Improving Scalability and Throughput of
Low Power Wide Area Networks

By

Muhammad Osama Shahid

A dissertation submitted in partial fulfillment of
the requirement of degree of

Doctor of Philosophy
(Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON
2025

Date of final oral examination: 06/10/2025

The dissertation is approved by the following members of the Final Oral Committee:

Bhuvana Krishnaswamy, Assistant Professor, ECE
Paramesh Ramanathan, Professor, ECE
Kassem Fawaz, Associate Professor, ECE
Jingyi Huang, Assistant Professor, Soil Science

ACKNOWLEDGMENTS

First and foremost, I thank the Almighty for blessing me with such favorable beginnings and guiding me along an exciting and beautifully unfolding path.

I would like to express my deepest appreciation and gratitude to the following individuals who have contributed significantly to the completion of this thesis:

I am deeply grateful to my parents for their unwavering love, constant encouragement, and the unwavering belief they've always had in my abilities. They've always been there for me. I also want to express my heartfelt gratitude to my amazing sisters for their love and support.

I extend my heartfelt thanks to my advisor, Dr. Bhuvana Krishnaswamy, whose unwavering support and guidance have been pivotal throughout this research journey. I am grateful for her mentorship, which has steered me in the right direction and has also given me the space to grow. Witnessing her journey as an exceptional researcher has been a constant source of inspiration.

I would like to extend my sincere gratitude to my committee members, Dr. Parmesh Ramanathan, Dr. Kassem Fawaz, and Dr. Jingyi Huang. Their valuable feedback has been invaluable in writing my dissertation.

A special thanks goes to my labmates, who created a warm and friendly environment in the laboratory. Yaman, a great mentor, has been a constant source of support. Daniel (an amazing project partner), Manan and Akhil (collaborators with immense potential), Yoganand, Muchen, Sumit and Jackie have been valuable teammates.

I must also highlight the staff of the ECE department who have always made sure to keep me well-informed and have made the process as smooth as they could. Kathy, Alex, Dan, David, Katrina, Delight, and everyone else make sure that we have access to all the resources we need to do the

work we want to do.

Finally, I acknowledge that my achievements are built upon the shoulders of the great teachers and mentors who have guided me throughout my life. In particular, I am grateful to Dr. Momin Uppal and Dr. Muhammad Tahir, my undergraduate advisors, whose teaching inspired me and whose wisdom and support laid a strong foundation for my journey. The collective knowledge, guidance, and inspiration I have received from the academic community have been indispensable in shaping my research endeavors.

To everyone mentioned above and all the friends and family who have contributed in various ways, I am sincerely grateful for your support, encouragement, and belief in me. Thank you all for being an integral part of this journey. This would not be possible without you.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The overall objective of this thesis is to find novel solutions to improve the throughput and scalability of LoRa Networks while preserving the low power and long range features offered by LoRa modulation.

LoRa has seen widespread adoption as a long range IoT technology. As the number of LoRa deployments grow, packet collisions undermine its overall network throughput. In Chapter Three, I propose a novel interference cancellation technique – Concurrent Interference Cancellation (CIC), that enables concurrent decoding of multiple collided LoRa packets. Through LoRa deployments using COTS devices, we demonstrate that CIC can increase the network throughput of standard LoRa by up to $10\times$ and up to $4\times$ over the state-of-the-art research.

In Chapter Four, I present Cloud-LoRa, the first practical Cloud Radio Access Network (CRAN) architecture for LoRa, to address the network throughput and scalability challenges of large-scale, multi-gateway LoRa networks. Cloud-LoRa improves network throughput by coherently combining signals across multiple gateways, and scales to multiple channels by implementing the receivers in the cloud. We deploy Cloud-LoRa in an agricultural field over multiple days with USRP as the gateway. 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.

In Chapter Five, I present BYOG, a LoRaWAN receiver that can receive and decode 10 channels simultaneously in real-time. Our novel and computationally lightweight end-to-end LoRa decoding algorithms can be implemented on any general-purpose software-defined radio and commercial cloud services, bringing down the cost and ease of LoRaWAN implementations.

