ChirpBox: An Infrastructure-Less LoRa Testbed

Pei Tian[1,3], Xiaoyuan Ma[2], Carlo Alberto Boano[5], Ye Liu[4], Fengxu Yang[1,6], Xin Tian[1], Dan Li[1✉], and Jianming Wei[1]

1. Shanghai Advanced Research Institute, Chinese Academy of Sciences, China 2. SKF Group, China
3. University of Chinese Academy of Sciences, China 4. Nanjing Agricultural University, China
5. Institute of Technical Informatics, Graz University of Technology, Austria
6. School of Information Science and Technology, ShanghaiTech University, China

{tianpei2018, tianx, lid, wjm}@sari.ac.cn ma.xiaoyuan.mail@gmail.com
cboano@tugraz.at yeliu@njau.edu.cn yangfx@shanghaitech.edu.cn

Abstract

A key obstacle hindering the development of large-scale LoRa testbeds outdoors is the common lack of a backbone infrastructure allowing to easily communicate with the 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 proper testing.

In this paper, we present ChirpBox, an infrastructure-less 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 and the collection of log traces. We achieve this, among others, by developing 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 deploy a test installation to evaluate its performance experimentally and showcase its operations.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*wireless communication*; C.4 [**Computer Systems Organization**]: Performance of Systems—*measurement techniques*

General Terms

Measurement, Experimentation, Performance, Design

Keywords

LoRa Communications, Testbeds, Wireless

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 [1, 2]. LPWAN technologies, 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 [3]. This empowers the deployment of large-scale systems and the creation of attractive IoT applications such as precision agriculture [4], smart metering [5], water distribution management [6], air quality monitoring [7], and smart lighting [8].

Among existing LPWAN technologies, LoRa is currently the most widespread and well-known [9]. 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.

On the other hand, in contrast to other solutions leveraging licensed bands and/or requiring subscription (e.g., NB-IoT, LTE-M, 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 [10], 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 [11], as well as solutions exploiting the concurrent transmissions principle [12, 13]. Besides novel networking protocols, the community has also been active in researching better coding schemes for data recovery [14, 15], enhanced link quality estimation techniques [16], improved strategies for an optimal parametrization of physical layer settings [17], as well as several other aspects improving the reliability and overall performance of LoRa-based communications [18].

LoRa testbeds and their limits. In order to develop, debug, and optimize such solutions, as well as to benchmark the performance of LoRa-based protocols, *testbed facilities play a crucial role*. Indeed, testbeds provide developers with a controlled (but realistic) testing environment and with many tools facilitating experimentation, such as the automated

This work was done when Xiaoyuan Ma was at Shanghai Advanced Research Institute, CAS.

International Conference on Embedded Wireless Systems and Networks (EWSN) 2021
17–19 February, Delft, The Netherlands
ISBN: 978-0-9949886-5-2

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 [19, 20].

Several testbed installations support experimentation on LoRa nodes [21]: well-established IoT facilities such as FlockLab 2 [22] and FIT IoT-Lab [23], newer heterogeneous testing environments such as LinkLab [24] and NITOS [25], as well as facilities specifically built for testing LoRa-based systems [26, 27, 28] and for performing city-wide evaluations [29, 30]. 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 [23, 24]. Given that LoRa is meant to be used outdoors for communicating over long distances, such installations are not representative of typical deployments. 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 key reasons 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 beginning 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 [31, 32, 33], given their ubiquitousness and easiness 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 rural and 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 paper 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 [22, 31]) used to host (i.e., to power, program, stimulate, and profile) one or more target nodes [34]. 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 [35, 36]. 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. Additionally, we demonstrate ChirpBox in action by benchmarking the performance of several LoRa-based protocols. Finally, we make our software implementation *open-source*[1], in order to provide the community with a ready-made low-cost solution and foster experimentation.

