A Cloud RAN Architecture for LoRa

Christophe Delacourt, Patrick Savelli, and Vincent Savaux

Abstract – This article deals with a cloud radio access network (CRAN) architecture for the LoRa system. In the suggested design, the gateway embeds a limited remote radio head (RRH), including the analog RF part, the digital-to-analog and analog-to-digital conversion, and the digital front end (DFE). The other LoRa network functions, including the physical (PHY) layer, the LoRaWAN medium access control layer, and the application and customer servers, are implemented as cloud resources. This approach leads to a flexible RAN that is robust to the variations of capacity needs. Furthermore, it allows us to test very specific LoRa features, such as detection or demodulation, while bypassing others, including the hardware RRH. The methodology and tools we used to deploy a LoRa CRAN are detailed, and results of the performance indicator (CPU load, memory consumption) are provided.

1. Introduction

LoRa is a low-power wide-area network operated in unlicensed frequency bands for Internet of Things (IoT) applications. This cost-effective, long-range, energy-efficient technology is being actively deployed globally by network operators, for both public and private network coverage.

The typical architecture of a LoRa network relies on gateways connected through an Internet protocol-based backhaul interface to the centralized LoRa network servers. The gateway is the network equipment that exchanges data with the IoT devices through the air interface and relays messages between these end devices and the network server.

The gateway typically implements the RF and physical (PHY) layer functions, including the encoding of the transmit signal, the radio frame generation and up-conversion in the uplink direction, and the received-signal down-conversion, demodulation, and channel decoding in the downlink direction.

Note that the PHY layer is a proprietary solution designed by Semtech. Basics of the modulation (chirp spread spectrum), channel coding, PHY header, and payload construction are described in [1].

The gateway is connected to the network server (NS), which implements the LoRaWAN medium access control (MAC) layer [2], and to the application server (AS), which is involved in some of the LoRaWAN security procedures (e.g., management of join requests and encryption of application payload). The AS provides an interface to the end user to collect and send data to the devices, typically through a web application. The LoRaWAN protocol is an open specification maintained by the LoRa Alliance.

In this article, we describe how we have implemented our own cloud radio access network (CRAN) LoRa architecture, independent of the commercial solutions. It is very flexible and at the same time mainly based on low-cost off-the-shelf components. The suggested solution is described in detail and widely illustrated. This article is dedicated to any researchers who would like to deploy their own CRAN LoRa system.

The architecture proposed in this contribution consists in offloading some of the processing functions of the gateway to the cloud infrastructure.

In this approach, the network is deployed with limited hardware equipment installed on the field, to implement the remote radio head (RRH) functions with a software-defined radio approach. Basically, this hardware contains the antennas, the RF analog parts, the high-speed analog-to-digital and digital-to-analog converters, and field-programmable gate array (FPGA) reconfigurable hardware to perform functions such as rate adaptation, filtering, channel multiplexing (on the transmit side) and demultiplexing (on the receive side), and generation of the modulation signal. An overview of the possible RRH solutions is provided in [3].

The other gateway functions, such as the PHY layer baseband processing (demodulation and channel encoding and decoding) and the NS and AS functions, are implemented on cloud resources managed through orchestration services, using virtualization technology. This functional split between the centralized baseband functions, also named the baseband unit, and the RRH installed on the top towers close to the antenna is an architecture evolution referred to as CRAN.

This approach offers easier deployment and scaling capability through dynamic shared resource allocation, where the software network functions are instantiated on the fly in cloud infrastructures according to the network load. It allows for easier network upgrades, enhancements, testing, monitoring, and maintenance. In addition, the centralization of the PHY processing enables advanced signal processing techniques such as the joint demodulation of frames received by multiple gateways.

The rest of the article is organized as follows: Section 2 is dedicated to the description of our proposed architecture. Sections 3 and 4 present the methodology

Manuscript received 26 May 2020.

