A Low-Cost and Infrastructure-Less LoRa Testbed

Pei Tian[a,b,d,*], Xiaoyuan Ma[c], Carlo Alberto Boano[d], Ye Liu[e], Fengxu Yang[f], Xin Tian[a], Dan Li[a], Jianming Wei[a]

[a]*Shanghai Advanced Research Institute, Chinese Academy of Sciences, China*
[b]*University of Chinese Academy of Sciences, China*
[c]*SKF Group, China*
[d]*Institute of Technical Informatics, Graz University of Technology, Austria*
[e]*Macau University of Science and Technology, China*
[f]*School of Information Science and Technology, ShanghaiTech University, China*

Abstract

One of the key obstacles hindering the development of large-scale LoRa testbeds outdoors is the lack of a backbone infrastructure allowing to easily communicate with the target nodes and supply them with power. As a result, many LoRa installations are just deployed indoors or only support a handful outdoor devices, which does not allow appropriate testing and benchmarking of low-power wide area networks. In this article, we present ChirpBox, an infrastructure-less and low-cost testbed in which the LoRa nodes are used not only to run experiments, but also to orchestrate all operations, including the dissemination of firmwares to be tested and the collection of log traces at the end of each test run. We achieve this, among others, by developing LoRaDisC: an all-to-all multi-channel protocol based on concurrent transmissions that allows an efficient communication over multi-hop LoRa networks. After presenting ChirpBox's design and implementation, we evaluate its performance experimentally and deploy a test installation to showcase its operations.

Keywords: Benchmarking, Concurrent Transmissions, Low-Power Wide Area Networks, LoRa Testbed, Wireless Systems

*Corresponding author
Email address: tianpei2018@sari.ac.cn (Pei Tian)

[1]Part of this work have been presented in [1] at the *18th International Conference on Embedded Wireless Systems and Networks (EWSN)*, Feb, 2021.

1. Introduction

Low-power wide area networks (LPWANs) are becoming a fundamental building block of the Internet of Things (IoT), as they allow to connect low-power devices over very large geographical areas using low-cost radio transceivers [2, 3]. LPWAN technologies, such as Sigfox, LoRa, Weightless, and NarrowBand-IoT, indeed, enable long-range communication up to tens of km in both rural and urban areas, with current consumptions in the order of tens of mA only [2, 4, 5, 6]. This empowers the deployment of large-scale systems and the creation of attractive IoT applications such as precision agriculture [7], smart metering [8], water distribution management [9], air quality monitoring [10], smart lighting [11], industrial monitoring [12], and intelligent transportation [13].

Among existing LPWAN technologies, LoRa is currently the most widespread and well-known [14]. On the one hand, this is due to the maturity and large availability of its chipset: LoRa was indeed one of the first LPWAN solutions to be designed (back in 2009 by Cycleo) and brought to the market. As a result, off-the-shelf LoRa modules are available today for less than 10 $. On the other hand, in contrast to other solutions leveraging licensed bands and/or requiring subscription (e.g., NarrowBand IoT, LTE-M, Weightless, and Sigfox), LoRa uses the sub-GHz unlicensed spectrum and allows users to freely deploy their own network, without any servicing costs and limitations on data traffic (besides regional duty-cycle constraints). Thus, one can enjoy lower operating costs while maintaining full ownership and control of the network infrastructure.

These properties, combined to a high receiver sensitivity thanks to the adoption of chirp spread spectrum modulation, have driven a large body of research and standardization activities proposing several networking solutions on top of LoRa. One of these is LoRaWAN [15], the reference architecture and medium access control protocol standardized by the LoRa Alliance, which specifies a star topology where end-devices interact with LoRaWAN gateways. Other examples include multi-hop communication protocols such as RLMAC [16], as well as solutions exploiting the concurrent transmissions principle [17, 18]. Besides novel networking protocols, the community has also been active in researching better coding schemes for data recovery [19, 20], enhanced link quality estimation techniques [21], improved

strategies for an optimal parametrization of physical layer settings [22], customized chirp signals to enable novel applications [23, 24, 25], as well as several other aspects improving the reliability and overall performance of LoRa-based communications [26, 27][28].

LoRa testbeds and their limits. In order to develop, debug, and optimize IoT solutions, as well as to benchmark the performance of low-power wireless protocols, *testbed facilities play a crucial role* [29]. Indeed, testbeds provide developers with a controlled (but realistic) testing environment and with tools facilitating experimentation, such as the automated scheduling of test runs, the reprogramming of target nodes with new firmware, the collection of log traces for offline evaluations, and the accurate tracing of GPIO events [30, 31, 32]. With the assistance of testbeds, researchers can easily observe the occurrence of an event within a wireless distributed system [33], better understand the capabilities of different physical layers [34, 35], and optimize protocol performance [36].

Several testbed installations support experimentation on LoRa nodes [37]: well-established IoT facilities such as FlockLab 2 [38] and FIT IoT-Lab [39], newer heterogeneous testing environments such as LinkLab [40] and NITOS [41], as well as facilities specifically built for testing LoRa-based systems [42, 43, 44] and for performing city-wide evaluations [45, 46]. However, most of these testbeds are *not open to the public* and, those who are, exhibit several limitations.

On the one hand, target nodes are often deployed in *indoor* environments with a very high density, e.g., in a single room [39, 40]. Given that LoRa is meant to be used outdoors for communicating over long distances, such installations are not representative of typical deployments and the characteristics of wireless links can be significantly different from that of real-world applications [47]. On the other hand, the few testbeds having nodes deployed outdoors (e.g., FlockLab 2) support *only a handful devices*, and hence do not allow for large-scale testing, which is very important when evaluating the performance of multi-hop protocols.

The gap to be filled. One of the root causes for the limited availability of outdoor testbed facilities supporting LoRa experimentation is the complexity in putting together a *backbone infrastructure*. The latter is often at the core of any IoT testbed: indoor facilities make often use of a wired backbone to (i) provide a stable power supply for the target nodes, (ii) disseminate the firmware to be tested and reprogram all devices, (iii) share a common time-

scale for determining the start of a test run and for computing time-related statistics, as well as to (iv) collect log traces for offline analysis.

Traditionally, Ethernet and USB connections are used for these purposes [32, 48, 49], given their ubiquitousness and ease of installation inside buildings. However, on rooftops and in rural areas, it is not only hard to permanently mount nodes and pull cables, but it may even not be possible to have power outlets at one's disposal. Although cellular technologies (e.g., 3G/4G) can be used to replace the wired Ethernet back-channel and allow a direct connection with the target nodes, they do not represent an ideal solution. On the one hand, the use of cellular networks for data transmission incurs traffic charges, which increases the operational costs. On the other hand, the availability of mobile service in remote areas cannot be given for granted: such settings are actually those targeted by LoRa systems. Besides, in case one is unable to power by mains the devices in the testbed (i.e., the target nodes need to be battery-powered), energy efficiency becomes a major concern, which precludes the use of power-hungry communication modules and calls for solutions allowing to communicate with the target nodes using a low-power wireless technology.

Our contributions. This article presents ChirpBox, a low-cost solution allowing to easily set up a LoRa testbed outdoors, even when no wired infrastructure is available to communicate with the target nodes and supply them with power.

Unlike most IoT testbeds, ChirpBox *does not make use of any observer node*. The latter is typically a power-hungry device (e.g., based on Raspberry Pis or BeagleBones [38, 32]) used to host (i.e., to power, program, stimulate, and profile) one or more target nodes [50]. Instead, in ChirpBox, target nodes are battery-powered independent entities connected via a multi-hop network to a *control node*, which serves as an interface with the user. This is achieved by equipping each target node with two firmwares, which are stored on separate memory banks: a *daemon* orchestrating the node's activities, and a *firmware under test* (FUT). The daemon takes care of (i) correctly receiving, storing, and executing the FUT provided by the user via the control node, (ii) diagnosing the connectivity among nodes, (iii) triggering the execution of a test run at a given time, as well as (iv) sending the log traces collected during a test run back to the control node. Unique feature of ChirpBox is that all communications between the testbed nodes to prepare, set up, and finalize a test run are carried out using *the same* LoRa radio on which the FUT

is deployed. This allows to employ the same hardware both for testing and for orchestrating the testbed operations, hence minimizing the overall costs. To efficiently disseminate and collect information across the testbed using LoRa, we implement *LoRaDisC*, an efficient all-to-all multi-channel protocol based on concurrent transmissions (CT) that can cope with LoRa's low data rate while ensuring that the regional duty-cycle constraints are met. Such protocol is also used to diagnose the connectivity among testbed nodes.

We build a prototype of ChirpBox using off-the-shelf components only, and deploy a test setup with 21 target nodes on a University campus to showcase its functionality. Our implementation makes use of target nodes based on an STM32 Nucleo board connected to an SX1276 radio – a common platform used by the LoRa community [51, 52]. The only hardware addition are a *real-time clock*, used to timely schedule the execution of test runs, as well as a *GNSS module*, used to provide all nodes with the same time-scale and to allow an accurate profiling of GPIO events. Each node, powered by Li-ion rechargeable batteries, is enclosed into a water-tight packaging and can be deployed freely, which guarantees maximum flexibility during installation.

We further carry out an experimental evaluation showing the performance of LoRaDisC's CT-based collection and dissemination primitives built on top of LoRa. The evaluation also quantifies the overhead of setting up and finalizing a test run in terms of both latency and energy consumption, which gives us the ability to infer the average lifetime of a target node as a function of the FUT size and of the network size. Additionally, we demonstrate ChirpBox in action by benchmarking the performance of several LoRa-based protocols and by observing how the quality of LoRa links varies as a function of ambient temperature by exploiting the built-in sensors in ChirpBox. Finally, we make our implementation *open-source*[2], in order to provide the community with a ready-made low-cost solution and foster experimentation.

This article proceeds as follows: we provide an overview of ChirpBox's architecture in Sect. 2 and detail its hardware and software components in Sect. 3. We provide a detailed description of the testbed management tools and of the APIs available for firmware development in Sect. 4. We then experimentally evaluate ChirpBox's performance in Sect. 5, and show exemplary uses of the testbed in Sect. 6. After summarizing related work in Sect. 7, we conclude this article in Sect. 8, along with a discussion of future work. Note

[2] `https://chirpbox.github.io`

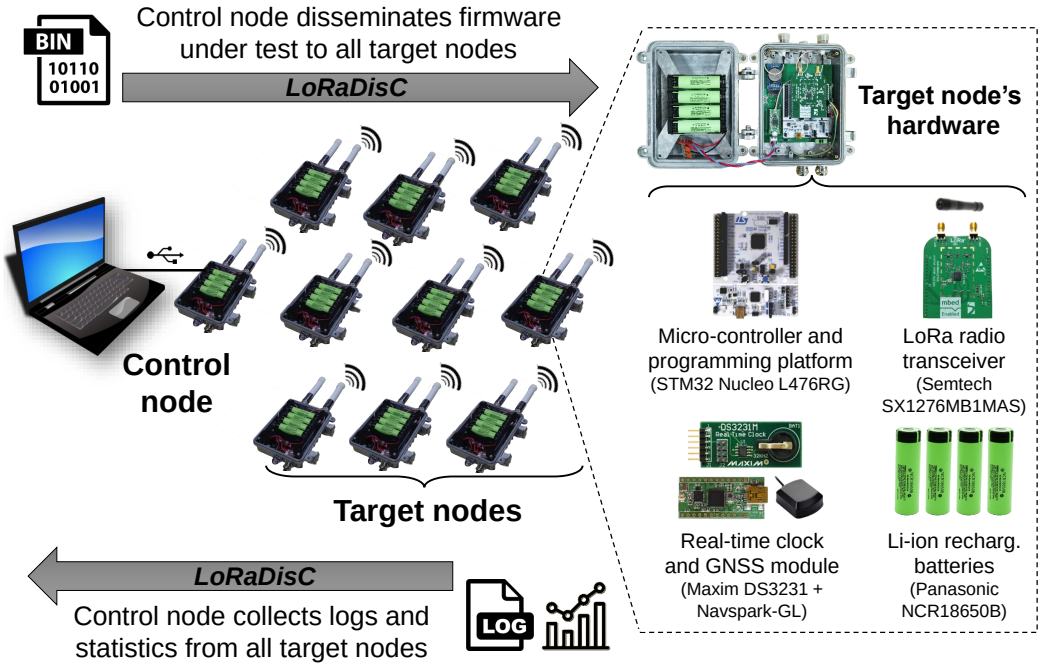


Figure 1: ChirpBox's architecture: the target nodes are used not only to run tests, but also to disseminate the FUT and a test run's settings, as well as to enable the collection of all logs after a test run has completed. All testbed nodes are homogeneous and based on off-the-shelf hardware.

that this article is an extended version of our EWSN'21 paper [1].

2. ChirpBox: Overview

Fig. 1 shows ChirpBox's architecture, which allows the deployment of a LoRa-based testbed in outdoor areas where no infrastructure is available – neither to communicate with the target nodes, nor to supply them with power. All nodes in ChirpBox are identical from an hardware viewpoint and entirely built using *off-the-shelf* components, as detailed in Sect. 3.1. The *target nodes* embedding a LoRa radio are used not only to run tests, but also to orchestrate the testbed's activities. ChirpBox, indeed, does not use any power-hungry observer: test runs are scheduled by the user via a *control node*, which disseminates the FUT and the run's configuration to all target nodes, as well as collects all the logs at the end of a test run *using LoRa*. Note that the lack of dedicated observer nodes, which are a key component

in traditional low-power wireless facilities [31, 32], allows to minimize the deployment costs and increases flexibility.

2.1. Control Node

ChirpBox's control node acts as an interface with the user: it consists of a desktop PC or laptop connected via USB to a LoRa node, which is based on an STM32L476RG board attached to a Semtech SX1276, as detailed in Sect. 3.1. The user can schedule a new test run by uploading a new firmware to be tested (`bin` file) and by specifying a number of settings, such as the duration of a test run, the target nodes that should be included or excluded in the test, and whether a check up of the testbed's health status should be carried out before the test run. Inspired by the D-Cube benchmarking infrastructure [53], ChirpBox also allows to binary patch the FUT, so to seamlessly change user-defined protocol parameters at runtime. This binary patching feature allows ChirpBox to test the same firmware using different parameters without the need of re-disseminating it to all target nodes. The test run's settings and user-defined protocol parameters are stored in a `json` file [54]. A `python` script parses this file as well as the provided FUT, and instructs the LoRa node via the serial port about the information that needs to be disseminated to the target nodes. The same script also collects the results at the end of a test run, such as the logs from each target node and the testbed's health status, returning them to the user.

2.2. LoRaDisC Protocol

In order to disseminate the FUT as well as the test run configuration to all battery-powered target nodes and in order to collect logs and health status information back to the control node, a *reliable* and *efficient* dissemination/collection protocol is fundamental. Indeed, LoRa's data rates are limited to hundreds or a few thousands bits per second (bps) depending on the employed spreading factor (SF), and every device needs to comply with the regulated duty cycle limits, which are set to 1% in most regions. Having the control node individually communicating with each target node (e.g., using LoRaWAN) is hence not viable, as the additional transmissions to disseminate and collect data to/from the target nodes would represent an overkill in terms of delay. Furthermore, a *multi-hop* dissemination and collection is desirable to avoid the need of having all target nodes in range of the control node, which would limit scalability. Note that, as illustrated in Sect. 6.2, even if connection to the control node (e.g., a LoRaWAN gateway)

can be maintained by increasing the SF, this should be avoided. Indeed, this leads to a higher amount of packet collisions as well as a lower data rate (in addition to the higher energy consumption and transmission delays intrinsically brought by the use of a higher SF).

