Cloud-LoRa: Enabling Cloud Radio Access LoRa Networks Using Reinforcement Learning Based Bandwidth-Adaptive Compression

Muhammad Osama Shahid, Daniel Koch, Jayaram Raghuram, and Bhuvana Krishnaswamy, *University of Wisconsin-Madison;* Krishna Chintalapudi, *Microsoft Research;* Suman Banerjee, *University of Wisconsin-Madison*

https://www.usenix.org/conference/nsdi24/presentation/shahid

This paper is included in the Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation.

April 16–18, 2024 • Santa Clara, CA, USA

978-1-939133-39-7

Open access to the Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation is sponsored by

Cloud-LoRa: Enabling Cloud Radio Access LoRa Networks Using Reinforcement Learning Based Bandwidth-Adaptive Compression

Muhammad Osama Shahid[*1] Daniel Koch[*1] Jayaram Raghuram[1]
Bhuvana Krishnaswamy[1] Krishna Chintalapudi[2] Suman Banerjee[1]
[1]*University of Wisconsin-Madison* [2]*Microsoft Research* [*]Co-Primary authors

Abstract

The Cloud Radio Access Network (CRAN) architecture has been proposed as a way of addressing the network throughput and scalability challenges of large-scale LoRa networks. CRANs can improve network throughput by coherently combining signals, and scale to multiple channels by implementing the receivers in the cloud. However, in remote LoRa deployments, a CRAN's demand for high-backhaul bandwidths can be challenging to meet. Therefore, bandwidth-aware compression of LoRa samples is needed to reap the benefits of CRANs. We introduce *Cloud-LoRa*, the first practical CRAN for LoRa, that can detect sub-noise LoRa signals and perform bandwidth-adaptive compression. To the best of our knowledge, this is the first demonstration of CRAN for LoRa operating in real-time. We deploy Cloud-LoRa in an agricultural field over multiple days with USRP as the gateway. A cellular backhaul hotspot is then used to stream the compressed samples to a Microsoft Azure server. We demonstrate SNR gains of over 6 dB using joint multi-gateway decoding and over 2x throughput improvement using state-of-the-art receivers, enabled by CRAN in real-world deployments.

1 Introduction

LoRa [1] is one of the most popular long-range, low-power wide area network (LPWAN) technology for IoT applications such as smart city [2, 3], smart agriculture [4, 5]. Operating in the license-free ISM band, anyone can independently deploy a LoRa LPWAN, where IoT devices make use of off-the-shelf LoRa radios to transmit messages to a gateway. Like most common wireless systems, a LoRa gateway performs all physical layer processing such as receiving and decoding transmissions. Often, gateways relay these decoded packets to the cloud to enable cloud-based IoT applications.

The last decade has seen the emergence of a new wireless architecture – a Cloud Radio Access Network (CRAN), where the gateway continuously streams the raw received digitized radio signals (I/Q samples) to a virtual gateway in the cloud over a back-haul link for physical-layer processing (Fig. 1). CRANs, applied to LoRa, offer three disruptive advantages.

- *Joint Multi-Gateway Packet Decoding:* Economic viability of a LoRa deployment is often dictated by its range – a long range necessitates fewer base-stations thereby reducing capital and operating costs. Weak signals from multiple gateways, when combined constructively, boost the signal-to-noise ratio (SNR) [6, 7], and in turn, extend the range. These approaches require centralization to jointly process the raw radio signals from multiple base-stations.
- *Rapid Physical-Layer Innovation to Boost Capacity:* A major challenge in dense areas is capacity scaling, as mushrooming uncoordinated LoRa deployments lead to increased collisions [8]. Recently, several promising PHY-layer demodulation techniques [9–16] have shown an order of magnitude improvement in capacity. CRAN enables their rapid deployment and A/B testing in the field, as virtual software receivers can be deployed in the cloud.
- *Elastic Scaling to Multiple Channels:* As capacity needs increase, traditional LoRa gateways need to be physically upgraded to high-end gateways with parallel receiver chains baked into their ASIC. CRAN gateways capture a wide spectrum and allow for the potential to dynamically scale the number of channels in the cloud depending on demand.

While researchers have demonstrated the potential of CRANs to enable the deployment of new physical layer techniques [6, 17–19], to the best of our knowledge, there is no end-to-end implementation of a LoRa CRAN till date where gateways continuously stream radio samples to virtual receivers in the cloud to be decoded in real-time. *The key contribution of this paper is* **Cloud-LoRa***, the first end-to-end practically-deployable LoRa CRAN for urban and rural deployments.* We demonstrate that Cloud-LoRa allows for the practical deployment of real-time joint multi-gateway packet decoding in the cloud with techniques such as Charm [6] and achieves **an SNR gain of over 6 dB**. This gain could translate to a doubling of the range. We also show that rapid deployment of novel decoding techniques such as [9, 10, 12] offer a **2X improvement in throughput** as well as scaling to multiple

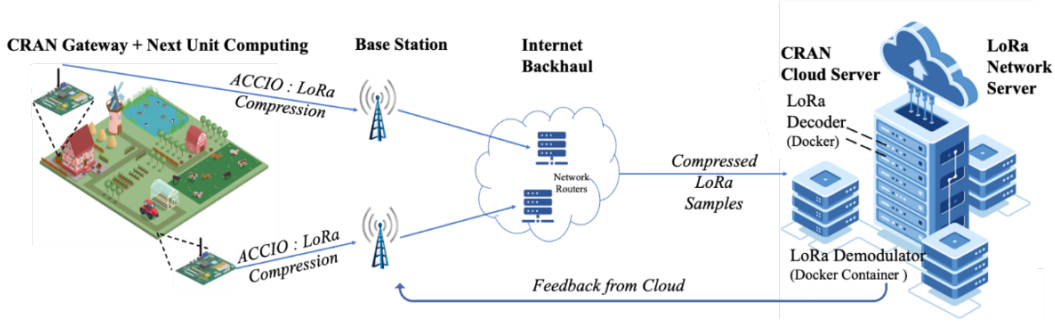


Fig. 1: An Illustration of CRAN and Cloud-LoRa

channels using per-channel virtual receivers in the cloud.

Cloud-LoRa comprises three key components: i) a *CRAN gateway* that can be deployed on software-defined radios (e.g., USRP and a NUC), ii) *ACCIO*, an online reinforcement learning-based adaptive compression algorithm, and iii) a cloud gateway with user-defined receivers. *Cloud-LoRa allows researchers to deploy their own physical layer demodulators* as containers at the cloud gateway (Figure 1).

Extreme Back-Haul Bandwidth Gap in a LoRa CRAN A common challenge to every CRAN is the need to stream a high volume of raw signals (I/Q samples) to the cloud. Each 1 MHz of radio spectrum generates a continuous data stream at 64 Mbps[1] to the cloud. However, LoRa LPWANs are intended for low-cost deployments and often, the only backhaul available in several rural and remote deployments is a cellular link offering as low as 500 kbps [20–22]. Therefore, a viable LoRa CRAN must be capable of streaming samples even on such low-bandwidth links to the cloud.

Detecting Sub-Noise LoRa signals buried in noise. At such low backhaul bandwidths, channel activity detection to avoid streaming noise signals to a cloud server is an integral component of a CRAN gateway. A majority of the existing activity detection methods rely on observing signal strength above a fixed threshold [23]. Due to its spread-spectrum technology, LoRa signals are received at sub-zero dB SNR, *i.e.*, they are buried in noise and hence signal strength-based thresholding fails. Even recent works, such as SparSDR [24] that uses time-frequency analysis for activity detection cannot distinguish sub-noise LoRa from noise in real-time. SparSDR can detect sub-noise signals at the cost of significantly higher false positives, a tradeoff that defeats the purpose of activity detection. In this work, we develop an activity detection approach that detects sub-noise LoRa signals in real-time across multiple channels with relatively low false positives.

Adaptive Lossy Compression and Streaming. A key component for any CRAN is compressing real-time streaming of radio samples to the cloud. CRANs typically employ lossless compression techniques since lossy compression degrades the quality of the signal and adversely affects its decodability. Further, it is hard to predict in advance at the gateway how much "lossiness" will allow a specific part of the signal to be decoded, since decodability of the signal depends on several dynamic factors such as SNR of the received wireless signal, the specific demodulation being used in the cloud etc.

As we evaluate in Section 5, state-of-the-art lossless compression techniques provide up to 70% compression for LoRa signals, depending on the SNR. At this compression, a single 500 kHz wide LoRa channel with 10% channel activity will generate a 960 kbps stream – about twice greater than the capacity of a rural cellular backhaul link of 500 kbps.

To address this challenge, we propose *ACCIO*, an online reinforcement learning (RL)-based wavelet compression that is lossy and adaptive, to compress sub-noise LoRa transmissions. *ACCIO* receives rewards (feedback) from a cloud receiver for decoding successfully, and learns to employ the right level of compression based on the available backhaul bandwidth and SNR at the gateway. *ACCIO* uses TCP BBR [25] to reliably deliver the lossy-compressed radio signals to the cloud. The bandwidth estimates from BBR, signal SNR estimates, and application buffer levels are then used by ACCIO's RL agent to dynamically set the appropriate level of compression. The motivation behind using RL for adaptive compression is discussed in detail in Section 3.2.

Open Source, Deployable Implementation. Our implementation of the Cloud-LoRa gateway performs channelization, activity detection, and runs *ACCIO* in real-time. Its cloud gateway runs on Azure and *implements several recent LoRa demodulators* including CIC [12] and Charm [6]. To demonstrate its practical viability, we deploy and test Cloud-LoRa in an 8-channel CRAN in two scenarios: **1)** a rural outdoor setting with cellular backhaul, and **2)** an urban outdoor setting. We have open-sourced our framework[2] with well-defined APIs to plug-in new physical-layer demodulators, and allow scalability with the number of channels. We hope that Cloud-LoRa will encourage and enable future researchers to deploy and compare novel physical-layer demodulators. The major contributions of this paper are:

- We present Cloud-LoRa, the first practically-deployable end-to-end LoRa CRAN solution. Our current implementation streams and processes signals from up to 8 LoRa channels in real-time to the cloud. We hope that researchers will be

[1] Two 32-bits for each complex-valued sample at 1 Msamples/s

[2] https://github.com/UW-CONNECT/cloud-lora.git

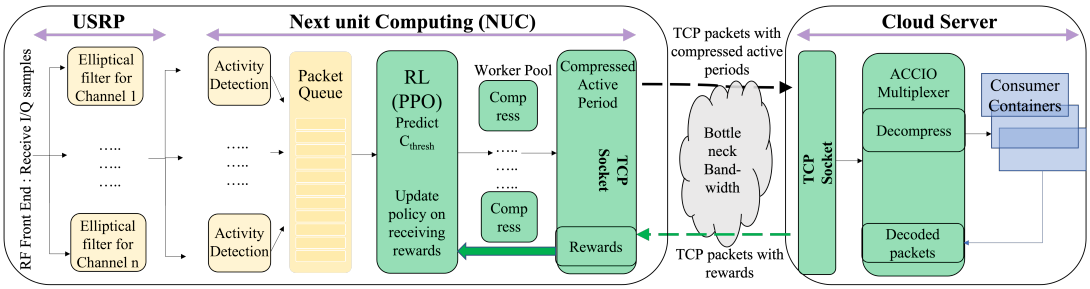


Fig. 2: Components of Cloud-LoRa : CRAN gateway (USRP) performs channelization, followed by activity detection at the NUC. ACCIO then compresses active periods using the RL agent and streams to the cloud server. The cloud server decodes the received packets and provides reward feedback to the ACCIO RL agent.

able to use Cloud-LoRa to deploy, test, and compare new physical layer demodulators in the field.