1 INTRODUCTION

The Internet of Things (IoT) is rapidly emerging as a transformative technology, reshaping industries and redefining how we engage with our environment. IoT comprises a vast network of devices, sensors, and objects that are capable of communicating and exchanging data. In the future, the global IoT ecosystem is expected to expand to billions of devices[1], which will generate an immense amount of data. A number of applications call for connectivity of such IoT devices at large distances. For instance, applications such as smart agriculture, smart cities, industrial IoT and environmental monitoring demand connectivity solutions capable of linking end devices and sensors across distances spanning several kilometers. Even in shorter-range applications like building smart homes, the need for robust long-range protocols persists, as walls and other structural obstructions can cause significant signal attenuation and SNR degradation. These applications also necessitate long battery life, as it is often impractical to manually recharge or replace the batteries of widely distributed end devices. Additionally, they typically exhibit relaxed data-rate requirements, given that the devices transmit small amounts of data infrequently. To meet this unique set of requirements, i.e. long range, low power consumption, and low data rate, Low Power Wide Area Network (LPWAN) technologies have gained traction. Among them, LoRa (Long Range) stands out as one of the most widely adopted solutions.

Deployed globally, LoRa [2] has emerged as a dominant IoT connectivity technology, enabling applications such as smart cities, smart homes [3, 4], smart agriculture [5, 6], and industrial IoT [7, 8]. In contrast to frequency shift keying, LoRa employs Chirp Spread Spectrum (CSS) modulation, which provides enhanced range and strong resistance to narrowband interference. This modulation technique enables LoRa end devices to communicate over distances of up to 10 kilometers or more

in line-of-sight conditions. LoRa is also known for its energy efficiency. Devices can operate for several years on a single coin cell battery. However, these advantages come with the trade-off of low data rates. LoRa typically supports only a few kilobits per second, which is sufficient for the low-power, low-bandwidth requirements of typical LPWAN applications. LoRa networks operate in the license-free ISM band, which means anyone can independently deploy a LoRa Low Power Wide Area Network (LPWAN). Off-the-shelf (OTS) LoRa radios are commercially available through which IoT devices transmit messages to a gateway.

LoRa networks follow a star topology, where end devices transmit data to a central gateway. The gateway performs physical layer processing, receiving and decoding packets and typically forwards the results to a cloud-based network server. A single gateway can cover several square kilometers, but large-scale deployments often require multiple gateways to ensure wide-area coverage. LoRa also supports multiple data-rates from 0.3 kbps to 11 kbps through a combination of bandwidth and Spreading Factor (SF) configurations. Lower data rates provide longer range but require more airtime, while higher data rates enable shorter-range transmissions. Higher data-rates offer short range of around 1 km whereas lower data-rates offer long range of around 10 kms.

With the rapid proliferation of Internet-of-Things (IoT) applications, the adoption of LoRa for low-power, large-scale communication has grown significantly. As a result, an increasing number of end devices now require connectivity across diverse use cases. As LoRa networks become increasingly dense, two major challenges emerge: scalability and throughput. Given their long range, several independently administered LoRa networks often interfere with each other [9]. Consequently, as LoRa gains popularity, packet collisions in LoRa networks significantly undermine their capacity [10, 11]. As the number of end devices requesting to join a LoRa network grows, managing their coexistence around a single gate-

way becomes a significant challenge. Multiple end devices may transmit simultaneously, leading to packet interference and potential data loss.

To address this, various Multiple Access Control (MAC) protocols, such as Time Division Multiple Access (TDMA) and Frequency Division Multiple Access (FDMA) exist. Given that one of LoRa's key advantages is its low-power operation, Time Division Multiple Access (TDMA) is not an ideal choice, as it requires additional overhead for time synchronization and coordination among all devices. LoRa does implement a form of Frequency Division Multiple Access (FDMA) by supporting multiple channels, each with a unique center frequency that devices select when they join the network. However, the LoRa standard supports up to only 64 channels. Given the limited channels and 100s/1000s of devices requesting to join the network, each channel must accommodate increasing number of end devices. Moreover, within each channel, standard LoRa uses Aloha based MAC protocol in which end devices transmit whenever they have some data to transmit. This leads to *packet interference and significant degradation of network throughput.*

Although, the LoRa standard allows end devices to choose from up to 64 channels, most low-cost commercial off-the-shelf (COTS) LoRa gateways support only 8 channels, with higher-end models extending support to 16. This limited channel support restricts the extent to which end devices can leverage frequency diversity. The root cause of this limitation lies in the *hardware complexity of handling multiple data-rates* within each channel. Specifically, decoding packets across different Spreading Factors (SFs) requires dedicated receiver chains for each SF. As a result, scaling LoRa gateways to support additional channels or SFs imposes significant hardware and cost overheads, ultimately limiting the system's scalability.

