ChirpOTLE: A Framework for Practical LoRaWAN Security Evaluation

Frank Hessel*
Secure Mobile Networking Lab
Department of Computer Science
TU Darmstadt, Germany
fhessel@seemoo.de

Lars Almon*
Secure Mobile Networking Lab
Department of Computer Science
TU Darmstadt, Germany
lalmon@seemoo.de

Flor Álvarez
Secure Mobile Networking Lab
Department of Computer Science
TU Darmstadt, Germany
falvarez@seemoo.de

ABSTRACT

Low-power wide-area networks (LPWANs) are becoming an integral part of the Internet of Things. As a consequence, businesses, administration, and, subsequently, society itself depend on the reliability and availability of these communication networks.

Released in 2015, LoRaWAN gained popularity and attracted the focus of security research, revealing a number of vulnerabilities. This lead to the revised LoRaWAN 1.1 specification in late 2017. Most of previous work focused on simulation and theoretical approaches. Interoperability and the variety of implementations complicate the risk assessment for a specific LoRaWAN network.

In this paper, we address these issues by introducing ChirpOTLE, a LoRa and LoRaWAN security evaluation framework suitable for rapid iteration and testing of attacks in testbeds and assessing the security of real-world networks. We demonstrate the potential of our framework by verifying the applicability of a novel denial-of-service attack targeting the adaptive data rate mechanism in a testbed using common off-the-shelf hardware. Furthermore, we show the feasibility of the Class B beacon spoofing attack, which has not been demonstrated in practice before.

CCS CONCEPTS

• **Security and privacy** → **Denial-of-service attacks**; **Mobile and wireless security**; • **Networks** → **Mobile networks**; • **General and reference** → *Experimentation.*

KEYWORDS

LoRaWAN, LPWAN, Internet of Things, Security, Framework, Denial-of-Service, Adaptive Data Rate

ACM Reference Format:
Frank Hessel, Lars Almon, and Flor Álvarez. 2020. ChirpOTLE: A Framework for Practical LoRaWAN Security Evaluation. In *13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '20), July 8–10, 2020, Linz (Virtual Event), Austria.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3395351.3399423

1 INTRODUCTION

Modern society increasingly relies on connected smart infrastructure. From the Internet of Things to smart cities and cyber-physical systems, connectivity becomes the key enabler. Low-power wide-area networks (LPWANs) are gaining more and more momentum to support this transition. With technologies like Sigfox, NB-IoT, LTE-M, LoRaWAN and others competing in this field, the question for their security arises and has moved into focus of the scientific community. A thorough analysis of their security properties, potential threats, and countermeasures is fundamental for the resilient and reliable operation of connected infrastructure.

We want to foster the research on LPWAN security with a focus on LoRaWAN. This protocol stands out for its open operator model and open-source software stacks like ChirpStack [6] and The Things Stack [18], which enable community-based networks.

While this accessibility of the technology clearly contributed to LoRaWAN's popularity, interoperability and implementation-specific details come with an additional class of security issues. Most currently known problems affect the LoRaWAN specification in version 1.0 and have been addressed in LoRaWAN 1.1 [13]. Research on this topic relies mostly on theoretical discussion or simulation to confirm potential findings. This focus and the heterogeneous environment of software versions and specifications raise the demand for practical assessment of the feasibility of attacks and countermeasures in LoRaWAN networks.

In this paper, we present ChirpOTLE, a novel LoRa and LoRaWAN security evaluation framework, to ease practical experimentation in testbeds and security assessments in real-world LoRaWAN networks. ChirpOTLE orchestrates distributed off-the-shelf LoRa nodes and comes with the building blocks to rapidly find and validate potential vulnerabilities.

After giving background information on LoRaWAN in Section 2 and presenting related work in Section 3, we make the following contributions:

- First, we present an open-source security evaluation framework for LoRaWAN (Section 4).[1]
- Second, we introduce the concept of wormholes in LoRaWAN to propose a novel denial-of-service (DoS) attack exploiting the adaptive data rate (ADR) mechanism (Section 5).
- Third, we present the first experimental evaluation of the Class B beacon spoofing attack (Section 6).
- Finally, we discuss results and possible countermeasures (Sections 7 and 8).

We conclude and summarize our work in Section 9.

*Both authors contributed equally to this research.

[1] https://github.com/seemoo-lab/chirpotle

WiSec '20, July 8–10, 2020, Linz (Virtual Event), Austria

ACM ISBN 978-1-4503-8006-5/20/07…$15.00
https://doi.org/10.1145/3395351.3399423

2 BACKGROUND ON LORAWAN

LoRaWAN is a specification for an infrastructure to connect sensor nodes with centrally managed applications and for medium access control (MAC) using LoRa as physical layer technology.

A LoRaWAN network consists of end devices (EDs) which communicate with a central network server (NS). Messages are forwarded through a LoRa link between the ED and one or more gateways (GWs) followed by a backing network connection to the NS. Application data is processed on application servers (ASs) after being forwarded to them by the NS. In advance to all other communication, EDs and the NS establish a device session either interactively using a mechanism called over-the-air activation (OTAA) or statically with activation by personalization (ABP).

In this paper, we focus on the wireless LoRa link. By default, all communication is initiated by the ED, which is called Class A operation. The ED transmits an uplink message and then waits for a period d_{rx_1} before opening the first receive window called "rx1". After an additional delay[2] of d_{rx_2}, a second receive window ("rx2") is opened. d_{rx_1} can be configured in a range between 1 s and 15 s, while d_{rx_2} is a fixed value of 1 s [8, Section 5.8]. We use the term *transaction* to refer to an uplink message and the optional downlink response in a corresponding receive window.

The payload of all data messages is encrypted, a message integrity code (MIC) is appended to messages to protect their integrity and authenticity, and distinct frame counters (FCnts) for each communication direction prevent replay attacks.

LoRaWAN manages its MAC layer by the exchange of so-called MAC commands. These MAC commands can either be sent in the payload field of a message, if no application data is pending, or piggy-backed to a data message in a special FOpts field.

Region-specific differences of the specification allow operation on the industrial, scientific and medical (ISM) bands. We focus on the EU868 region. We now introduce two relevant LoRaWAN features for our analysis: ADR and Class B Operation.

2.1 Adaptive Data Rate

Due to the nature of a chirp-spread spectrum (CSS) modulation like LoRa, transmission times grow quadratic with decreasing data rate. The same payload can occupy the medium for milliseconds or seconds, which makes selecting an appropriate data rate (DR) crucial for performance in dense, large-scale networks, especially in constrained ISM bands. LoRaWAN employs ADR to address this issue [8, Section 4.3.1.1]. Its goal is to keep each ED at a certain demodulation margin, which is the set screw between quality of service and efficient bandwidth allocation.

An ED activates ADR by setting the ADR flag in its uplink messages. The NS then collects information on the reception quality for the ED and once enough data is gathered, issues a LinkADRReq MAC command, which contains a target DR and transmission power (TP) for the ED. To cope with frame loss due to collisions, the command can also be used to limit the ED to specific channels and to request redundant transmissions using the nbTrans field. By accepting the request using the LinkADRAns MAC command, the ED adjusts its DR for further transmissions.

To assure connectivity, the ED keeps track of the number of transactions since the last downlink message in a value called ADR_ACK_CNT. If this value exceeds a configurable threshold called ADR_ACK_LIMIT (default: 64), it sets the ADRACKReq flag in each uplink to encourage the NS to schedule a downlink message. Whenever the ED receives a downlink message, it resets its ADR_ACK_CNT to 0. If ADR_ACK_CNT, however, exceeds ADR_ACK_LIMIT + ADR_ACK_DELAY, with the latter being another configurable parameter, the ED starts to increase its TP and DR stepwise every ADR_ACK_DELAY transactions, until it eventually receives a response.