- We propose a novel LoRa activity detection approach that detects even sub-noise LoRa signals across multiple channels in real-time, without demodulating them.
- We propose ACCIO, a reinforcement learning-based adaptive compression technique that aims to maximize packet decodability in the cloud CRAN receiver. ACCIO also compresses the active LoRa signals to meet the available backhaul bandwidth and desired latency requirements.
- We provide an open-sourced implementation of Cloud-LoRa including CRAN gateway (on USRP), ACCIO, and several LoRa receivers as Dockerized containers.
- We demonstrate Cloud-LoRa through rural field deployments and by testing other recently published techniques [6, 12]. We show an SNR gain of over 6dB when signals from multiple gateways were jointly decoded in the cloud. We see a 2X boost in network throughput using novel physical-layer innovations deployed in the cloud. We process up to 8 LoRa channels in real-time, further improving throughput.

We attest that this work complies with the applicable ethical standards of our home institution.

2 Background and Motivation

LoRa Modulation-DeModulation: LoRa uses Chirp Spread Spectrum (CSS) modulation, a spread-spectrum technology that enables LoRa to operate at sub-zero dB SNR due to its resilience to noise and interference. The spreading factor (SF), which defines the number of bits per symbol, along with the RF bandwidth of a LoRa channel (BW) determine the symbol duration, datarate, energy consumption, and range of communication. As the SF increases, the range will also increase, but at the cost of reduced datarate. As SF and BW are predetermined for a transmitter, a LoRa receiver searches for preambles of a single SF and BW to detect the start of a packet. LoRa demodulates by dechirping the received signal, which enables it to receive sub-noise signals [26].

Variable Backhaul Bandwidths: While access to broadband connectivity has been expanding, available bandwidths still vary vastly across the country [22, 27]. FCC 2020 reports on broadband access determined that potentially over 50% of rural Americans lack broadband access [20, 28] of 25 Mbps download/3 Mbps upload speeds. Broadband speeds lower than 1 Mbps have been identified as a bottleneck for the adoption of precision agriculture [21, 29, 30]. Additionally, significant variability in data-rates can be expected over wireless links even in urban areas due to changes in load, environment, service providers, among other factors [31, 32].

Recent works on CRAN-based LoRa: The benefits of CRAN-based LoRa have been identified in recent works such as Charm [6], Nephalai [19], and OPR [7]. Charm and OPR demonstrate that the range of communication can be improved by coherently combining signals using CRANs. Nephalai proposes a static compression technique using compressed sensing [33] to stream multiple LoRa channels to the cloud to improve the network throughput. More generally, physical-layer-agnostic CRAN for IoT has been proposed in works such as SparSDR [24] and CharIoT [34]. While these existing works have identified and demonstrated the benefits of CRAN, dynamic compression that adapts to the signal characteristics and meets the variable backhaul bandwidths of LoRa gateways remains an open challenge. We propose an RL-based adaptive compression to address this challenge.

3 Proposed System - Cloud LoRa

Towards a practical, real-time LoRa CRAN, our Cloud-LoRa framework consists of three components, illustrated in Fig. 2:

1. CRAN gateway : a software defined radio (SDR) gateway that continuously streams samples from a wideband spectrum. The gateway performs channelization to filter LoRa channels and detects activity in each individual channel. The activity detection module at the gateway is designed to detect even sub-noise LoRa signals, and stream only those signals corresponding to active LoRa transmissions.
2. ACCIO (green blocks in Figure 2) : the active LoRa transmissions need further compression. We propose ACCIO, an online RL-based compression algorithm that adaptively

predicts the compression threshold for each active period. ACCIO's goal is to maximize the total packets decoded in the cloud gateway, while meeting the backhaul-bandwidth and latency constraints.

3. Cloud Server : we implement standard LoRa as well as user-defined LoRa receivers in a Microsoft Azure cloud server as Docker containers. The cloud server reconstructs the compressed samples, which are then demodulated and decoded. The number of packets decoded per active period is sent as reward feedback to ACCIO's RL agent.

3.1 CRAN Gateway

LoRa transmitters typically have a low duty-cycle to conserve their battery. As a result, a majority of the samples captured at a CRAN gateway are noise. Since it is wasteful to transport noise to the cloud, activity detection is critical in CRAN.

Multi-Channel Filter. The gateway performs channelization to filter an individual channel from the wideband spectrum before detecting activity. We first convert each channel to baseband and then apply a 4th-order IIR Elliptical filter (Figure. 2) [35]. This light-weight filter both suppresses other channels by 100 dB and offers a small transition band, ensuring minimal cross-channel leakage and real-time operation.

Sub-Noise LoRa Activity Detection. Activity detection is typically performed using energy-based approaches such as carrier sensing, which fail to distinguish low-SNR LoRa signals from noise [24]. At received SNRs below $0\,dB$, the energy of LoRa samples becomes comparable to that of noise.

A standard LoRa receiver performs dechirping followed by Fast Fourier Transform (FFT) to accumulate energy in a single frequency, in-turn distinguishing noise from LoRa samples [26, 36]. However, dechirping is specific to a spreading factor (SF). Current multi-channel LoRa gateways have a dedicated RF front-end for each SF. A naive sub-noise LoRa activity detection is to dechirp the received samples with each possible SF (7 through 12), and then perform energy-based detection. This is computationally intensive and requires 6× more multiplications than a single demodulator. Therefore, an SF-agnostic activity detection is desirable for LoRa CRAN.

We propose an SF-agnostic LoRa activity detection algorithm to detect sub-noise LoRa signals at the CRAN gateway. Our activity detection leverages two properties of LoRa: **1)**. Two LoRa signals of different SFs are Pseudo-orthogonal (more in Appendix.A.1). *e.g.*, dechirping an SF7 signal with SF8 downchirp would result in pseudo-random noise. **2)** For a given bandwidth, the downchirp of one SF is a time-scaled function of the downchirp of another SF. Based on these properties, we design ***superDC****, a custom downchirp, that can dechirp and hence detect the activity of more than one SF by superimposing downchirps of multiple SFs.*

The CRAN gateway continuously dechirps an array of samples with the *superDC*, followed by an FFT. *An active LoRa transmission results in a sharp peak above the noise floor in the FFT, triggering activity detection at the gateway.* For instance, a superDC that superimposes SF7, SF8, and SF9 downchirps detects an activity *only if* the active signals are in SF7 through SF9. Since an SF9 downchirp is 4× as long as that of SF7, and 2× as that of SF8, we construct the superDC by superposing one SF9 downchirp with two consecutive SF8 and four consecutive SF7 downchirps (more in Appendix.A).

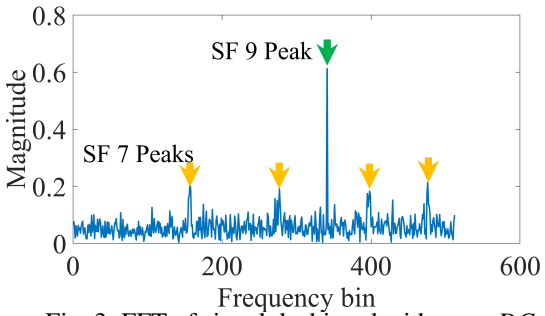


Fig. 3: FFT of signal dechirped with *superDC*

Figure 3 shows the FFT of a signal containing SF7 and SF9 chirps, each with 10 dB SNR, dechirped with this *superDC*. We observe four SF7 peaks and one SF9 peak since the superDC includes four SF7 and one SF9 downchirps.

The active LoRa signals that can be detected by the *superDC* are determined by the superposed downchirps, which in turn determine the length of the superDC. A superDC to detect all SFs (7 to 12) must accommodate at least one SF12, two SF11, four SF10, eight SF9, sixteen SF8, and thirty-two SF7 downchirps. In this case, the FFT peak-gain (ratio of the maximum peak in an FFT window to its noise floor [3]) of an SF7 symbol is 32 times lesser than that of an SF12 symbol. Hence, low-SNR SF7 symbols could go undetected. On the other hand, using narrower windows would lead to missing higher SF symbols. To combat this challenge, we design two superDCs: $superDC_{low}$ to detect symbols with SF 7 through 9 and $superDC_{high}$ to detect symbols with SF 10 through 12. The former can be defined in time domain as

$$superDC_{low}(t) = \sum_{m=0}^{3} C(t - mT_{SF7}, 7) + \sum_{m=0}^{1} C(t - mT_{SF8}, 8) + C(t, 9), \quad 0 \le t \le T_{SF9}, \quad (1)$$

where $T_{SF} = \frac{2^{SF}}{BW}$ and $C(t,i)$ is the downchirp of SFi. We can similarly define $superDC_{high}$. By choosing two groups of SF, we minimize the impact of excessive window sizes, while still maintaining SF-agnostic detection. The received samples are dechirped using both $superDC_{low}$ and $superDC_{high}$ to detect activity. As we demonstrate in our evaluation, the two groups of superDC signals can detect all LoRa activity in real-time.

As the SNR of the received samples decreases, the peak-gain of the received signal dechirped with a superDC signal

[3] We maintain a running estimate of the noise floor to confer resilience against temporal variations.

also decreases. This could result in spurious samples triggering activity detection. To reduce such false positives, the gateway signals activity in the channel whenever a minimum of 3 consecutive peak-gains, which correspond to 12 symbols of the smallest SF (*i.e.*, SF7 or SF10), are observed to be higher than a threshold (average peak gain for noise signal). We push such an active period's I/Q samples to the *Packet Queue* for compression and transport to the cloud server.

In summary, dechirping received samples using our custom-designed superDCs ($\text{superDC}_{\text{low}}$ and $\text{superDC}_{\text{high}}$) provides the processing gain needed to detect LoRa activity even when the received signals are much below zero dB SNR. The proposed activity detection is agnostic to the SF of the transmitter, making it a general-purpose front-end, with only $2\times$ multiplications of a single LoRa demodulator, as opposed to the state-of-the-art gateways that incur $6\times$ multiplications.

3.2 ACCIO : Reinforcement Learning-based Adaptive Compression

While activity detection reduces the volume of noise samples streamed, when the network traffic increases, even active period samples can be too high for some backhaul bandwidths to support. Even with perfect activity detection, the required bandwidths for 64 channels with $\approx 10\%$ channel occupancy is over 200 Mbps. Nephalai [19], a recent work on LoRa CRAN, utilizes downsampling and compressive sensing for compression. But, novel demodulators leverage oversampling to resolve packet collisions [9, 11–13], and hence compression without downsampling is necessary. Moreover, LoRa's chirp spread-spectrum (CSS) modulation renders dictionary-based lossless compression methods ineffective. We propose ACCIO , a light-weight RL-based adaptive compression algorithm that works on top of a Discrete Wavelet Transform (DWT)-based lossy compression scheme in order to maximize the number of packets decoded at the cloud server, without exceeding the backhaul bandwidth and latency constraints.