Furthermore, COTS LoRaWAN gateways are inflexible, typically hard-coded to perform standard LoRa demodulation. As a result, promising new physical-layer decoding techniques often cannot be adopted without

significant hardware redesign. This *rigid, hardware-centric architecture* limits the adaptibility and scalability of deployed infrastructure.

In summary, current LoRa networks face three major limitations that hinder their ability to scale to large, diverse deployments:

1. As LoRa networks scale, packet interference becomes increasingly problematic, leading to a significant degradation in overall network throughput.

2. The lack of hardware flexibility in existing gateways hinders the adoption of state-of-the-art techniques and technological advancements, thereby constraining the overall efficiency of LoRa networks.

3. The computational complexity of LoRa packet decoding limits most gateways to processing only 4 or 8 channels, preventing the system from utilizing additional channels and restricting overall network scalability.

This thesis tackles above challenges and presents a unified exploration of techniques that aim to improve overall scalability and throughput of LoRa networks. Chapter 2 provides a detailed technical background on LoRa, equipping readers with the foundational knowledge required to engage with the rest of the thesis. Chapter 3 surveys the existing literature, highlighting state-of-the-art LPWAN solutions developed to enhance the scalability and performance of LoRa networks. The core of the thesis then introduces three key contributions, each designed to address a specific aspect of the scalability-throughput trade-off.

1- Resolving LoRa packet collisions: In the Chapter 4, I present my work Concurrent Interference Cancellation (CIC): a technique that proposes a novel PHY-layer demodulation approach that enables simultaneous decoding of multiple overlapping LoRa packets. Unlike prior methods that rely on differentiating transmissions based on hardware fingerprints,

CIC achieves this cancellation by exploiting the best available time and frequency resolution of LoRa chirp signals, to disentangle collided packets such that gateways can decode each to boost overall network throughput.

2- Proposing an amenable and flexible framework for LoRa: Adoption of CIC and many other LoRa decoding techniques in real-world deployments is hindered by a key limitation: the rigid, hardware-centric design of existing LoRa gateways. A cloud-based centralization promises such capability to run demodulators of choice and offers many more benefits. In the Chapter 5, I present Cloud-LoRa in which I propose Cloud Radio Access Network (CRAN) architecture as a way of addressing the network throughput, scalability and flexibility challenges of large-scale LoRa networks. By streaming raw radio samples to the cloud, Cloud-LoRa enables joint multi-gateway decoding to expand gateway coverage, elastic scaling to multiple LoRa channels to improve scalability and opportunity to run state-of-the-art LoRa demodulators in the cloud.

3- Reducing computational complexity of Std. LoRa Decoding: I introduce BYOG (Bring Your Own Gateway), a lightweight, SF-agnostic packet decoder that operates on general-purpose SDRs. BYOG uses a novel self-dechirping algorithm to detect preambles and estimate SFs without exhaustive matching, reducing the computational cost of decoding by 6× per channel. This approach democratizes gateway infrastructure, enabling low-cost, programmable, and easily deployable solutions.

Together, these three works CIC, Cloud-LoRa, and BYOG contribute to a layered, interoperable strategy for improving the scalability and throughput of LoRa networks. By integrating these innovations, this thesis advances LoRa network design, paving the way for more resilient, scalable, and high-throughput IoT deployments. Finally, I discuss future research directions in Chapter 7 and then finally conclude my thesis.

2 LORA BACKGROUND

LoRa uses Chirp Spread Spectrum (CSS) modulation, where the data symbols (each data symbol encodes multiple bits) are transmitted as chirps – signals with linearly increasing frequency. Use of chirps enables long distance communication and provides immunity against interference from other transmissions [12].

In this chapter, we provide the necessary background on LoRa PHY layer required in the rest of the thesis.