Summarized, the ADR mechanism is device-induced but network-controlled. The LoRaWAN specification does not define the exact algorithm to use on the NS, but Semtech has published a recommendation for an algorithm [10], that has found its way into LoRaWAN software like ChirpStack[3]. It collects server-side signal-to-noise ratio (SNR) measurements for up to 20 frames and uses their maximum to issue the LinkADRReq command. The authors justify the usage of the maximum by interference being the main reason for packet loss. Furthermore and in contrast to LoRaWAN 1.0 [9, Section 4.3.1.1], LoRaWAN 1.1 explicates that ADR should be requested for stationary EDs even if the NS signals its inability to appropriately estimate a good configuration. We show how both of these decisions endanger the reliability of a network under attack.

2.2 Class B Operation

In Class A, downlink traffic must follow uplink messages. Applications with requirements on downlink latency can operate in Class B, whereby EDs open additional receive windows. Temporal synchronization is achieved by transmitting network-global beacons every 128 s from GWs based on a global, GPS-synchronized clock.

The EDs use beacon metadata (time of arrival), beacon payload (GPS timestamp), and device properties (device address) to calculate receive window offsets. Including the address leads to pseudo-random offsets for each ED and reduces collisions.

Since an ED may be within reach of multiple GWs and to allow the coexistence of networks from different operators, all beacons are sent simultaneously to aim for constructive interference [8, Section 15.2, Note 1]. As a direct consequence, the payload of beacons cannot undergo meaningful encryption nor authentication, since keys would need to be available for all network operators.

To enable Class B, the ED starts by searching and locking to beacons frames. Once a stable lock exists, it transmits all further uplink messages with the ClassB flag set to notify the NS about the class switch. This process can be sped up by requesting the current time from the NS using the DeviceTimeReq MAC command, allowing the ED to estimate the next arrival of a beacon. When the lock is lost, the ED runs at least two hours of "beacon-less operation", in which it gradually widens all Class B receive windows before eventually switching back to Class A [8, Section 5.12].

3 RELATED WORK

We present the relevant related work in three parts: Studies on ADR performance, on jamming and wormholes, and on beacon spoofing.

Most previous work on ADR covers performance aspects. Li et al. give a simulation-based performance evaluation of Semtech's ADR

[2] In contrast to the use of RECEIVE_DELAY2 in the LoRaWAN specification, we define d_{rx_2} as the delay *between* receive windows.

[3] cf. https://github.com/brocaar/chirpstack-network-server/blob/v3.6.0/internal/adr/

algorithm with a focus on time to convergence [16]. Their results show that approaching convergence from high DRs is notably more time-consuming than from low DRs, meaning that over-optimizing the DR causes a prolonged decline of ED availability.

Alternative server-side algorithms for ADR have been proposed for a variety of optimization goals. Bor and Roedig suggest using a probing-based mechanism to determine the best DR for each device [5]. Most other approaches, however, strive for global optimization by considering all EDs within a certain region. Reynders et al. propose a cell-based algorithm that aims for fairer packet error rates in the outer region of a cell based on node distance or path-loss estimation [19]. Abdelfadeel et al. extend this work and suggest to use fairness regarding equal overall collision probability as a goal [1]. Cuomo et al. designate the greediness of the default ADR algorithm to use high DRs as a main cause for network congestion and also propose two alternative schemes. They exploit the orthogonality of different spreading factors (SFs) to equalize DR usage either by the number of devices or by expected air time [12]. To our best knowledge, none of the proposed algorithms considers the presence of intentionally manipulated SNR measurements.

Due to its CSS modulation, jamming LoRa is most effective by exploiting coexistence issues with CSS signals. Goursaud and Gorce compare CSS to ultra-narrow band (UNB) modulation and show that LoRa mainly interferes with other LoRa signals [15]. They prove that interference between LoRa frames of different SFs is low while a signal of the same SF has to be at least 6 dB stronger to demodulate it by benefiting from the capture effect. Croce et al. underline these findings by simulations and experiments and also point out that orthogonality between SFs is limited [11].

Aras et al. use this knowledge to create a LoRa jammer based on off-the-shelf hardware which can perform triggered and payload-based reactive jamming [2, 3]. They also evaluate a unidirectional store-and-forward wormhole based on two nodes which handles frames of SF10 and above reliably.

Miller mentions a beacon spoofing attack shortly after the publication of LoRaWAN 1.0.0, either to manipulate the beacon's payload or to tamper with the calculation of receive windows [17]. Van Es et al. formally verify the possibility of injecting malicious beacons with modified time values to provoke calculating invalid receive window offsets on the EDs [22]. Yang et al. also discuss the impact of modification to different beacon payload fields [24]. Contrary to their proposition, it is not possible to change the wakeup periodicity and drain ED's battery by changing the beacon payload. Butun et al. assess beacon spoofing to be relevant for LoRaWAN 1.1 [7].

4 SECURITY EVALUATION FRAMEWORK

LoRaWAN is a distributed and heterogeneous system built on an infrastructure consisting of NSs, ASs, and GWs. The different protocols and the network infrastructure connecting all of these components provide many options for conducting a security evaluation, but also demand for a specific attacker model when it comes to security evaluation.

4.1 Attacker and Threat Model

We limit our scope to the wireless link between EDs and GWs. This decision comes with the least preconditions in the attacker model, as the wireless communication channel is, by its nature, accessible to everyone within proximity.

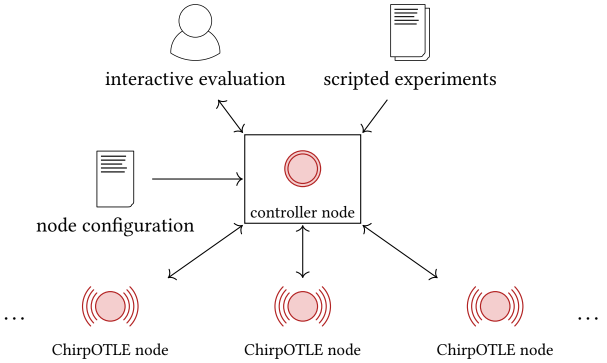

Figure 1: Architecture of the ChirpOTLE framework

Furthermore, we limit the attacker to use only inexpensive off-the-shelf LoRa hardware to emphasize the low requirements for reproducing the attacks. Using only LoRa transceivers intended for use in EDs, however, comes with challenges: Chips like Semtech's SX127x series can only handle one channel and DR at a time.

We assume the attacker to be able to transmit and receive arbitrary LoRa frames on a given channel. The location of devices, network configuration, and other publicly observable metadata like uplink periodicity of EDs is assumed to be known by the attacker. However, the attacker is not in possession of any cryptographic keys or other data that is marked as protected in the LoRaWAN specification, nor she is able to break cryptographic primitives.

4.2 Framework Design and Architecture

Based on this attacker model, we design and implement ChirpOTLE, our LoRa and LoRaWAN security evaluation framework. The main challenge is the physical layer interaction within the coverage area of the network under test. Furthermore, attacks involving wormholes require forwarding frames between different locations, while some decisions are made and actions are run locally on the node, due to strict timing constraints. Selective jamming, for instance, needs quick decisions for or against jamming a frame while it is still in transmission. From these requirements, we identify two main components of the framework: First, a set of LoRa field nodes with real-time support and a network interface for configuration and out-of-band communication. Second, a controller that orchestrates their actions in complex interaction patterns.

Figure 1 gives a high-level overview of the resulting architecture. Using a flexible, file-based node configuration allows fast adaption to given network topology and available hardware. With Python as a high-level language on the controller node, we can use its built-in REPL interface for interactive assessment of vulnerabilities and evaluation of new attacks. This environment inherently allows running scripts to generate quantitative experimental results without manual intervention.