Christophe Delacourt, Patrick Savelli, and Vincent Savaux are with b<>com, 1219 avenue des Champs Blancs, 35510 Cesson-Sévigné, France

christophe.delacourt@b-com.com

patrick.savelli@b-com.com

vincent.savaux@b-com.com

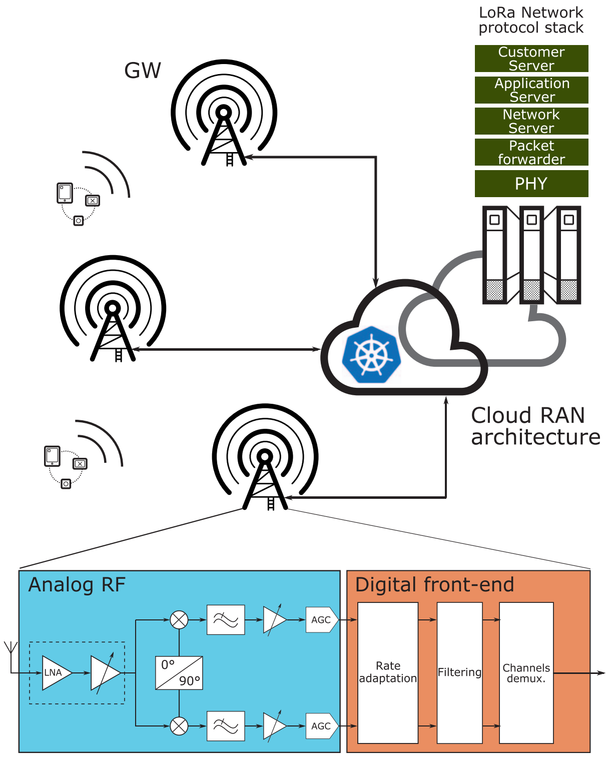


Figure 1. CRAN architecture: the gateways (GWs) only embed the analog RF and the DFE; the whole LoRa protocol stack is processed in external servers.

and the results, respectively, and Section 5 concludes the article.

2. CRAN Architecture

2.1 Overview

The CRAN IoT architecture considered in this article is depicted in Figure 1, where only the receiver blocks are illustrated (simpler blocks are used in the transmitter). It is composed of two main components: a hardware RRH located on the antenna towers, and the other functions of the architecture, which are offloaded to the cloud infrastructure. The hardware RRH is, in turn, is composed of analog and digital functions: the analog RF blocks of amplification, down-conversion, filtering, and analog-to-digital conversion in the receiver. A reconfigurable FPGA embeds the digital front-end (DFE) functions of rate adaptation [4], low-pass filtering [5], and channel demultiplexing. Note that the RRH also contains the transmitter functions.

As illustrated in Figure 1, the LoRa protocol stack is fully implemented in software in the cloud. This includes the PHY layer and packet forwarder, usually embedded in the antenna tower hardware in state-of-the-art solutions. The PHY layer contains the Rx digital processing chain: synchronization, symbol demodulation and demapping, and channel decoding. It also implements the channel encoding for the Tx path. The

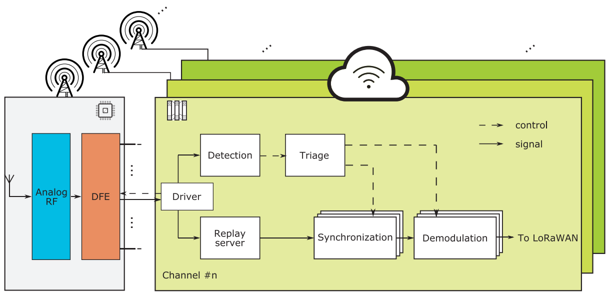


Figure 2. A possible implementation of the LoRa PHY layer hardware/software split.

processing is distributed in “dockerized” micro-services (https://www.docker.com/), interfaced through remote procedure call-based interfaces, which allows for flexibility and dynamic scaling according to the network processing load. The PHY layer is interfaced with the LoRa network server through the packet forwarder interface. The network server, application server, and customer server are based on open-source software. The software functions are orchestrated with Kubernetes (https://kubernetes.io/), which offers deployment automation facilities and efficiently handles the maintainability and scalability of the platform.

2.2 Detailed Functions

The PHY layer software architecture relies on a micro-service approach, where each different functional component of the Tx and Rx chains is a separated entity that provides a specific service with a dedicated interface. Each of the different micro-services runs on a separate docker container, to provide an abstraction that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as an isolated process in user space.

Docker containers are self-contained software environments for development that include all elements needed to run: code, run time, system tools, and libraries. The use of containers ensures portability between different platforms and clouds, as well as easier scaling of the application.

All communication between the services is made with gRPC (https://grpc.io/), a modern open-source high-performance remote procedure call framework based on HTTP2. This allows us to use standard cloud components like reverse proxies and load balancers. It should be noted that although we have made our own implementation of the LoRa receiver, we support all the features of the PHY layers.