Fig. 2.1: LoRa packet format : every symbol is transmitted with a predetermined SF

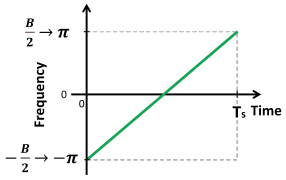


Fig. 2.2: Time-Frequency variation of base upchirp $C_0(t)$

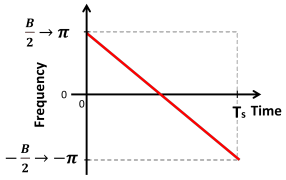


Fig. 2.3: Time-Frequency variation of downchirp $C_0^*(t)$

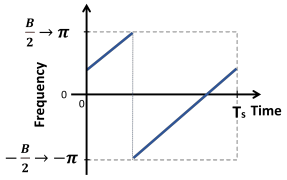


Fig. 2.4: Time-Frequency variation of Datachirp $C_\phi(t)$

Chirp Modulation in LoRa: LoRa uses chirp spread spectrum (CSS) modulation to communicate over long distances. CSS uses linear chirps, whose frequency varies with time, to represent a data symbol. A LoRa packet consists of a preamble of 8 consecutive upchirp (C_0) symbols, followed by two SYNC symbols (C_x, C_y ($x \neq 0, y = x + 8$)) and 2.25 down-chirps (C_0^*), as illustrated in Fig. 2.1. A base upchirp C_0 is that whose frequency increases linearly along with time from $-\frac{B}{2}$ to $\frac{B}{2}$ over a symbol duration T_s, where B is the bandwidth of transmission. An

upchirp is shown in the continuous chirp representation in Figure 2.2. A downchrip is the complex conjugate of an upchirp (Figure. 2.3). In LoRa, every data symbol C_ϕ is derived by shifting the fundamental symbol C_0 by a frequency f_ϕ.

$$C_\phi(t) = C_0(t) \cdot e^{j2\pi f_\phi t}; \tag{2.1}$$

$$C_0(t) = \begin{cases} e^{\,,0 \leqslant t \leqslant T_s} \\ 0, \text{otherwise} \end{cases} j2\pi \left(0.5\frac{B^2}{2^{SF}}t^2 - \frac{B}{2}t\right)\} \tag{2.2}$$

In Eqn 2.1, $\phi \in \{0, 1, \cdots, 2^{SF} - 1\}$ and $T_s = \frac{2^{SF}}{B}$. The spreading factor, $SF \in \{7, 8, 9, 10, 11, 12\}$, fixed for each LoRa packet, dictates the transmission data rate. Using a larger SF extends transmission range at the cost of using a lower data rate.

Demodulation Using De-chirping in Standard LoRa: To demodulate a symbol, the LoRa demodulator must estimate f_ϕ. A LoRa receiver first de-chirps a symbol by multiplying it with *down-chirp* $C_0^*(t)$, the complex conjugate of C_0, over a window aligned with the symbol's boundaries (Eqn 2.3). De-chirping converts the symbol into a sinusoid of constant frequency f_ϕ. f_ϕ is recovered by locating the peak in an FFT of the de-chirped symbol (Figure 4.1).

$$C_\phi(t)C_0^*(t) = e^{j2\pi f_\phi t} \tag{2.3}$$

$$\Phi(C_\phi(t)) = FFT(C_\phi(t)C_0^*(t)) = sinc(T_s(f - f_\phi)) \tag{2.4}$$

In Eqn 2.4, $\Phi(C_\phi(t))$ represents de-chirping and a 2^{SF} point FFT over a symbol $C_\phi(t)$. FFT accumulates the energy from the entire window into one FFT bin corresponding to the starting frequency of the data chirp; we get a peak at that frequency in FFT.

Carrier Frequency Offset (CFO): Carrier Frequency Offset (CFO),δf, is the small difference in generated carrier frequencies between the transmitter and the receiver due to manufacturing imperfections. CFO manifests

itself as a constant shift in the estimated symbol frequency so that the peak of a symbol C_ϕ would be located as $f_\phi + \delta f$. Consequently, receivers must estimate and subtract δf

Packet Detection using the LoRa Preamble: Before the receiver can begin demodulating, it must first reliably detect the onset of a new transmission, determine the exact positions of the boundaries of the symbols within the packet in order to perform de-chirping and FFT and, estimate f_ϕ. A preamble preceding the data symbols, facilitates all these functions. As shown in Figure. 2.1, the LoRa preamble comprises a sequence of 8 consecutive C_0 symbols, followed by two SYNC symbols C_x, C_y $(x \neq 0, y = x+8)$ and 2.25 down-chirps C_0^*. To detect a new transmission, the receiver continuously de-chirps and performs an FFT until it finds 8 consecutive peaks with the same frequency. The SYNC words and down-chirps then help locate the packet's symbol boundary positions, δf estimation and to confirm onset of a new packet.