To fulfill the real-time requirements and to address a great variety of off-the-shelf hardware, we implement a low-level *companion application* based on RIOT [4], an operating system for microcontroller

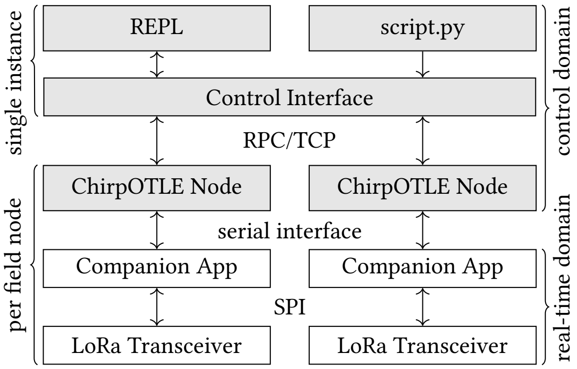

Figure 2: Control flow in the security evaluation framework

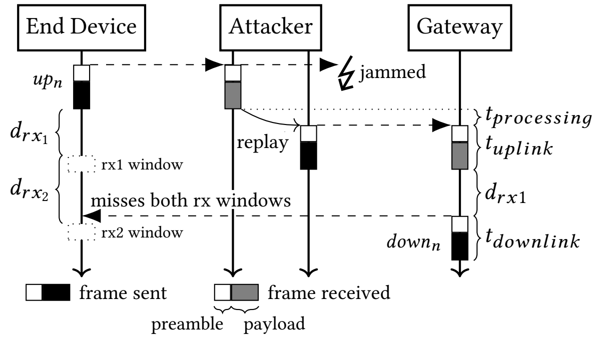

Figure 3: Message flow of the unidirectional wormhole

units (MCUs). This application connects to SX127x-compatible LoRa radios through SPI and provides the host with a serial interface for higher-level commands like switching the channel or activating a jammer.

To combine both, the low-level MCU application and the high-level scripting environment for experiment design, we use the TPy framework [21]. It simplifies deployment and control in distributed network experiments. Together with the node configuration, its remote procedure call (RPC) abstraction layer hides communication details between controller and ChirpOTLE nodes in the field, as shown in Figure 2. One or more MCUs executing the companion application can be hooked up to a lightweight RPC host, for example, a Raspberry Pi, that handles network communication and translates RPC calls into commands for the serial interface of the MCU.

5 WORMHOLES IN LORAWAN

The concept of wormholes in LoRaWAN is different from the typical understanding, as LoRa is only utilized on a single hop between EDs and GWs. Usually, wormholes exploit a faster out-of-bound connection to relay messages of a multi-hop network, exceeding the network's own forwarding speed. All LoRaWAN wormholes follow a store-and-forward principle to bridge a single hop.

Since communication of LoRaWAN Class A EDs is only uplink-induced, bidirectional LoRaWAN wormholes are constrained by the receive windows of a transaction to forward downlink messages. We use a simple unidirectional wormhole as a foundation to present two more sophisticated, downlink-enabled approaches. We then incorporate them into an attack exploiting the ADR mechanism of LoRaWAN.

5.1 Unidirectional Wormhole

Figure 3 shows the message flow within a unidirectional, uplink-only wormhole: One of the attacker's transceiver acts as the entry node and is placed near the target ED. Another transceiver serves as an exit node near the GWs. A LoRa frame up_n is captured in its entirety by the entry node and forwarded to the wormhole's exit node through an out-of-bound channel. The exit node then replays the message up'_n for the GW to receive.

If the ED is within reach of the GW, the exit node can be configured to selectively jam the reception of frames while the entry node still receives them. This approach differs from the LoRa wormhole introduced in [3] by the jammer's trigger. We discovered that listening on the *exit node* and quickly switching it to jamming is significantly faster than triggering the jammer by the sniffer via network. This allows selective jamming based on the DevAddr even for SF7 and still receiving the messages at the entry node, if the jammer's signal is at least 6 dB weaker at the source [15].

A wormhole cannot alter the LoRaWAN message content, as it is protected by a MIC. However, the attacker can modify metadata of the message, in particular the timing, the location, SNR, and received signal strength indication (RSSI) values. If the device and network under attack both conform to the LoRaWAN 1.1 specification, an attacker has to replay uplink messages on the same channel and DR, as those parameters take part in the MIC calculation in the updated specification [8, Section 4.4.2]. That was not the case in LoRaWAN 1.0.x [9, Section 4.4], which allows replay attacks with different transmission parameters.

5.2 RX2 Wormhole

The attacker cannot replay the downlink message in the rx1 window of a transaction. Figure 3 illustrates how the rx1 window starts at a fixed delay of d_{rx_1} measured from the end of the uplink message up_n. So when receiving the replayed message up'_n, the NS will respond not earlier than d_{rx_1} after that.

The transmission parameters of LoRaWAN downlink messages are not protected by the message's MIC, even in LoRaWAN 1.1 [8, Section 4.4.1]. Therefore, an attacker may aim for the second receive window to forward the downlink message. The additional delay can be exploited to circumvent the timing constraints.

Figure 4 illustrates the principle of the *rx2 wormhole*. Downlink messages are scheduled to match the rx2 window. The diagram also illustrates the remaining timing constraints of such a wormhole. The duration of the replayed uplink t_{uplink}, of the original downlink $t_{downlink}$, and additional time for processing, t_{proc_1} and t_{proc_2}, all together must not exceed the duration of the time between both receive windows:

$$
\begin{aligned}
t_{proc_1} + t_{uplink} + d_{rx_1} + t_{downlink} + t_{proc_2} &\leq d_{rx_1} + d_{rx_2} \\
t_{proc_1} + t_{uplink} + t_{downlink} + t_{proc_2} &\leq 1\,\text{s} \quad (= d_{rx_2})
\end{aligned}
\tag{1}
$$

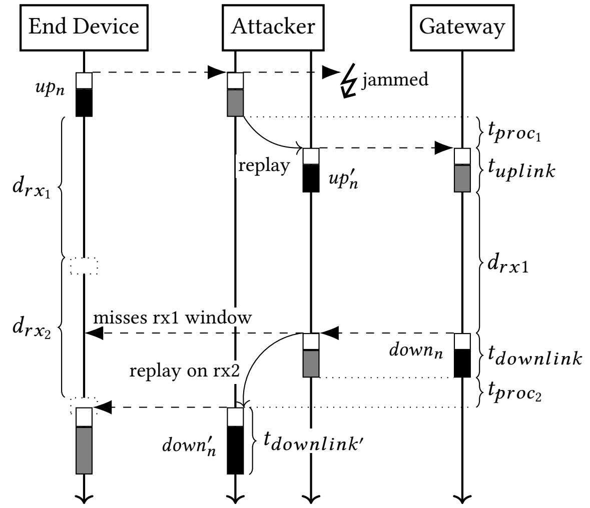

Figure 4: Message flow of the rx2 wormhole

Assuming a negligible processing time and equal frame lengths for uplink and downlink, we get an upper boundary for the transmission time of 500 ms per frame. A LoRa frame containing the LoRaWAN data has a length of at least 12 bytes. Figure 5 puts both of these limitations in relation to the DRs for the EU868 region.

It is evident that the rx2 wormhole breaks the timing constraints for DR0 and DR1. For DR2 and DR3, it can be used if small or medium-sized LoRaWAN payloads are expected. These downsides are compensated by the advantage of keeping both frames in the same transaction. This is in particular important for confirmed uplinks, as the MIC of the corresponding downlink is only valid during the same transaction in LoRaWAN 1.1 [8, Section 4.4.1] as a response to the ACK spoofing attack on LoRaWAN 1.0 [13, 23].