The different micro-services that compose the suggested LoRa PHY software are summarized as follows, and illustrated in Figure 2:

1. Driver: The driver is in charge of configuring the RF and the FPGA components (DFE for the Rx path, modulator for Tx) and encapsu-

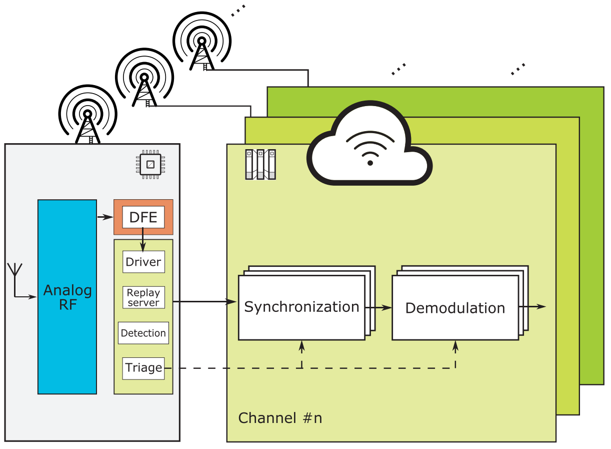


Figure 3. Alternative implementation of the LoRa PHY layer hardware/software split: driver, detection, triage, and replay server are deported to a low-cost ARM CPU-based card.

lates the IQ data stream from each channel in gRPC streams. In the basic configuration, we use eight channels with a sampling rate of 250,000 samples/s, for a total of 2,000,000 samples/s. Each sample consists of 32 bits (16 bits I and Q), and therefore the resulting network bandwidth is 8 MB/s. The currently used RF card is USB based, so the Driver service can be deported on the RRH depending on the setup.

2. Replay server: The replay server is intended to buffer the incoming IQ packet stream for retransmission when the synchronization or demodulation service requests it. Its main goal is to overcome the network latency introduced by the micro-services. In any case, the introduced latency is negligible (of order 10^{-2} s) compared to the duration of the LoRa transmission window.
3. Detection: This component implements the received LoRa frame detection for spreading factors (SFs) 7 to 12. The detection function is a continuous process, as required by the LoRa standard, and therefore it generates a constant CPU load per channel.
4. Triage: This service coordinates the whole reception chain. Triggered by a detection event, it manages synchronization and demodulation processes.
5. Synchronization: The synchronization service starts by requesting retransmission of the packets to the replay server, then processes the frame time and frequency synchronization.
6. Demodulation: The demodulation service is responsible for demodulating and decoding the LoRa PHY frame. This frame includes the PHY header and payload parts. The service starts by requesting retransmission of the packets to the replay server, applies synchronization corrections, and then decodes the header part to determine the parameters required for the payload decoding (channel coding rate [CR], payload length, presence or not of a cyclic redundancy check [CRC] in the payload). It then performs the channel decoding of the PHY payload part, prior transmitting the decoded frame to the MAC layer. We support all the features of the PHY layer, and thus any message encoded with any combination of SF and CR.