To support multi-hop communication, we design and implement *LoRaDisC*, an all-to-all protocol based on *concurrent transmissions* that allows a reliable and efficient multi-hop collection and dissemination across all LoRa nodes in the testbed. LoRaDisC uses multiple flooding rounds during which nodes communicate on multiple channels: this allows to minimize collisions, maximize throughput, and speed-up the information exchange across nodes, while ensuring compliance to the regional duty-cycle regulations. Moreover, as detailed in Sect. 3.3, LoRaDisC embeds optimizations to support multiple SFs, avoid wasteful receptions, and to increase the reliability of communications.

2.3. Target Node's Operations

To use the target nodes for managing the testbed operations and for experimentation interchangeably, we exploit the *dual-bank flash memory* feature of the STM32L476RG micro-controller, which allows us to equip each target node with two firmwares. We store on the first memory bank a *daemon* firmware that takes care of the synchronization to the control node, as well as of receiving, transmitting, and forwarding messages in the context of LoRaDisC's data dissemination and collection primitives. The daemon is also in charge of storing and verifying the received FUT, of triggering the test run's execution based on the control node's instructions, as well as of conveying the log traces collected during the last test run back to the control node. The FUT is stored in the second memory bank and can log information at a given flash address by calling the `log_to_flash()` function provided by ChirpBox's application programming interface (API). Moreover, with the help of an RTC module, ChirpBox switches from the FUT back to the daemon on the first memory bank as soon as a test run is completed. More details are provided in Sect. 3.2.

2.4. Testbed Management Features

In order to facilitate experimentation, ChirpBox also provides the user with means to monitor the testbed's health status and evaluate the connectivity among nodes. When scheduling a new test run, the user can for example set the `connectivity_evaluation` flag, which instructs ChirpBox

to add a short all-to-all communication phase to estimate the link quality and packet reception rate across all nodes. This way, the developer can infer each node's neighbours, as well as the presence of asymmetric links. During this phase, the evaluation results along with the nodes' health status (e.g., the battery consumption) are returned to the user using the same collection primitives employed to convey the log traces back to the control node. The connectivity information and nodes' health status tools of LoRaHop help developers in performing a sanity check. Malfunctioning nodes can be omitted from a test by configuring the test run's settings accordingly, as discussed in Sect. 2.1. ChirpBox also provides tools to update the daemon wirelessly, as well as additional API commands to timestamp the logs and integrate external sensors: we elaborate on these additional features in Sect. 4.

3. ChirpBox: Design and Implementation

We next describe ChirpBox's design and implementation in detail. We start by describing a target node's hardware (Sect. 3.1), and we then illustrate its internal activities, flash space allocation, and RTC operations (Sect. 3.2). We further present the design of LoRaDisC and explain how it is used for data dissemination and collection (Sect. 3.3).

3.1. Anatomy of a ChirpBox Node

As shown in Fig. 1, ChirpBox's nodes employ an STM32L476RG Nucleo board attached to a Semtech SX1276MB1MAS shield, which supports operations in the 470 and 868 MHz ISM bands. This is a popular combination that is also used to build some of the LoRa nodes forming "*The Things Network*" public community initiative [55]. The read-only unique identifier (UID) provided by the micro-controller is mapped to a static logical ID, e.g., 1 to 10 if the testbed consists of 9 target nodes and a control node. The logical IDs of all the nodes in the testbed should be provided prior the compilation of the daemon firmware, as the built-in multi-hop protocol (LoRaDisC) requires knowledge of the testbed's scale in advance (see Sect. 3.3).

We further equip ChirpBox's nodes with a GNSS module and antenna (NavSpark-GL [56]) to provide all nodes with an absolute and common timescale, as well as with a real-time clock (RTC) module (Maxim DS3231MPMB1 [57]) connected with the reset pin of the micro-controller to precisely control the duration of each test run. Each target node is also equipped with four Li-ion rechargeable batteries with a capacity of 3400 mAh

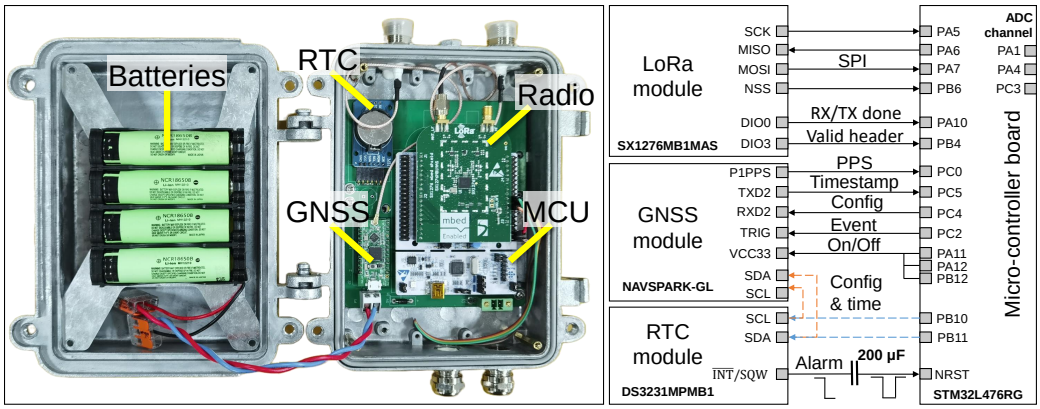


Figure 2: Simplified sketch of the interconnections between the main hardware components in a ChirpBox node. In principle, the RTC module can be connected to the micro-controller directly (blue dashed lines). To avoid misconfiguration or tampering of the RTC module while the FUT is running, the RTC module can only be accessed via the programmable GNSS module (orange dashed lines).

(Panasonic NCR18650B) and is enclosed into a watertight IP 67 casing. Note that the control node does not require batteries, as it can be powered via USB. Besides control and target nodes, no additional hardware is necessary to set up ChirpBox, which allows to keep costs low: excluding the main board and the LoRa radio, the remaining components are inexpensive and well below 100$ per node.

Fig. 2 further shows the interconnections between the main hardware blocks in ChirpBox. The LoRa transceiver communicates with the STM32L476RG micro-controller via SPI and it is further connected to two interrupt pins to signal when a packet has been transmitted or received (`DIO 0`) as well as when a valid packet header is received (`DIO 3`). With the help of these interrupt pins, we can implement support for CT-based LoRa communication and filter invalid headers, as explained in Sect. 3.3. The micro-controller can retrieve the current time when second-level accuracy is acceptable and the use of GNSS is not possible or too energy-expensive. Moreover, the micro-controller can instruct the RTC module via the programmable GNSS module to generate an alarm interrupt (falling edge) at a given time: this is used to precisely time the duration of an experiment and trigger the completion of a test run. Specifically, the $\overline{INT}/SQW$ pin is connected to the reset pin of the STM32L476RG: a 200 μF capacitor transforms the falling edge into a negative pulse that can reset the micro-controller. In

principle, the RTC module could also be controlled by the STM32L476RG micro-controller via I^2C directly (as illustrated by the blue dashed lines in Fig. 2). However, this increases the risk of a misconfiguration or of deliberate tampering by the FUT, which would result in target nodes being no longer under the control of the daemon. To avoid such cases, in our implementation, the RTC module can only be accessed via the GNSS module (as highlighted by orange dashed lines in Fig. 2) with secure commands, as detailed in Sect. 3.2. The STM32L476RG micro-controller is also directly connected to the Navspark-GL using multiple GPIO pins: this allows to turn on/off the GNSS module at runtime and minimize the energy consumption. Using the Navspark-GL, the micro-controller can synchronize the local clock with the GNSS time by exploiting the PPS signal. Furthermore, the micro-controller can also accurately timestamp events using the `TRIG` pin and receive the GNSS time at which they occurred on the `PC5` pin – a useful feature to enable fine-grained debugging of distributed events.

The STM32L476RG micro-controller at the heart of a ChirpBox node embeds *two flash memory banks* and can be configured to boot from any of the two by setting the `BFB2` (i.e., "boot from bank 2") bit accordingly at runtime. We load a daemon taking care of orchestrating a target node's activities on the first memory bank, and keep the second memory bank to store the FUT and the logs of the current test run, as illustrated in Fig. 3. At the beginning and at the end of a test run, the micro-controller is reset and the boot configuration is changed accordingly. The daemon embeds several functions that are used to coordinate the target node's operations, as detailed in Sect. 3.2, most of which are based on LoRaDisC's dissemination and collection primitives. ChirpBox further provides an API facilitating logging and abstracting the functions provided by the GNSS and RTC modules.

Note that the dual-bank flash memory is not a rare feature for modern MCUs. For example, the STM32L0/1/4 series [58], TI MSP432 series [59], Renesas RA6M4/4 series [60], and Atmel ATSAM3A4C [61] also offer such functionality. For the MCUs with only one flash memory bank, in principle, such bank switch can be imitated by *swapping image* in the bootloader [62]. The image swapping functionality can select whether to run the daemon or the FUT. The only requirement when having only one flash memory bank available is that the flash space should be sufficiently large to store the daemon, the FUT, as well as the logs.

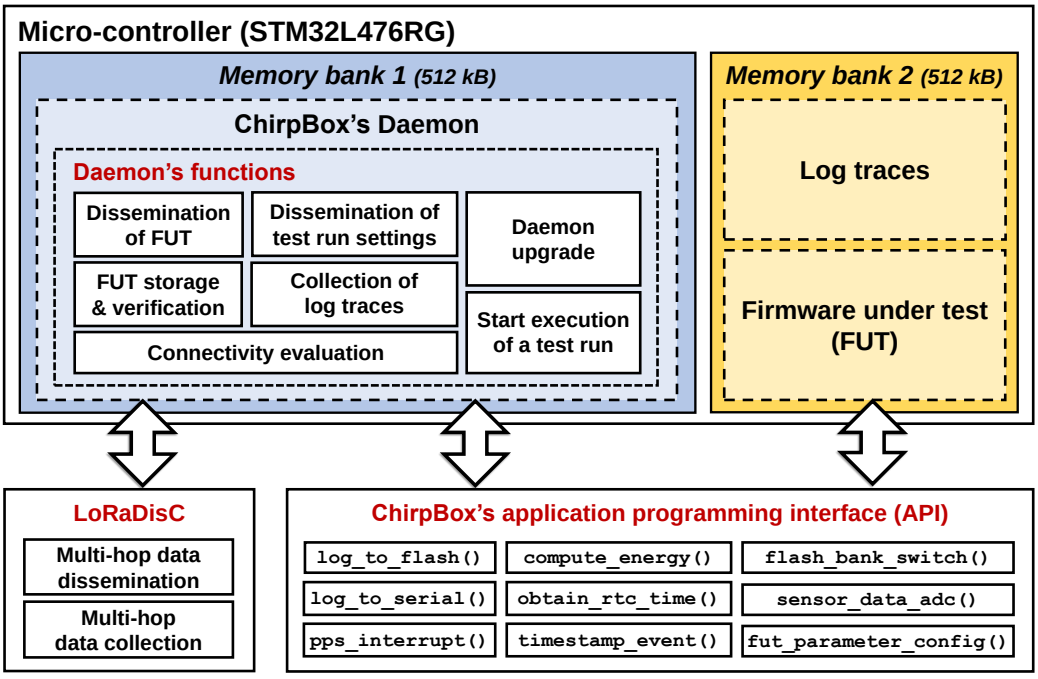


Figure 3: Each target node contains a daemon (coordinating its activities) in one memory bank and the FUT on a second bank. Many of the daemon's functions are built upon LoRaDisC's communication primitives. An API provides access to common functions and abstracts the underlying hardware, simplifying the development of the FUT.

3.2. Target Node Operations

The daemon orchestrates a target node's activities by perpetually following the sequence of steps illustrated in Fig. 4. By default, nodes are idle (i.e., in low-power mode) and periodically listen for `SYNC` messages originated by the control node (every 60 seconds in our implementation). These messages are immediately re-transmitted by each target node and flooded through the whole network. The `SYNC` messages inform the target nodes whether any test run is scheduled for execution and – if this is the case – notify the daemon about the next step to be performed to prepare or run the next test. Such steps follow a pre-defined order: at first, an optional connectivity evaluation ❶ can be carried out using LoRaDisC as described in Sect. 4.1.2; after this step is completed, the control node requests all target nodes to convey back the collected results, also using LoRaDisC ❷.

The `SYNC` message can then notify the target nodes that its subsequent

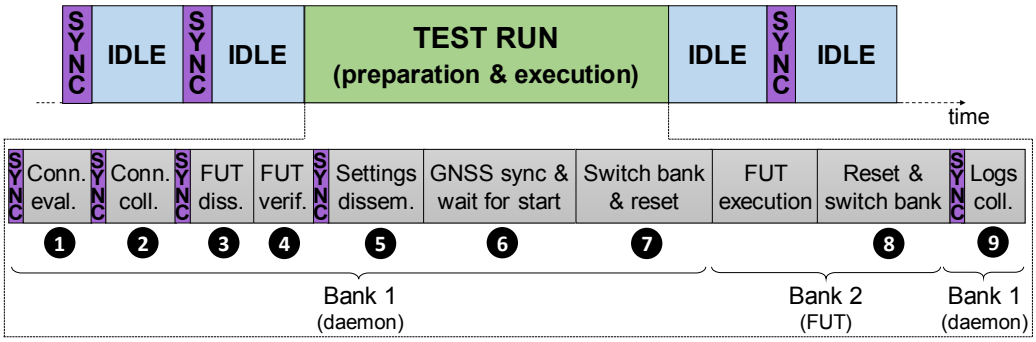


Figure 4: Operations of a target node in ChirpBox: nodes are typically idle and periodically look for `SYNC` messages from the control node. When a test run needs to be executed, the `SYNC` message triggers the execution of different activities.

dissemination messages sent using LoRaDisC will contain the next firmware to be run ❸. The firmware is received and stored on the second memory bank on the fly[3]; its integrity is verified by the daemon as soon as the dissemination is completed ❹. Note that these two steps are also optional, as the next test run could make use of the previous firmware with different parameters, as explained in Sect. 2. Similarly, the control node also disseminates the next run's settings ❺, e.g., start time, duration, and FUT parameters.

If the target node is expected to actively take part in the next test run, the daemon shortly activates the GNSS module (by setting the `On/Off` pin as shown in Fig. 2) and synchronizes the target node's local clock to UTC (if necessary) by exploiting the PPS signal. The synchronization takes about 40 seconds until the GNSS module completes its cold start phase. The daemon then configures the RTC module to trigger alarm interrupts at the start time and the expected completion time of the test run, respectively. Afterwards, the micro-controller enters the low-power mode to save energy and wake up until the instructed start time of the test run ❻. At this point, the daemon configures the `BFB2` bit to `1` and resets the micro-controller[4] ❼: this way, the execution of the FUT from the second memory bank is triggered

[3]When using the STM32L476RG micro-controller, it is possible to write on a bank without disrupting code execution, i.e., a piece of code executing in one bank can read and write the content restored in the other bank.