This paper 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 then experimentally evaluate ChirpBox's performance in Sect. 4, and show exemplary uses of the testbed in Sect. 5. After summarizing related work in Sect. 6, we conclude this paper in Sect. 7, along with a discussion of future work.

2 ChirpBox: Overview

Fig. 1 shows ChirpBox's architecture, which allows deployments in outdoor areas where no infrastructure is available – neither to communicate with the target nodes, nor to

[1] https://github.com/sari-wesg/ChirpBox.git

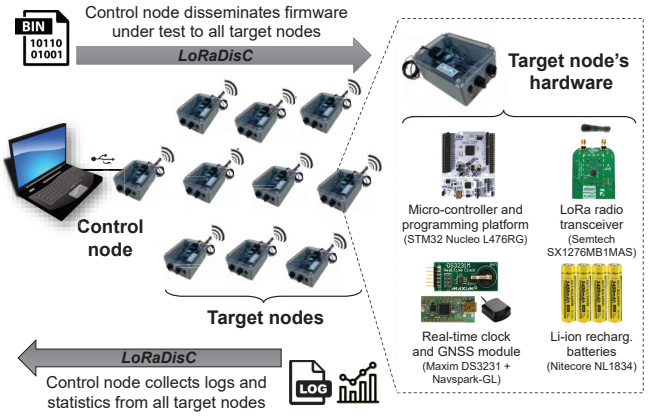


Figure 1. ChirpBox's architecture: the target nodes are used not only to run tests, but also to disseminate the FUT and a test run's settings, as well as to enable the collection of all logs after a test run has completed. All testbed nodes are homogeneous and based on off-the-shelf hardware.

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*.

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. ChirpBox also allows to binary patch the FUT, so to seamlessly change user-defined protocol parameters. 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 [37]. The test run's settings and user-defined protocol parameters are stored in a `json` file. A `python` script parses this file as well as the provided FUT, and instructs the LoRa node via 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.

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 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. To this end, 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.

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). ChirpBox switches between the two memory banks with the help of the daemon and RTC module, as detailed in Sect. 3.2.

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. ChirpBox also provides tools to update the daemon wirelessly, as well as additional API commands to timestamp the logs: we elaborate on these additional features in Sect. 3.4.

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 and

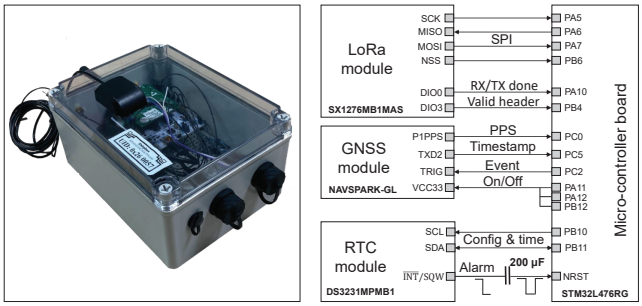


Figure 2. Simplified sketch of the interconnections between the main hardware components in a ChirpBox node.

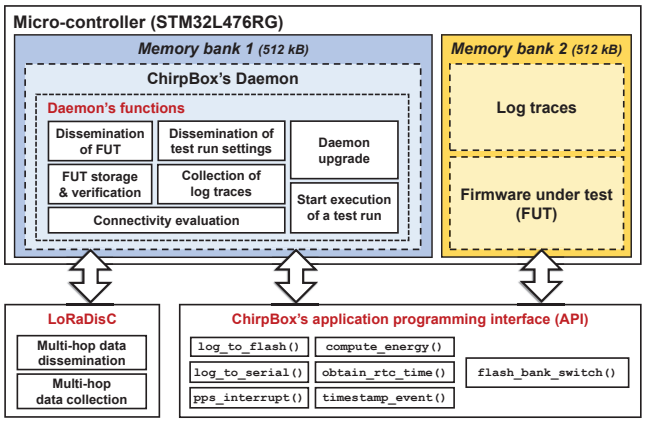


Figure 3. Each target node contains a daemon (coordinating its activities) in one memory bank and the FUT on a second bank. Many of the daemon's functions are built upon LoRaDisC's communication primitives. An API provides access to common functions and abstracts the underlying hardware, simplifying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

flash space allocation (Sect. 3.2). We further present the design of LoRaDisC and explain how it is used for dissemination and collection (Sect. 3.3), as well as shed light on ChirpBox's testbed management features (Sect. 3.4).