5.3 Downlink-Delayed Wormhole

By expanding the wormhole concept over two consecutive transactions, an attacker achieves the ability to target low DRs at the cost of not being able to handle confirmed uplink messages in LoRaWAN 1.1. We call this approach the *downlink-delayed wormhole*. Figure 6 shows how the uplink message up_n is intercepted, sniffed, and then replayed as up'_n. If the NS responds with a downlink message $down_n$, it is not immediately forwarded but stored by the attacker.

Once the ED transmits the next uplink message up_{n+1}, it is again sniffed and jammed. If a pending downlink message has been stored by the attacker, the priority is to forward it back to the ED through the entry node. As the entire message is already available, the attacker can aim for the rx1 window. When the downlink message has been delivered, the attacker eventually forwards the stored uplink message up_{n+1} through the exit node to the NS, which may return the next pending downlink $down_{n+1}$.

This approach overcomes the issues with high DRs at the cost of crossing transaction boundaries. For multi-channel networks, this also adds complications regarding the downlink FCnt. When staying within the same transaction, a downlink message from the NS always contains the current maximum for the FCnt. This is important since the ED will only accept messages with a higher FCnt. If an attacker cannot observe all channels at all DRs, she may miss a transaction in which a higher downlink FCnt is processed by the ED. In that case, the stored downlink frame of the attacker becomes outdated, as its FCnt prevents it from being accepted.

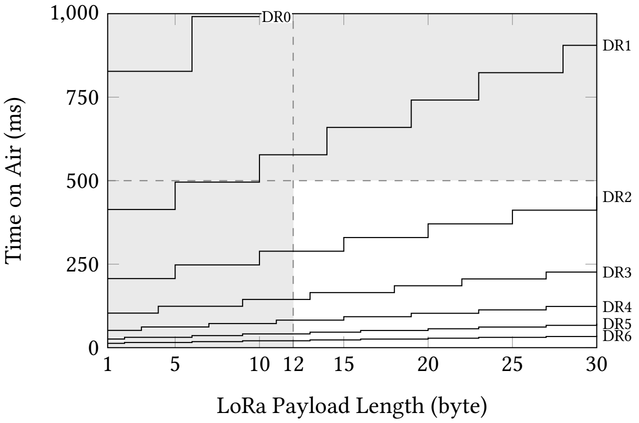

Figure 5: LoRa frame transmission time in relation to DR (EU868) and payload length. LoRaWAN uplink configuration with header and payload CRC on physical layer

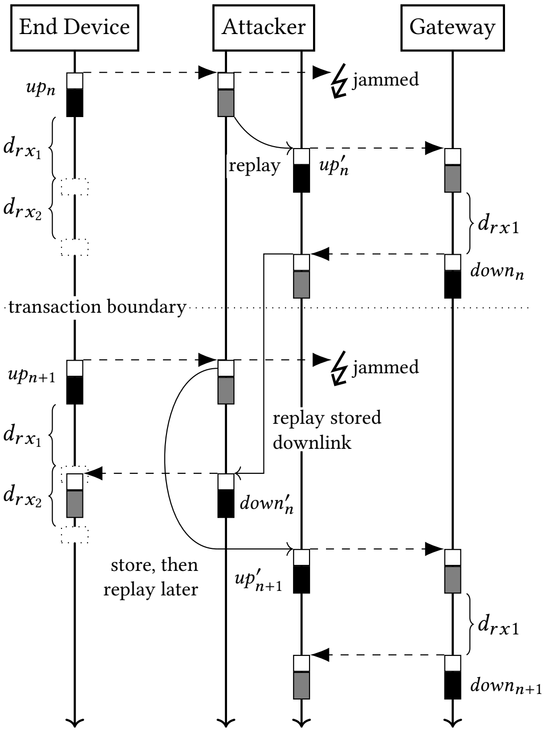

Figure 6: Message flow of the downlink-delayed wormhole

5.4 From Wormholes to ADR Spoofing

The *ADR spoofing attack* exploits the ADR mechanism to force the ED into using a TP and DR at which it is unable to communicate with any GW. An attacker can intentionally create such a situation by selectively forwarding messages through a wormhole to manipulate their metadata. The ED's RSSI and SNR then appear as being higher for the NS. Employing these values in the ADR calculation leads to too optimistic estimates for the target TP and DR.

Once the ED adapted to the new DR, the attacker decides which messages to forward to the NS. To keep the ED in this state, at least one transaction out of ADR_ACK_LIMIT + ADR_ACK_DELAY must pass the wormhole. This prevents the ED from increasing TP and lowering the DR because its ADR_ACK_CNT is reset to 0.

From this description, we can identify two phases of the attack: First, the spoofing phase, in which the attacker intercepts messages on the initial DR, and second, the retention phase, in which the ED is forced to keep its settings by selective forwarding.

While the decision to enable ADR is made on the ED, the choice of the actual parameters is made on the NS and announced in a LinkADRReq MAC command. Therefore, the attacker has two concerns during the spoofing phase: First, to forward uplink frames with good reception parameters to trigger a LinkADRReq command with a high DR, and second, to forward the downlink message containing this command to the ED. This calls for the deployment of a bidirectional wormhole. Since the attack is most effective if the ED can only communicate on low DRs by default, the attacker may need to pick the downlink-delayed variant for spoofing.

Once the ED has processed the LinkADRReq, it immediately switches to the higher DR and confirms the new settings with a LinkADRAns in the next uplink message. The attacker does not forward this message, as a chance exists that the LinkADRReq will remain unacknowledged in the NS's MAC command queue. If that is the case, the NS itself will push the ED back to the higher DR without any additional effort, should a link be reestablished. Since MAC commands are only acknowledged once [8, Section 5, Note 2], this can create a self-reinforcing situation until the LinkADRReq at the NS times out.

To not rely only on the NS, the attacker also creates a wormhole on the "optimized" DR. For this DR, the rx2 wormhole is sufficient, which comes with fewer complications. We assume an ideal case of no communication between the ED and GW without the help of the attacker. The most reliable strategy for the attacker to achieve her goals then would be to selectively forward a transaction if the ADRAckReq flag is set. The NS processes the message and ideally issues a downlink message to reset the ADR_ACK_CNT of the ED. If the ADRACKReq flag is unset in the next transaction, the ED has processed the downlink and ADR_ACK_CNT is reset.

The most critical task for the attacker is to ascertain whether and when the ED changed the DR. The plain LinkADRReq command can only be observed if it is transmitted piggy-backed in a LoRaWAN 1.0 setup. In all other situations, MAC commands are encrypted, which only allows inferring the total length of all MAC commands in the message. Furthermore, observing downlink messages does not guarantee that they are received and processed by the ED. With the given capabilities and a single node, the attacker cannot receive the uplink message containing LinkADRAns after DR adaption and still observe the initial channel. In that case, a probabilistic approach makes sense. With n being the number of channels on the network and f_{up} being the ED's uplink periodicity, we calculate:

$$t_{timeout} = \frac{1}{f_{up}} * \left\lceil \log_{\frac{n-1}{n}}(0.01) \right\rceil \qquad (2)$$

After not receiving messages for $t_{timeout}$, an attacker listening to a single channel can assume that the ED has switched the DR in 99% of all cases and proceed to the retention phase. For a network with 3 channels, this corresponds to 12 uplink messages.