[4]The daemon also enables the flash write protection of memory bank 1, in order to prevent unintended (and unwanted) modifications by the FUT.

seamlessly.

Once the duration of a test run has elapsed, the target node is reset by the alarm interrupt generated from the RTC module. Using the current GNSS time, the reset handler[5] in the STM32L476RG double-checks that the RTC alarm time has passed (it otherwise waits until then), and triggers a soft reset after clearing the `BFB2` bit and re-enabling the flash write protection of bank 1 ❽. Hence, the target node then boots from the first memory bank and starts running the ChirpBox daemon, which resumes its operations. The next `SYNC` message would instruct all target nodes to send the logs stored during the previous run in a given flash portion to the control node using LoRaDisC's collection primitive ❾. Thereafter, the target node enters low-power mode and wakes up at regular intervals to receive the `SYNC` messages and react accordingly. Note that during a `SYNC` flood, each node uses channel hopping and adopts a "*listen before talk*" (LBT) strategy to comply with the local duty cycle regulations – the same principle used by LoRaDisC and detailed in Sect. 3.3.

Note that the FUT (running on the second memory bank) should be stopped by the RTC module's alarm signal indicating to the micro-controller that a test run is complete: this is the only way to give back the control to the daemon. This means that the RTC module must reliably trigger the micro-controller at the end of a test run: if this is not the case, a target node would never switch back to the daemon firmware. This could be the case if the expiration of the timer triggering the alarm signal that was preset by the daemon has been accidentally misconfigured or was deliberately tampered by the FUT. To avoid this without the need of additional hardware, the RTC module is connected to the GNSS module's I^2C interfaces rather than being connected to the micro-controller directly, as illustrated in Fig. 2. By using an easily-programmable Arduino-like interface [63] provided by the GNSS module, we implement a set of private commands – only known by the daemon – on the GNSS module to configure and fetch the time/alarm time of the RTC module. This way, the RTC module can be controlled by the micro-controller in the daemon, hence remaining hidden to the FUT.

[5]The reset handler is called once the micro-controller reboots. Such logic can be easily integrated into the FUT by adding the `toggle.c` and `ll_flash.c` files, provided by ChirpBox, to the FUT's project. This way, the reset handler of the FUT can be reloaded.

3.3. LoRaDisC

Most of ChirpBox's operations build on top of LoRaDisC, a multi-hop protocol that can support both *one-to-all* data dissemination and *all-to-one* data collection on top of LoRa nodes. LoRaDisC exploits *concurrent transmissions (CT)*, retaining their simplicity and advantages (such as low end-to-end latency, high reliability, high energy efficiency, and multi-hop communication without the need of explicit routing strategies [64]), as well as *network coding* to increase reliability and throughput.

We review these two fundamental concepts in Sect. 3.3.1 and 3.3.2 before providing an overview of LoRaDisC in the remainder of this section.

3.3.1. Concurrent Transmissions

The transmission of multiple overlapping packets over the air typically results in a *collision*, and a wireless device is often unable to decode a packet correctly. However, it was recently shown that by means of concurrent transmissions (CT), i.e., by letting devices transmit their packets simultaneously and by accurately aligning the start of the transmissions, there is still a high chance to correctly decode packets [64]. Specifically, a target packet has a high chance to be decoded correctly if it is received with a greater power (>3 dB) than the sum of all the other signals at the receiver: this effect is called *capture effect* [65], and it has been analyzed theoretically [66, 67] and verified experimentally on several off-the-shelf low-power radio transceivers [18, 68].

CT have been especially used to achieve dependable wireless data collection and dissemination, as highlighted – among others – by the results of the EWSN dependability competition series [69]. In an exemplary naïve CT-based network made up of multiple sequential flooding rounds, as shown in Fig. 5(a), one node initiates the flooding round by actively sending a packet in a pre-defined order (node A initiates the flooding round in which packets are marked in dark blue, node B initiates the flooding round in which packets are marked in cyan, and so on). As soon as a node receives a packet, it re-broadcasts the packet during the next CT slot. Each node is allowed to transmit at most for N_T times per flooding round to save energy and each flooding round consists of N_S slots ($N_T = 1$ and $N_S = 3$ in Fig. 5(a)). By doing so, messages can be disseminated to all nodes in the network.

CT were first implemented based on IEEE 802.15.4 narrowband transceivers [64, 68, 70, 71], and its feasibility was subsequently verified on IEEE 802.15.4 ultra-wideband radios [72] and on Bluetooth Low Energy

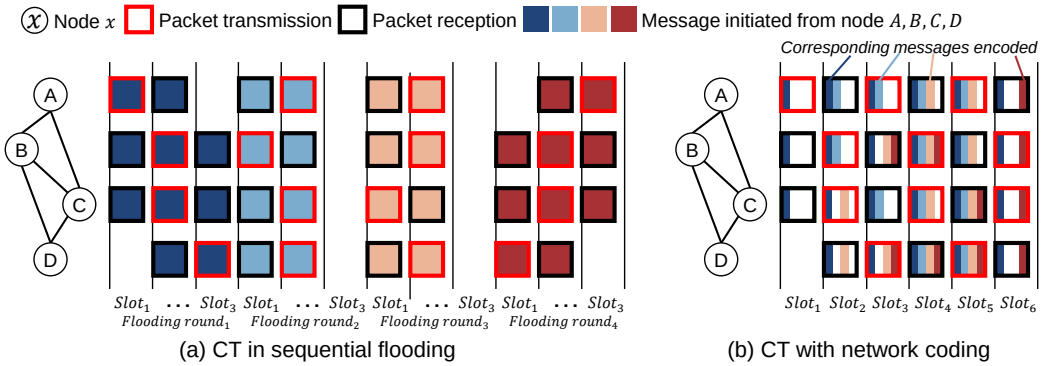


Figure 5: Example of data dissemination when using sequential concurrent transmissions (a) and when applying network coding applied on top of concurrent transmissions (b).

(BLE) devices [34, 73]. In LoRa, CT were first investigated by Bor et al. [17] and further verified by Liao et al. [18].

3.3.2. Network Coding

Network coding has been applied to CT-based networks in order to increase their throughput and reliability [74, 75]. The essential idea of network coding is to deliver encoded packets and reduce packet loss by removing the redundancy (e.g., fountain codes [76]) or/and to optimize the number of exchanged packets by exploiting the redundancy (e.g., LT codes [77] and random linear network coding (RLNC) [78]). An example of RLNC based on CT is as represented in Fig. 5(b). By exploiting network coding, the number of CT slot required by a data dissemination protocol decreases from 12 (in Fig. 5 (a)) to 6. Different from the sequential CT flooding, the transmitted packets are not identical when RLNC is applied. The colorful blocks in Fig. 5(b) indicate that the corresponding messages are encoded in the packet, e.g., using the XOR operation. The receiver decodes a packet based on the known messages: for instance, in Fig. 5(b), node C can decode B's message at the third CT slot, as it has decoded A's message during slot 1.

3.3.3. Overview of LoRaDisC

LoRaDisC consists of a series of flooding rounds, the first of which is a *one-to-all* round that contains the configuration information for the following ones, as illustrated in Fig. 6. The configuration information notifies the receiving nodes about: (i) the primitive type, which is either dissemination or collection, (ii) the number of messages M to be disseminated or the start/end

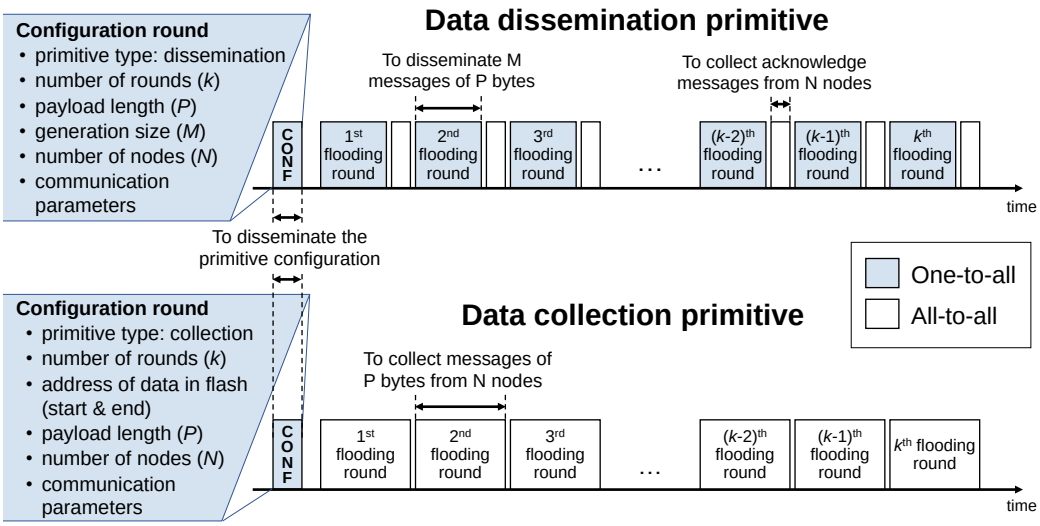


Figure 6: LoRaDisC supports 2 communication primitives: data dissemination and collection, consisting of an initial flooding round and a series of one-to-all or all-to-one rounds to disseminate, acknowledge, and collect information.

address in flash of the data that needs to be collected, (iii) the payload length P used for the subsequent messages, (iv) the number of subsequent flooding rounds k, (v) the number of nodes in the network N, as well as (vi) LoRa communication parameters such as SF, bandwidth, transmission power, and coding rate to be used in the following rounds[6]. Each flooding round contains a series of CT slots, during which nodes blend packets using RNLC and transmit random linear combinations of previously received packets as in Mixer [79]. This allows to improve the reliability during data dissemination and the overall throughput during data collection. Moreover, the use of coded packets in LoRaDisC helps increasing the chances of successful delivery over unreliable channels.

During data dissemination, after a one-to-all flooding round, an all-to-all round is used to ensure that all nodes have received the previous information (i.e., to acknowledge the correct reception of the data transmitted during the previous round). To this end, each node sets a bit corresponding to itself in a dedicated `coding_vector` field of the LoRaDisC header if all M messages have been received in the previous one-to-all round. The node disseminat-

[6]The physical layer settings used in the configuration round (i.e., SF, bandwidth, transmission power, and coding rate) are set at compile time based on the size of the network.

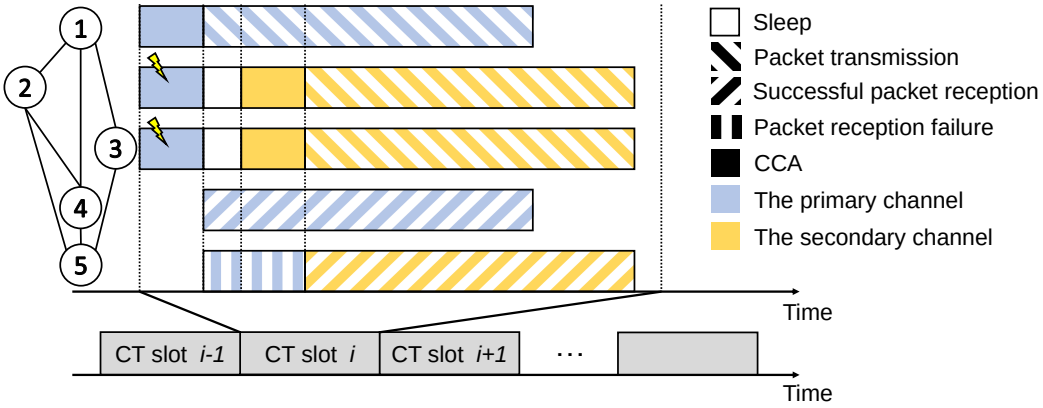


Figure 7: LBT & AFA mechanisms introduced in LoRaDisC.

ing information (e.g., the control node) can then double-check whether all the bits are set to one: if this is not the case, multiple blocks would be retransmitted in the next one-to-all round.

3.3.4. Use of LBT and AFA

LoRaDisC's design is strongly influenced by the spectrum access regulations for LoRa-based systems, which severely limit the amount of transmissions that a device is allowed to perform. For example, in Europe, it is required that each LoRa device works with a transmission duty cycle as low as 0.1 or 1% (depending on the employed channel) when no polite spectrum access technique is used. This corresponds to a maximum transmission time of 3.6 s or 36 s per hour on the selected channel, which is usually followed by LoRaWAN Class A end-devices, which are based on an ALOHA medium access control scheme. However, when a LoRa device uses polite spectrum access by means of *listen-before-talk* (LBT) and *adaptive frequency agility* (AFA), the duty cycle restrictions are much less rigid [80]. LBT prescribes that a device carries out a clear channel assessment (CCA): in case of a busy channel, the device must either wait for a random back-off time or change the employed frequency before the next CCA check. The use of at least two frequencies for transmission is referred to as AFA. When both LBT and AFA are implemented, the duty cycle restriction is loosened to 100 s of cumulative transmission time per channel per hour, which corresponds to a duty cycle ratio of 2.7% per channel [81].

In LoRaDisC, we hence embed both LBT and AFA to enjoy a higher

duty cycle per channel and to significantly speed up the data transfer, as shown in Fig. 7[7]. According to the spectrum access regulations [81, p. 55], before each transmission, a node needs to keep listening for 5 ms to make sure whether the channel is clear. Therefore, in contrast to classical CT-based protocols, where received messages can be immediately re-transmitted (e.g., within 196 μs for IEEE 802.15.4 devices [68]), in LoRaDisC nodes need to first complete the CCA check. The de-synchronization error introduced by this additional LBT delay can be tolerated due to LoRa's low data rate compared to IEEE 802.15.4.

Each CT slot is assigned a primary and a secondary channel, as illustrated in Fig. 7. By default, at the beginning of a CT slot, all nodes that are willing to transmit information (nodes 1, 2, and 3 in this example) perform a CCA check on the primary channel. If the latter is found to be idle, the transmission is initiated after 5 ms (node 1). If the channel is found busy due to surrounding RF activities (this is the case for nodes 2 and 3), the node backs off for a given time (e.g., 10 ms) and performs a new CCA check on the secondary channel (initiating the transmission after 5 ms if this is found to be idle). Devices receiving information (nodes 4 and 5) listen by default on the primary channel: if data is available, they lock on the channel and receive it (node 4). If no information is available on the primary channel, nodes switch to the secondary channel (node 5). In our current implementation, the employed channels vary over time to maximize throughput while complying to the local regulations: nodes derive the primary and secondary channel to be used from the flooding round number and the CT slot number, respectively. The duration of a CT slot is determined after the configuration round according to the payload length and PHY settings (e.g., SF, bandwidth, and coding rate).

