A Cloud RAN Architecture for LoRa

Christophe Delacourt [1], Patrick Savelli [1], and Vincent Savaux [2]

[1]Affiliation not available
[2]b¡¿com

October 30, 2023

Abstract

This paper 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 radio-frequency (RF) analog part, the digital-to-analog and analog-to-digital conversion, and a digital front-end (DFE). The other LoRa network functions, including the physical (PHY) layer, the LoRaWAN medium access control (MAC) layer, and the application and customer servers are implemented as cloud resources. The presented 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 the detection or demodulation, while bypassing the other ones including the hardware RRH. The methodology and tools we used to deploy a LoRa cloud RAN are detailed, and results concerning the performance indicator (CPU load, memory consumption) are provided as well.

URSI GASS 2020, Rome, Italy, 29 August - 5 September 2020

A Cloud RAN Architecture for LoRa

Christophe Delacourt(1), Patrick Savelli*(1), and Vincent Savaux(1)
(1) b<>com, Rennes, France

Abstract

This paper 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 radio-frequency (RF) analog part, the digital-to-analog and analog-to-digital conversion, and a digital front-end (DFE). The other LoRa network functions, including the physical (PHY) layer, the LoRaWAN medium access control (MAC) layer, and the application and customer servers are implemented as cloud resources. The presented 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 the detection or demodulation, while bypassing the other ones including the hardware RRH. The methodology and tools we used to deploy a LoRa cloud RAN are detailed, and results concerning the performance indicator (CPU load, memory consumption) are provided as well.

1 Introduction

LoRa is a low power wide area network (LPWAN) operated in unlicensed frequency bands for Internet of things (IoT) applications. This cost effective, long range and energy efficient technology is being actively deployed globally by network operators, both for public or private networks coverage. The typical architecture of a LoRa network relies on gateways connected through an Internet protocal (IP) 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 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), that implements the LoRaWAN®MAC layer [2], and to the application server (AS) that is involved in some of the LoRaWAN security procedures (e.g. management of join requests, 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®.
The architecture proposed in this contribution consists in offloading some of the processing functions of the gateway in 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 (SDR) approach. Basically, this hardware contains the antennas, the RF analog parts, the high speed analog-to-digital and digital-to-analog converters, and a field programmable gate array (FPGA) reconfigurable hardware to perform functions such as rate adaptation, filtering, channels multiplexing (in the transmit side) and de-multiplexing (in the receive side), and the 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), as well as 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 (BBU), and the RRH installed on the top towers close to the antenna is an architecture evolution referred to as C-RAN. This approach offers easier deployment and scaling capability through dynamic shared resources 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 additionn the centralization of the PHY processing natively enables advanced signal processing techniques such as the joint demodulation of frames received by multiple gateways

The rest of the paper is organized as follows: Section 2 is dedicated to the description of our proposed architecture. Sections 3 and 4 present the methodology and the results, respectively, and Section 5 concludes this paper.

2 Cloud RAN Architecture

2.1 Overview

The cloud-RAN IoT architecture considered in this paper is depicted in Fig. 1, where the receiver blocks only are il-

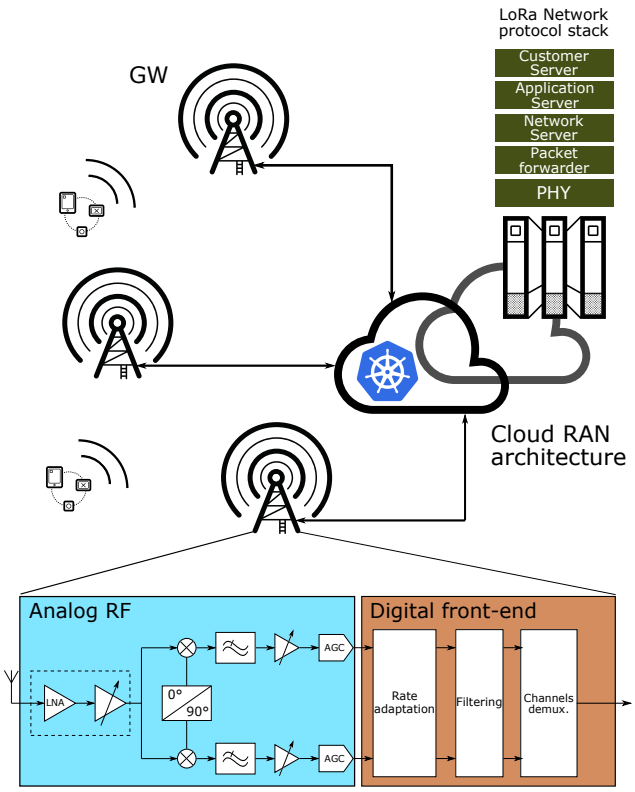

Figure 1. Cloud RAN Architecture: the gateways (GWs) only embed the analog RF and the DFE, the whole LoRa protocol stack is processed in external servers.

lustrated (simpler blocks are used in the transmitter). It is composed of two main components: hardware RRH located on the antenna towers, and the other functions of the architecture that are offloaded in the cloud infrastructure. The hardware RRH is, in turn, is composed of analog and digital functions: the analog radio-frequency (RF) blocks of amplification, down conversion, filtering, and analog-to-digital conversion (ADC) in the receiver. A reconfigurable FPGA embeds the DFE functions of rate adaptation [4], low-pass filtering [5], and the channel demultiplexing. Note that the RRH also contains the transmitter functions.