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 [38].

We further equip ChirpBox's nodes with a GNSS module and antenna (NavSpark-GL [39]) to provide all nodes with an absolute and common timescale, as well as with a real-time clock (RTC) module (Maxim DS3231MPMB1 [40]) 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 (Nitecore NL1834) 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 (`DIO0`) as well as when a valid packet header is received (`DIO3`). 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 is connected to the RTC module via I^2C and 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 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. The STM32L476RG 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.

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. 3.4; after this step is com-

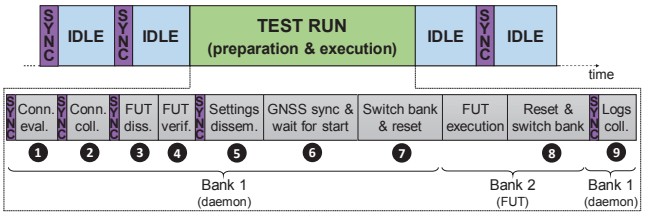


Figure 4. Operations of a target node in ChirpBox: nodes are typically idle and periodically look for `SYNC` messages from the control node. When a test run needs to be executed, the `SYNC` message triggers the execution of different activities.

pleted, 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 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[2]; 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 and synchronizes the target node's local clock to UTC (if necessary) by exploiting the PPS signal; it then configures the RTC module to trigger an alarm interrupt at the expected completion time of the test run, and remains in low-power mode 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[3] ❼: this way, the execution of the FUT from the second memory bank is triggered 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 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.

[2] 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.

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

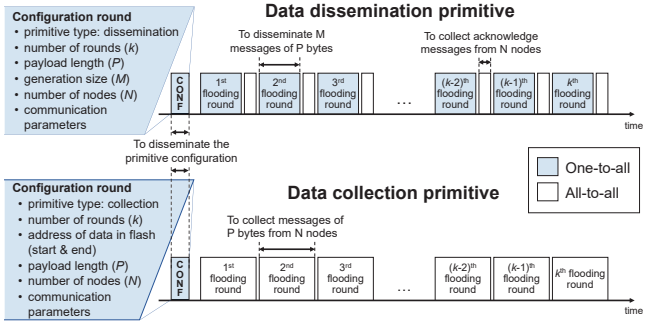


Figure 5. LoRaDisC supports 2 communication primitives: data dissemination and collection, consisting of an initial flooding round and a series of one-to-all or all-to-one rounds to disseminate, acknowledge, and collect information.

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, multi-hop communication without routing strategy, and high energy efficiency [41].

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. 5. 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 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. Each flooding round contains a series of CT slots, during which nodes blend packets using random linear network coding (RLNC) and transmit random linear combinations of previously received packets as in Mixer [42]. 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 disseminating 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.

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

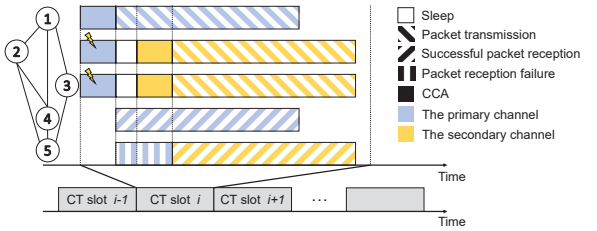


Figure 6. LBT & AFA mechanisms introduced in LoRaDisC.

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 or 36 s per hour on the selected channel. 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 [43]. 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 [44].

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. 6[4]. According to the spectrum access regulations [44, 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 [45]), 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. 6. 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 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.