3.3.5. Physical (PHY) settings and frame length

For the first all-to-all configuration round of LoRaDisC (as well as for ChirpBox's `SYNC` messages), the employed PHY settings are fixed at compile time. Based on the data exchanged during the configuration round, new PHY settings can then be used in successive rounds. However, as PHY settings such as the SF largely influence the time on-air of frames (and hence

[7]Our implementation of LoRaDisC is *generic* and allows, in principle, to disable the use of LBT and AFA, as shown in Sect. 5. However, this is not recommended, as this results in longer delays when operating the testbed.

the amount of transmissions a device can perform), LoRaDisC needs to *automatically* adjust the maximum payload length over time in order to comply to local regulations. Indeed, whilst packets with up to 255 bytes payload are supported in the LoRa PHY, local regulations may limit the maximum transmit time. Therefore, before calling the LoRaDisC primitive (dissemination or collection), the transmitter (e.g., ChirpBox's control node) computes the payload size according to which SF is selected in the next one-to-all/all-to-all flooding round. In our implementation, the LoRaDisC header and footer is 14 bytes in total for a network with 20 nodes: for this reason, only spreading factors up to 11 can be supported in Europe[8].

3.3.6. Use of Hardware Interrupts

Several protocols based on CT have been developed recently, the vast majority of which on top of IEEE 802.15.4 radios [64] and, quite recently, of BLE transceivers [82]. A key difference when practically implementing CT on top of LoRa is the lack of an interrupt signal indicating that a start-of-frame delimiter (SFD) is transmitted or received. The latter is typically used to synchronize a node during a CT slot [79]. As no SFD interrupt signal is provided in most LoRa radios (including the SX1276 used in ChirpBox), we make use instead of the *RX_done* and *TX_done* interrupts (`DIO 0`). The jitters of these signals (1.48 ms and 43.6 μs respectively [83]) are acceptable given that a CT implementation on LoRa can tolerate alignment errors of 3 symbol times, i.e., up to 3.072 ms for the fastest configuration (SF=7) [18]. We exploit also a second interrupt signal (`DIO 3`), triggered upon the reception of a *valid header*, to further improve the efficiency of LoRaDisC. As the data rate of LoRa is rather low (e.g., a packet with a 182-byte payload requires 312.58 ms to be received with SF=7 and almost 1 s with SF=9), we instruct nodes to immediately verify if the byte following the received header corresponds to the well-known LoRaDisC header. If this is not the case, the node quickly turns off the radio to preserve its limited energy budget[9]: this is especially useful to filter the transmissions from co-located LPWANs.

[8]The European Telecomm. Standards Institute, for instance, imposes a maximum transmit time of 1 s [81]: this corresponds to a payload of at most 10 bytes when using a SF of 12, a bandwidth of 125 kHz, a coding rate of 4/5, a preamble length of 8 symbols, with explicit header and CRC enabled.

[9]When receiving a 232-byte payload with SF=7, a node can avoid to unnecessarily keep its radio active for 353.28 ms – saving 92% of energy.

4. Testbed Services

As discussed in Sect. 2, ChirpBox embeds a number of features providing the user with means to monitor the testbed's health status (Sect. 4.1.1) and connectivity (Sect. 4.1.2), as well as to upgrade the daemon's firmware and generate patch files (Sect. 4.1.3). We provide an application programming interface (API) that can be called by the FUT to timestamp events and to store log files into the flash memory (Sect. 4.2.1). In addition, software interfaces for external sensors (e.g., acoustic sensors) are also supported for further application extensions (Sect. 4.2.2).

4.1. Services for Testbed Management

The control node can instruct all target nodes to periodically collect and send back information in order to get a fine-grained picture of the testbed health status and connectivity. For example, target nodes may periodically be asked to measure their battery voltage or their current on-board temperature, and to report this information back to the control node. The voltage can be used to infer the need for a battery replacement of specific target nodes.

4.1.1. Health Monitor

The battery voltage is one of the health status indicators provided by ChirpBox, and is periodically collected from the target nodes. Once the battery voltage is lower than the voltage start-up threshold, a target node may malfunction, as a reset may cause an unexpected switch across memory banks that would interrupt a test run. Therefore, we design an under-voltage protection mode, in which a battery voltage check is performed at the beginning of the daemon firmware. As soon as an under-voltage condition is detected, the daemon enters the under-voltage protection mode, during which the daemon does nothing but periodically (i.e., every second) checking the voltage, and does not exit until the voltage becomes stable again and well above the start-up threshold.

Note that ChirpBox can also be easily powered by solar panels. Using only an additional 12-to-3.3 V adapter, a ChirpBox node can be installed on an existing solar street lamp posts and powered by the solar panels as shown in Fig. 9(b). With the help of the solar panel, the battery voltage can be kept at a sufficiently high value (e.g., ≥ 4 V), as shown in Fig. 8. This practically makes ChirpBox maintenance-free and allows perpetual operations, as it removes the need to periodically replace the battery of the target nodes.

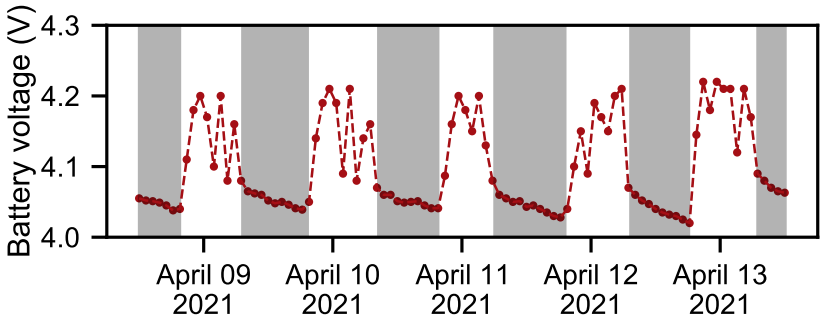


Figure 8: Evolution of the battery voltage on a ChirpBox target node powered by a solar panel. The voltage increases during daytime and slowly declines during night (gray areas).

The on-board temperature is another health status indicator that can be used to detect hardware anomalies (e.g., a short circuit). The temperature is measured using the built-in sensor of the SX1276 transceiver. As we show in Sect. 6.4, one can use these measurements to observe the impact of ambient temperature on link quality. Differently from the battery voltage measurements, the temperature values are piggybacked in the connectivity status information (see Sect. 4.1.2).

4.1.2. Connectivity Monitor

Any node in ChirpBox is required to have at least one neighbor node in order to receive and relay information from/to the control node. Therefore, connectivity across network nodes should be monitored and ideally guaranteed in advance, e.g., through a careful deployment of target nodes. To aid with this, ChirpBox provides a tool to measure the connectivity in the network, thereby facilitating the analysis of the link quality between nodes. The connectivity evaluation itself consists in a firmware instructing the target nodes to periodically transmit probe packets in a round-robin manner, thereby guaranteeing no collision. One can observe how the network topology changes with different PHY settings (e.g., when using different SFs) by modifying the PHY settings of the connectivity evaluation firmware accordingly. Statistics about the packet reception ratio (PRR) are updated by each node whenever receiving a valid packet and persistently stored in flash. Once the connectivity evaluation has completed, the statistics from all nodes are collected by the control node using LoRaDisC's collection primitive (marked with ❷ in Fig. 4).

Recent studies have shown a correlation between ambient temperature

and link quality [52]. To allow investigation of such environmental impact on LoRa communication, on-board temperature measurements are piggy-backed to the data collected by the control node at the end of a connectivity evaluation phase.

4.1.3. Daemon Upgrade

Whenever the user wishes to fix bugs and upgrade the daemon with additional features, or to simply change LoRaDisC's configuration, a change in the daemon firmware is necessary. To aid this task, ChirpBox provides a function generating a patch file using *jojodiff* [84] and making this available to the control node. When the `patching` flag in a `SYNC` message is enabled, the aforementioned patch file along with an MD5 verification is disseminated from the control node with LoRaDisC's communication primitive.

Upon reception of the patch file, ChirpBox's daemon generates a patched daemon image in the second memory bank and verifies it using *janpatch* [85]. If the produced image is valid, the daemon switches to the second bank in the same way as when executing a FUT. When running, the patched daemon can realize the current memory bank using the `SYSCFG_MEMRMP` register: if it currently runs on bank 2, it copies itself on bank 1. Thereafter, in the same way as at the end of a FUT execution, ChirpBox resumes the execution of the new patched daemon residing on the first bank. The only difference compared to the execution of a classical FUT is that the flash write protection of the first memory bank should not be enabled before switching to the second one: this would otherwise prevent the patched daemon to copy itself to bank 1. Note that the same patching tools *can be applied to the FUTs*, especially if two consecutive firmwares differ only minimally: this allows the control node to drastically shorten the time required to disseminate the FUT.

4.2. Application Programming Interface (API)

ChirpBox provides an API that can be called by the FUT to timestamp events and to store log files (Sect. 4.2.1), and to support external sensors and application extensions (Sect. 4.2.2), as discussed next.

4.2.1. Basic API

Since there is no observer connected to the target nodes in ChirpBox, the ability of logging messages to flash with an accurate timestamp is beneficial to developers investigating or debugging protocol performance. To support developers in this task, ChirpBox provides a low-level API giving access to

common functions and abstracting the operations of the underlying RTC and GNSS modules. Specifically, ChirpBox allows to accurately timestamp `GPIO` events (on a ms-scale) with a 6-byte Unix timestamp by calling the `timestamp_event()` function. When calling the `log_to_flash()` function, a target node automatically writes in flash the requested information along with the time of the request.

To minimize memory usage, the `log_to_flash()` function stores the formatted strings (as chars) and variables (as binary) separately (rather than converting everything into string directly). By doing so, based on our experience, a 310 kB log file (generated by a test of 3 hours from 21 nodes) in `.txt` format needs only about 3.75 kB of flash memory. The stored log messages are then collected by the control node automatically after the test run completes (marked with ❾ in Fig. 4). With the help of a `Python` script, the laptop connected with the control node converts these log messages to a readable format and save them as `.txt` files.

Note that ChirpBox also foresees *on-site inspections*, where every target node can be connected to a laptop, thereby redirecting logs to be outputted on the serial port (using the `log_to_serial()` function).

4.2.2. APIs for External Sensors

One can easily extend the functionality of ChirpBox by attaching additional sensors to its target nodes. For example, one can connect a microphone [86] to an analog pin of the target node's micro-controller and call the API's `sensor_data_adc()` function to read its values at runtime. By integrating such external sensors to the target nodes, one can generate event-based traffic (rather than periodic traffic), which better resembles the traffic pattern of real-world applications. For example, sensors can be integrated on ChirpBox's target node to test and verify LoRa-based applications such as the interaction of sensor nodes and actuator nodes in realistic environments. Alternatively, one can also use the target node as a low-power data acquisition device (e.g., recording real-world data for later analysis). For example, sound data can be stored locally and sent back to the control node once a test run is over and after switching back to the daemon.

5. Evaluation

We evaluate ChirpBox's performance experimentally by setting up a testbed of 21 nodes in a university campus inside Shanghai over an area

of 28 hectars, as shown in Fig. 10. An assembled ChirpBox target node is shown in Fig. 9(a) and an example of solar-powered target node is shown in Fig. 9(b). The deployment of two gateways is shown in Fig. 9(c). We run a series of tests to answer the following questions:

- What is the reliability of LoRaDisC during data collection and dissemination? How does its performance vary as a function of the file size, the employed SF, and the number of network nodes? How does the position of the control node affect performance? What are the benefits introduced by the LBT and AFA mechanisms? (Sect. 5.1)
- What is the overhead introduced by ChirpBox to orchestrate the testbed's operations? How much energy is consumed to prepare, run, and collect the results of an experiment? How long can ChirpBox's nodes operate before their battery depletes? (Sect. 5.2)

5.1. Performance of LoRaDisC

We start by evaluating the performance of LoRaDisC when collecting and disseminating firmwares throughout the network shown in Fig. 10. In our experiments, we make use of a transmission power of 0 dBm: this results in a multi-hop network with a diameter of 2 and 3 when using a SF of 11 and 7, respectively. Unless differently specified, in all our experiments we use a generation size $M = 16$, a payload length $P = 232$ bytes, and $SF = 7$; we

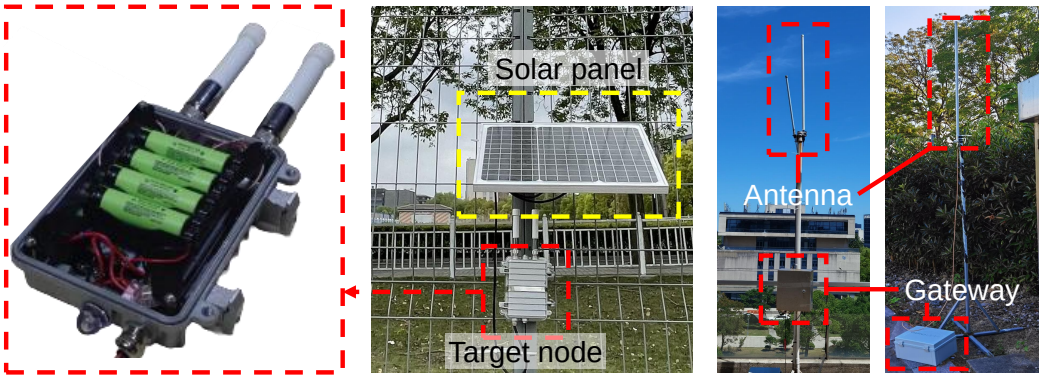


Figure 9: Example of ChirpBox target node in its enclosure (a), as well as deployed target nodes (b) and LoRaWAN gateways (c) in the streets of Shanghai.

also enable both LBT and AFA mechanisms. Each experiment is repeated 3 times. The control node logs the number of flooding rounds and records the time necessary to complete a data collection or dissemination using GNSS timestamps. All target nodes compute their energy consumption in software using Energest [87], following the methodology described in Sect. 5.2.

5.1.1. Performance as a Function of the File Size

We let the control node disseminate firmwares and collect logs of different size to/from 20 target nodes using SF=7. We select a file size between 0.1 and 100 kB, as these values are representative of typical firmwares and logs, (see Table 1).