LoRaWAN: LoRaWAN architecture, illustrated in Figure 6.1, consists of LoRa end device, LoRaWAN network server, and application server. The end devices perform modulation and demodulation as described above and take care of the physical layer processing. LoRaWAN provides medium access control (MAC) algorithms through various device classes (Class A, B, and C). LoRaWAN gateway receives the RF samples and relays them to the network server as IP packets through traditional backhaul. Multiple gateways can receive the data broadcast by an end device. These copies are used to improve reliability at the application server. In addition to the messages, metadata such as signal strength, timestamp are used by the network server to respond to the received data with acknowledgments, perform ADR, handling join requests, security, among other network activities.

Adaptive Data Rate: ADR is a mechanism for LoRaWAN gateways to optimize network throughput [13]. ADR allows end devices to change

their spreading factors and transmit powers based on the perceived channel quality. ADR is typically initiated by the network server. Every device begins with a default SF and transmit power. Upon receiving a request to update ADR settings, the network server provides feedback that indicates the appropriate SF for the channel quality. Devices closest to the gateway will be received with a high SNR and hence are instructed to operate at a low SF by the server. Since lower SF is more suitable for a shorter range and offers a higher data rate; by choosing lower SF for nearby devices, the overall network throughput can be improved. Packets from farther devices will be received with a lower SNR and hence assigned a higher SF so that packets can be received at the gateway. ADR is preferred for networks with stable RF conditions where the channel does not change often. In a dynamic and/or mobile network, ADR might be initiated by the end device. A key enabling factor for ADR is the capability of the LoRaWAN gateway to receive multiple SFs simultaneously. Commercial LoRaWAN gateways include a **preamble search engine** [14], as detailed in the patent for LoRaWAN gateway that searches through all SFs to determine the appropriate SF. This exhaustive search increases the cost of commercial gateways and hence 64 channel gateways are only available for carriers. Such an exhaustive search also limits real-time implementation on SDRs.

3 LITERATURE SURVEY

As one of the most widely adopted LPWAN technologies, LoRa has become a key enabler of long-range, low-power connectivity for a wide array of IoT applications. However, the increasing density and scale of LoRa deployments have exposed limitations in throughput and scalability, prompting significant research attention. Recent efforts have sought to leverage LoRa's unique PHY-layer characteristics and flexible network architecture to overcome these limitations.

Physical-Layer Collision Resolution Due to the uncoordinated and asynchronous nature of LoRa transmissions, packet collisions are a primary bottleneck for network throughput. Traditional LoRa receivers can decode only one packet per channel at a time, severely limiting scalability in dense networks. Early research leveraged LoRa's inherent robustness to interference to decode overlapping packets by grouping received symbols based on common features. Choir[15] pioneered this approach by grouping symbols based on carrier frequency offset (CFO) differences. FTrack[16] introduced time-frequency symbol tracking using Short-Time Fourier Transforms (STFT), sliding a fixed window over the de-chirped signal to identify symbol tracks. However, FTrack's accuracy degrades under low-SNR conditions and incurs computational overhead from symbol-by-symbol tracking. Later works, such as mLoRa[17] and CoLoRa[18], used power-based clustering to group symbols. mLoRa applied successive interference cancellation (SIC), iteratively assigning high-power symbols to a presumed transmitter and subtracting them from the composite signal. While SIC is effective for high-SNR scenarios, it leads to accumulation of symbol decoding errors in low SNR cases. NScale [19] focused on improving low-SNR performance through non-stationary scaling and matched symbol detection across frames. While these techniques improved performance under specific assumptions, they all fundamentally relied on

symbol grouping and transmitter fingerprinting strategies that struggle in highly dynamic or noisy environments. CIC (Concurrent Interference Cancellation), introduced in this thesis, breaks from this paradigm by avoiding explicit transmitter matching. CIC instead exploits both time and frequency resolution within a single LoRa symbol, using sub-symbol windows to isolate spectral components unaffected by collisions. Following CIC, several works have attempted to push LoRa's performance limits further. Pyramid[20] combines phase-correlation and interference rejection to improve packet recovery under collision. AlignTrack[21] uses temporal alignment techniques to enhance decoding of partially overlapped packets. NeLoRa [22] explores learning-based packet decoding to improve LoRa performance under weak signal conditions. However, these works are typically restricted to single-channel settings or operate best only in specific SNR regimes, limiting their generalization.