Lossy Active-Period Compression. We propose to utilize the Discrete Wavelet Transform (DWT) [37] as our lossy compression scheme for oversampled active LoRa signals. DWT, being a multi-resolution time-frequency analysis, is a suitable compression tool for CSS modulation which uses both time and frequency for modulation. Each DWT coefficient represents the energy of the received signal corresponding to a particular frequency (level) and time (shift). Also, DWT's linear complexity ($O(N)$, where N is the length of the signal) makes it light-weight, allowing it to compress in real-time. More background on DWT is presented in Appendix B.

We compress active LoRa signals by first applying DWT to the signals, and then retaining only those DWT coefficients with magnitude greater than a threshold C_{thresh}. The compressed signals can be reconstructed if sufficient energy is retained in the DWT coefficients. As we increase C_{thresh}, we retain fewer coefficients and compress more; but the energy of the signal (coefficients) retained will decrease, leading to lossy compression. Determining the optimum threshold that ensures reliable reconstruction of signals at the receiver (cloud server), while maintaining a compression to match the network bandwidth is a challenging problem.

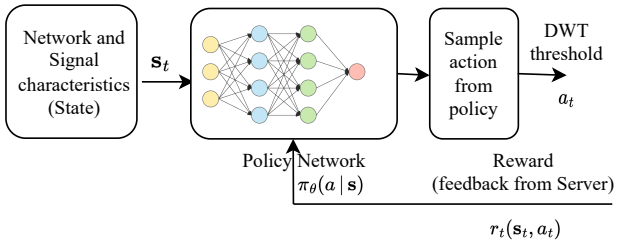
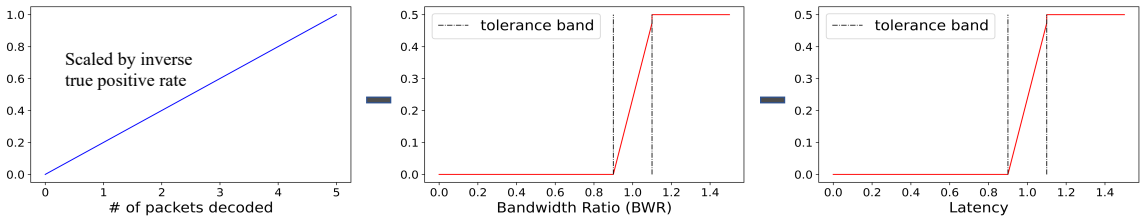


Fig. 4: Overview of the RL algorithm of ACCIO

Bandwidth-Adaptive Compression. The optimal compression threshold of DWT coefficients has a non-linear dependence on three factors: i) the SNR of samples at the gateway, ii) the backhaul network conditions, and iii) the LoRa demodulator at the cloud. Current compression approaches are static [19, 24] and do not adapt to these factors. Therefore, an adaptive compression that can learn this dependence is required. While DWT provides an effective way to compress based on the time-frequency characteristics of the signals, we still need a method to determine the appropriate amount of compression based on the backhaul network conditions (which are usually non-stationary). To address this, we propose an RL-based (DWT) threshold prediction algorithm that adaptively selects the level of compression for any given active period, with the goal of maximizing the cumulative number of packets decoded at the cloud server.

Choice of RL.

While supervised learning methods (particularly DNNs) are effective at modeling non-linear dependencies between the input and the target, they are designed for an offline scenario where the data distribution is not changing over time [38]. This makes them less effective when the network traffic and conditions (e.g., duty cycle, SNR, and the number of channels) are non-stationary, and also when there is lack of visibility into demodulator design used at the cloud receiver. Reinforcement learning is particularly well suited for this scenario since by design it learns an agent or policy in an unknown, dynamic environment such that the agent can perform a sequence of actions with the goal of maximizing its cumulative reward feedback [39]. In our setting, the agent performs the task of adaptively selecting the DWT threshold for each active period (based on various signal and network characteristics), with the goal of maximizing the total number of packets decoded at the cloud receiver over a transmission interval. Moreover, our choice of an online policy gradient-based RL algorithm does *not require pretraining* on a large collection of offline data from the target (or a similar) environment. It can start learning the compression policy from scratch based on data



(a) Reward for packet decoding (b) Penalty for bandwidth ratio (c) Penalty for latency

Fig. 5: Components of the reward function : (a) $u_{dec}(\mathbf{s},a)$, (b) $u_{band}(\mathbf{s},a)$, (c) $u_{lat}(\mathbf{s},a)$.

from the target environment, and still learn a good stable policy (see § 5.6). Hence, it can be applied to a wide variety of applications and deployments.

An RL agent at the gateway learns to take sequential actions based on the current state of the environment such that its cumulative-discounted rewards received from the environment over multiple time-episodes is maximized (more background in Appen. D). Crucial to the success of an RL agent are the design of the state variables and the reward function. Based on a careful study and evaluation, *we propose suitable state variables and a reward function that enable the agent to adaptively predict the (DWT) threshold in order to achieve high decodability under varying signal and backhaul conditions*. A unique challenge to the design of the reward function in this setting is the lack of perfect ground truth for providing the reward signal. In a typical RL system, the environment would have ground truth for providing the reward. This occurs because some of the transmitted active periods could be false positives (without actual packets), and it is unknown to the server whether a received active period is a false positive or not. We address this in the design of our reward function.

We focus on the Policy Gradient class of RL methods [40, 41], whose goal is to directly learn an optimal policy function that is parameterized by a neural network. Specifically, we use the Proximal Policy Optimization (PPO) method with clipped objective [42] for online training of the RL agent. PPO is widely adopted as a state-of-the-art online policy-gradient method due to its better computational and sample efficiency, and stable policy function updates. We next discuss the components of our RL algorithm.

Action. The action of the RL agent a corresponds to selecting the DWT threshold C_{thresh}. In principle, the threshold is continuous-valued. However, we simplify the design by choosing a discrete set of eight threshold levels. Specifically, if $a \in \{0,1,\cdots,7\}$ is the action taken, then $C_{thresh} = 5\,d_{avg}\,a$, where d_{avg} is the average DWT coefficient value over the current active period. (The factor 5 is chosen to cover a wider range of thresholds.). Our action space is discrete and the policy function $\pi_{\boldsymbol{\theta}}(a\,|\,\mathbf{s})$ will be a conditional probability mass function that sums to 1 over all the actions.

States. We provide the RL agent with a state vector $\mathbf{s}$ that broadly consists of the network (pipeline) characteristics and the signal characteristics from the recent active periods. The state variables based on the network are functions of the estimated bandwidth and the current fraction of the packet buffer that is filled. The state variables based on the signal characteristics include the normalized packet length, the ratio of signal-to-noise magnitude in the time domain, and a histogram of the peak-gain values of the active period. The full list of states with a description is given in Table 1.

State variable	Description
norm_pkt_len	(AP size in samples) / (sampling rate); AP - Active period
mag_time	(AP magnitude in the time domain) / (noise magnitude)
dcmp_avg	Average value of the first-level decomposition DWT coefficients
PG_hist	A 4-bin histogram of the peak gain values
BW_obs	log(BW_btl / 50,000,000); BW_btl - estimated bottleneck bandwidth
BW_ratio	(#bits sent to cloud in the last 10s) / $\int BW\_btl$ over the last 10s
BW_ratio_5	(#bits sent to cloud over last 5 AP) / $\int BW\_btl$ over last 5 AP
BW_ratio_10	(#bits sent to cloud over last 10 AP) / $\int BW\_btl$ over last 5 AP
buffer_size	Fraction of the packet queue filled with AP

Table 1: State variables used by ACCIO . The first four states capture LoRa characteristics and the rest capture the network. $\int BW\_btl$ is computed as a Riemann sum over time period.

Reward Function. A well-designed reward function is a crucial part of the RL design. As discussed earlier, the goal of ACCIO is to compress the LoRa signals in the active periods such that it maximizes the number of packets correctly decoded, while also meeting the bandwidth and latency constraints. We design the reward function as a sum of four terms: **i)** a positive reward term $u_{dec}(\mathbf{s},a)$ that is a weighted count of the number of packets decoded correctly (Fig. 5 a); **ii)** a negative penalty term $u_{band}(\mathbf{s},a)$ that strongly discourages the bandwidth utilized from getting very close to the available bandwidth (Fig. 5 b); **iii)** a negative penalty term $u_{lat}(\mathbf{s},a)$ that strongly discourages the overall latency (from client-side processing and network delays) from getting very close to a preset limit (*e.g.*, 2 seconds) (Fig. 5 c); and **iv)** a strong negative penalty $u_{over}(\mathbf{s},a)$ (equal to -10) that prevents the packet queue from filling up close to its limit (dictated by the hardware). The last penalty term $u_{over}(\mathbf{s},a)$ is applied preemptively at the client side whenever an action of the RL agent could potentially lead to a transmission that will cause buffer overflow and/or exceed the acceptable transmission time. The overall reward function is given by,

$$r(\mathbf{s},a) \;=\; u_{dec}(\mathbf{s},a) - u_{band}(\mathbf{s},a) - u_{lat}(\mathbf{s},a) - u_{over}(\mathbf{s},a),$$

The reward terms are discussed formally in Appen. C. Details of our PPO implementation is given in Sec. 4, and more background on the PPO algorithm is given in Appen. D.

False Positives & Reward Feedback. The cloud server does

not know the ground truth about LoRa packets, *i.e.*, a transmitted active period could just be noise (false positive). Moreover, in a low-duty-cycle network, the cloud receiver may decode only a small number of packets relative to the total number of active periods. Therefore, the RL agent could learn to compress more since there is a higher chance of incurring penalties from overshooting BW and/or latency limits, while the positive rewards for decoding the occasional packets are small. This could drastically increase the overall learning time necessary for the RL to reach an optimal policy. To address this, in the reward term $u_{dec}(\mathbf{s}, a)$, we weight the number of packets decoded by the *inverse of the true positive rate*, which is estimated as the fraction of LoRa packets decoded correctly over the last 100 detected active periods.

3.3 CRAN Cloud Server

The active LoRa signals compressed using ACCIO are streamed to the cloud server using a reliable TCP connection. Each compressed active period is packetized with metadata such as gateway ID, time-stamp at the gateway, length of the active period, sampling rate, channel number, number of DWT levels, among others. The cloud server in our architecture receives the packet, reads the metadata, and performs inverse DWT to reconstruct the signal. It is then input to user-defined LoRa receivers, implemented as Docker containers in the cloud. We separate the LoRa demodulator and decoder so that a custom LoRa demodulator can be deployed by simply updating the demodulator, while retaining the rest of the cloud implementation. The number of decoded packets is sent back to the CRAN gateway as a reward (component) to ACCIO , which then uses the reward to update its RL policy.