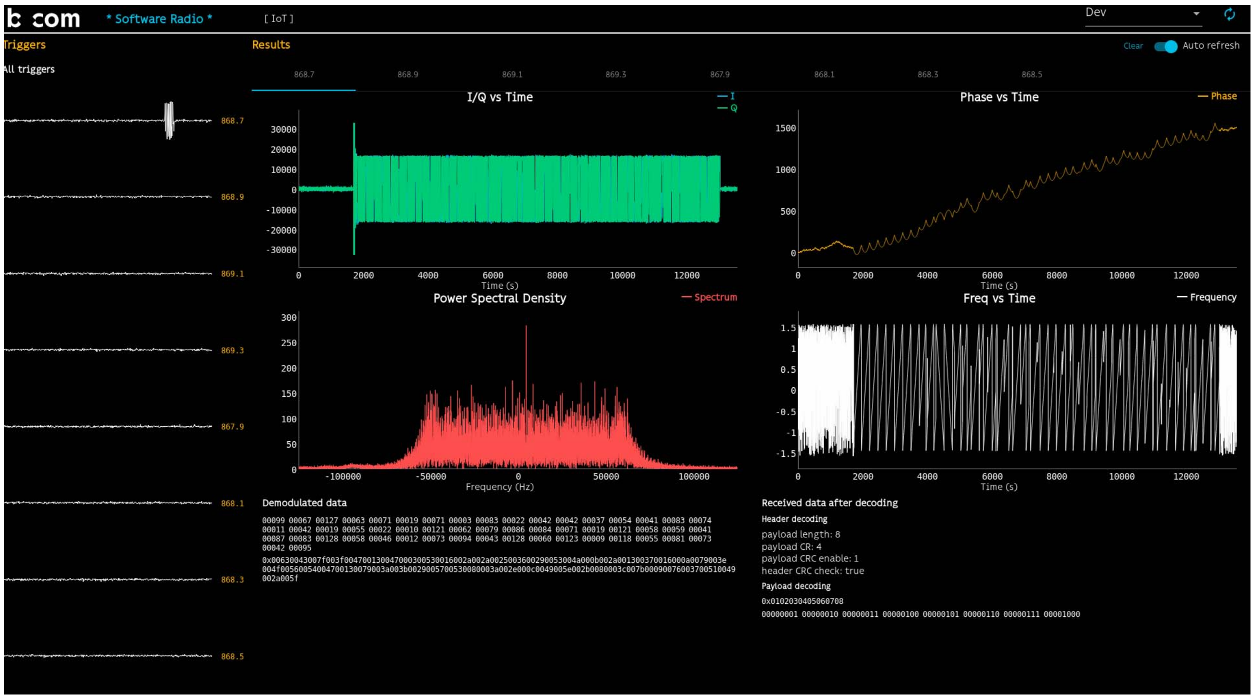
It must be specified that each LoRa gateway channel has exactly one replay server and one detection and triage service each. However, as depicted in Figure 2, synchronization and demodulation services are stateless and can be scaled/mutualized among multiple gateways, depending on the load of the system. This feature allows for a very flexible CRAN implementation.

In addition to the full-CRAN architecture, we suggest an alternative implementation of a LoRa PHY layer hardware/software split, where fewer blocks are deported to the cloud. This architecture, depicted in Figure 3, is especially relevant in use cases where the bandwidth between the RRH and the cloud services is limited. We keep the same hardware DFE, and propose an ARM-based cost-effective board embedding parts of the micro-services. In this configuration, only synchronization and demodulation are performed in the cloud, and thus the bandwidth between the gateway and the cloud is only required when a message is detected. The peak load in this case is 1 MB/s per channel when a frame is detected, which is the average load in the first functional split proposed in Figure 2.

3. Methodology and Tools

In order to ease development and tests of the micro-services, we developed a hardware simulator called LiveRF. Given an input scenario described in JSON format, LiveRF is able to generate, on the fly, the IQ data stream for each LoRa channel. This micro-service can replace the driver micro-service to perform receiver performance simulations, including signal generation and addition of noise and RF effects and impairments such as frequency error and drift, phase rotation, automatic gain control, and quadrature error. LiveRF is used to perform exhaustive message parameter testing, load tests, and complex tests such as frame collision, which is difficult to reproduce in real environment conditions. It is also used for continuous integration, as it allows functional non-regression testing, and sensitivity performance evaluation of the RX chain.

Moreover, the Monitor tool has been developed, a specialized monitoring micro-service which consolidates all the information and events provided by the Rx chain. Based on Monitor, we are able to develop a number of tools, most notably a dedicated PHY monitoring web application, shown in Figure 4. This



Demodulated data

```
00099 00067 00127 00063 00071 00019 00071 00003 00083 00022 00042 00042 00037 00054 00041 00083 00074
00011 00042 00019 00055 00022 00010 00121 00062 00079 00086 00084 00071 00019 00121 00058 00059 00041
00087 00083 00128 00058 00046 00012 00073 00094 00043 00128 00060 00123 00009 00118 00055 00081 00073
00042 00095

0x00630043007f003f0047001300470003005300160002a002a002500360029005300 4a000b002a001300370016000a0079003e
004f005600540047001300790003a003b0029005700530080003a002e000c0049005e002b0080003c007b0009007600370051 0049
002a005f
```

Received data after decoding

Header decoding

```
payload length: 8
payload CR: 4
payload CRC enable: 1
header CRC check: true
```

Payload decoding

```
0x0102030405060708
00000001 00000010 00000011 00000100 00000101 00000110 00000111 00001000
```

Figure 4. LoRa PHY monitoring web application.

application is a viewer of the I and Q signals, phase and frequency variations of the gateway received signal, and parameters like the received signal strength indication. It also displays the SF, the demodulated symbols, the decoded frames, and the CRC decoding results.