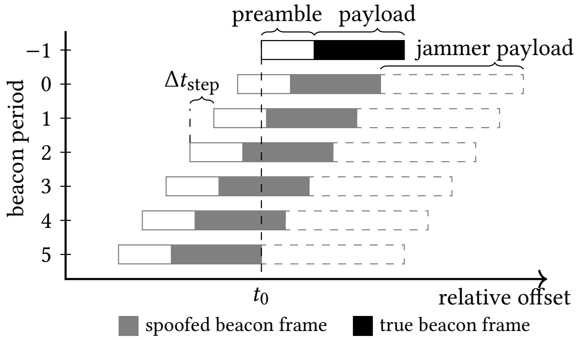

Figure 7: Concept of the beacon drifting attack

6 BEACON SPOOFING

Another attack that creates a DoS situation by tampering with network parameters is beacon spoofing. In contrast to ADR spoofing, beacon spoofing affects all Class B EDs within reach of the attacker. The affected devices lose their ability to receive network-induced downlink traffic.

The concept behind this attack has been discussed [7, 22], but to our knowledge, no proof exists showing its practical applicability. It is based on everyone's ability to create valid Class B beacon frames, which inherently is required for the coexistence of LoRaWAN networks with different operators. If an attacker transmits beacon frames with an offset, all EDs locked to the fake beacon do not open their receive windows on time.

While periodically transmitting frames is a quite feasible task, forcing already locked EDs to switch to the spoofed beacons requires a bit more effort. Locked devices open their beacon receive window only just before they expect to receive a beacon. EDs searching for beacons utilize DeviceTimeReq commands to reduce the size of their beacon search window. Targeting only EDs in the beacon acquisition phase reduces the applicability of the attack drastically. Therefore, we focus on already locked EDs only.

By first locking to the legitimate beacon herself, an attacker synchronizes with the network's time without access to GPS. Then, she starts the beacon drifting as shown in Figure 7. While staying within the receive window tolerance of the EDs, she slowly prepones the transmission of the spoofed beacon by Δt_{step} each beacon period. If the step size is chosen small enough, the EDs shift their beacon receive window together with the spoofed beacon. Once the accumulated drift corresponds to at least an entire beacon frame length, the attacker can stop shifting. Using this stop criterion aims at minimizing the collision time with the legitimate beacon. The beacon frame duration in EU868 is 152.58 ms. For our experiments, we add a margin of 5 symbols, which corresponds to half a beacon preamble, leading to a total drift of 173.06 ms. EDs open their receive windows early by the same offset and close them before a downlink transmission starts.

As the beacon frames are sent in implicit header mode without a CRC on the physical layer, the attacker can bring in another measure to increase her success rate. Bytes added to the end of the beacon frame are discarded by all receivers, as the expected frame length has to be known in this transmission mode. The attacker can exploit this by adding random data as a jamming payload after the beacon data, which then coexists with the legitimate beacon frame, making it harder for EDs to stay locked with it.

If the attacker is successful, the ED sticks to the spoofed beacon as long as the attacker transmits it, because the ED can only detect beacon presence, but not downlink availability. Once the attacker stops transmitting beacons, the EDs perform two hours of beaconless operation and eventually return to Class A.

7 EXPERIMENTAL SETUP

An experimental evaluation of the two attacks requires bringing the ChirpOTLE nodes together with LoRaWAN network entities in a testbed. This section summarizes the network configuration and topology used for our experiments.

7.1 Network Configuration

As the specification does not cover all implementation-specific details for the operation of a network, running an experimental evaluation of LoRaWAN always yields results in the context of the selected software. All experiments in this study use ChirpStack 3.6.

LoRaWAN libraries for EDs come in much greater variety than software for network infrastructure, with a reference implementation being available. We deploy its LoRaWAN 1.1 branch[4] on an ST Nucleo L476 evaluation board with an SX1276MB1xAS LoRa transceiver to create our device under test. The software already comes with sample applications for Class A and B operation.

We configure these applications to use ABP and extend them with remote-control to allow for automation. These modifications do not affect the behavior of the LoRaWAN implementation with the only exception of forcing the `nbTrans` parameter to a constant value of 1. This decision creates comparable conditions for each trial even if the NS decides for a different redundancy configuration.

We limit the NS and ED to use only the three default channels from the EU868 region (cf. ETSI EN-330 200-1, Section 7.2.3 [14]), at 868.1 MHz, 868.3 MHz and 868.5 MHz. This allows verifying that a single LoRa transceiver is sufficient to run attacks against a multi-channel network. In Section 8.1, we extrapolate the insights from our experiments to networks with larger channel lists. On the uplink channels, DR0 to DR5 are enabled, which correspond to SFs 12 to 7, respectively, all at 125 kHz channel bandwidth. For the rx2 downlink window and Class B downlink transmissions, we use the default channel of 869.525 MHz at DR0. `RECEIVE_DELAY1` is set to 1 second.

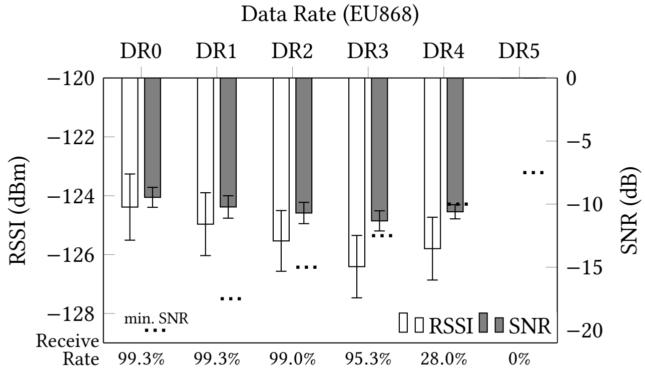

Figure 8: Channel from ED to GW without attacker at highest TP. Required SNR according to [10] (n=300 per DR)

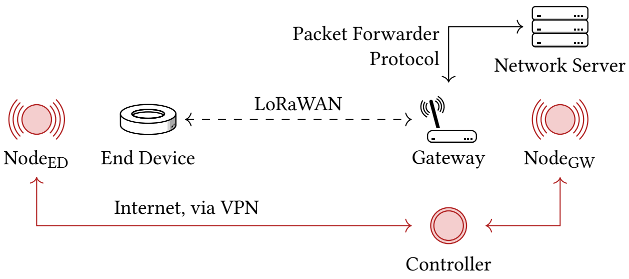
Packet Forwarder Protocol
Network Server
LoRaWAN
Node_{ED}
End Device
Gateway
Node_{GW}
Internet, via VPN
Controller

Figure 9: Experiment topology: network under test and ChirpOTLE nodes and controller (red)

7.2 Topology

The ED and a GW are placed at two locations in a way that the channel attenuates the LoRa signal by at least 6 dB, which is a requirement for jamming-based attacks [15]. This is particularly important if an attacker needs to sniff frames at their source and simultaneously prevent them from reaching their destination.

We also equip the GW's antenna port with an attenuator to fine-tune the SNR for frames from the ED to be just below −7.5 dB, which is the required SNR for receiving on DR5 [10]. This allows us to test the ADR spoofing attack for a variety of initial DRs.

Baseline measurements of the channel properties after applying both measures are depicted in Figure 8. A nearly perfect receive rate for the lower DRs and its drop for DR4 and DR5 show that the setup is tuned well for the experiments.

With the network being configured, we introduce the ChirpOTLE nodes to the topology, as depicted in Figure 9. Both LoRaWAN network entities, the ED and the GW, are each collocated with a single attacker node, called Node_{ED} and Node_{GW}, respectively.

Each ChirpOTLE node consists of a Pycom LoPy 4 connected to a Raspberry Pi. It also runs the TPy *LoRa Node* interface for remote-controlling the LoPy 4. The ChirpOTLE controller, the ChirpStack NS API, and the ED's output are monitored centrally to collect measurements during the experiments.