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 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 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[5].

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 [41] and, quite recently, of BLE transceivers [46]. 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 [42]. 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 [47]) 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) [13]. 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 immediately turns off the radio to preserve its limited energy budget[6]: this is especially useful to filter the transmissions from co-located LPWANs.

3.4 Testbed Management Features

As discussed in Sect. 2, ChirpBox embeds a number of features providing the user with means to monitor the testbed's health status and connectivity, to enable a fine-grained time-stamping of events, as well as to upgrade the daemon's firmware and generate patch files.

[4] Our implementation of LoRaDisC is *generic* and allows, in principle, to disable the use of LBT and AFA, as shown in Sect. 4. However, this is not recommended, as this results in longer delays when operating the testbed.

[5] The European Telecomm. Standards Institute, for instance, imposes a maximum transmit time of 1 s [44]: 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.

[6] 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.

Monitoring the testbed status. 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. The on-board temperature can be linked to the communication performance observed during a test run, as recent studies have shown a strong correlation between the two [36]. All this information can be piggybacked to the data collected by the control node at the end of a connectivity evaluation phase (marked with ❷ in Fig. 4). The connectivity evaluation itself consists in a firmware instructing the target nodes to periodically transmit probe packets in a round-robin manner. By reusing the GNSS timestamp, we can make sure that only one target node acts as a transmitter at anytime, which allows to gather a picture about the connectivity across the various nodes in the testbed using different PHY settings, such as SF, transmission power, and frame length. 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.

Application programming interface. 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 them in this task, ChirpBox provides a low-level application programming interface (API) giving access to common functions and abstracting the operations of the underlying RTC and GNSS modules. For example, when calling the `log_to_flash()` function, a target node automatically writes in flash the requested information along with the time of the request. ChirpBox also allows to accurately timestamp `GPIO` events with a 6-byte Unix timestamp by calling the `timestamp_event()` function. Furthermore, it also foresees on-site inspections where every target node can be connected to a laptop, thereby redirecting logs to be output on the serial port (`log_to_serial()` function).

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* [48] 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* [49]. 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 Evaluation

We evaluate ChirpBox's performance experimentally. To this end, we set up a testbed of 21 nodes in a university campus over an area of 28 hectars, as shown in Fig. 7, and 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? What are the benefits introduced by the LBT and AFA mechanisms? (Sect. 4.1)
- 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. 4.2)

4.1 Performance of LoRaDisC

We start by evaluating the performance of LoRaDisC when collecting and disseminating firmwares throughout the network shown in Fig. 7. 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 also enable both LBT and AFA mechanisms. Each experiment is repeated 3 times. The control node logs the number of flooding rounds and records

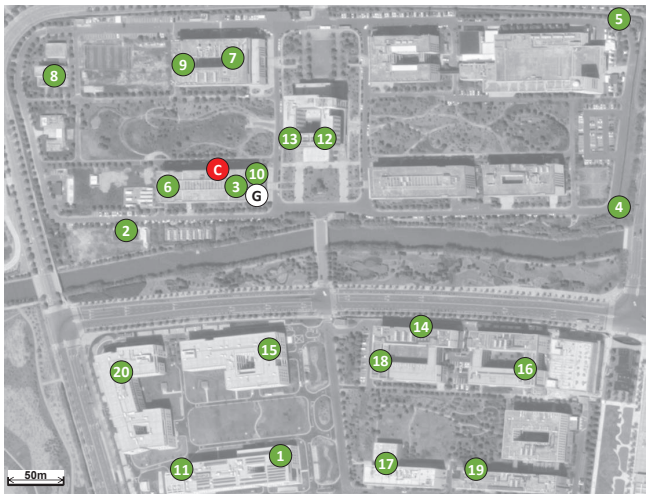
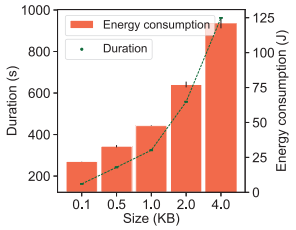
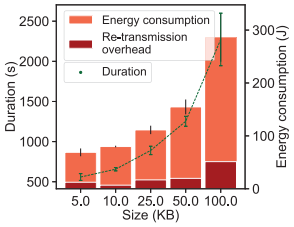


Figure 7. ChirpBox test installation in a University campus with twenty target nodes (green) and one control node (red). The white circle marks the location of a LoRaWAN gateway.