As illustrated in Fig. 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 de-mapping, channel decoding. It also implements the channel encoding for the Tx path. The processing is distributed in "dockerized" micro-services[1], 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[2], 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 run 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 isolated processes in user space. Docker containers are self contained software environments for development that include all elements needed to run: code, runtime, system tools and libraries. The use of containers ensures portability between different platforms and clouds, and enable easier scaling of the application.
The different micro services that compose the suggested LoRa PHY software are summarized as follows, and illustrated illustrated in Fig. 2:

- **Driver**: the driver is in charge of configuring the RF and the FPGA components (DFE for the Rx path, and modulator for Tx), and encapsulates IQ data stream from each channel in network packets.
- **Replay server**: The replay server is intended to buffer the incoming IQ packets stream for retransmission when the synchronization or demodulation service requests it. Its main goal is to overcome the network latency introduced by the microservices. In any case the introduced latency is negligible (of order 10^{-2}s) compared to the duration of the LoRa transmission window.
- **Detection**: this component implements the received LoRa frame detection for all spreading factors. The detection function is a continuous process as required by LoRa standard and therefore it generates a constant CPU load per channel.
- **Triage**: this service coordinates all the reception chain. Triggered by a detection event, it manages synchronization and demodulation processes.
- **Synchronization**: the synchronization service starts by requesting retransmission of the packets to the replay server, then it processes the frame time and frequency synchronization.
- **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

[1] https://www.docker.com/

[2] https://kubernetes.io/

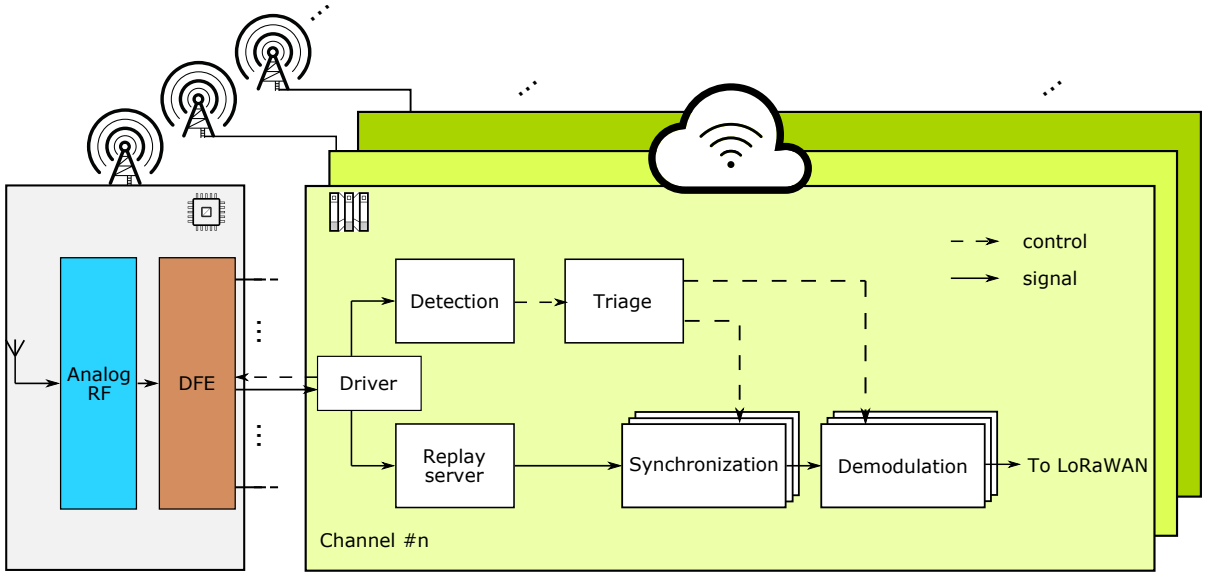

Figure 2. A possible implementation of LoRa PHY Layer hardware/software split.

part to determine the parameters required for the payload decoding (channel coding rate, 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 to the decoded frame transmission to the MAC layer.

It must be precised that each LoRa gateway channel has exactly one replay server, detection and triage services. However, as depicted in Fig. 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 previous full-CRAN architecture, we also suggest an alternative implementation of LoRa PHY layer hardware/software split, where less blocks are deported in the cloud. This architecture, depicted in Fig. 3, is especially relevant in use cases where the bandwidth between the RRH and the cloud services is limited. Thus, we keep the same hardware DFE, and we 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 1MB/s per channel when a frame is detected, which is the average load in the first functional plit proposed on the Fig. 2.