We have also developed a frame capture tool that has proven to be very useful for debugging purposes. This tool captures I and Q signals of frames that can be replayed using LiveRF for further analysis. This process can be triggered according to a set a filtering events. For example, it can capture only the frames that have been unsuccessfully decoded, for a given SF, or on a given frequency channel.

Finally, Monitor is also used within our automated gateway sensitivity measurement framework, used both in real conditions with a controlled LoRa RF generator and in simulation mode with LiveRF. In addition, during the whole development phase we performed regular tests with real LoRa devices and ran numerous sensitivity performance campaigns to validate the gateway and calibrate the simulator.

4. Results

Based on the proposed architecture and methodology, we have developed an efficient, robust, and perfectly interoperable platform with existing LoRa devices and LoRaWAN servers. The network architecture has been deployed and tested using commercial devices in a real environment. The inherent flexibility of the architecture enables quick prototyping of any kind of new processing algorithms and tools. Addition of new features and upgrades is also greatly simplified. Intensive automated testing significantly reduces the development cycle, thanks to continuous deployment and Kubernetes.

The major drawback of the approach is the network overhead mechanically introduced by the micro-services. Furthermore, compared to a monolithic implementation, a lot of memory optimization is prevented, leading to a slightly slower execution time. However, this limitation is largely mitigated by the excellent scaling capabilities of the architecture in a cluster environment. The frame detector has a fixed CPU cost and memory usage (continuous detection), whereas synchronization and demodulation service load depends on the number of detected messages. Those two services are stateless and can be scaled or mutualized depending on the network load. Furthermore, there is still a lot of room for fine-level software optimization.

We performed preliminary load tests on a Kubernetes cluster composed of five virtual machines with four cores and 8 GB RAM, for a total of 20 cores and 40 GB RAM. The underlying hardware is an Intel Xeon Gold 6138 CPU at 2.00 GHz. Results are given in terms of CPU load (100% corresponds to one full loaded core) and memory used (in MB). They are measured after 10 min of a constant load of 6 LoRa frames/s, 14 frames/s, and 24 frames/s, as depicted in Table 1. It must be noticed that 24 frames/s greatly overestimates a typical LoRa device deployment. We conclude that the proposed architecture is able to support practical use cases on small clusters.

Table 1. Preliminary load test results

Frames/s	CPU load (%)	Memory used (MB)
6	321	733
14	442	741
24	593	1263

5. Conclusion and Future Work

In this article we have presented a new CRAN architecture for the LoRa system. This architecture is flexible and has been further derived into two configurations: a full-RAN solution and a cost-effective ARM-based board to limit the required bandwidth between the RRH and the cloud.

For the LoRa standard, we plan to work on joint demodulation of messages between multiple gateways to further improve reception performance. Finally, our goal is to extend this flexible and reconfigurable CRAN solution to support multiple IoT protocols.

6. References

1. A. Augustin, J. Yi, T. Clausen, and W. M. Townsley, "A Study of LoRa: Long Range & Low Power Networks for the Internet of Things," *Sensors*, **16**, 9, September 2016, p. 1466.
2. LoRa Alliance Technical Committee, "LoRaWAN® specification v1.1," https://lora-alliance.org/resource-hub/lorawanr-specification-v11 (Accessed 4 November 2020).
3. Patrick Savelli, Vincent Savaux, Pauline Desnos, Ali Zeineddine, Matthieu Kanj, and Christophe Delacourt "Flexible Multi-Standard Digital Front-End for LPWA Technologies", URSI Radio Science Letters, vol. 2, 2020, pp. 1 - 5
4. A. Zeineddine, S. Paquelet, A. Nafkha, P.-Y. Jezequel, and C. Moy, "Efficient Arbitrary Sample Rate Conversion for Multi-Standard Digital Front-Ends," 2019 17th International IEEE NEW Circuits and Systems Conference (NEWCAS), Munich, Germany, June 2019, pp. 1 - 4.
5. S. Paquelet and V. Savaux, "On the Symmetry of FIR Filter With Linear Phase," *Elsevier Digital Signal Processing*, **81**, 10, October 2018, pp. 57-60.