[4] We use commit 92e37147 of the `feature-5.0.0` branch in https://github.com/Lora-net/LoRaMac-node.

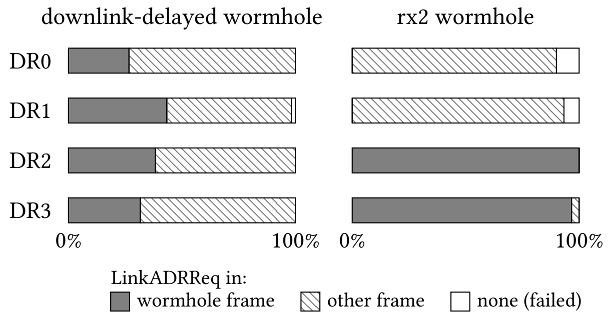

Figure 10: Trigger for the ED to switch its DR (n=60)

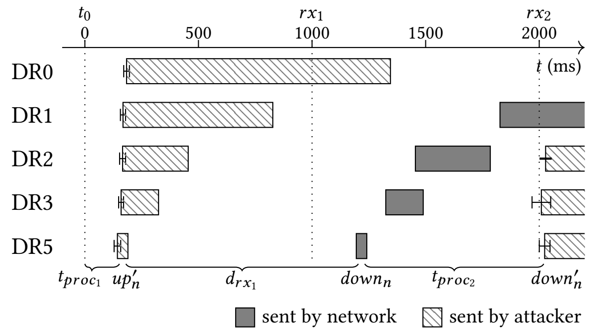

Figure 11: Measured timing of the rx2 wormhole

8 RESULTS

For the evaluation, we ran 20 trials for each configuration of independent variables. Before each trial, the ED was reactivated on the NS using ABP. Then the ED was reset and the attacker's state was cleared to guarantee independence between trials.

8.1 ADR Spoofing

For the wormhole attack on ADR, we use Node$_{\text{ED}}$ as the entry node and Node$_{\text{GW}}$ as the exit node of the wormholes (cf. Figure 9).

We vary the parameters shown in Table 1. We evaluate both wormhole types with all DRs to see if the actual behavior of the network matches our expectation of the rx2 wormhole being incapable of handling DR0 and DR1 traffic. The LoRaWAN application data length is set to a single byte to keep it well in the acceptable range for DR2 and DR3. Changing the number of preceding uplink messages between reset of the ED and start of the attack fills the server-side time series of SNR measurements at different levels.

8.1.1 Wormhole Type and Data Rate. First, we look upon the success of the attacker in the spoofing phase to verify the attack's effectiveness. We define success as the processing of a LinkADRReq with the target DR by the ED. Figure 10 shows that this is achieved in most trials, with a few exceptions for the low DRs and the rx2 wormhole. In all other cases, the ED is forced into the higher DR.

Table 1: Evaluation parameters for the ADR spoofing attack

Parameter	Evaluated Values
spoofing phase: wormhole type	downlink-delayed, rx2
spoofing phase: data rate	DR0, DR1, DR2, DR3
uplinks preceding the attack	1, 10, 20

Table 2: Number of transactions from start of attack until the DR is adjusted to the target DR

Preceding Uplinks	Number of Transactions (mean, SD)	*n*
1	7.74 ± 5.0	158
10	7.58 ± 5.79	153
20	7.92 ± 5.57	158

Figure 10 also depicts how the message containing the critical LinkADRReq command reaches the ED. The results match our expectations about the different capabilities of the wormhole types.

For the rx2 wormhole, a clear distinction exists between DRs. On DR2 and DR3, the attacker could directly forward the message to the ED. Notably, the attack does not fail for the lower DRs. Frames with high SNR values still reach the NS, and even though the downlink traffic cannot be sent back by the wormhole, the NS adds a LinkADRReq to the ED's MAC command queue. From there, it is delivered on a frequency not observed by the attacker.

Figure 11 shows the actual timing of the rx2 wormhole during the attack, also including measurements for DR5 from the retention phase. The attacker is incapable of handling DR1 and DR0 as the downlink frame is not complete when the rx2 window opens. All other DRs leave a sufficiently large margin for processing.

For the downlink-delayed wormhole, the results are different, with roughly a third of the messages reaching the ED through the wormhole for all DRs. The reason for this ratio is the network's channel list's size. For forwarding, the downlink-delayed wormhole needs two consecutive transactions on the same frequency. For our three-channel-network, this happens at 33.3% after observing the first transaction, if transactions are treated independently. So it is reasonable to assume that a full eight-channel-network can be attacked as well with an increase in time to success.

8.1.2 ADR Algorithm State. In the next step, we verify that the previous state of the ADR algorithm on the NS does not affect the attack. Therefore, we vary the number of uplinks preceding the attack and measure the number of transactions before the end device is reconfigured through ADR. The result is shown in Table 2.

As expected, the fill level of the SNR table on the NS did not affect the ADR decision. The reason for this is the usage of the max operator instead of averaging over the collected values.

8.1.3 Retaining the Device. Once the ED processes the LinkADRReq, the attacker proceeds to the retention phase. The initial DR, wormhole type, and ADR algorithm state do not affect success in retaining, so we aggregate results from all trials reaching this phase.

We first evaluate the overall success of the attacker in this phase, which we define by successfully resetting the ADR_ACK_CNT of the ED and by a low ratio of successful uplinks. In all 469 trials that made it into the retention phase, the attacker was able to retain the ED on its high DR. For 95% of trials, the uplink success rate

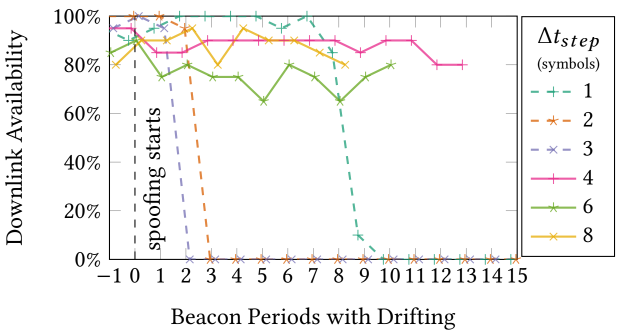

Figure 12: Downlink availability during beacon drifting

dropped below 2.88%, for 99%, it was still below 3.11%. This value cannot reach 0% while remaining successful in keeping the ED in its state, as ADRAckReq flags have to be answered to. For our ED with a low ADR_ACK_LIMIT of 32, nearly all of the uplink messages are intentional passes through the wormhole.

8.1.4 Countermeasures. We have seen that tricking the ED in using unsuitable DRs is possible with a high success rate even for LoRaWAN 1.1. The attack does not have a single enabling vulnerability, but several contributing factors.

The missing relation of messages within a transaction enables the wormhole attacks. As a reaction to ACK spoofing in LoRaWAN 1.1, the uplink FCnt was included in a downlink MIC, but only if the uplink ACK flag is set. If that was the case for all Class A messages, the downlink-delayed wormhole would be prevented. Also, transmission parameters like DR and frequency are only included in the uplink MIC. Including them also in downlink MICs would prevent the rx2 wormhole. The FHDR field containing the ADR and ADRACKReq flags lacks confidentiality protection. Protecting FHDR restrains the attacker from identifying frames that need forwarding.

ADR algorithms are designed with performance in mind, but not security or robustness. For example, averaging the SNR over several messages can prevent abrupt changes from a single data point. If only uplink messages with plausible SNR values are directly answered with LinkADRReqs and a strong relation between messages within a transaction is implemented, the success of ADR spoofing would decrease significantly.