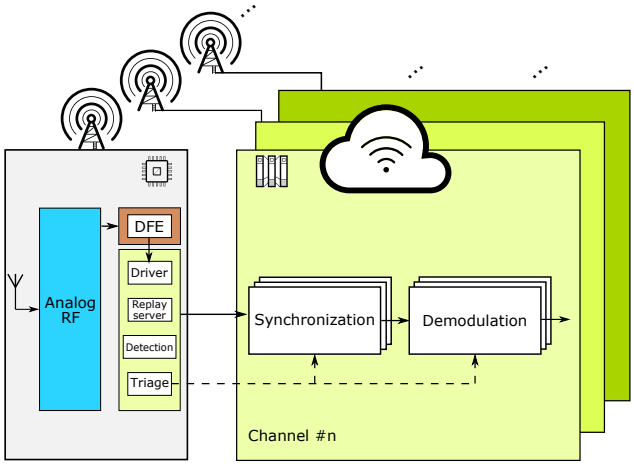

Figure 3. Alternative implementation of LoRa PHY Layer hardware/software split: driver, detection, triage, and replay server are deported in a low-cost ARM CPU based card.

3 Methodology & Tools

In order to ease development and tests of the microservices, 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 microservice can replace the driver microservice in order to perform receiver performance simulations, including signal generation, addition of noise and RF effects and impairments such as frequency error and drift, phase rotation, AGC, quadrature error, etc. *LiveRF* is used to perform exhaustive message parameters testing, load tests, and complex test such as frames collision, which is difficult to reproduce in real enviromnent conditions. It is also used for continuous integration, as it allows to perform functional non regression testing, and sensitivity performance evaluation of the RX chain.

Moreover, the tool *Monitor* has been developed, a specialized monitoring microservice 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 showed in figure 4. This application is a viewer of the I and Q

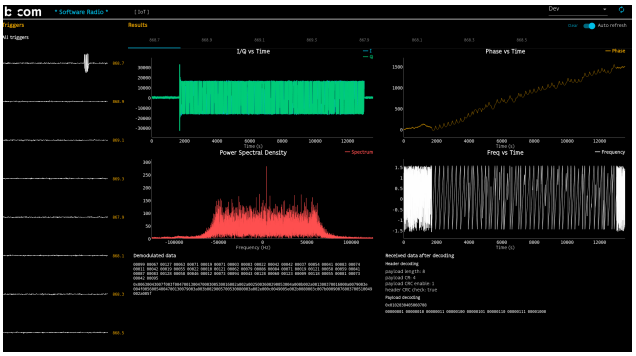

Figure 4. LoRa PHY monitoring Web Application

signals, phase and frequency variations of the gateway received signal, as well as parameters like the received signal strength indication (RSSI). It also displays the spreading factor (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, one may 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 GW sensitivity measurement framework, used both in real condition with a controlled LoRa RF generator, or in simulation mode with *LiveRF*. In addition, during the whole development phase, we have performed regular tests with real LoRa devices and run 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. 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 are 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 microservices. Futhermore, comparatively to a monolithic implementation, a lot of memory optimization are 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 services 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 optimizations.

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

Table 1. Preliminary load test results

Anyway, we have performed preliminary load tests on a Kubernetes cluster composed of 5 virtual machines with 4 cores and 8GB Ram, for a total of 20 cores and 40GB RAM. The underlaying hardware is an Intel Xeon Gold 6138 CPU @ 2.00GHz. Results are given in terms of CPU load (100% corresponds to 1 full loaded core) and used memory in MB. They are measured after 10 minutes of a constant load of 6, 14 and 24 LoRa frame/s as depicted in Table 1. It must be noticed that 24 frames/s greatly overestimates a usual LoRa devices deployment. We conclude that the proposed architecture is able to support practical use cases on small size clusters.

5 Conclusion & Future work

In this paper we have presented a new cloud RAN architecture for the LoRa system. This architecture is flexible and has been 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 have planned 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.

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*, September 2016.

[2] "Lorawan™specification," LoRa Alliance Technical Committee, October 2017, ver. 1.1.

[3] P. Desnos, A. Zeineddine, V. Savaux, P. Savelli, M. Kanj, and C. Delacourt, "Flexible multi-standard digital front-end for lpwa technologies," in *General Assembly and Scientific Symposium (GASS) of the International Union of Radio Science (Union Radio Scientifique Internationale-URSI)*, August 2020.

[4] A. Zeineddine, S. Paquelet, A. Nafkha, P.-Y. Jezequel, and C. Moy, "Efficient arbitrary sample rate conversion for multi-standard digital front-ends," in *2019 17th International IEEE NEW Circuits and Systems Conference (NEWCAS)*, jun 2019.

[5] S. Paquelet and V. Savaux, "On the symmetry of FIR filter with linear phase," *Elsevier Digital Signal Processing*, vol. 81, no. 10, pp. 57 – 60, October 2018.