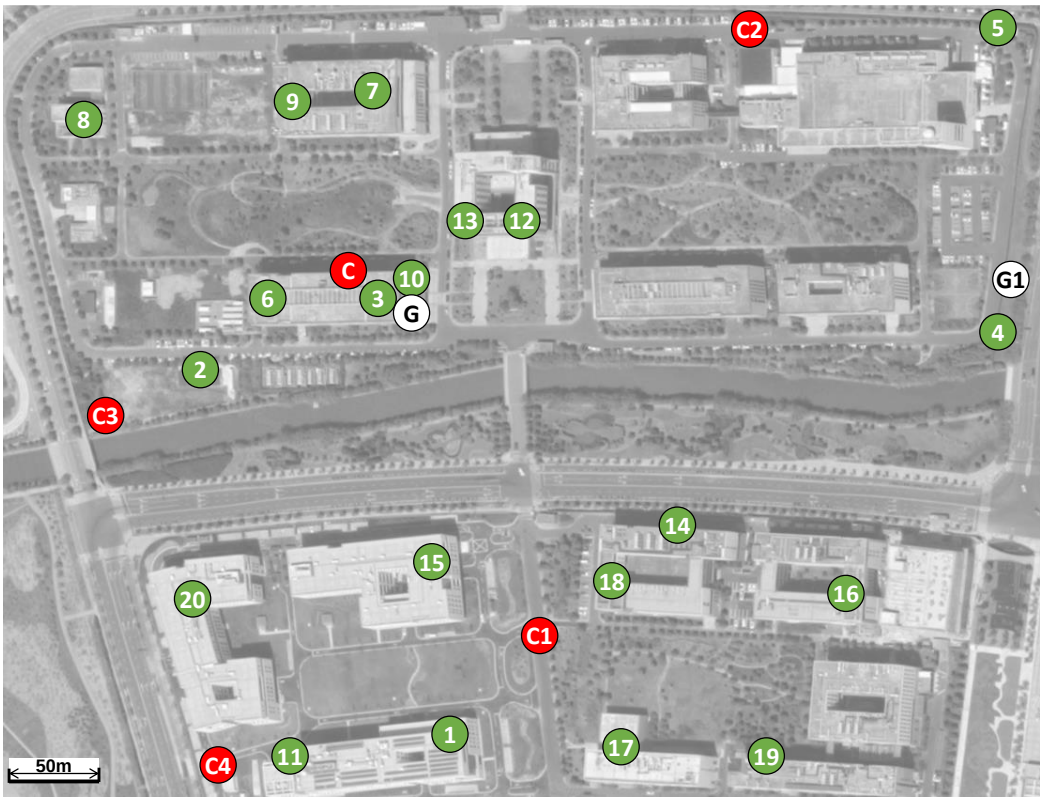
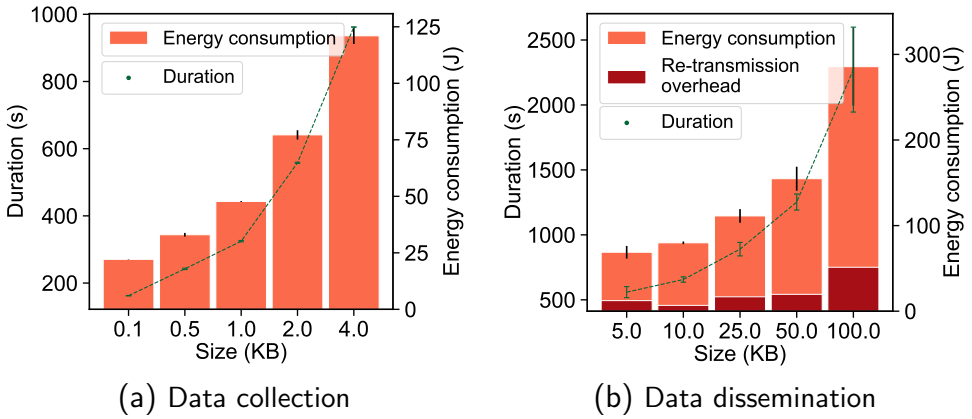


Figure 10: ChirpBox test installation in a university campus with twenty target nodes (green) and one control node (red). The control node is deployed at five different positions (C, C1, C2, C3, C4). Unless differently stated, the experiments are conducted with the control node at position C. The white circles G and G1 mark the location of LoRaWAN gateways.



(a) Data collection (b) Data dissemination

Figure 11: LoRaDisC's performance as a function of file size.

Table 1: Size of exemplary firmwares and log traces.

Data to be transferred	**Size (kB)**	**Primitive**
LED toggling firmware	9.07	Dissemination
LoRaBlink firmware	61.0	Dissemination
Patch (LoRaBlink firmware with different PHY settings)	0.11	Dissemination
ChirpBox's daemon	131.0	Dissemination
Log traces for 512 GPIO events with UNIX timestamp	4.0	Collection
Connectivity logs for 25 nodes (single channel & SF)	0.1	Collection

Fig. 11 depicts the time necessary to complete the collection/dissemination, as well as the average energy consumption of all target nodes. As expected, the larger the file size, the longer the duration of the data exchange and the energy consumption. However, one interesting observation is that the increase in energy and duration is not very significant between 0.1, 0.5, and 1 kB: this is due to the overhead caused by LoRaDisC's configuration round, which employs SF=11.

As a comparison, if one would use a LoRaWAN gateway to disseminate a FUT of 100 kB to all target nodes, at least 462 packets with a payload of 222 bytes are necessary (without accounting for any re-transmission), which requires approximately 280 minutes with a 1% duty cycle (more than

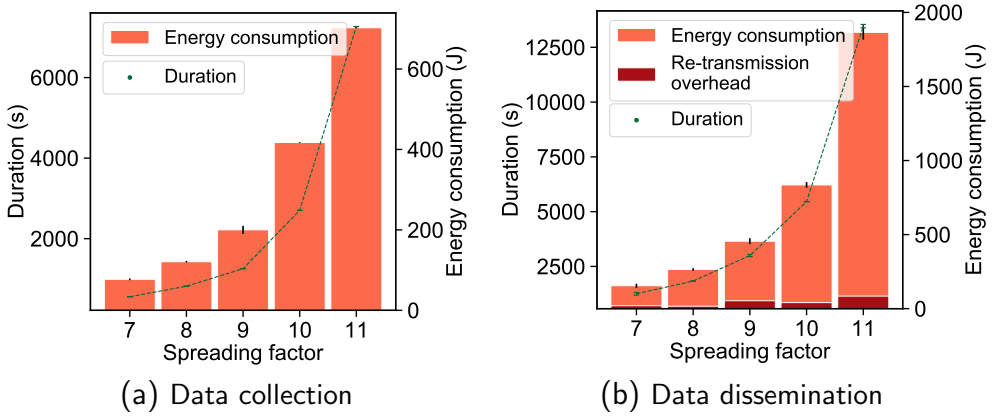


Figure 12: LoRaDisC's performance as a function of SF.

seven times the amount of time taken by LoRaDisC). When collecting small amount of data using a LoRaWAN gateway, one would actually require less time than LoRaDisC (e.g., 747.6 s instead of 962.0 s when collecting 4 kB). This is due to the larger network diameter when using LoRaDisC over a multi-hop network operating at low-power (0 dBm). In order for the LoRaWAN gateway shown in Fig. 10 to successfully communicate to all target nodes in a star topology, a few target nodes need to increase their transmission power to 14 dBm.

In our experiments, *all* firmwares and logs have been reliably exchanged. Note that the numbers shown in Fig. 11 account for the necessary retransmissions: in average, when disseminating data, the control node initiated 13.9% more flooding rounds to carry data blocks that have not been acknowledged by all nodes: the energy consumption caused by these additional rounds is shown in dark red in Fig. 11b.

5.1.2. Performance as a Function of the SF

Fig. 12 shows the time needed to complete the collection of 2 kB logs and the dissemination of a 50 kB firmware image, as well as the average energy consumption of all target nodes when using different SFs. As discussed in Sect. 3.3, we only support SFs from 7 to 11, given that at most 10 bytes can be sent with SF=12 in 1 s due to the European regulations [81].

As expected, the higher the SF, the longer the duration of the data exchange and, correspondingly, the energy consumption – despite the fact that

a lower network diameter can be achieved with a greater SF. Although the data rate achievable when using SF=10 (or 8) is two times slower than that of SF=11 (or 9), the relative differences in the time necessary to complete the collection or dissemination between these SFs are larger than two: this is linked to the limitation on the maximum duration of a transmission (1 s), which negatively affects the performance of higher SFs. Moreover, during dissemination, one can see that the proportion of the retransmission overhead decreases from 12.5% at SF=7 to only 4.5% at SF=11: this confirms that the use of a higher SF increases the robustness of LoRa communications [17]. Note also that, as discussed in Sect. 3.3, according to the spectrum access regulations [81], LoRaDisC automatically restricts the frame length to 232, 232, 182, 84, and 30 bytes for SF 7 to 11, respectively.

5.1.3. Performance as a Function of the Network Size

We execute a series of experiments selecting randomly only a portion (5, 10, and 15) of the target nodes in the network: this allows us to observe the impact of the network size on the performance of LoRaDisC's collection and dissemination. Fig. 13 shows the time necessary to complete the collection of 2 kB logs and the dissemination of a 50 kB firmware to/from a different number of target nodes. The figure also depicts the average energy consumption of the target nodes, indicating the relative differences across different scenarios. For example, when the number of nodes reduces from 15 to 10, the average energy consumption can be reduced by 22.9% during data collection and by 18.2% during data dissemination. As one would expect, the duration of the data exchange and the energy consumption increase with the number of target nodes. Nevertheless, the figure clearly shows the benefits of the CT-based approach embedded in LoRaDisC, as well as those deriving from the use of network coding. Indeed, the increase in both the duration of the data exchange and the energy consumption when doubling or tripling the network size (e.g., from 5 to 10 or 15 nodes) is relatively small: this is because a node is likely to obtain a missing frame from one of its neighbours throughout the flood. Note also that, as in all previous experiments, *all* firmwares and logs have been reliably exchanged.

5.1.4. Performance of Control Node at Different Locations

To explore the impact brought by the control node's position, we evaluate the performance of ChirpBox when collecting 2 kB-large logs and when disseminating 50 kB-large FUTs. We deploy the control node at five dif-

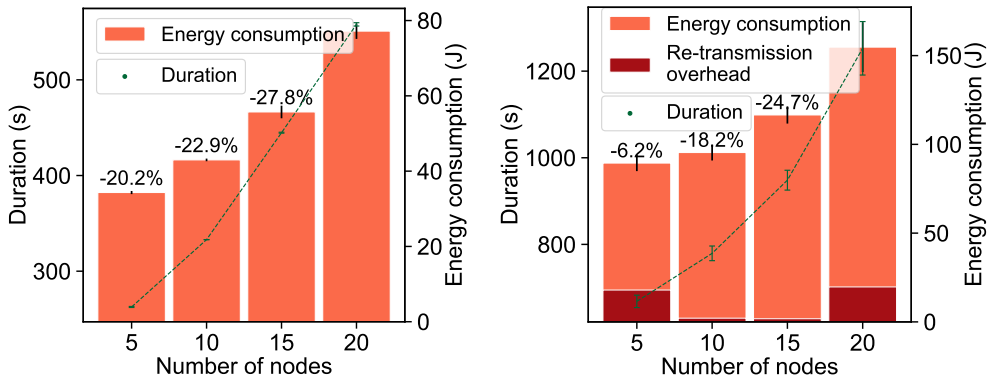


Figure 13: LoRaDisC's performance as a function of the number of target nodes in the network.

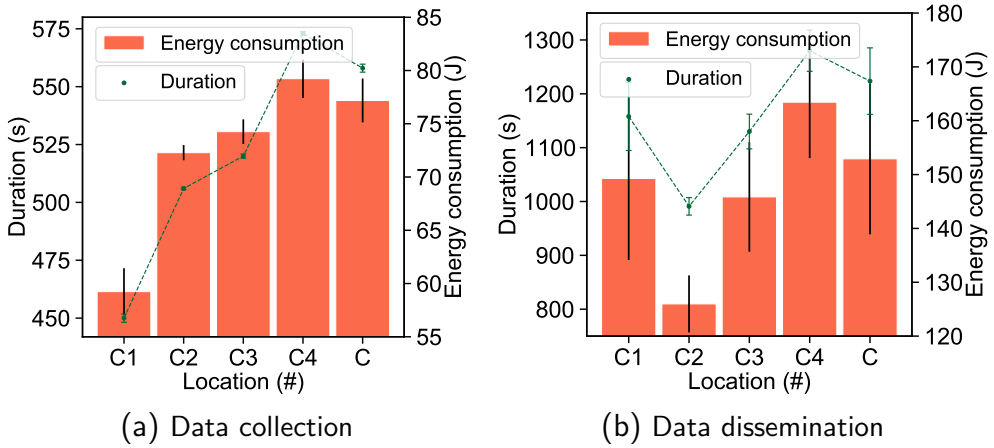


Figure 14: LoRaDisC's performance when placing the control node at different locations.

ferent positions, as marked in Fig. 10. LoRaDisC achieves 100% reliability regardless of the position at which the control node is deployed. However, the collection/dissemination latency and the overall energy consumption do vary slightly.

The results relative to the data collection are illustrated in Fig. 14a. The smallest average energy consumption and the shortest duration are achieved when the control node is placed at location C1 (23.3% less energy than at location C), whereas the most energy is consumed when the control node is

placed at location C4 (2.7% more energy than at location C). By exploiting ChirpBox's connectivity monitor functionality, we find that the control node at C1 has the largest number of neighbors, which indicates that the data from the target nodes can be collected more easily in this configuration. In contrast, when the control node is placed at location C4, it has the least number of neighbors, which leads to the highest energy consumption.

The results relative to the data collection are illustrated in Fig. 14b. The most efficient performance is achieved when the control node is at location C2 (i.e., 21.4% less energy than at location C). Similar to the data collection results, the worst case is when the control node is at location C4 (i.e., 6.4% more energy than at location C). By exploiting ChirpBox's connectivity monitor functionality, we can infer that nodes 4 and 5 have weak links with the rest of the network, which means that nodes 4 and 5 require much more attempts to send their acknowledgements. Because of this, the best performance is achieved when placing the control node at location C2. The number of neighboring target nodes is hence a key factor when selecting the control node's position, and should be carefully dimensioned during deployment.

5.1.5. Benefits Introduced by LBT and AFA

Fig. 15 shows the benefits introduced by the adoption of the LBT and AFA mechanisms in LoRaDisC. The figure shows the effective throughput *over one hour* for data collection and dissemination on a network with 20 target nodes when the two mechanisms are enabled or disabled. When disabling AFA, LoRaDisC only makes use of one channel. When AFA is enabled, 10 channels are used, which means that each flooding round is assigned to 2 out of the 10 channels as the primary channel and the secondary channel, respectively. LoRaDisC ensures that all nodes comply with the regulations, i.e., with a transmission time of at most 100 s and 36 s per hour per channel when LBT is enabled or disabled, respectively. Once the radio usage quota of a channel is used up, a node would keep silent (i.e., stop transmitting data) on this channel until it is allowed again.

When using a SF of 7, LoRaDisC can disseminate 171.5 kB and collect 16 kB of data every hour when LBT and AFA are enabled. This is 1.4, 5.7, and 14.3 times more than when using AFA only, LBT only, and neither of the two. Only 240 bytes and 80 bytes can be disseminated and collected, respectively, when using a SF of 11 without LBT and AFA. This is 2.0, 9.7, and 37.2 times less than when using AFA only, LBT only, and both of them. Note that, when disabling both mechanisms, the regulations limit

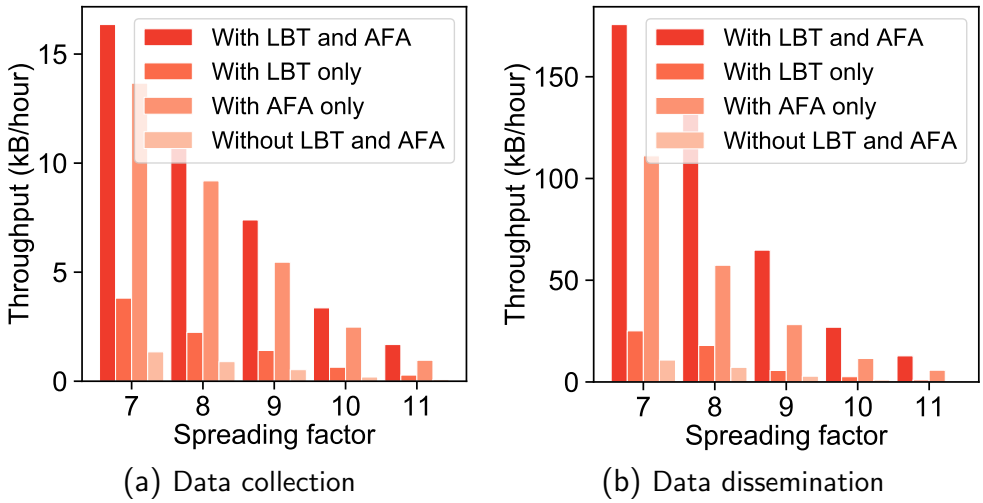


Figure 15: LoRaDisC's throughput when enabling or disabling the LBT/AFA mechanisms for different SFs.

data transmission to 36 s per hour: this especially impacts larger SFs (due to their slower data rate) and emphasizes the benefits of LoRaDisC's flooding of information across the testbed.