Table 3: Evaluated values for Δt_{step} and estimated duration of the beacon drifting attack

Δt_{step}		full beacon length reached after	
symbols	time	beacon periods	time
1	4.096 ms	38	81:04 min
2	8.192 ms	19	40:32 min
3	12.288 ms	13	27:44 min
4	16.384 ms	10	21:20 min
6	24.576 ms	7	14:56 min
8	32.768 ms	5	10:40 min

Without these changes, a workaround is to deploy a denser GW distribution with fewer EDs being only in the vicinity of a single GW or losing connection on high DRs. This is, however, contradictory to the LPWAN's paradigm of using a sparse infrastructure.

8.2 Beacon Spoofing

Beacon spoofing requires only Node$_{\mathrm{ED}}$ near the target ED. We vary the drifting step size Δt_{step} as shown in Table 3. For each trial, we wait until at least one downlink message is received in a ping slot after resetting the ED to avoid setup problems being falsely attributed to the attacker's success. We quantify Class B downlink availability under attack by queuing a downlink message in each beacon period and counting its arrival at the ED. The measurements include the period directly before the attack and three periods after shifting has stopped.

8.2.1 Impact of Step Size. Our results in Figure 12 reveal two outcomes: For Δt_{step} of 1, 2, and 3 symbols, downlink availability drops significantly after 9, 3, and 2 beacon periods, respectively. For greater values of Δt_{step}, it remains mostly unaffected at roughly 80% and above. We conclude that the attack fails if the beacon is shifted too aggressively, most likely by exceeding the ED's receive window tolerance.

The downlink degrades faster for higher values of Δt_{step}. Relating Δt_{step} and the beacon period during which the communication breaks down yields a threshold of around 9 symbol lengths. Once the beacon is shifted further, the timing of the downlink windows between the ED and NS diverges too far to communicate.

We want to verify that the degrade in downlink quality is indeed caused by the ED being locked to the spoofed beacon. Transmitting a distinct value in the attacker's frames allows distinguishing them from the true ones. The GwInfo field is well-suited for this purpose, as it does not take part in the calculation of downlink windows.

Figure 13 shows the received beacon type over time for each value of Δt_{step}. The ED may either receive the true, or the spoofed beacon, or no beacon at all. If a beacon frame cannot be decoded correctly, we count it as lost, since ping slot offsets cannot be calculated without the payload. Consistent with our previous results, we see that the ED is locked to the spoofed beacon if $\Delta t_{step} \leq 3$ symbols. For greater values, the ED does not lock to the spoofed beacon but loses track of the true one. The effect remains for several periods depending on the step size. For $\Delta t_{step} = 4$ symbols, the ED recovers after 7 periods, while it only takes 3 periods if the attacker uses $\Delta t_{step} = 8$ symbols. So, contrary to our expectations, appending jamming payload does not prevent the ED from re-locking.

To examine the situation further, Figure 14 depicts the beacon SNR measured at the ED during the attack. For low values of Δt_{step}, the SNR increases significantly with the start of the attack and remains high with low variance. In these cases, the nearby attacker node exploits the capture effect. The situation is different for the trials with a fast drift. SNR levels before the attack starts are comparable. While the beacon is lost, no values can be measured. Then, the measurements plateau around a value of −5 dB, but with a higher variance, which can be ascribed to the presence of the attacker's signal. Jamming with the extended payload most likely fails since its random symbols do not disturb the receiver's autocorrelation-based detection and synchronization mechanism.

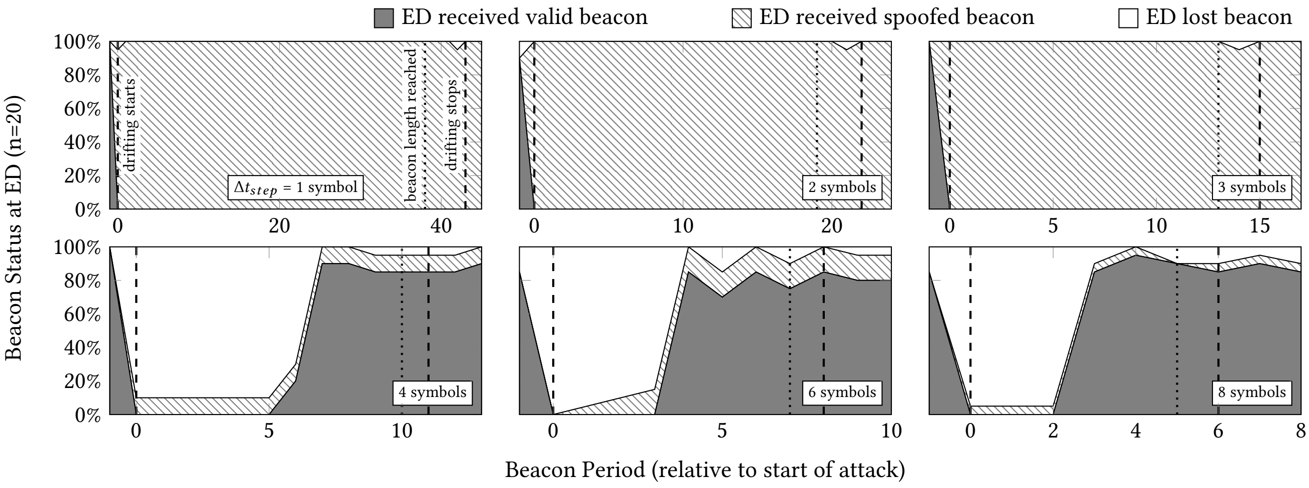

Figure 13: Beacon status at the ED under attack

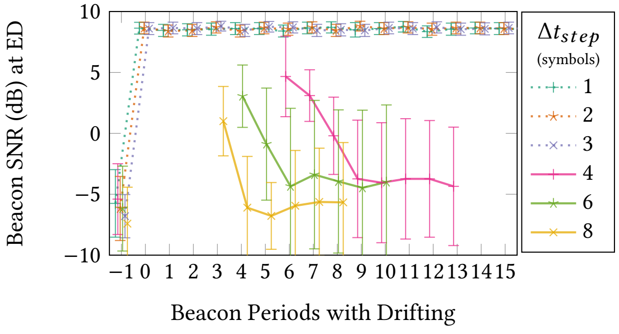

Figure 14: Beacon SNR during attack. Limited to values for which at least 5 beacons were received

8.2.2 Countermeasures. We have shown that an attacker can disrupt the Class B downlink of EDs in proximity repeatably in less than ten minutes. This implies severe consequences for applications relying on a guaranteed downlink latency, especially if no periodic uplink allows for graceful degradation of the service.

With the current requirement for network-spanning beaconing, no effective authentication-based countermeasure is applicable. Loosening this requirement and returning to network-specific beacons as specified in earlier drafts of LoRaWAN 1.0 [20, Section 15.1] would allow adding an authentication code to the beacon. Not transmitting beacons every 128 s based on the GPS epoch, but with an additional network-specific offset in the [0 s, 127 s] interval, could assure network coexistence. Since beacons are transmitted with inverse polarity to downlink traffic, additional beacon frames are negligible as a source of interference for the actual downlink traffic on the same channel. By transmitting the beacon authentication key during OTAA for specific devices, exposing it in non-volatile device memory is avoided and compromised keys can be replaced.

Without modification of the specification, the attack cannot be prevented. An ED can only try to detect it, for example by observing changes in the beacon's SNR or a leaping Class A downlink FCnt. If an attack is suspected, a compensation strategy should be used. Periodic uplinks assure a minimum level of downlink latency in that case but come at the cost of higher energy consumption. If the NS has evidence that Class B downlink fails, its only option would be to use a `ForceRejoinReq` MAC command on the next uplink to force the ED to rejoin the network in Class A [8, Section 9].