Two key objectives of our cloud-server design are i) scalability and ii) ease of deployment of user-defined LoRa receivers. To address scalability, we deploy parallel Docker containers per consumer. As the network scales, the cloud server increases the number of consumers to keep up. To facilitate user-defined LoRa receivers, we include a multiplexer that receives compressed signals and routes them to consumer containers based on their metadata. The link between the multiplexer and the consumers is simply a set of sockets, where each consumer listens on a unique port. The consumer is unaware of the compression and reconstruction. A configuration file maintains the global mappings of the (base-station, channel) pairs to ports. Note that users can map multiple base stations to the same ports to easily apply coherent combining such as Charm [6] atop our implementation. Each Cloud-LoRa packet contains time-stamps for coarse time synchronization between the base stations.

4 Implementation

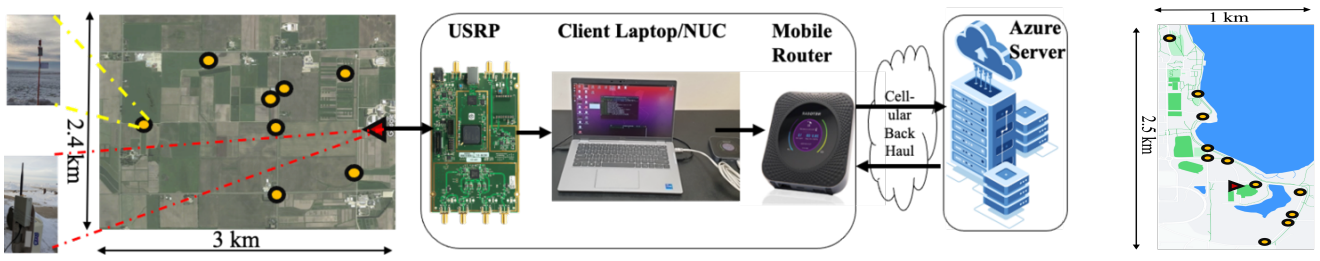
We describe in detail our end-to-end implementation of the three components of Cloud-LoRa : 1) SDR as CRAN gateway 2) ACCIO , the RL-based compression, 3) the cloud server.

CRAN Gateway - Activity Detection. We use a USRP B200 [43] as the CRAN gateway to capture a 2 MHz spectrum that includes 8 LoRa channels (125 kHz bandwidth and 75 kHz guard band, as per LoRaWAN specs). Channelization is performed using eight parallel 4th-order elliptical low-pass filters. Each filtered channel is input to the activity detection module implemented using Python. We use $superDC_{low}$ and $superDC_{high}$ to detect active periods of SF7 to SF9 and SF10 to SF12 respectively. We advance the superDC windows every 1/3-rd of the lengths of respective window samples, to ensure alignment with higher SFs.

CRAN Gateway - ACCIO The adaptive compression of ACCIO is implemented on a client laptop. On detecting active periods in each channel, the corresponding I/Q samples are pushed to the *Application Packet Queue*. The RL agent pops the oldest active period and extracts the state variables from the current network and the active period's coefficients. Table 1 lists the state variables used by the RL agent.

The RL agent was trained using the PPO algorithm [42], whose implementation is provided in the Keras and TensorFlow libraries [44]. Both the policy function and the value (or advantage) function in our PPO-based agent are realized using a fully-connected neural network with two hidden layers of sizes 48 and 32 respectively. This small network enables lightweight training and action determination, while maintaining sufficient complexity for complex approximations. The output layer of the policy network uses the Softmax activation to return probabilities over the set of actions. We set the discount factor of the cumulative rewards to $\gamma = 0.9$. We use the variant of PPO with a clipped objective, and set the clip ratio ε to 0.2. Optimization is based on the stochastic gradient descent method Adam [45], whose learning rate for the policy network and value-estimation network are set respectively to 0.00025 and 0.009. The episode length was defined as 50 active periods, *i.e.*, the agent performs online training, and after every 50 active periods, rewards are returned from the server. To maintain strict reward ordering, each active period (once popped from the application packet queue) is given an ID counter. The active period statistics are then cached and re-ordered after the decoding information is returned. As our implementation is run online, this re-ordering and training process runs in a background daemon.

The RL agent chooses an action that determines the compression threshold C_{thresh}, and DWT coefficients with magnitude $< C_{thresh}$ are set to zero. We further compress the DWT coefficients using Lz4 (preferred over Gzip due to its faster compression). The compressed coefficients are packetized with metadata and sent through TCP to the cloud server. Packet metadata includes the SDR gateway's ID, the active period's ID, the channel it was received on, sampling rate, time the active period was received at the client, as well as other useful information such as the data-section size and its



(a) (b)

Fig. 6: (a) RURAL : Outdoor rural deployment where LoRa Tx (yellow circles) transmit to a USRP B200 (red triangle), which is then connected to a client running ACCIO that streams to Azure server through cellular hotspot in real-time. (b) URBAN : LoRa Tx transmit to a USRP which stores the received samples in a local file.

DWT level sizes (needed for Inverse DWT).

We utilize BBR as the TCP variant; it provides the estimated network bandwidth to the client. We use the socket statistics tool to obtain the bottleneck bandwidth, delivery rate estimated by BBR, and the link's average round-trip time. The bottleneck bandwidth is a key state used by the RL agent to determine a compression threshold.

LoRa receiver at the Cloud Server. The cloud server is implemented in Microsoft Azure as Docker Containers [46], demodulating packets and sending rewards back in real-time(Cloud-LoRa is amenable to deployment on other cloud providers as well.) Our server utilized 8 Docker containers, each reading on unique ports corresponding to each LoRa channel. The Docker containers were booted using Docker Compose [47] and each container was running on an Azure VM. The first module of the cloud server is a multiplexer that decompresses the received samples: it first performs the inverse of Lz4, reads the metadata, and then decompresses using Inverse DWT. Using the metadata, the multiplexer routes the decompressed DWT coefficients to the corresponding user-defined consumer (demodulator). In other words, the multiplexer is responsible for reconstructing the active periods and placing them in the queue of the appropriate consumer based on the metadata of the received TCP packets. The consumers are Docker containers that take the reconstructed active period samples as input, and run the user-defined LoRa demodulator algorithm that outputs symbols, followed by the LoRa decoder that outputs bits. The number of packets decoded per active period, weighted by the inverse-true-positive-rate is used by the RL agent as a reward component in the feedback channel back to the corresponding CRAN gateway.

5 Evaluation

We have deployed the first LoRa CRAN operating in real-time, in two practical outdoor deployments/scenarios. In RURAL(Fig 6(a)), we deployed eight LoRa transmitters in an agricultural farm. Here, we use a cellular backhaul, whose bandwidth varies with time; the backhaul bandwidth is the bottleneck in the network. We show the real-time operation of Cloud-LoRa in this scenario averaged across multiple days. URBAN(Fig 6(b)) shows an urban deployment where the backhaul does not pose a limitation in bandwidth. We leverage this scenario to perform controlled experiments, evaluate the micro-benchmarks and perform an ablation study. Towards evaluating Cloud-LoRa , we answer the following questions.

1. How well does Cloud-LoRa perform in rural settings with impoverished cellular backhaul?
2. Can Cloud-LoRa enable real-time joint decoding of LoRa packets from multiple gateways in the cloud to improve coverage or capacity?
3. Can Cloud-LoRa enable rapid deployment of recently developed state-of-the-art LoRa demodulators?
4. Can Cloud-LoRa scale elastically to provision for network capacity by increasing the number of channels?
5. How does ACCIO adapt in real-time to changing backhaul bandwidth, network latency, and channel quality?
6. How does ACCIO's adaptive compression respond to varying backhaul network latency, and how well does it adapt to bandwidth variations?

5.1 Real-world Deployment Settings

We describe our rural and urban deployments in detail below.

RURAL : Rural deployment scenario. As shown in Fig. 6(a), our deployment includes 8 LoRa transmitters (yellow circles) that broadcast data from humidity sensors to a CRAN gateway (red triangle). Each transmitter operates in a dedicated 125 kHz BW LoRa channel and chooses a random SF and packet length to emulate rate adaptation in LoRaWAN networks while actively transmits 10% of the time. Our CRAN gateway receives over 902.2 MHz to 904.2 MHz and receives samples over 8 different LoRa channels one for each transmitter. It uses a Netgear cellular mobile hotspot as backhaul to a CRAN cloud server in Microsoft Azure (Fig. 6(a)). The backhaul bandwidths achieved by the cellular hotspot varied over a wide range: 1 Mbps to 15 Mbps at different locations and times. As shown in Fig. 6(a), Cloud-LoRa streams samples to the cloud server in real-time, using ACCIO to learn and adapt to the varying available bandwidth. The transmitters were left in the field over 2 days with a total of ≈470000 packets transmitted. The CRAN gateway did not

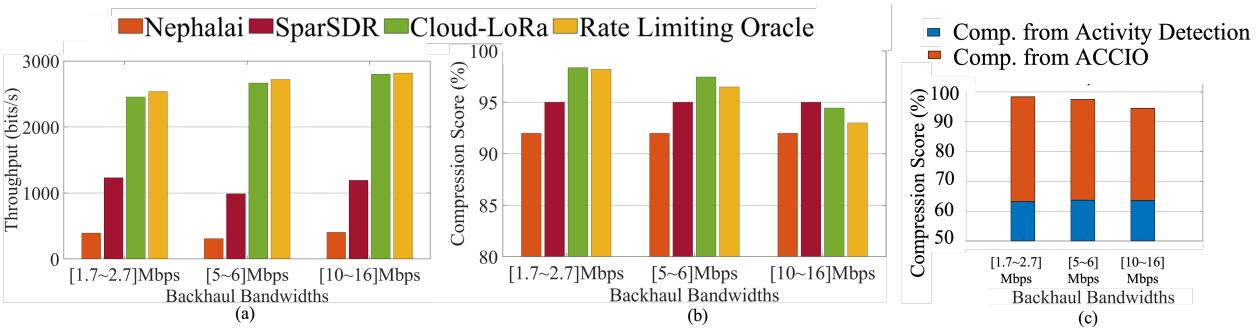


Fig. 7: (a) : Cloud-LoRa throughput performance across 8 parallel LoRa channels, averaged over multiple days in RURAL scenario. (b) Corresponding compression performance. (c) Ablation study on Compression.

have any pre-trained model for ACCIO to use; instead, the RL agent learned from scratch and adapted in real-time.

URBAN: Urban deployment used for Ablation Study. We deploy 9 off-the-shelf LoRa transmitters, each operating in a different channel in an urban, outdoor setting (Figure 6(b)). The transmitters were deployed over an area of 2.5 km x 1 km. The CRAN gateway receives the samples from all the transmitter over a wide bandwidth and stores them locally with time stamps to enable replay. The stored samples are then replayed in real-time to the cloud server to emulate real-time streaming. We connect the USRP to the cloud server via a router. This activity is to ensure consistency across multiple microbenchmark experiments that run with different parameters. This setup allows us to simulate different backhaul bandwidths and latencies to the cloud by using Linux Traffic Control (TC) [48] at the router, a tool for shaping traffic. We perform controlled, comparative, and ablation studies using this deployment by varying various factors such as backhaul bandwidths(Sec. 5.6), LoRa channel quality(Sec. 5.7), network load(Sec. 5.5), backhaul latency(Sec. 5.7), and others.