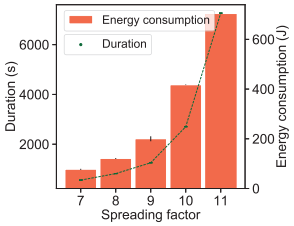


(a) Data collection

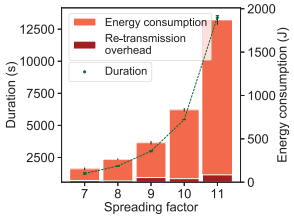


(b) Data dissemination

Figure 8. LoRaDisC's performance as a function of file size.



(a) Data collection



(b) Data dissemination

Figure 9. LoRaDisC's performance as a function of SF.

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

the time necessary to complete a data collection or dissemination using GNSS timestamps. All target nodes compute their energy consumption in software using Energest [50], following the methodology described in Sect. 4.2.

4.1.1 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).

Fig. 8 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 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. 7 to successfully communicate to all target nodes in a star topology, however, a transmission power of 14 dBm is necessary.

In our experiments, *all* firmwares and logs have been reliably exchanged. Note that the numbers shown in Fig. 8 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. 8b.

4.1.2 Performance as a function of the SF

Fig. 9 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 [44].

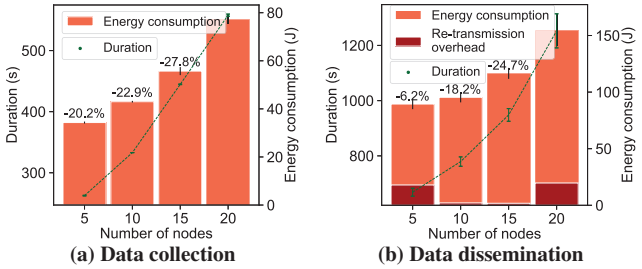
As expected, the higher the SF, the longer the duration of the data exchange and, correspondingly, the energy consumption – despite the fact that a lower network diameter can be achieved with a greater SF. Although the data rate achievable when using SF=10 (or 8) is two times slower than that of SF=11 (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 notice that the proportion of the re-transmission 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 [12].

4.1.3 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. 10 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. 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.

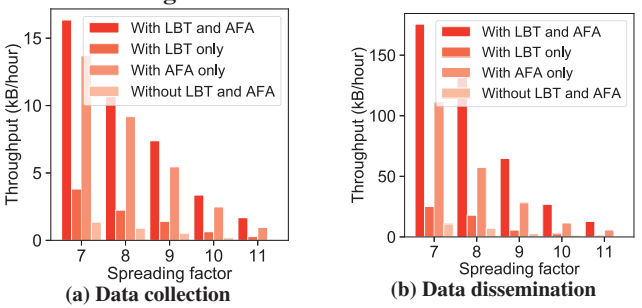
4.1.4 Benefits introduced by LBT and AFA

Fig. 11 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



(a) Data collection (b) Data dissemination

Figure 10: LoRaDisC's performance as a function of the number of target nodes in the network.



(a) Data collection (b) Data dissemination

Figure 11. LoRaDisC's throughput when enabling or disabling the LBT/AFA mechanisms for different SFs.

disabling AFA, LoRaDisC only makes use of one channel; when AFA is enabled, 10 channels are used. LoRaDisC ensures that all nodes comply with the regulations, i.e., a transmission time of at most 100 s and 36 s per hour per channel when LBT is enabled or disabled, respectively.