9 CONCLUSION

In this paper, we introduced two attacks affecting the availability of LoRaWAN networks and demonstrated their practical feasibility using ChirpOTLE, our LoRaWAN security evaluation framework.

As the foundation for the novel ADR spoofing attack, we introduced two concepts of bidirectional wormholes for LoRaWAN which apply even to the latest specification of LoRaWAN 1.1. Our results show that these wormholes are capable of manipulating frame metadata in LoRaWAN networks. This enables the ADR spoofing attack, which allows disrupting the communication for EDs located at the edge of the network with a high success rate.

We also introduced the concept of beacon drifting as a concrete attack for the vulnerability of missing beacon authentication, which has been mentioned in literature before. We found that by gradually shifting malicious beacon frames, an attacker may disrupt Class B downlink communication within an area in less than ten minutes.

We propose countermeasures for both attacks, which require revising the current specification. Without a change to the specification, the attacks can only be complicated, but not prevented.

ACKNOWLEDGMENTS

This work has been co-funded by the LOEWE initiative (Hessen State Ministry for Higher Education, Research and the Arts, Germany) within the emergenCITY centre and by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

REFERENCES

[1] Khaled Q Abdelfadeel, Victor Cionca, and Dirk Pesch. 2018. Fair Adaptive Data Rate Allocation and Power Control in LoRaWAN. In *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. IEEE, Chania, Greece, 14–15. https://doi.org/10.1109/WoWMoM.2018.8449737

[2] Emekcan Aras, Gowri Sankar Ramachandran, Piers Lawrence, and Danny Hughes. 2017. Exploring the Security Vulnerabilities of LoRa. In *Cybernetics (CYBCONF), 2017 3rd IEEE International Conference on*. IEEE, Exeter, UK, 1–6. https://doi.org/10.1109/CYBConf.2017.7985777

[3] Emekcan Aras, Nicolas Small, Gowri Sankar Ramachandran, Stéphane Delbruel, Wouter Joosen, and Danny Hughes. 2017. Selective Jamming of LoRaWAN Using Commodity Hardware. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2017)*. Association for Computing Machinery, Melbourne, VIC, Australia, 363–372. https://doi.org/10.1145/3144457.3144478

[4] Emmanuel Baccelli, Cenk Gündoğan, Oliver Hahm, Peter Kietzmann, Martine S Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C Schmidt, and Matthias Wählisch. 2018. RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT. *IEEE Internet of Things Journal* 5, 6 (2018), 4428–4440. https://doi.org/10.1109/JIOT.2018.2815038

[5] Martin Bor and Utz Roedig. 2017. LoRa Transmission Parameter Selection. In *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, Ottawa, ON, Canada, 27–34. https://doi.org/10.1109/DCOSS.2017.10

[6] Orne Brocaar. 2020. Chirpstack Network Server. GitHub Repository. https://github.com/brocaar/chirpstack-network-server

[7] Ismail Butun, Nuno Pereira, and Mikael Gidlund. 2018. Analysis of LoRaWAN v1.1 Security. In *Proceedings of the 4th ACM MobiHoc Workshop on Experiences with the Design and Implementation of Smart Objects (SMARTOBJECTS '18)*. ACM, Association for Computing Machinery, Los Angeles, California, 1–6. https://doi.org/10.1145/3213299.3213304

[8] LoRa Alliance Technical Committee. 2017. *LoRaWAN™ Specification V1.1*. Technical Report. LoRa Alliance. https://lora-alliance.org/resource-hub/lorawantm-specification-v11

[9] LoRa Alliance Technical Committee. 2018. *LoRaWAN™ Specification V1.0.3*. Technical Report. LoRa Alliance. https://lora-alliance.org/resource-hub/lorawantm-specification-v103

[10] Semtech Corporation. 2016. *LoRaWAN – simple rate adaptation recommended algorithm*. Technical Report. Semtech Corporation.

[11] Daniele Croce, Michele Gucciardo, Stefano Mangione, Giuseppe Santaromita, and Ilenia Tinnirello. 2018. Impact of LoRa Imperfect Orthogonality: Analysis of Link-Level Performance. *IEEE Communications Letters* 22, 4 (Jan. 2018), 796–799. https://doi.org/10.1109/LCOMM.2018.2797057

[12] Francesca Cuomo, Manuel Campo, Alberto Caponi, Giuseppe Bianchi, Giampaolo Rossini, and Patrizio Pisani. 2017. EXPLoRa: Extending the performance of LoRa by suitable spreading factor allocations. In *2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, Rome, Italy, 1–8. https://doi.org/10.1109/WiMOB.2017.8115779

[13] Tahsin CM Dönmez and Ethiopia Nigussie. 2018. Security of LoRaWAN v1.1 in Backward Compatibility Scenarios. *Procedia computer science* 134 (2018), 51–58. https://doi.org/10.1016/j.procs.2018.07.143

[14] ETSI. 2012. *ETSI EN 300 220-1: Electromagnetic compatibilityand Radio spectrum Matters (ERM); Short Range Devices (SRD); Radio equipment to be used in the 25 MHz to 1 000 MHz frequency range with power levels ranging up to 500 mW; Part 1: Technical characteristics and test methods*. Technical Report. European Telecommunications Standards Institute.

[15] Claire Goursaud and Jean-Marie Gorce. 2015. Dedicated networks for IoT: PHY/MAC state of the art and challenges. *EAI Endorsed Transactions on Internet of Things* 1, 1 (Oct. 2015), 1–11. https://doi.org/10.4108/eai.26-10-2015.150597

[16] Shengyang Li, Usman Raza, and Aftab Khan. 2018. How Agile is the Adaptive Data Rate Mechanism of LoRaWAN?. In *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, Abu Dhabi, United Arab Emirates, United Arab Emirates, 206–212. https://doi.org/10.1109/GLOCOM.2018.8647469

[17] Robert Miller. 2016. Lora Security: Building a Secure LoRa Solution. MWR Labs Whitepaper.

[18] The Things Network. 2020. The Things Stack. GitHub Repository. https://github.com/TheThingsNetwork/lorawan-stack

[19] Brecht Reynders, Wannes Meert, and Sofie Pollin. 2017. Power and spreading factor control in low power wide area networks. In *2017 IEEE International Conference on Communications (ICC)*. IEEE, Paris, France, 1–6. https://doi.org/10.1109/ICC.2017.7996380

[20] Nicolas Sornin, Miguel Luis, Thomas Eirich, Thorsten Kramp, and Olivier Hersent. 2016. *LoRaWAN™ Specification V1.0.2*. Technical Report. LoRa Alliance. https://lora-alliance.org/resource-hub/lorawantm-specification-v102

[21] Daniel Steinmetzer, Milan Stute, and Matthias Hollick. 2018. TPy: A Lightweight Framework for Agile Distributed Network Experiments. In *Proceedings of the 12th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization (WiNTECH '18)*. Association for Computing Machinery, New Delhi, India, 38–45. https://doi.org/10.1145/3267204.3267214

[22] Eef van Es, Harald Vranken, and Arjen Hommersom. 2018. Denial-of-Service Attacks on LoRaWAN. In *Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES 2018)*. ACM, Association for Computing Machinery, Hamburg, Germany, 1–6. https://doi.org/10.1145/3230833.3232804

[23] Xueying Yang. 2017. *LoRaWAN: Vulnerability Analysis and Practical Exploitation*. Master's thesis. Delft University of Technology.

[24] Xueying Yang, Evgenios Karampatzakis, Christian Doerr, and Fernando Kuipers. 2018. Security Vulnerabilities in LoRaWAN. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, IEEE, Orlando, FL, USA, 129–140. https://doi.org/10.1109/IoTDI.2018.00022