5.2. Overhead and Energy Consumption

We analyze next the overhead introduced by ChirpBox when orchestrating the testbed's operations and break down the energy expenditure in each of the phases used to prepare, run, and finalize an experiment.

5.2.1. Energy Measurements

We use a Keithley DMM7510 digital multimeter to monitor the current and voltage of a ChirpBox node during different states. The corresponding power draw and the duration of each state is listed in Table 2. Among others, one can note how the LoRa radio consumes more power while transmitting than while receiving, and that the energy consumption of flash operations differs for the two memory banks.

We use the values in Table 2 as input for Energest [87], the software-based energy estimation used throughout this evaluation. To ensure that this provides sufficiently accurate estimates, we compare the energy consumption estimated by Energest with the one measured in hardware using D-Cube observers [32] attached to two of the target nodes while the testbed

Table 2: Measured energy consumption in various states.

State	**Power (mW)**	**Duration (s)**	**Energy (mJ)**
MCU @running mode	80.96	1	80.96
MCU @sleep mode	53.51	1	53.51
MCU @deep-sleep mode	18.84	1	18.84
Flash writing (bank 1)	47.92	0.022 (2 kB)	1.06
Flash erasing (bank 1)	46.03	0.022 (2 kB)	1.01
Flash writing (bank 2)	73.13	0.022 (2 kB)	1.61
Flash erasing (bank 2)	62.86	0.022 (2 kB)	1.38
Radio TX (14 dBm)	287.65	0.389	111.90
Radio TX (10 dBm)	244.79	0.389	95.23
Radio TX (0 dBm)	207.37	0.389	80.67
Radio RX (BW 125 kHz)	181.72	0.389	70.69
Obtain GNSS time	324.71	0.04	12.99
Switch memory bank	116.10	0.077	8.94

is operational. Specifically, we let D-Cube sample voltage and current simultaneously at a rate of 1 kHz and calibrate its measurements with the Keithley DMM7510.

We measure the energy consumption of ChirpBox while setting up and running a LoRaWAN Class A firmware image of 60.1 kB[10] for half an hour, as well as while collecting 2 kB logs from each target node at the end of the run. In order to compute the energy consumption during the FUT execution, we call `compute_energy()` from the FUT, which makes Energest available to the developer via ChirpBox's API.

Fig. 16 shows the energy consumption of two target nodes broken down for the different operational phases of ChirpBox explained in Fig. 4. Specifically, the different bars of the figure display the energy estimated in software using Energest and that measured in hardware using D-Cube. One can see that the software-based estimation is quite accurate, with an average underestimation in the order of 3% and never above 6.5%. Note that phase ❼ and ❽ are not displayed in the figure: these correspond to a reset operation and their

[10]The FUT lets a node send a 8-byte application payload every 10 s. Ten channels are used by the LoRaWAN gateway to exchange data.

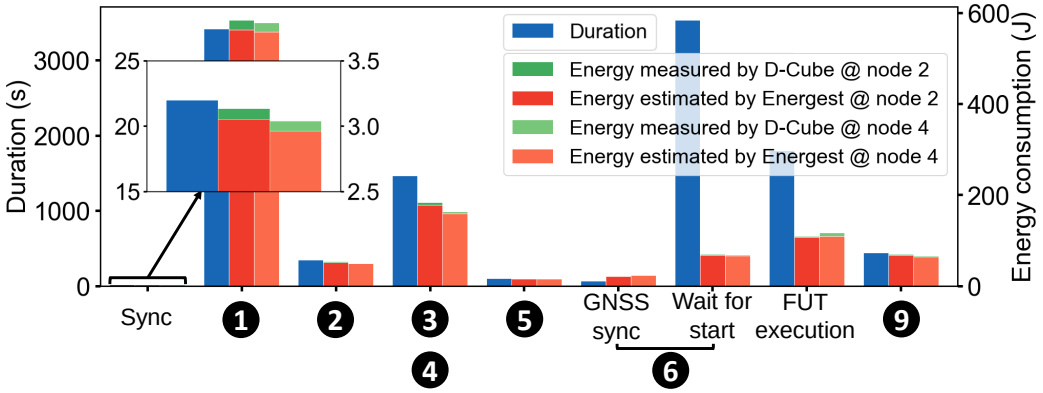


Figure 16: Breakdown of the energy consumed by a ChirpBox node when preparing, running, and finalizing an experiment. The different phases refer to those described in Fig. 4. The four bars compare the energy estimated using Energest with the one measured in hardware on two nodes.

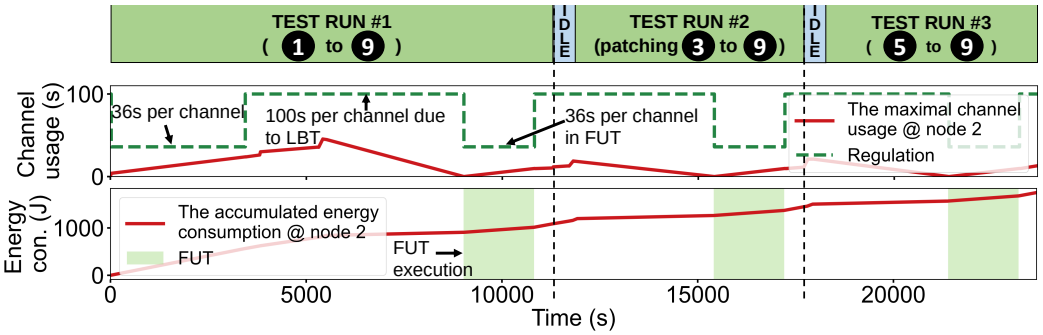


Figure 17: Energy consumption and channel usage of three consecutive test runs. The FUT executed in the second run is a minor modification of the previous one, whereas the third run is a repetition of the second one with different settings.

duration as well as energy consumption are negligible.

5.2.2. Breakdown of Energy Consumption per Phase

Fig. 16 also allows us to analyze the energy consumption for the different operational phases of ChirpBox explained in Fig. 4. Phase ❶ accounts for 30.3% of the test run's duration: this is because the connectivity evaluation is carried out using all possible spreading factors on multiple channels. The waiting time for starting an experiment (phase ❻) also represents a large portion of a test run's duration, as a device needs to wait up to one hour to

free the channel usage for the FUT. Phase ❹, ❼ and ❽ account together for less than 0.1% of the whole run and are hence negligible. The remaining phases ❷, ❸, ❺, and ❾ account for 3.1%, 12.9%, 0.9%, and 3.9%, respectively. Finally, the overhead of a `SYNC` flood is only 0.2% of the entire run.

We further show the energy consumption and the channel usage of a ChirpBox node when running three consecutive test runs. The first run consists in a connectivity evaluation, and in the dissemination of the same firmware employed earlier (60.1 kB), i.e., all nine phases ❶ to ❾ are executed. In the second and third run, no connectivity evaluation is carried out, and the firmware has only minor changes compared to the one used in the first run. ChirpBox hence disseminates only a 4.2 kB patch file in the second run, as discussed in Sect. 4.1.3, executing only phases ❸ to ❾. The third run makes use of the same firmware as the second run but with a different run settings, which allows ChirpBox to only execute phases from ❺ to ❾.

Fig. 17 (bottom) shows the results. It takes more than two and a half hours and about 900 J for each node to prepare, run, and collect the results of an experiment when executing all nine phases. Instead, it takes 4130 and 3725 s, as well as about 180 and 126 J to prepare, run, and collect the results of the second and third experiment. This is due to the high duration and energy consumption of the connectivity evaluation phase, as shown in Fig. 16. If the user updates the FUT with ChirpBox's patching functionality, the duration of the dissemination and the consumed energy is reduced by 74 and 70%, respectively, compared to when disseminating the full-size firmware. Fig. 17 (top) also illustrate the maximum channel usage (red solid line) among all nodes in the network: one can see that the maximum channel usage permitted by the regulations (dashed green line) is never exceeded[11].

5.2.3. Lifetime of ChirpBox nodes

Based on the previous results, we can estimate the lifetime of ChirpBox nodes as a function of the available battery capacity. As discussed in Sect. 3.1, we use four Panasonic NCR18650B batteries in our implementation. These batteries have a nominal capacity of 3400 mAh when used until 2.7 V. However, ChirpBox operates only at voltages above 3.3 V: we measure that the four batteries can provide at most 10.3 Ah until this voltage is reached. This

[11] During a connectivity evaluation and while running a LoRaWAN Class A FUT, a node can transmit for at most 36 s per hour per channel, as LBT is disabled. In the remaining time, LBT is always enabled, resulting in up to 100 s of transmissions per hour per channel.

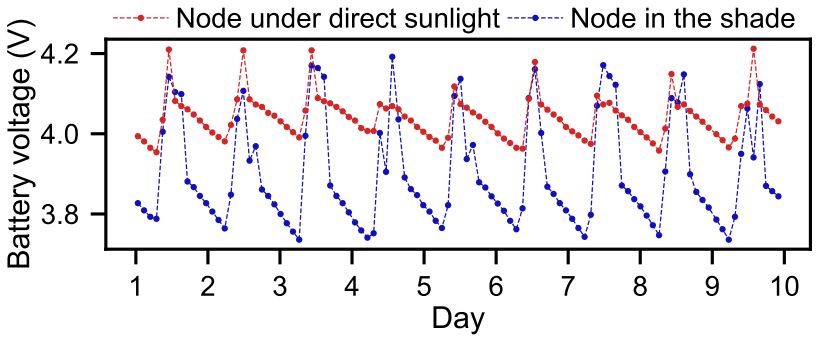


Figure 18: Exemplary battery voltage of one solar-powered node receiving direct sunlight throughout the day (red line) and one solar-powered node placed in the shade (blue line). Both nodes exhibit a sufficient voltage level, ensuring stable and perpetual operations.

results in about 112 consecutive runs (≈ 14 days) including all nine phases. When making use of the patching function as in the second run shown in Fig. 17, the nodes can execute about 343 runs (≈ 24 days). About 400 runs (≈ 27 days) can be performed when simply repeating the experiment as in the third run. These results are clearly specific to our implementation. Nevertheless, depending on the user's requirements, a larger battery can be selected, or a solar panel can be mounted on top of the nodes to recharge the batteries over time and prolong the achievable battery lifetime.

We also perform evaluations on battery-powered nodes that are charged by solar panels. Specifically, we carry out the connectivity evaluation (phase ❶) every 2 hours on three channels ranging from SF 7 to SF 12 for up to 10 days (with an average of 10.3 hours of daylight each). The results are illustrated in Fig. 18, and demonstrate that – despite the hindrance of shading resulting in a lack of direct sunlight – the nodes are able to sustain a high and constant battery voltage. This ensured stable and continuous operation of the nodes exclusively through solar energy.

6. ChirpBox in Action

We finally make use of ChirpBox to benchmark the performance of LoRa-based protocols and to evaluate the impact of temperature variations on network connectivity.

6.1. Benchmarking Protocol Performance

We compare the end-to-end latency, energy consumption, and channel usage of three LoRa-based protocols (LoRaDisC with LBT and AFA,

LoRaBlink, and LoRaWAN) when disseminating a firmware image of 50 kB across the test installation shown in Fig. 10. We use LoRaDisC following the same settings as in the previous section: as the protocol uses the LBT mechanism, transmissions up to 100 s per hour per channel are allowed by the regulations. LoRaBlink [17] is a multi-hop protocol that enables nodes to transmit identical packets concurrently by exploiting the capture effect to avoid collisions. To evaluate its performance, we use the open LoRaBlink firmware[12]. In our LoRaBlink evaluation, we set the control node C to flood beacon packets every 7.5 s (epoch) to avoid nodes being desynchronized. Since there is no LBT mechanism in LoRaBlink, a node can only transmit data for at most 36 s per hour per channel. Whilst LoRaDisC and LoRaBlink are used on top of a multi-hop network, LoRaWAN forms a star network rooted at the gateway G shown in Fig. 10. LoRaWAN has its own firmware update over the air (FUOTA) process [88], where a LoRaWAN gateway delivers the firmware to multiple end-devices (in a multicast group) by using the LoRaWAN Class C protocol. Specifically, end devices can listen all the time except in transmit mode, resulting in low-latency communication. The gateway operates at a 10% duty cycle; nodes in the same multicast group keep listening on a specific channel, where transmissions for up to 360 s per hour are allowed. To make a fair comparison, we mainly focus on the dissemination of chunks and do not consider the digital signature procedure of the firmware.

Fig. 19 shows the performance of the tree protocols over three different runs. LoRaDisC achieves 100% reliability (Fig. 19a) and exhibits the least energy consumption (Fig. 19b) while keeping the channel usage well below the one imposed by the regulations (Fig. 19c). As expected, LoRaWAN Class C is the fastest protocol to complete the dissemination, requiring 20% less time compared to LoRaDisC. However, despite the short duration of the dissemination, the protocol still requires 15% more energy compared to LoRaDisC due to the continuous listening activity. Using a LoRaWAN Class A firmware would result in a 6.6 times higher duration and a 3.4 higher energy consumption when trying to meet the regulations[13] (results omitted due to space constraints). LoRaBlink exhibits the lowest reliability, the longest dissem-

[12] Dec. 14, 2015 version, available at `https://www.lancaster.ac.uk/scc/sites/lora/lorablinkkit.html`

[13] A gateway cannot multicast packets to end nodes in LoRaWAN Class A.

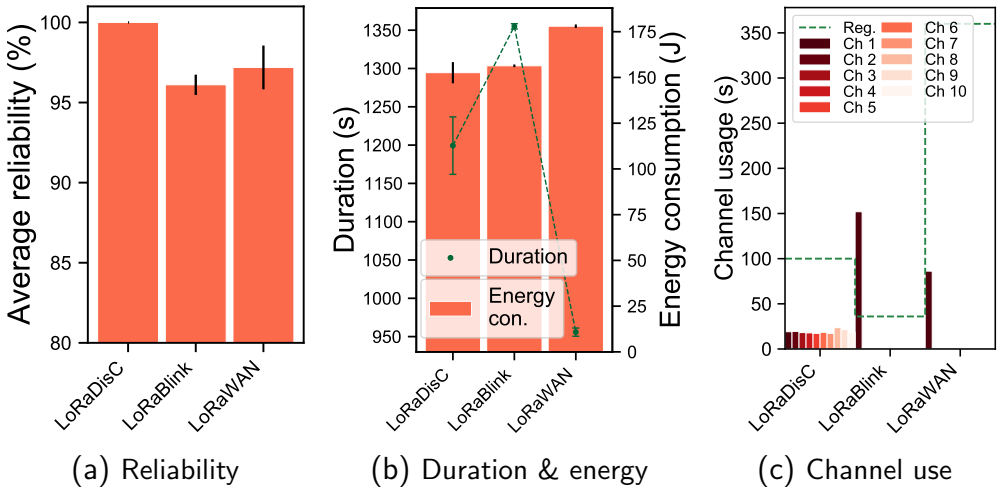


Figure 19: Performance of LoRaDisC, LoRaBlink, and LoRaWAN when disseminating a 50 kB file across a network.

ination latency, and a comparable energy expenditure as LoRaDisC – this despite making use of the channel for approximately 4 times higher than what would be allowed by the regulations (Fig. 19c). Instead, if LoRaBlink would meet the channel usage regulations, the duration of its dissemination would increase by 6.6 times[14].