Backhaul Compression Baselines compared. We implement and compare the compression and throughput performance of Cloud-LoRa against five baselines: 1) **Standard LoRa** – a LoRa gateway that demodulates each packet at the gateway *i.e.*, without CRAN; 2) CRAN with **No compression**; 3) **Nephalai** [19], which proposes a compressed-sensing-based static compression; 4) **SparSDR** [24] – a sparsity-aware compression which is agnostic to the PHY-layer technology; and 5) **Rate-limiting Oracle**. This oracle provides a theoretical upper bound on the throughput and compression performance. We assume that the oracle has a global view of the incoming traffic and bottleneck bandwidth, is not limited by computational resources, performs perfect activity detection with zero false positives and, is able to compress the active periods exactly to meet the available bandwidth.

5.2 Performance in a Rural, Bandwidth-Constrained Deployment

In RURAL we repeated the deployment over three separate 8hr sessions. In each session, the cellular hotspot was placed at roughly the same location (within a 2m radius). Despite using roughly the same location for the hotspot, we found significant variation in the backhaul bandwidth ranges in these three sessions. Arranging the sessions in increasing average backhaul bandwidth, the ranges were 1.7-2.7Mbps, 5-6Mbps and 10-16Mbps. The distribution of the received SNR collected over all 24hrs from all the transmitters (≈470000 packets) are shown in Fig. 8. As seen from Fig. 8, there is a wide variation in received SNRs at the gateway from -15dB to 30dB. ACCIO continuously learns and adapts its compression to meet the available cellular backhaul bandwidths (Fig. 6(a)) and simultaneously streams 8 channels. We plot the average LoRa throughput over all 8 channels (bits/second) achieved by Cloud-LoRa in Fig. 7(a) for the three different sessions. We also compare the achieved average network throughput of Cloud-LoRa with Nephalai, SparSDR, and Rate-limiting Oracle. The maximum achievable throughput *with compression* is upper bounded by the rate-limiting oracle. We observe that the throughput of Cloud-LoRa approaches that of the oracle at higher backhaul bandwidths of 10-16 Mbps, while achieving ≈ 96% of that of the Oracle even at 2Mbps backhaul badwidths. Further, Cloud-LoRa is able to use its adaptive compression effectively and significantly outperforms other LoRa compression solutions such as Nephalai by 7x and 6.2x, and SparSDR by 2.3x and 1.9x (on average).

In Fig. 7(b), we plot the compression score, defined as (# samples streamed to the cloud) / (# samples captured by the CRAN gateway), for the same backhaul bandwidths. We observe that Cloud-LoRa has a higher compression score of 98.4% at 1.7 to 2.7 Mbps, while a lower compression of 94% at 10 to 16 Mbps, *i.e.*, it compresses more at low backhaul bandwidths. On the other hand, Nephalai and SparSDR adopt static compression and hence face packet losses when the compression does not meet the backhaul bandwidth. The most appropriate compression necessary using CRAN is that of the Oracle, as it has a global view; Cloud-LoRa is within 98.5% of the oracle's comrpession score on average.

Two key reasons for the improved throughput performance of Cloud-LoRa over existing approaches are i) sub-noise activity detection and ii) adaptive compression. The contribution of each of these modules to the overall compression is shown in Fig. 7(c). On average, the activity detection achieves a

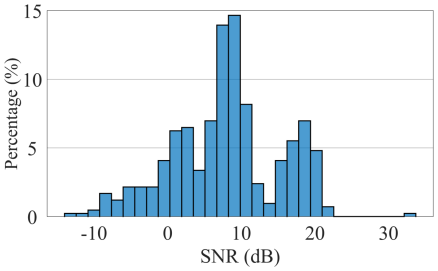


Fig. 8: Distribution of the received SNR at the Gateway in RURAL

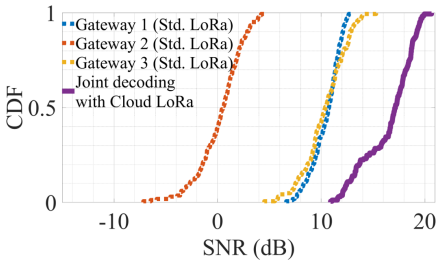


Fig. 9: SNR Gains from Joint Multi-Gateway Packet Decoding

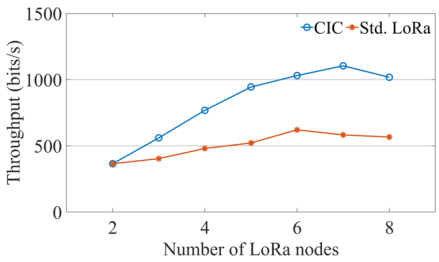


Fig. 10: Throughput Improvements due to Rapid Deployment of state-of-the-art

compression score of 63% in our setting by distinguishing active LoRa transmission from noise samples. The remaining compression is achieved by ACCIO , which varies the compression threshold of the active period to meet the backhaul bandwidth. While the compression score achieved by the activity detection block does not change with backhaul or LoRa signal characteristics, that of ACCIO changes with backhaul bandwidth, as evident in Fig. 7(c).

5.3 Joint Multi-Gateway Packet Decoding

Cloud-LoRa offers centralization, which is key to jointly process raw radio signals across multiple gateways such that the SNR of weak LoRa links could be enhanced through coherent combining with relatively stronger links. In this section, we deploy Charm [6] using Cloud-LoRa using 3 Cloud-LoRa gateways deployed on the floor of a large building spanning over 100m $\times$ 50m along with a LoRa transmitter transmitting packets. On detecting activity, the gateways would attach a time stamp, gateway ID and relay the samples to the cloud. In the cloud, we deploy Charm which coherently combines samples from the 3 gateways based on time stamps and packet/gateway IDs and decodes 100% packets. In Fig. 9, we plot the CDF of packet SNR received across 3 USRP gateways streaming samples. Jointly decoded packets achieve a mean SNR of 16dB, a 6dB improvement from even the stronger links, i.e. Gateways 1 and 3. This SNR boost almost doubles the coverage area. At each gateway, ACCIO achieves 93% compression. We also plot the network throughput achieved by joint decoding when each gateway faces varying backhaul bandwidths. These results are in Appendix E.

5.4 Rapid Deployment of state-of-the-art

Another key advantage of LoRa CRAN is the rapid deployment of novel PHY techniques to practice. In this section, we evaluate a state-of-the-art LoRa receiver in the cloud server, i.e. Concurrent Interference Cancellation (CIC) [12] published as recently as 2021 using Cloud-LoRa . CIC improves LoRa network throughput by decoding multi-packet collisions. In Fig. 10, we plot the network throughput with CIC as the demodulator in the cloud and compare it against Std. LoRa in the cloud. We increase the number of concurrent nodes (same SF and BW) in a single channel in the x-axis and stream the samples to the cloud, where CIC is used as the demodulator. Cloud-LoRa was capable of transporting the samples in real-time for CIC to demodulate. Hence, the network throughput shows an improvement of 1.9x over standard LoRa when 7 nodes are colliding. The throughput begins to drop beyond 7, the maximum collisions CIC can resolve.

5.5 Elastic Scaling to Multiple Channels

Cloud-LoRa allows for scaling to provision for higher capacity by increasing the number of channels from single to multiple channels without any hardware change. However, as the number of channels increases, the volume of samples increases. With limited backhaul bandwidths, such an increase in samples demands higher compression. In this section, we answer the question of *how well ACCIO adapts to an increasing number of channels for a given bottleneck bandwidth*. We plot the number of packets decoded in the cloud at 10 Mbps bottleneck bandwidth for increasing number of channels in Fig. 11. When only one channel has active LoRa signals, a 10Mbps backhaul can support low compression. As the number of active channels increases, the load and hence the number of samples increases. At 10 Mbps bandwidth, a static compression of 50% for Nephalai is best suited for up to 2 channels since packet losses due to compression would be the bottleneck in this case. With 4 or more channels, Nephalai-75 is better suited since packet losses due to network congestion would dominate. ACCIO on the other hand adapts its compression to meet the bandwidth, despite the increase in traffic load. Cloud-LoRa 's network throughput increases linearly by an order of magnitude as the number of channels increases from 1 to 8. Cloud-LoRa decodes about 2X and 4X more packets than Nephalai-75 and Nephalai-50 respectively for 4 channels, and 12X and 20X for 8 channels.

5.6 Varying Backhaul Conditions

Bandwidth-Aware ACCIO In the practical deployment, we have witnessed the adaptation of Cloud-LoRa 's compression performance to different backhaul conditions. However, due

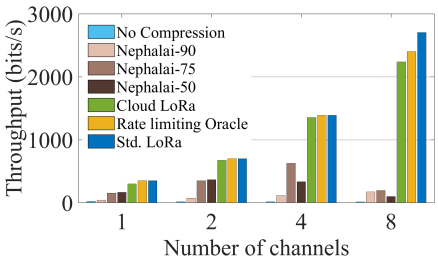


Fig. 11: Elastic Scaling to Multiple Channels - Throughput Vs # of channels

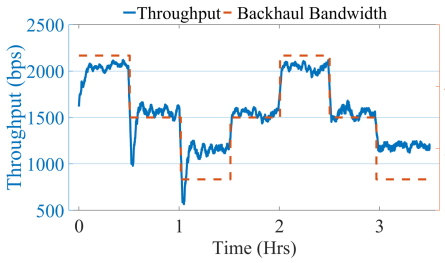


Fig. 12: ACCIO Adaption to Varying Backhaul Bandwidths

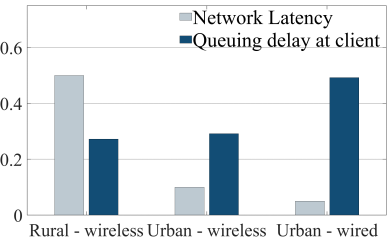


Fig. 13: ACCIO Adaption to Varying Backhaul Latency

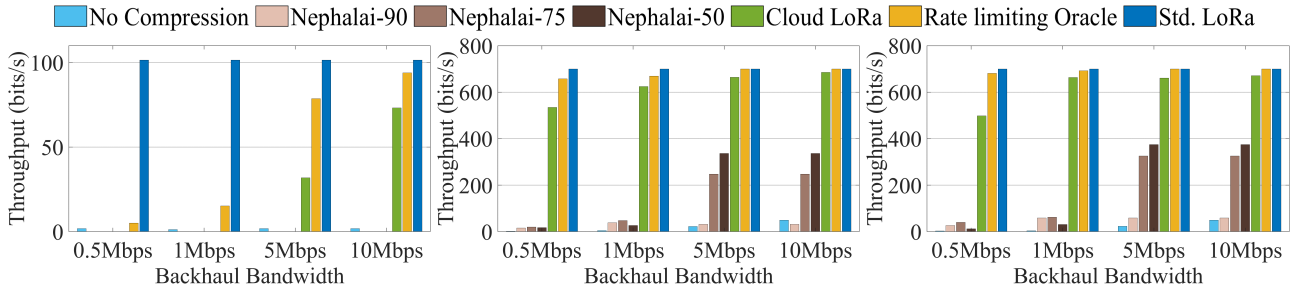


Fig. 14: ACCIO adaptation to varying LoRa Channel Quality
(a) Low SNR (-20 to -5 dB) (b) Medium SNR (-5 to 10 dB) (c) High SNR (10 to 25 dB) LoRa signals.

