppNonce is replaced with the **JoinNonce**, is a device specific counter value (that never repeats itself) provided by the Join Server and used by the end-device to derive the session keys FNwkSIntKey, SNwkSIntKey, NwkSEncKey and AppSKey. JoinNonce is incremented with every Join-Accept message [22] (page 54). *Impact on findings: countermeasure against finding*
F	In v1.0.2, a Join-Accept message contains an **AppNonce**, **NetID**, DevAddr, CFList, RxDelay, and CFList [20] (page 34). In v1.1, a Join-Accept message contains a **JoinNonce**, **Home_NetID**, DevAddr, CFList, RxDelay, and CFList [22] (page 53). *Impact on findings: JoinNonce is a countermeasure against finding*

LoRaWAN v1.1 end-device activation changes – continued

#	Description
G	In v1.0.2, the DLsettings contains **RFU**, RX1DRoffset, and RX2 data rate [20] (page 35). In v1.1the DLsettings contains **OptNeg**, RX1DRoffset, and RX2 data rate [22] (page 52). The OptNeg bit indicates whether the Network Server implements the LoRaWAN v1.0 protocol version (unset) or v1.1 and later (set). *Impact on findings: none*
H	In v1.1, a device MAY periodically transmit a **Rejoin-request message** on top of its normal applicative traffic. This Rejoin-request message periodically gives the backend the opportunity to initialize a new session context for the end-device. For this purpose, the network replies with a Join-Accept message. This may be used to hand-over a device between two networks or to rekey and/or change DevAddr of a device on a given network [22] (page 57). *Impact on findings: none*