6.2. Investigating LoRaWAN's Performance

We deploy a LoRaWAN gateway at location G and G1, respectively, as marked in Fig. 10, and enable target nodes to send packets periodically using LoRaWAN Class A. The LoRaWAN gateway listens to eight channels, and ChirpBox's target nodes upload packets on a random channel (one out of these eight channels). We observe the reliability of communications by running tests with a transmission power of 0 dBm and SFs from 7 to 12. In order to investigate how collisions affect the reliability of such an ALOHA network, we also let target nodes send packets in a round-robin manner (i.e., we ensure absence of internal interference) by exploiting the GNSS module of ChirpBox as a reference.

[14] A LoRaBlink node needs to transmit at least 420.36 ms data per epoch to flood a packet: with 1% duty cycle, an epoch is 42.036 s, which translates to 8995.7 s when transmitting 50 kB (214 epochs).

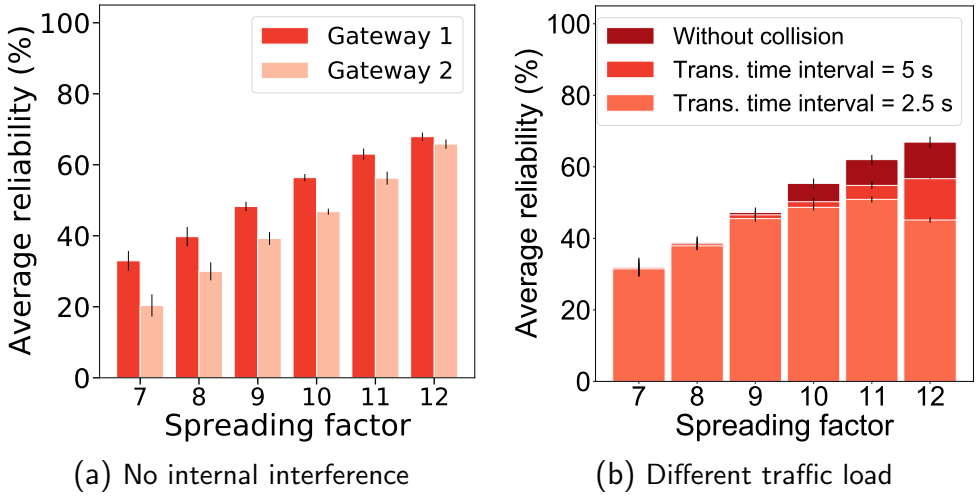


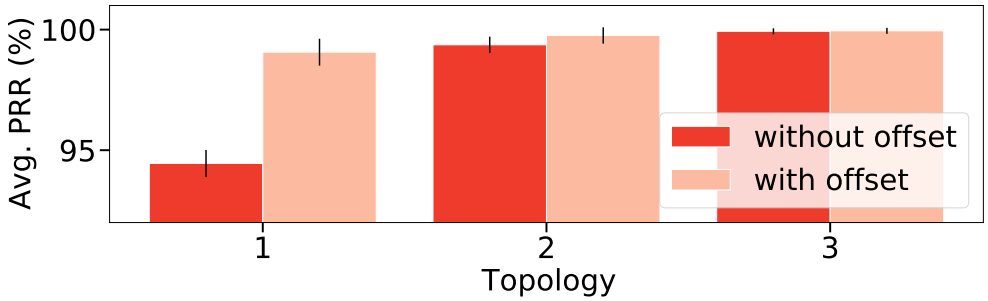
Figure 20: Reliability of LoRaWAN communications when using different gateway locations and with different traffic loads.

Fig. 20a shows the average PRR of such collision-free test when the gateway is placed at location G and G1, respectively. Since greater SFs enable a larger coverage at the price of a lower data rate, the reliability and energy consumption both increase significantly when using a high SF (i.e., the PRR at SF=12 at least doubles, but about 26 times more energy is consumed compared to SF=7). We also find that the gateway's position becomes less important when using a higher SF. The reliability of G1 is lower than that of G when sending packets with a lower SF (e.g., 12.6% less than G when using SF=7), and is almost the same when the employed SF increases (e.g., only 2.1% lower reliability than G when using SF=12).

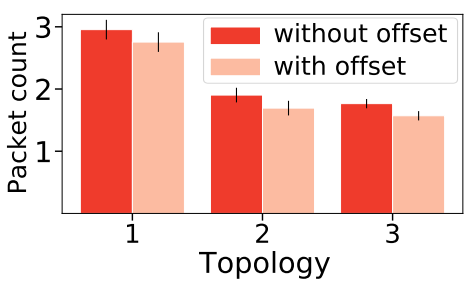
Fig. 20b illustrates the reliability of LoRaWAN communications for different traffic loads, i.e., when uploading data at different times – every 5 s and 2.5 s, respectively. In these experiments, the gateway is placed at G. We find that the slow data rate limits the throughput (the packet time on air is nearly 1.5 s when using SF=12). The average PRR, instead, does not increase as that of the collision-free tests; instead, it drops slightly when a period of 2.5 s and SF=12 is adopted.

6.3. Investigating the "Offset Insert" Concept

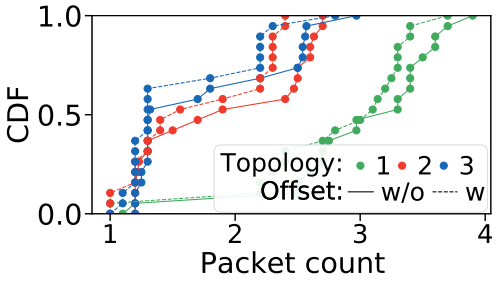
Liao et al. [18] verified the feasibility of CT over LoRa, and improved its reliability by inserting an additional time offset in each CT slot. The latter



(a) Reliability as a function of the chosen topology



(b) Avg. first received packet



(c) CDF of first received packet

Figure 21: Evaluating the impact of the *offset insert* concept.

helps spreading the energy of interference (i.e., signals from other CT nodes) in the time domain, thereby ensuring that packets are decoded successfully.

We verify this *offset insert* concept on the deployed ChirpBox network presented before. In order to make our study topology-independent, we set nodes 5, 11, and 8 as flooding initiators and denote these three configurations as topologies 1–3, respectively. The initiator triggers the network by actively sending a packet, whereas the remaining nodes simply re-broadcast received packets immediately, following an *rx-tx-rx-tx* pattern. Each node has 5 transmission attempts during each flooding round. We also set SF=7 in order to create a larger multi-hop network. The logged stats contain PRR information as well as an indication of the first packet being received.

Fig. 21a shows the reliability in terms of average PRR: one can notice that the position of the flood initiator does affect performance. Specifically, the reliability degrades as the network diameter increases (i.e., the average PRR is only 94% in topology 1 with the conventional CT flooding). However, by using the *offset insert*, the reliability of topology 1 is improved significantly,

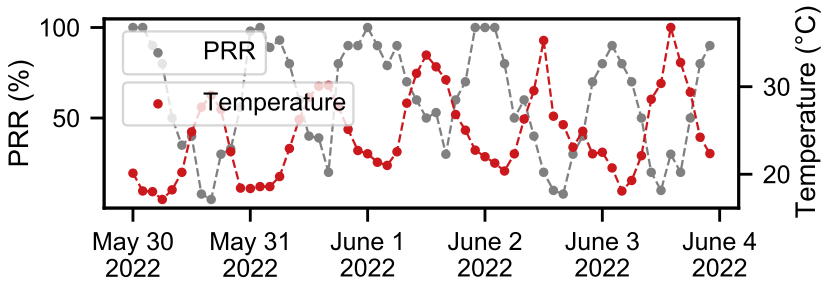


Figure 22: Impact of ambient temperature on the PRR between `node 7 and 3`. We use 480 MHz as channel and SF=7.

with the PRR approaching 100%.

With the help of the logged information about the first packet being received, we can show that the *offset insert* really increases the chances to decode a CT packet correctly, thereby decreasing the necessary number of packet retransmissions. Fig. 21b and Fig. 21c confirm that less retransmissions are required in average to receive the first packet when applying the offset. This result is consistent for all the three considered topologies.

6.4. Observing the Impact of the Environment on LoRa Networks

Using the testbed health status information collected by ChirpBox as described in Sect. 4.1.2, one can seamlessly collect information about the on-board temperature of each target node over time and inspect whether temperature variations have an impact on the connectivity in the testbed. We record the connectivity information in the nodes deployed in the university campus as a function of temperature over more than one year. Fig. 22 shows the exemplary link quality between node 7 and node 3 over the course of a week. The PRR measured at node 3 fluctuates periodically, decreasing significantly when the temperature rises (i.e., during daytime) and recovering when the temperature decreases (i.e., during nighttime). Fig. 23 shows the RSS recorded by node 1 when receiving packets from node 20 over a 12-month period (May 2021 to April 2022) when utilizing 480 MHz as channel and when employing SF=7. The RSS values clearly correlate with temperature, with lower RSS values in summer (higher temperature) and higher RSS values in winter (lower temperature), respectively. These results confirm earlier studies [35, 52] and indicate that higher temperatures negatively affects LoRa communication.

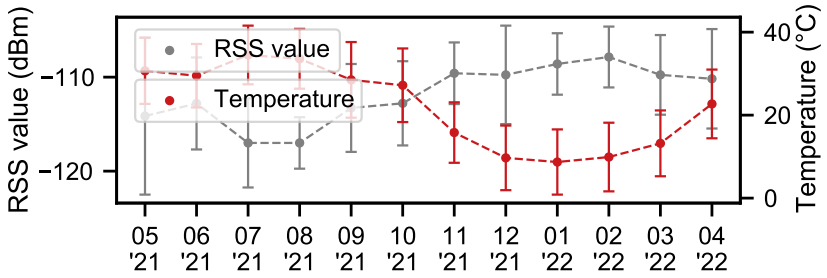


Figure 23: Impact of ambient temperature on the RSS values recorded by `node 1` when receiving packets from `node 20` over a 12-month period (May 2021 to April 2022). We use 480 MHz as channel and SF=7.

7. Related Work

We now analyse related work w.r.t. existing LoRa-based testbed facilities and CT-based protocols on top of LoRa.

LoRa-based Testbed Infrastructures Several testbeds have been purposely developed to evaluate the reliability and scalability of LoRa systems [11, 37, 40, 41, 42, 43, 44, 45, 46], but most of these facilities are not publicly available. Recently, a few LoRa devices have also been supported in public testbed infrastructures such as FlockLab 2 [38, 50], FIT IoT-Lab [39], and UMBRELLA [89], but mostly with a limited number of indoor nodes[15]. To simplify the reprogramming and management of the target LoRa nodes, all these facilities make use of an existing network backbone, typically based on Ethernet or cellular networks [51, 90, 89]. As a network backbone is not always available and as the use of cellular networks incurs high operational costs, Kazdaridis et al. [41] have only used LoRa nodes for experimentation. Their approach consists in exploiting LoRaWAN gateways to send commands with the parameter configuration to the target nodes and collect statistics. However, the firmware needs to be programmed manually prior deployment and only star topologies are supported, which limits scalability. In ChirpBox, instead, we propose a full-fledged multi-hop testbed that supports re-programming of LoRa nodes and an efficient collection of log traces.

Concurrent Transmissions on top of LoRa CT have been widely pop-

[15]UMBRELLA supports more than 200 LoRa nodes, but nodes are deployed in a line topology – a placement that is rather rare in real-world applications.

ular in the context of low-power wireless systems since the development of Glossy [68]. This influential work led to the design of a large number of CT-based data collection and dissemination protocols for IEEE 802.15.4 systems [64]. Recent studies have also shown the feasibility of CT on top of other technologies, such as Bluetooth Low Energy [82], ultra-wideband [91], and LoRa [17, 18]. Specifically, when it comes to LoRa, Bor et al. [17] have been the first to experimentally demonstrate the existence of non-destructive CTs under given conditions (i.e., packets time offset, power difference, bit rate). These results have been extended by Liao et al. [18], who have theoretically analyzed the feasibility as well as developed a prototypic implementation of CT on top of LoRa. Our work builds upon these two studies and proposes, to the best of our knowledge, *the very first CT-based multi-hop data collection and dissemination protocol for LoRa-based networks*. Such protocol, LoRaDisC, copes with LoRa's limited data rate and regional duty-cycle constraints and allows ChirpBox to sustain a reliable and efficient data exchange across the testbed. LoRaDisC embeds several of the features that have increased the reliability and efficiency of state-of-the-art CT-based protocols for IEEE 802.15.4 networks, such as the use of ACK rounds and network coding [79, 92].

8. Conclusions and Future Work

We have presented ChirpBox, a low-cost and infrastructure-less LoRa testbed that can be deployed in remote areas without cellular coverage and without a backbone infrastructure allowing to efficiently communicate with the target nodes and supply them with power. Thanks to LoRaDisC, the first CT-based all-to-all multi-channel protocol on top of LoRa, ChirpBox can reliably and efficiently disseminate as well as collect data through the network, which allows an easy orchestration of the testbed activities by reusing the LoRa-based target nodes, as demonstrated in our experimental evaluation. We believe that ChirpBox's low-cost and open-source availability will simplify the deployment of outdoor testbeds and the benchmarking of communication protocols, thereby fostering research on LoRa-based systems in the years to come.

Looking ahead, we see the need for aperiodic traffic generation in order to emulate event-driven applications, which can be achieved with the help of the programmable GNSS module. To support the addition or removal of target nodes after deployment, the built-in LoRaDisC protocol needs to periodically

form a network (i.e., it should update the online targets): we will enable this functionality in future work. Another promising avenue is the design of a common and openly-accessible cloud service for several ChirpBox instances (where a ChirpBox instance is defined as a network managed by a single control node).

Acknowledgement

This work is partially funded by the Special Fund for Basic Research on Scientific Instruments of the National Natural Science Foundation of China (No. 51827814), the Science and Technology Innovation Plan of Shanghai Science and Technology Commission (20DZ1201002), and the China Scholarship Council (CSC). This work is also partially supported by the TU Graz LEAD project "Dependable IoT in Adverse Environments".

References

[1] P. Tian, et al., ChirpBox: An Infrastructure-Less LoRa Testbed, in: Proc. of the 18th EWSN Conf., 2021.