to the uncontrolled cellular backhaul, it is challenging to observe its time to learn and adapt. In this section, we study the adaptation of Cloud-LoRa to varying backhaul bandwidths in a controlled setting. Using the I/Q samples collected and stored at the CRAN gateway in the outdoor RURAL, we replay the real-time streaming of samples to a cloud server *i.e.*, the gateway streams samples stored in a file to the cloud server, where the samples are decoded, which are then used as rewards by ACCIO at the gateway. During this replay, we control the backhaul bandwidths using Linux-TC. We vary the backhaul bandwidths every 0.5 hours, as shown by the dotted orange line in Fig. 12, from 5 Mbps down to 1 Mbps, and back to 5 Mbps, and then 1 Mbps in steps of 2 Mbps every 0.5 hours. During the first ramp down, LoRa throughput drops and then ramps up every time backhaul bandwidth changes. This is because, in the first 1.5 hours, the ACCIO module at the gateway is untrained *i.e.*, it has not faced the new backhaul conditions before, and hence takes time to learn a new policy at the gateway. Once it learns the policy, the throughput flattens. This is evident from the dip in throughput (blue curve) at 0, 0.5, and 1 hour mark in Fig. 12. The RL agent takes approximately 10 minutes to learn a new policy when it faces a new backhaul bandwidth. After the 1.5 hour mark, when the backhaul bandwidth ramps up to 5 Mbps, LoRa throughput approaches the steady state quickly at 1.5, 2, 2.3, and 3 hour marks as the RL agent has learned a policy for these backhaul bandwidths in the first 1.5 hours. The adaptation time of Cloud-LoRa to varying backhaul conditions therefore depends on the historical data.

Latency-Aware ACCIO Figure 13 plots the queueing delay at the SDR gateway for three networks with different network latencies. In each of these networks, the application latency requirement is designed to be 2 seconds. Therefore, the RL agent learns to drop packets at the client and/or compress more when the network latency is high such that the overall latency is below 2 seconds. In the case of networks with low latency, the RL agent tolerates more queueing delay at the client, allowing it to send more packets.

5.7 Varying LoRa Channel Quality

Compression depends on the SNR of the LoRa samples at CRAN gateway. We evaluate the throughput and compression performance of ACCIO as a function of SNR at different backhaul bandwidths in Figure 14. The corresponding compression performance are presented in Appen. E. Figure 14 (a), (b), (c) correspond to SNR : low (-20 to -5 dB), medium (-5 to 10 dB), and high (10 to 25 dB) respectively. At SNR $<$ 0 dB, Nephalai decodes less than 5% of the packets even at 50% compression (Nephalai-50). This is due to its inability to differentiate between noise and active LoRa signal. Cloud-LoRa improves network throughput by over 20X compared to Nephalai at low backhaul bandwidths. Cloud-LoRa's throughput performance is limited by the false positive rate of the activity detection, which results in a higher volume of active LoRa samples to be transported. At a backhaul bandwidth of 1 Mbps, the RL agent chooses compression scores of over 99% to meet the backhaul constraints (see Fig. 19 in Appendix),

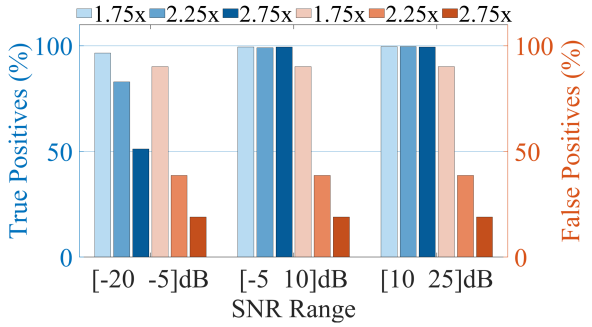


Fig. 15: Activity detection performance

resulting in poor reconstruction in the cloud. As the backhaul bandwidth increases, the achievable throughput improves to over 70% of that of a standard gateway. The network throughput of Cloud-LoRa for Medium SNR signals (green bars in Fig. 14(b)) is about 90% at 1 Mbps backhaul with a compression score of approximately 91%. Nephalai fails to decode more than 10% at 1 Mbps; this can be attributed to the lack of activity detection in Nephalai which leads to redundant samples taking up available bandwidth.

Lossless compression such as Lz4-0 and Gzip9 offers better decodability than ACCIO at bandwidths >= 5Mbps. As the bandwidth decreases, Lz4-0 faces over 50% packet loss due to network losses, similar to Nephalai-50. While Gzip9 achieves 75% compression and has lower packet loss even at low bandwidths, it is too slow to use for real time compression. Even at a backhaul bandwidth of over 5 Mbps, Nephalai-50% compression decodes only about 50% of the packets, while Cloud-LoRa decodes more than 95% of the packets with an impressive compression of over 91%. For high SNRs ACCIO transports over 95% of packets (Fig. 14(c)) to the cloud with an average compression score of 98%.

5.8 Activity Detection of Cloud-LoRa

We evaluate the tradeoff between the sensitivity to low SNR and false positive rate in detecting active LoRa packets of our proposed activity detection algorithm. The proposed multi-channel, sub-noise LoRa activity detection is agnostic to the transmitters' SF. Therefore, the sensitivity of the module determines the minimum SNR that can be detected. We plot the percentage of true active periods and false positives as a function of SNR in Figure 15. The two y-axes presented are true positives, the percentage of active periods correctly detected, normalized to that of a standard LoRa gateway (blue bars), and the percentage of active periods that were detected but did not contain a packet (false positives), normalized to the total samples received (red bars). Each bar represents a different sensitivity used by the activity detection module. A higher sensitivity results in detecting more packets even at lower SNRs at the cost of higher false positives. As SNR increases, the true positive does not vary with the threshold, as the signal energy is high. However, false positives increase with increased sensitivity, even at high SNR. Therefore, the right choice of the threshold is particularly critical in detecting the most active period at low SNRs, without trading off too many false positives. Cloud-LoRa settings that detect over 80% of the packets (Cloud-LoRa -2.25x) only result in 40% false positives, which is further compressed by RL. We observe that, combined with the RL compression, the volume of non-active samples transported to the cloud by Cloud-LoRa is minimal. In contrast, SparSDR incurs more than 90% overheads to accommodate near-zero dB SNR.

6 Discussions and Future Work

To the best of our knowledge, Cloud-LoRa is the first end-to-end practically deployable LoRa CRAN. There are plenty of promising future research directions emerging from this framework. In Cloud-LoRa , we spawn a new process for each channel and stream upto 9 channels in real-time. We can scale to the maximum 64 channels using a NUC with more cores, and a more efficient software implementation. Beyond standard LoRa, recent research such as CurvingLoRa [49] and Falcon [16] that changes the transmitter requires further study to understand the impact of compression on packet decodability. Further research is needed to incorporate performance guarantees such as throughput fairness across channels.

7 Conclusions

We proposed, designed, and implemented Cloud-LoRa , the first practical CRAN for LoRa networks. Cloud-LoRa streams LoRa signals to a cloud server that performs baseband signal processing, in turn providing opportunities for dynamic network scaling and rapid deployment of novel LoRa receivers in the cloud. Towards realizing this end-to-end framework, we developed an *activity detection* algorithm that can detect sub-noise active LoRa signals and reduce signals being streamed to the cloud. We also developed ACCIO, an *RL-based adaptive compression* technique, whose compression threshold adapts to variations in backhaul bandwidth, latency requirements, and input-signal characteristics at the gateway in real-time. We implement and deploy Cloud-LoRa as a Docker container in an Azure server, and experimentally show the feasibility of CRAN for real-world LoRa deployments.

8 Acknowledgements

We would like to thank our shepherd Kate Lin and the anonymous reviewers for the valuable comments and for helping us improve the paper. The authors are partially supported through the following NSF grants : CCSS-2034415, CNS-2142978, 2213688, 1838733, 2112562, 1719336, 1647152, 1629833, 2107060, 2212688, and 2003129 and the US Department of Commerce award 70NANB21H043.

References

[1] LoRa. https://www.semtech.com/lora.

[2] E. Asimakopoulou and N. Bessis. Buildings and crowds: Forming smart cities for more effective disaster management. In *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 229–234, 2011.

[3] María V Moreno, Miguel A Zamora, and Antonio F Skarmeta. User-centric smart buildings for energy sustainable smart cities. *Transactions on emerging telecommunications technologies*, 25(1):41–55, 2014.

[4] Achim Walter, Robert Finger, Robert Huber, and Nina Buchmann. Opinion: Smart farming is key to developing sustainable agriculture. *Proceedings of the National Academy of Sciences*, 114(24):6148–6150, 2017.

[5] Climate Smart Agriculture. https://www.worldbank.org/en/topic/climate-smart-agriculture.

[6] Adwait Dongare, Revathy Narayanan, Akshay Gadre, Anh Luong, Artur Balanuta, Swarun Kumar, Bob Iannucci, and Anthony Rowe. Charm: exploiting geographical diversity through coherent combining in low-power wide-area networks. In *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 60–71. IEEE, 2018.

[7] Artur Balanuta, Nuno Pereira, Swarun Kumar, and Anthony Rowe. A cloud-optimized link layer for low-power wide-area networks. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 247–259, 2020.

[8] Branden Ghena, Joshua Adkins, Longfei Shangguan, Kyle Jamieson, Philip Levis, and Prabal Dutta. Challenge: Unlicensed lpwans are not yet the path to ubiquitous connectivity. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–12, 2019.

[9] Xianjin Xia, Yuanqing Zheng, and Tao Gu. Ftrack: Parallel decoding for lora transmissions. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, pages 192–204, 2019.

[10] Shuai Tong, Jiliang Wang, and Yunhao Liu. Combating packet collisions using non-stationary signal scaling in lpwans. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 234–246, 2020.

[11] Rashad Eletreby, Diana Zhang, Swarun Kumar, and Osman Yağan. Empowering low-power wide area networks in urban settings. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 309–321, 2017.

[12] Muhammad Osama Shahid, Millan Philipose, Krishna Chintalapudi, Suman Banerjee, and Bhuvana Krishnaswamy. Concurrent interference cancellation: decoding multi-packet collisions in lora. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 503–515, 2021.

[13] Chenning Li, Hanqing Guo, Shuai Tong, Xiao Zeng, Zhichao Cao, Mi Zhang, Qiben Yan, Li Xiao, Jiliang Wang, and Yunhao Liu. Nelora: Towards ultra-low snr lora communication with neural-enhanced demodulation. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, pages 56–68, 2021.

[14] Qian Chen and Jiliang Wang. AlignTrack: Push the limit of lora collision decoding. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2021.

[15] Zhenqiang Xu, Pengjin Xie, and Jiliang Wang. Pyramid: Real-time lora collision decoding with peak tracking. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2021.