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 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.

4.2 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.

4.2.1 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 [50], 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

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

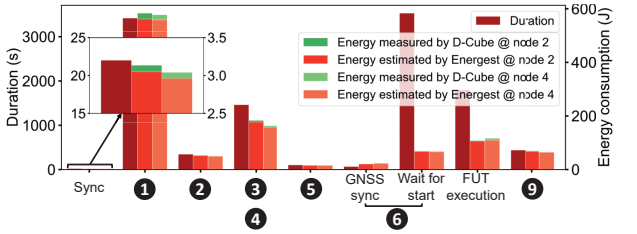


Figure 12. Breakdown of the energy consumed by a ChirpBox node when preparing, running, and finalizing an experiment. The different phases refer to those described in Fig. 4. The four bars compare the energy estimated using Energest with the one measured in hardware on two nodes.

observers [31] attached to two of the target nodes while the testbed 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[7] 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. 12 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 duration as well as energy consumption are negligible.

4.2.2 Breakdown of energy consumption per phase

Fig. 12 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: this is

[7] 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.

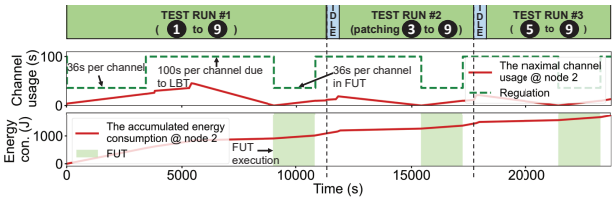


Figure 13. Energy consumption and channel usage of three consecutive test runs. The FUT executed in the second run is a minor modification of the previous one, whereas the third run is a repetition of the second one with different settings.

because a device needs to wait up to one hour in order 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. In the first run, the connectivity evaluation is carried out and the same firmware employed earlier is disseminated in its entirety (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 modifications compared to the one used in the first run. This results in ChirpBox disseminating only a 4.2 kB patch file in the second run, as discussed in Sect. 3.4, i.e., only phases ❸ to ❾ are executed. 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. 13 (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. 12. 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. 13 (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[8].

4.2.3 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 Nitecore NL 1834 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 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. 13, the nodes can execute about 343 runs (≈24 days). Of course, these results are specific to our implementation. In principle, 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.

5 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.

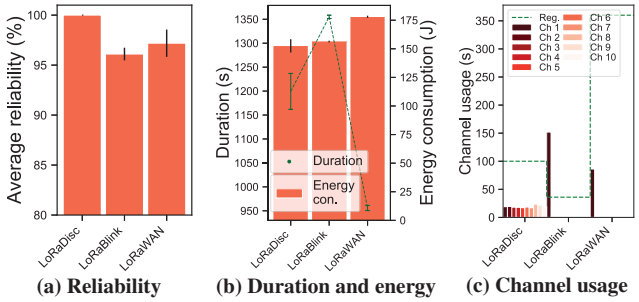
Benchmarking protocol performance. We compare the end-to-end latency, energy consumption, and channel usage of three LoRa-based protocols (LoRaDisC with LBT and AFA, LoRaBlink, and LoRaWAN) when disseminating a firmware image of 50 kB across the test installation shown in Fig. 7. We use LoRaDisC following 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 [12] 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[9]. 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. 7. In order to speed up the firmware dissemination when using LoRaWAN, all nodes follow the Class C specification (i.e., end devices can listen all the time except in transmit mode, resulting in low-latency communication). The gateway operates at a 10% duty cycle; all nodes are in the same multicast group and keep listening on a specific channel, where transmissions for up to 360 s per hour are allowed.