[2] U. Raza, et al., Low Power Wide Area Networks: An Overview, IEEE Communications Surveys & Tutorials 19 (2) (2017).

[3] K. Mekki, et al., A Comparative Study of LPWAN Technologies for Large-Scale IoT Deployment, ICT Express 5 (1) (2019).

[4] J. Petäjäjärvi, et al., On the Coverage of LPWANs: Range Evaluation and Channel Attenuation Model for LoRa Technology, in: Proc. of the 14th ITST Conf., 2015.

[5] K.E. Nolan, et al., An Evaluation of Low Power Wide Area Network Technologies for The Internet of Things, in: Proc. of the 12th IWCMC Conf., IEEE, 2016.

[6] R.S. Sinha, et al., A Survey on LPWA Technology: LoRa and NB-IoT, Ict Express 3 (1) (2017).

[7] N. Silva, et al., Low-Cost IoT LoRa Solutions for Precision Agriculture Monitoring Practices, in: Proc. of the 19th EPIA Conf., 2019.

[8] Y. Cheng, et al., Secure Smart Metering based on LoRa Technology, in: Proc. of the 4th ISBA Conf., 2018.

[9] M. Cattani, et al., Adige: An Efficient Smart Water Network based on Long-Range Wireless Technology, in: Proc. of the 3rd CySWATER Workshop, 2017.

[10] P. Basford, et al., LoRaWAN for Smart City IoT Deployments: A Long Term Evaluation, Sensors 20 (3) (2020).

[11] G. Pasolini, et al., Smart City Pilot Projects Using LoRa and IEEE 802.15.4 Technologies, Sensors 18 (4) (2018).

[12] L. Tessaro, et al., LoRa Performance in Short Range Industrial Applications, in: Proc. of the 23rd SPEEDAM Symp., 2018.

[13] Huawei, The Race is on: NB-IoT Boosts Smart Bike Sharing for ofo, [Online] `https://bit.ly/2QMxkeT` – Last accessed: 2022-11-21.

[14] B. Foubert, et al., Long-Range Wireless Radio Technologies: A Survey, Future Internet 12 (1) (2020).

[15] LoRa Alliance, LoRaWAN Specification, v1.1, [Online] `http://bit.ly/3NYM3QF` – Last access: 2022-11-21 (2017).

[16] B. Sartori, et al., Enabling RPL Multihop Communications based on LoRa, in: Proc. of the 13th WiMob Conf., 2017.

[17] M. Bor, et al., LoRa for the Internet of Things, in: Proc. of the 1st MadCom Workshop, 2016.

[18] C.-H. Liao, et al., Multi-Hop LoRa Networks Enabled by Concurrent Transmission, IEEE Access 5 (2017).

[19] P. Marcelis, et al., DaRe: Data Recovery through Application Layer Coding for LoRaWAN, in: Proc. of the 2nd IoTDI Conf., 2017.

[20] M. Sandell, et al., Application Layer Coding for IoT: Benefits, Limitations, and Implementation Aspects, IEEE Syst J. 13 (1) (2019).

[21] S. Demetri, et al., Automated Estimation of Link Quality for LoRa: A Remote Sensing Approach, in: Proc. of the 18th IPSN Conf., 2019.

[22] M. Bor, et al., LoRa Transmission Parameter Selection, in: Proc. of the 13th DCOSS Conf., 2017.

[23] F. Yang, et al., EMU: Increasing the Performance and Applicability of LoRa through Chirp Emulation, Snipping, and Multiplexing, in: Proc. of the 21st IPSN Conf., IEEE, 2022.

[24] S. Tong, et al., De-spreading Over the Air: Long-Range CTC for Diverse Receivers with LoRa, in: Proc. of the 28th MobiCom Conf., 2022.

[25] J. Shi, et al., LoRaBee: Cross-Technology Communication from LoRa to ZigBee via Payload Encoding, in: Proc. of the 27th ICNP Conf., 2019.

[26] J. Sundaram, et al., A Survey on LoRa Networking: Research Problems, Current Solutions, and Open Issues, IEEE Commun. Surv. Tutor. 22 (1) (2020).

[27] C. Li, et al., NELoRa: Towards Ultra-low SNR LoRa Communication with Neural-enhanced Demodulation, in: Proc. of the 19th ACM SenSys conf., 2021.

[28] C. Li, et al., CurvingLoRa to Boost LoRa Network Throughput via Concurrent Transmission, in: Proc. of the 19th NSDI Conf., 2022.

[29] C. Boano, et al., IoTBench: Towards a Benchmark for Low-power Wireless Networking, in: Proc. of the 1^{st} CPSBench Worksh., 2018.

[30] G. Werner-Allen, et al., MoteLab: A Wireless Sensor Network Testbed, in: Proc. of the 4th IPSN Conf., 2005.

[31] R. Lim, et al., FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems, in: Proc. of the 12th IPSN Conf., 2013.

[32] M. Schuß, et al., A Competition to Push the Dependability of Low-Power Wireless Protocols to the Edge, in: Proc. of the 14th EWSN Conf., 2017.

[33] R. Lim, et al., Testbed Assisted Control Flow Tracing for Wireless Embedded Systems, in: Proc. of the 14th EWSN Conf., 2017.

[34] M. Baddeley, et al., The Impact of the Physical Layer on the Performance of Concurrent Transmissions, in: Proc. of the 28th ICNP Conf., 2020.

[35] M. Cattani, et al., An Experimental Evaluation of the Reliability of LoRa Long-Range Low-Power Wireless Communication, Journal of Sensor and Actuator Networks (JSAN) 6 (2) (2017).

[36] A. Gamage, et al., LMAC: Efficient Carrier-Sense Multiple Access for LoRa, in: Proc. of the 26th MobiCom Conf., 2020.

[37] J. Marais, et al., LoRa and LoRaWAN Testbeds: A Review, in: Proc. of the IEEE AFRICON Conf., 2017.

[38] R. Trüb, et al., FlockLab 2: Multi-Modal Testing and Validation for Wireless IoT, in: Proc. of the 3rd CPS-IoTBench Workshop, 2020.

[39] C. Adjih, et al., FIT IoT-LAB: A Large Scale Open Experimental IoT Testbed, in: Proc. of the 2nd WF-IoT Forum, 2015.

[40] Y. Gao, et al., LinkLab: A Scalable and Heterogeneous Testbed for Remotely Developing and Experimenting IoT Applications, in: Proc. of the 5th IoTDI Conf., 2020.

[41] G. Kazdaridis, et al., Evaluation of LoRa Performance in a City-Wide Testbed: Experimentation Insights and Findings, in: Proc. of the 13th WiNTECH Workshop, 2019.

[42] A. Yousuf, et al., A Low-cost LoRaWAN Testbed for IoT: Implementation and Measurements, in: Proc. of the 4th WF-IoT Forum, 2018.

[43] Z. Wang, et al., Dandelion: An Online Testbed for LoRa Development, in: Proc. of the 15th MSN Conf., 2019.

[44] J. Marais, et al., Evaluating the LoRaWAN Protocol using a Permanent Outdoor Testbed, IEEE Sensors 19 (12) (2019).

[45] I. Rodriguez, et al., The Gigantium Smart City Living Lab: A Multi-Arena LoRa-based Testbed, in: Proc. of the 15th ISWCS Symp., 2018.

[46] J. Struye, et al., The CityLab Testbed – Large-scale Multi-technology Wireless Experimentation in a City Environment, in: Proc. of the INFOCOM Workshops, 2018.

[47] C. Hakkenberg, Experimental Evaluation of LoRa(WAN) in Indoor and Outdoor Environments, Master's thesis, University of Twente (2016).

[48] P. Appavoo, et al., Indriya2: A Heterogeneous Wireless Sensor Network (WSN) Testbed, in: Proc. of the 13th TridentCom Conf., 2018.

[49] R. Lim, et al., TraceLab: A Testbed for Fine-Grained Tracing of Time Sensitive Behavior in Wireless Sensor Networks, in: Proc. of the LCN Workshops, 2015.

[50] R. Trüb, et al., A Testbed for Long-Range LoRa Communication, in: Proc. of the 18^{th} IPSN Conf., demo session, 2019.

[51] Q. Lone, et al., WiSH-WalT: A Framework for Controllable and Reproducible LoRa Testbeds, in: Proc. of the 29th PIMRC Symp., 2018.

[52] C. Boano, et al., Impact of Temperature Variations on the Reliability of LoRa: An Experimental Evaluation, in: Proc. of the 7th SENSORNETS Conf., 2018.

[53] M. Schuß, et al., Moving Beyond Competitions: Extending D-Cube to Seamlessly Benchmark Low-Power Wireless Systems, in: Proc. of the 1st CPSBench Workshop, 2018.

[54] R. Jacob, et al., Towards a Methodology for Experimental Evaluation in Low-Power Wireless Networking, in: Proc. of the 2nd CPS-IoTBench Workshop, 2019.

[55] The Things Network, STM32L476 Nucleo-64 + SX1276RF1IAS, [Online] `https://bit.ly/2FhCvRX` – Last accessed: 2022-11-21.

[56] NavSpark, Arduino-compatible Dev. Board with GPS/GLONASS, [Online] `https://bit.ly/3ipDVYE` – Last accessed: 2022-11-21.

[57] Maxim Integrated, DS3231-Extremely Accurate I2C-Integrated RTC/TCXO/Crystal, [Online] `https://bit.ly/3hpkH5C` – Last accessed: 2022-11-21.

[58] STMicroelectronics, On-the-fly Firmware Update for Dual Bank STM32 Microcontrollers, [Online] bit.ly/3Xpnxwn – Last accessed: 2022-11-21 (2019).

[59] Texas Instruments, MSP432P401R, MSP432P401M Mixed-Signal Microcontrollers, [Online] bit.ly/3VCh67x – Last accessed: 2022-11-21 (2015).

[60] Renesas RA Family, RA6 Secure Firmware Update using MCUboot and Flash Dual Bank, [Online] bit.ly/3GEngPV – Last accessed: 2022-11-21 (2022).

[61] Microchip, SAM3X/SAM3A Series Atmel SMART ARM-based MCU DATASHEET, [Online] bit.ly/3XpD1jS – Last accessed: 2022-11-21 (2021).

[62] STMicroelectronics, STM32 Microcontroller System Memory Boot Mode, [Online] bit.ly/3XulYNt – Last accessed: 2022-11-21 (2022).

[63] Arduino, Arduino® UNO R3 Datasheet, [Online] bit.ly/3gAN2Ki – Last accessed: 2022-11-21 (2022).

[64] M. Zimmerling, et al., Synchronous Transmissions in Low-Power Wireless: A Survey of Communication Protocols and Network Services, ACM Computing Surveys (CSUR) 53 (6) (2020).

[65] K. Leentvaar, et al., The Capture Effect in FM Receivers, IEEE Transactions on Communications 24 (5) (1976).

[66] V. Rao, et al., Murphy loves CI: Unfolding and improving constructive interference in WSNs, in: Proc. of the 35th INFOCOM Conf., 2016.

[67] Y. Wang, et al., Exploiting Constructive Interference for Scalable Flooding in Wireless Networks, IEEE/ACM Transactions on Networking 21 (6) (2013).

[68] F. Ferrari, et al., Efficient Network Flooding and Time Synchronization with Glossy, in: Proc. of the 10th IPSN Conf., 2011.

[69] C. Boano, et al., EWSN Dependability Competition: Experiences and Lessons Learned, IEEE Internet of Things Newsletter (Mar. 2017).

[70] P. Dutta, et al., Wireless ACK Collisions Not Considered Harmful, in: Proc. of the 7th HotNets Workshop, 2008.

[71] T. Chang, et al., Constructive Interference in 802.15.4: A Tutorial, IEEE Communications Surveys & Tutorials 21 (1) (2019).

[72] M. Trobinger, et al., One Flood to Route Them All: Ultra-Fast Convergecast of Concurrent Flows over UWB, in: Proc. of the 18th SenSys Conf., ACM, 2020.

[73] B.A. Nahas, et al., BlueFlood: Concurrent Transmissions for Multi-Hop Bluetooth 5 – Modeling and Evaluation, ACM Transactions on Internet of Things 2 (4) (2021).

[74] W. Du, et al., When Pipelines Meet Fountain: Fast Data Dissemination in Wireless Sensor Networks, in: Proc. of the 13th SenSys Conf., 2015.

[75] M. Mohammad, et al., Codecast: Supporting Data Driven In-network Processing for Low-power Wireless Sensor Networks, in: Proc. of the 17th IPSN Conf., IEEE, 2018.

[76] D. MacKay, Fountain Codes, IEE Proceedings-Communications 152 (6) (2005).

[77] M. Luby, LT Codes, in: Proc. of the 43rd FOCS Conf., 2002.

[78] T. Ho, et al., A Random Linear Network Coding Approach to Multicast, IEEE Transactions on information theory 52 (10) (2006).

[79] C. Herrmann, et al., Mixer: Efficient Many-to-All Broadcast in Dynamic Wireless Mesh Networks, in: Proc. of the 16th SenSys Conf., 2018.

[80] M. Saelens, et al., Impact of EU Duty Cycle and Transmission Power Limitations for Sub-GHz LPWAN SRDs: An Overview and Future Challenges, Journal on Wireless Comm. and Networking (2019).

[81] European Telecommunications Standards Institute (ETSI), Electromagnetic Compatibility and Radio Spectrum Matters (ERM); Short Range Devices; Part 1, [Online] `bit.ly/3ESxIly` – Last accessed: 2022-11-21 (2012).

[82] B. Nahas, et al., Concurrent Transmissions for Multi-Hop Bluetooth 5, in: Proc. of the 16th EWSN Conf., 2019.

[83] C. Ramirez, et al., LongShoT: Long-Range Synchronization of Time, in: Proc. of the 18^{th} IPSN Conf., 2019.

[84] J. Heirbaut, JojoDiff - diff utility for binary files, [Online] `http://jojodiff.sourceforge.net/` – Last accessed: 2022-11-21.

[85] J. Jongboom, Jojo AlterNative Patch (JANPatch), [Online] `https://github.com/janjongboom/janpatch` – Last accessed: 2022-11-21.

[86] SparkFun, SparkFun Sound Detector, [Online] `bit.ly/3i3bBzU` – Last accessed: 2022-11-21.

[87] A. Dunkels, et al., Software-based On-line Energy Estimation for Sensor Nodes, in: Proc. of 4th EmNetS Workshop, 2007.

[88] FUOTA Working Group of the LoRa Alliance Technical Committee, FUOTA Process Summary Technical Recommendation TR002 v1.0.0, [Online] `bit.ly/3U0anV1` – Last accessed: 2022-11-21 (2019).

[89] UMBRELLA, The UMBRELLA Testbed, [Online] `bit.ly/3OIMhLO` – Last accessed: 2022-11-21 (2022).

[90] A. Sikora, et al., Test and Measurement of LPWAN and Cellular IoT Networks in a Unified Testbed, in: Proc. of the INDIN Conf., 2019.

[91] D. Lobba, et al., Concurrent Transmissions for Multi-hop Communication on UWB Radios, in: Proc. of the 17th EWSN Conf., 2020.

[92] X. Ma, et al., Harmony: Saving Concurrent Transmissions from Harsh RF Interference, in: Proc. of the 39th INFOCOM Conf., 2020.