[16] Shuai Tong, Zilin Shen, Yunhao Liu, and Jiliang Wang. Combating link dynamics for reliable lora connection in urban settings. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 642–655, 2021.

[17] Christophe Delacourt, Patrick Savelli, and Vincent Savaux. A cloud ran architecture for lora. In *Radio Science Letters*, 2020.

[18] Eryk Schiller, Silas Weber, and Burkhard Stiller. Design and evaluation of an sdr-based lora cloud radio access network. In *2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 1–7. IEEE, 2020.

[19] Jun Liu, Weitao Xu, Sanjay Jha, and Wen Hu. Nephalai: towards lpwan c-ran with physical layer compression. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–12, 2020.

[20] FCC Report on Broadband Aceess. https://www.fcc.gov/reports-research/reports/broadband-progress-reports/2020-broadband-deployment-report.

[21] Tyler B Mark, Terry W Griffin, and Brian E Whitacre. The role of wireless broadband connectivity on 'big data'and the agricultural industry in the united states and australia. *International Food and Agribusiness Management Review*, 19(1030-2016-83150):43–56, 2016.

[22] FCC Working Paper on Digital Divide . https://www.fcc.gov/reports-research/working-papers/digital-divide-us-mobile-technology-and-speeds.

[23] Amalinda Gamage, Jansen Christian Liando, Chaojie Gu, Rui Tan, and Mo Li. Lmac: Efficient carrier-sense multiple access for lora. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–13, 2020.

[24] Moein Khazraee, Yeswanth Guddeti, Sam Crow, Alex C Snoeren, Kirill Levchenko, Dinesh Bharadia, and Aaron Schulman. Sparsdr: Sparsity-proportional backhaul and compute for sdrs. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pages 391–403, 2019.

[25] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-based congestion control. *ACM Queue*, 14, September-October:20 – 53, 2016.

[26] LoRa and LoRaWAN. https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan//.

[27] James E Prieger. The broadband digital divide and the economic benefits of mobile broadband for rural areas. *Telecommunications Policy*, 37(6-7):483–502, 2013.

[28] Christopher Ali. The politics of good enough: Rural broadband and policy failure in the united states. *International Journal of Communication*, 14:23, 2020.

[29] John Lai and Nicole O Widmar. Revisiting the digital divide in the covid-19 era. *Applied economic perspectives and policy*, 43(1):458–464, 2021.

[30] FCC Task Force. www.fcc.gov/task-force-reviewing-connectivity-and-technology-needs-precision-agriculture-united-states.

[31] Muhammad Iqbal Rochman, Vanlin Sathya, Norlen Nunez, Damian Fernandez, Monisha Ghosh, Ahmed S Ibrahim, and William Payne. A comparison study of cellular deployments in chicago and miami using apps on smartphones. In *Proceedings of the 15th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & CHaracterization*, pages 61–68, 2022.

[32] Yuanjie Li, Chunyi Peng, Zhehui Zhang, Zhaowei Tan, Haotian Deng, Jinghao Zhao, Qianru Li, Yunqi Guo, Kai Ling, Boyan Ding, et al. Experience: a five-year retrospective of mobileinsight. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 28–41, 2021.

[33] David L Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[34] Revathy Narayanan, Swarun Kumar, and Siva Ram Murthy. Cross technology distributed mimo for low power iot. *IEEE Transactions on Mobile Computing*, 2020.

[35] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall Press, USA, 3rd edition, 2009.

[36] LoRa Modulation Basics. https://www.frugalprototype.com/wp-content/uploads/2016/08/an1200.22.pdf.

[37] Stéphane Mallat. *A wavelet tour of signal processing*. Elsevier, 1999.

[38] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[39] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[40] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063. The MIT Press, 1999.

[41] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897. PMLR, 2015.

[42] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[43] USRP B200. https://www.ettus.com/all-products/ub200-kit/.

[44] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[45] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[46] Docker containers on Azure. https://docs.docker.com/cloud/aci-integration/.

[47] Docker Compose. https://docs.docker.com/compose/.

[48] Linux Traffic Control. https://man7.org/linux/man-pages/man8/tc.8.html.

[49] Chenning Li, Xiuzhen Guo, Longfei Shangguan, Zhichao Cao, and Kyle Jamieson. {CurvingLoRa} to boost {LoRa} network throughput via concurrent transmission. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 879–895, 2022.

[50] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998.

[51] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

Appendix

A LoRa Modulation and Demodulation

LoRa uses Chirp Spread Spectrum (CSS) as its PHY layer modulation. In CSS, the instantaneous frequency of the signal increases linearly with time within a predefined Bandwidth BW over a symbol duration of T_s as shown by chirp equation 2. The start frequency f_{sym} of the data chirp $S(t, f_{sym})$ encodes the data to be transmitted. The slope of the datachirp in frequency-time plot as shown in Fig.16 denotes the Spreading Factor SF which in turn determines the symbol duration $T_s = \frac{2^{SF}}{BW}$, Data Rate and range of operation. Higher SF offers longer range at the cost of reduced data rate. SF can take values $\in \{7,8,9,10,11,12\}$ and for increasing SF, symbol duration doubles. That means an $SF8$ chirp is twice the length of an $SF7$ chirp and an $SF12$ chirp is 32 times the length of an $SF7$ chirp.

$$C(t) = e^{j2\pi(\frac{BW^2}{2\times 2^{SF}}t - \frac{BW}{2})t}, 0 \leq t \leq T_s \quad (2)$$

$$S(t, f_{sym}) = C(t) \cdot e^{j2\pi f_{sym} t} \quad (3)$$

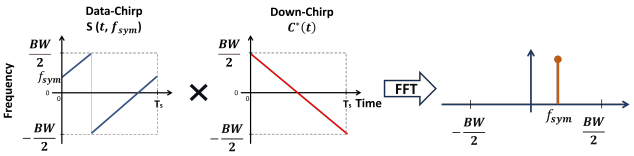


Fig. 16: LoRa Demodulation

At the receiver, LoRa demodulation starts by correlating the received buffer with a preamble (8 base upchirps $C(t)$) to determine the start of LoRa packet. The receiver correlates the buffer with preambles of all possible SFs to reveal the spreading factor SF_x of received packet. It then demodulates the data by multiplying the data symbols with downchirp of same SF_x as shown in Fig.16. Downchirp is a conjugate of base upchirp whose frequency decreases linearly with time. This multiplication is called dechirping and it concentrates the signal energy into a single frequency which is equal to the start frequency of the data symbol. Index of the peak in the FFT of the dechirped signal gives us the demodulated data.

A.1 Chirps of different SF are Pseudo-orthogonal.

As discussed above, to detect the presence and thus start of a LoRa packet, receiver correlates the incoming I/Q samples with base upchirps of all SFs. Once a packet of specific $SF = x$ is detected, datachirps are then demodulated using a downchirp of $SF = y : x = y$. Two chirps (a datachirp and a downchirp) for which $x \neq y$ cannot concentrate energy of the datachirp through dechirping since chirps of unequal SFs

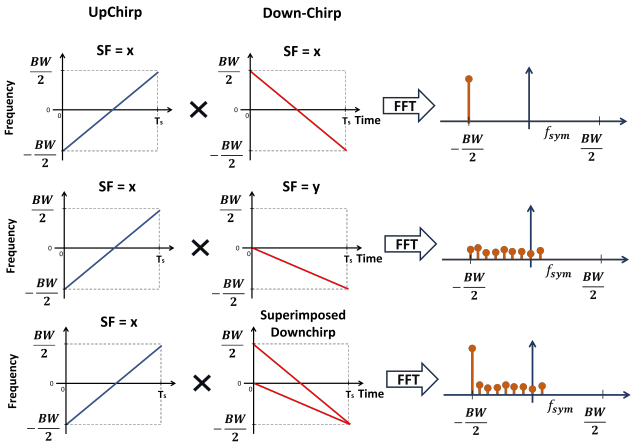


Fig. 17: Pseudo-Orthogonal LoRa chirps

are pseudo-orthogonal. For such cases, dechirping only removes the linear increase in frequency-time trend of the chirps when the magnitude of frequency-time slope of both chirps is equal as shown in top plot of Fig.17. It therefore results in clear peak in the FFT of the dechirped signal. In contrast, if the two chirps have unequal SF, and therefore different magnitude of slopes as shown in the middle plot of Fig.17, dechirping followed by FFT will not concentrate signal energy into a single peak. Instead, the energy is spread over multiple frequencies (hence the term *pseudo*) depending upon the difference in x and y. Therefore, pseuso-orthogonality of SF means that chirps of similar SF can only dechirp signals to a single frequency whereas chirps of unequal SF do not show this behaviour.

Based on this observation, if we design a downchirp which is superposition of two downcirps of $SF = x$ & y respectively and use it to dechirp a datachirp of $SF = x$, we will obtain energy concentration due to downchirp of $SF = x$ and will obtain an increased noise floor due to downchirp of $SF = y$ as shown in bottom plot of Fig.17.

B Primer on the Discrete Wavelet Transform

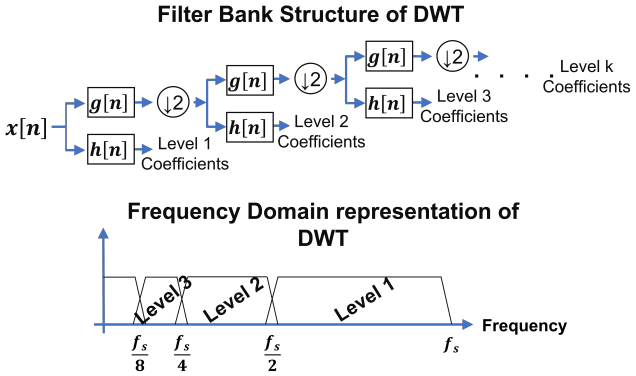


Fig. 18: Filter Bank Implementation of DWT

DWT is a multi-resolution time-frequency analysis tool that

is widely used in image signal processing. It decomposes the input signal into a set of mutually orthogonal wavelet basis functions, that are shifted and scaled versions of a *mother wavelet*. With the appropriate choice of the mother wavelet, DWT provides high time and frequency resolution. Hence, it is an efficient tool to de-noise and compress signals with varying frequency content, such as chirp spread spectrum used by LoRa.

Figure 18(top) shows a k-stage DWT that uses a series of high-pass (h[n]) and low-pass (g[n]) filter banks to decompose the signal into coefficients at multiple levels. This filter-bank implementation of DWT of an input signal of length N has a complexity of $O(N)$, making it computationally a minimally intensive operation, and hence enabling compression in real-time. The DWT coefficients from higher levels of decomposition provide detailed time information. The coefficients from lower levels of the DWT provide the more detailed frequency information. Each DWT coefficient represents the energy of the received signal corresponding to the frequency (level) and time (shift).