Fig. 14 shows the performance of the tree protocols over three different runs. LoRaDisC achieves 100% reliability (Fig. 14a) and exhibits the least energy consumption (Fig. 14b) while keeping the channel usage well below the one imposed by the regulations (Fig. 14c). 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[10] (results omitted due to space constraints). LoRaBlink exhibits the lowest reliability, the longest dissemination 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. 14c). Instead, if LoRaBlink

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

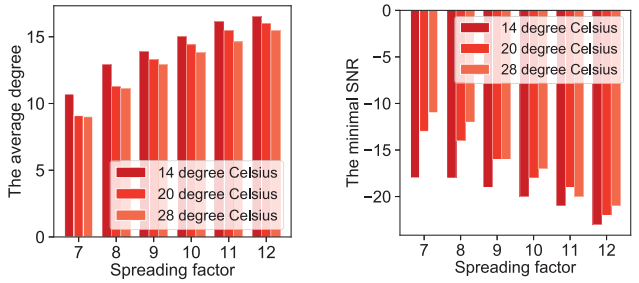
[9] Dec. 14, 2015 version, available at https://www.lancaster.ac.uk/scc/sites/lora/lorablinkkit.html

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



(a) Reliability **(b) Duration and energy** **(c) Channel usage**

Figure 14: Performance of LoRaDisC, LoRaBlink, and LoRaWAN when disseminating a 50 kB file across a network.



(a) Average number of neighbours **(b) Minimum SNR sustained under successful communications**

Figure 15. Impact of temperature on network connectivity.

would meet the channel usage regulations, the duration of its dissemination would increase by 6.6 times[11].

Impact of temperature on LoRa networks. Using the testbed health status information collected by ChirpBox as described in Sect. 3.4, 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. Fig. 15 shows the average number of neighbours in the testbed installation shown in Fig. 7, as well as the minimum SNR recorded during a successful communication in three different time instances (characterized by three different average temperatures). As we can see, higher temperatures negatively affect the communication between LoRa nodes, as experimentally shown by earlier works [36, 51]. Specifically, the average number of neighbours decreases by 1.4, and the SNR increases by up to 7 dBm at SF=7 when the temperature increases by 14 °C.

6 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 [8, 21, 24, 25, 26, 27, 28, 29, 30], 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 [22, 34] and FIT IoT-Lab [23], but only with a limited number of nodes that are mostly deployed indoors. All these facilities, in order to simplify the reprogramming and management of the target LoRa nodes, make use of an existing network backbone, typically based on Ethernet or cellular networks [35, 52]. As a network backbone is not always available and as the use of cellular networks incurs high operational costs, Kazdaridis et al. [25] have tried to only use 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 can support reprogramming of LoRa nodes and the efficient collection of traces. In a prior poster publication [53], we described the idea behind ChirpBox and how it would be of help to the community: this paper describes the concrete realization of this vision into a tangible solution, giving details on its design and implementation.

Concurrent transmissions on top of LoRa. CT have been widely popular in the context of low-power wireless systems since the development of Glossy [45]. 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 [41]. Recent studies have also shown the feasibility of CT on top of other technologies, such as Bluetooth Low Energy [46], ultra-wideband [54], and LoRa [12, 13]. Specifically, when it comes to LoRa, Bor et al. [12] 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. [13], 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 flooding rounds and network coding [42, 55].

7 Conclusions and Future Work

We have presented ChirpBox, an 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 re-using 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.

In the future, we plan to support additional LoRa platforms, to add energy harvesting modules (hence avoiding the

[11] 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).

need for battery replacement), and to enable the interruption of an ongoing test run if a channel usage violation is detected.

8 Acknowledgments

This work is partially funded by the National Science Foundation of China (No. 61902188) and the Strategic Priority Research Program of Chinese Academy of Sciences (No. XDC02070800). We would like to thank Sixuan Chen for her support regarding the water-proofed mechanical design.

9 References

[1] U. Raza *et al.*, "Low Power Wide Area Networks: An Overview," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, 2017.

[2] K. Mekki *et al.*, "A Comparative Study of LPWAN Technologies for Large-Scale IoT Deployment," *ICT Express*, vol. 5, no. 1, 2019.