C Reward Function Shaping

We discuss the specific choice of the reward function used in our RL algorithm. Recall that the reward function is designed as follows:

$$r(\mathbf{s},a) \;=\; u_{dec}(\mathbf{s},a) - u_{band}(\mathbf{s},a) - u_{lat}(\mathbf{s},a) - u_{over}(\mathbf{s},a). \tag{4}$$

The first reward term encourages the RL algorithm to maximize the total number of LoRa packets decoded correctly over an episode of compression, consisting of (potentially) multiple APs. As discussed in Section 3.2, we would like to weight the number of packets decoded by the estimated true positive rate (TPR) in order to account for the false positives included in the count. Since, the TPR cannot be estimated exactly due to absence of ground truth about when an actual LoRa transmission occurs, we estimate the TPR as the fraction of LoRa packets decoded correctly over the last 100 detected active periods. Suppose N_{dec} is the number of packets decoded correctly and $\hat{P}_{tpr}$ is the estimated TPR, then

$$u_{dec}(\mathbf{s},a) \;=\; \frac{N_{dec}}{\hat{P}_{tpr}}. \tag{5}$$

The second term in the reward function $u_{band}(\mathbf{s},a)$ penalizes the bandwidth utilized B from getting very close to the available bandwidth $B_{\max}$ and is defined as follows:

$$u_{band}(\mathbf{s},a) \;=\; \begin{cases} 0, & \text{if } \frac{B}{B_{\max}} \in [0,\alpha] \\ \frac{0.5\alpha}{1-\alpha^2}\left(\frac{B}{B_{\max}} - \alpha\right), & \text{if } \frac{B}{B_{\max}} \in [\alpha, \frac{1}{\alpha}] \\ 0.5, & \text{if } \frac{B}{B_{\max}} > \frac{1}{\alpha}. \end{cases} \tag{6}$$

Here $\alpha \in (0,1)$ is a constant that determines how close to the maximum bandwidth we want to start penalizing the RL algorithm. We set $\alpha = 0.9$ in our experiments. As shown in Fig. 5, the graph of $u_{band}(\mathbf{s},a)$ as a function of the bandwidth ratio (BWR) $B/B_{\max}$ would be a constant 0 for BWR values less than α; a straight line ranging from 0 to 0.5 for BWR values in the interval $[\alpha, 1/\alpha]$; and a constant value of 0.5 for BWR values larger than $1/\alpha$. The idea is that we start giving as negative reward (penalty) as the BWR approaches 1 and the penalty increases until the BWR exceeds a value slightly larger than 1.

The third term in the reward function $u_{lat}(\mathbf{s},a)$ is very similar to the second term. We simply replace the BWR with the ratio of the overall latency T_{lat} to the maximum latency $T_{\max}$, and it is given below for completeness

$$u_{lat}(\mathbf{s},a) \;=\; \begin{cases} 0, & \text{if } T_{lat} \in [0,\alpha] \\ \frac{0.5\alpha}{1-\alpha^2}\left(\frac{T_{lat}}{T_{\max}} - \alpha\right), & \text{if } \frac{T_{lat}}{T_{\max}} \in [\alpha, \frac{1}{\alpha}] \\ 0.5, & \text{if } \frac{T_{lat}}{T_{\max}} > \frac{1}{\alpha}. \end{cases} \tag{7}$$

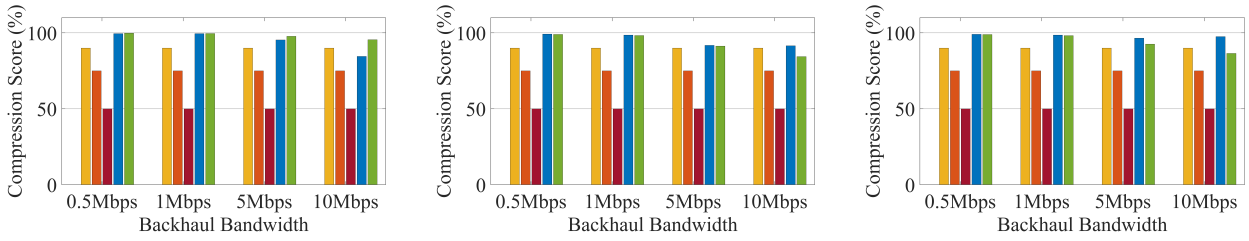
We set $T_{\max} = 0.2$ in our experiments. The factor 0.5 is included in both the penalty terms in order to balance out the rewards due to correct packet decoding and the two penalty terms. We want to avoid giving a higher overall weight to the penalty terms in order to encourage the RL to focus on its main goal of accurate packet decoding.

As discussed in Section 3.2, the final penalty term $u_{over}(\mathbf{s},a)$ is included to preemptively avoid packet buffer overflow, which could result in dropped packets. It is set to a constant -10 whenever an action of the RL agent could result in a packet buffer overflow.

D Background on Reinforcement Learning

Reinforcement learning (RL) is a machine learning paradigm of sequential decision making where an agent acting in an uncertain (stochastic) environment learns to perform actions by interacting with the environment using trial and error (rather than direct supervision) in a way that it maximizes its cumulative reward [39, 50]. This is a popular setting for learning when it is hard to obtain supervised data (*e.g.*, labeled inputs) and the data distribution is non-stationary.

The main component of RL is an agent (*e.g.*, a self-driving car), which is the decision-making center that acts in an uncertain environment (*e.g.*, a street). At any time step t, the agent senses the environment and obtains a signal or reading known as the state $\mathbf{s}_t \in \mathcal{S} \subseteq \mathbb{R}^{d_s}$ (*e.g.*, processed sensor inputs from cameras, Radar, LIDAR etc), which is typically a vector denoted in bold. Given the state $\mathbf{s}_t$, the agent performs an action $\mathbf{a}_t \in \mathcal{A}$ (*e.g.*, the steering angle, brake position *etc.*) that has an effect on the environment. Here $\mathcal{S}$ and $\mathcal{A}$ are the state space and action space respectively. The environment transitions to a new state $\mathbf{s}_{t+1}$ according to its (usually unknown) *state-transition function* $\mathbb{P}(\mathbf{s}_{t+1} \,|\, \mathbf{s}_t, \mathbf{a}_t)$ that governs its dynamics. This is a conditional probability distribution of the state at time $t+1$ given the state and action at time t. The environment



(a) (b) (c)

Fig. 19: (a-c): Compression performance of Cloud-LoRa in a single channel (a) Low SNR (-20 to -5 dB) (b) Medium SNR (-5 to 10 dB) and (c) High SNR (10 to 25 dB) LoRa signals.

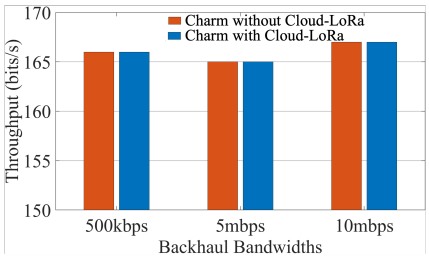


Fig. 20: Throughput performance of Charm with and w/o Cloud-LoRa

provides feedback to the agent in the form of a *reward signal* $r_t := r(\mathbf{s}_t, \mathbf{a}_t) \in \mathbb{R}$ that informs the agent of how good or bad its action $\mathbf{a}_t$ was in the state $\mathbf{s}_t$. Starting from an initial state $\mathbf{s}_0$, this sequence of state observation, action, reward, and state transition is repeated for a number of time steps, known as *an epsiode or trajectory* $\tau := (\mathbf{s}_0, \mathbf{a}_0, r_0, \mathbf{s}_1, \mathbf{a}_1, r_1, \cdots, \mathbf{s}_T, \mathbf{a}_T, r_T)$.

The agent learns a strategy or policy to perform actions in different states by repeatedly interacting with the environment over several episodes, with the goal of maximizing its total rewards. Formally, the policy $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0,1]$ of the agent is a conditional probability distribution over the set of actions given the state, *i.e.*, $\pi(\mathbf{a} \,|\, \mathbf{s}) := \mathbb{P}(\mathbf{A}_t = \mathbf{a} \,|\, \mathbf{S}_t = s)$. We define the *discounted return* U_t at time t as the *discounted cumulative future reward*, with a discount factor $\gamma \in [0,1]$, given by $U_t = R_t + \gamma R_{t+1} + \cdots + \gamma^{T-t} R_T$ [4]. The discount factor γ determines the relative importance of the current reward over future rewards. For a given policy π, the *state-value function* $V_\pi(\mathbf{s}_t)$ and *action-value function* $Q_\pi(\mathbf{s}_t, \mathbf{a}_t)$ are important quantities that define the value of a given state or a state-action pair in terms of the (discounted) cumulative future rewards. The action-value function captures how good an action $\mathbf{a}_t$ is while being in state $\mathbf{s}_t$, and is defined as

$$Q_\pi(\mathbf{s}_t, \mathbf{a}_t) \;=\; \mathbb{E}[U_t \,|\, \mathbf{S}_t = \mathbf{s}_t, \mathbf{A}_t = \mathbf{a}_t],$$

where the expectation is over all the future states and actions, and defined by the policy and the state transition function. The state-value function (or simply value function) captures how good a given state is under the policy π, and defined as

$$V_\pi(\mathbf{s}_t) \;=\; \mathbb{E}[U_t \,|\, \mathbf{S}_t = \mathbf{s}_t] \;=\; \mathbb{E}_{\mathbf{A} \sim \pi(\cdot \,|\, \mathbf{s}_t)}[Q_\pi(\mathbf{s}_t, \mathbf{A})].$$

The goal of an RL agent is to find an optimal policy that maximizes the expected value function $\mathbb{E}_{\mathbf{S}}[V_\pi(\mathbf{S})]$. It is common to use the *advantage function* $A_\pi(\mathbf{s}_t, \mathbf{a}_t) = Q_\pi(\mathbf{s}_t, \mathbf{a}_t) - V_\pi(\mathbf{s}_t)$, which captures the excess return (in a state $\mathbf{s}_t$) obtained by performing action $\mathbf{a}_t$, compared to the expected return obtained over all possible actions.

There are many classes of RL methods, and recently deep learning methods have been adopted to solve the traditionally challenging setting of large and continuous state/action spaces [51]. We focus on a particular type of on-policy RL known as Policy Gradient methods [40, 41], whose goal is to directly learn an optimal policy function that is parameterized by a neural network. Specifically, we use the Proximal Policy Optimization (PPO) method with clipped objective [42] for online training of the RL agent. PPO is widely adopted as a state-of-the-art online policy-gradient method due to its better computational and sample efficiency, and stable policy function updates.

E Supplementary Results

The amount of compression by ACCIO depends on the SNR of the LoRa samples at CRAN gateway. The compression performance of ACCIO for varying LoRa channel quality are presented in Figure 19. Figure 19 (a), (b), (c) correspond to low SNR (-20 to -5 dB), medium SNR (-5 to 10 dB), and high SNR (10 to 25 dB) respectively.

In Fig. 20, we plot the network throughput of Charm with 3 USRP gateways streaming samples to the cloud. Charm coherently combines samples from the 3 gateways and decodes the packets. We compare Charm with Cloud-LoRa , against Charm on a local machine without any compression (*i.e.*, Charm without a bottleneck backhaul). It can be noted that the throughput is unaffected by Cloud-LoRa even at bandwidths as low as 500 kbps, indicating that the compression did not affect the quality of the LoRa signals, and still allows Charm to coherently combine signals in the cloud.

[4] The state, action, and reward random variables are denoted in uppercase, taking on specific values denoted in lowercase.