[3] 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.

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

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

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

[7] P. J. Basford *et al.*, "LoRaWAN for Smart City IoT Deployments: A Long Term Evaluation," *Sensors*, vol. 20, no. 3, 2020.

[8] G. Pasolini *et al.*, "Smart City Pilot Projects Using LoRa and IEEE 802.15.4 Technologies," *Sensors*, vol. 18, no. 4, 2018.

[9] B. Foubert and N. Mitton, "Long-Range Wireless Radio Technologies: A Survey," *Future Internet*, vol. 12, no. 1, 2020.

[10] LoRa Alliance, "LoRaWAN 1.1 Specification, v1.1," 2017, [Online] https://bit.ly/2F0QbQM – Last accessed: 2020-10-16.

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

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

[13] C.-H. Liao *et al.*, "Multi-hop LoRa Networks Enabled by Concurrent Transmission," *IEEE Access*, vol. 5, 2017.

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

[15] M. Sandell and U. Raza, "Application Layer Coding for IoT: Benefits, Limitations, and Implementation Aspects," *IEEE Systems Journal*, vol. 13, no. 1, 2019.

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

[17] M. Bor and U. Roedig, "LoRa Transmission Parameter Selection," in *Proc. of the* 13th *DCOSS Conf.*, 2017.

[18] J. P. Shanmuga Sundaram *et al.*, "A Survey on LoRa Networking: Research Problems, Current Solutions, and Open Issues," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, 2020.

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

[20] 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.

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

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

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

[24] 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.

[25] 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.

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

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

[28] J. M. Marais *et al.*, "Evaluating the LoRaWAN Protocol using a Permanent Outdoor Testbed," *IEEE Sensors*, vol. 19, no. 12, 2019.

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

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

[31] 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.

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

[33] 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.

[34] R. Trüb *et al.*, "A Testbed for Long-Range LoRa Communication," in *Proc. of the* 18th *IPSN Conf., demo session*, 2019.

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

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

[37] 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.

[38] The Things Network, "STM32L476 Nucleo-64 + SX1276RF1IAS," [Online] https://bit.ly/2FhCvRX – Last accessed: 2020-10-16.

[39] NavSpark, "Arduino-compatible Dev. Board with GPS/GLONASS," [Online] https://bit.ly/3ipDVYE – Last accessed: 2020-10-16.

[40] Maxim Integrated, "DS3231-Extremely Accurate I2C-Integrated RTC/TCXO/Crystal," [Online] https://bit.ly/3hpkH5C – Last accessed: 2020-10-16.

[41] M. Zimmerling *et al.*, "Synchronous Transmissions in Low-Power Wireless: A Survey of Communication Protocols and Network Services," *CORR – arXiv preprint 2001.08557*, 2020.

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

[43] 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.

[44] "Electromagnetic compatibility and Radio spectrum Matters (ERM); Short Range Devices; Part 1," European Telecommunications Standards Institute (ETSI), Tech. Rep., 2012, ETSI EN 300 220-1 v2.4.1.

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

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

[47] C. G. Ramirez *et al.*, "LongShoT: Long-Range Synchronization of Time," in *Proc. of the* 18th *IPSN Conf.*, 2019.

[48] J. Heirbaut, "JojoDiff - diff utility for binary files," [Online] http://jojodiff.sourceforge.net/ – Last accessed: 2020-10-16.

[49] J. Jongboom, "Jojo AlterNative Patch (JANPatch)," [Online] https://github.com/janjongboom/janpatch – Last accessed: 2020-10-16.

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

[51] M. Cattani *et al.*, "An Experimental Evaluation of the Reliability of LoRa Long-Range Low-Power Wireless Communication," *JSAN*, vol. 6, no. 2, 2017.

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

[53] Authors and title masked due to double-blind submission, in *Blinded Conf., poster session*, 2020.

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

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