Cloud-Based LoRa Architectures (***CRAN for LPWAN***) As LoRa networks grow in density and scale, local processing at gateways becomes a bottleneck. The Cloud Radio Access Network (CRAN) architecture offers a powerful alternative by centralizing PHY-layer processing in the cloud, enabling flexible decoding, joint signal processing, and dynamic resource allocation. Initial CRAN-inspired LPWAN works include Charm[23] and OPR[24], which showed that cloud-side coherent combining can extend communication range and reliability. Nephalai [25] introduced a compressed sensing-based gateway to stream multiple LoRa channels to the cloud. However, it uses static compression, which is not well-suited for networks with variable SNR or backhaul capacity. Cloud-native frameworks such as SparSDR[26] and CharIoT[27] demonstrated the potential for offloading PHY tasks from edge devices to the cloud. However, these systems are not designed to handle sub-noise signals and do not adapt to the dynamic traffic and link conditions common in LoRa deployments. To fill this gap, Cloud-LoRa, introduced in this thesis, proposes a practi-

cal CRAN-based architecture for LoRa. Cloud-LoRa enables gateways to stream raw I/Q samples to the cloud using a reliable transport protocol (TCP BBR), where advanced decoding algorithms like CIC or BYOG can be deployed.

Gateway Capacity and Multi-Channel Scalability

While physical-layer techniques address collisions within a channel, scalability also requires expanding the number of channels a gateway can support. The LoRa standard defines up to 64 channels, but COTS gateways typically support only 8, due to the computational cost of decoding multiple spreading factors (SFs)/data-rates in parallel. Hou et al.[28] and Koch et al.[29] proposed multi-SF decoding schemes and scalable gateway architectures, but these often come at the expense of reduced SNR sensitivity or higher energy consumption. These approaches also struggle to support the real-time demands of high-throughput LoRa deployments. To address these issues, BYOG (Bring Your Own Gateway), introduced in this thesis, proposes a computationally lightweight and SF-agnostic decoding algorithm using general-purpose SDRs. The core innovation, self-dechirping, enables detection and decoding of LoRa packets without exhaustive SF-specific preamble matching. BYOG supports real-time decoding of 10 channels, matching standard LoRa performance while enabling broader scaling at lower cost and complexity.

Adaptive MAC Protocols and Network Coordination

Apart from PHY-layer enhancements, several works focus on improving medium access control (MAC) and scheduling to reduce collision probability and boost throughput. LoRa-TS[30] proposes a time-slot based scheduling mechanism to avoid concurrent transmissions. Free[31] and Burst-MAC[32] explore backscatter and lightweight beaconing mechanisms for coordinating LoRa transmissions without relying on centralized synchronization. LoFi[33] employs frequency diversity and preamble spreading to improve collision resilience under dense deployments.

LMAC [34] and BSMA [35] introduce a CSMA technique to resolve channel access for LoRa end devices. More recent systems such as FCA-LoRa[36] and MAB[37] adopt learning-based MAC protocols, dynamically adjusting transmission timing or SF assignments to balance load and reduce contention. While these efforts are promising, many require modifications at the end-device or assume centralized scheduling, which conflicts with LoRa's decentralized, low-power capability. The solutions proposed in this thesis, including CIC and BYOG, operate entirely at the gateway or backend, maintaining full compatibility with unmodified end devices.

4 RESOLVING PACKET COLLISIONS IN LORA

LoRa typically employs star topology where end devices transmit their messages to the central gateway. It uses an ALOHA-based Medium Access Control (MAC) protocol, allowing devices to transmit data whenever they have something to send, without coordination. This approach works effectively in networks with a small number of nodes. However, as the network scales and more devices are added, the likelihood of multiple nodes transmitting simultaneously increases. This results in packet collisions, which standard LoRa demodulators at the gateway cannot resolve. Consequently, the network experiences a significant degradation in throughput due to the increased rate of packet loss.

As discussed in Chapter 2, a standard LoRa receiver de-chirps each received symbol by multiplying it with a down-chirp, a chirp of linearly decreasing frequency. Fourier transform (FFT) of the de-chirped signal is then used to identify this frequency and hence the symbol. Figure 4.1

Fig. 4.1: Standard LoRa decoding under no collisions – each symbol maps to one unique frequency

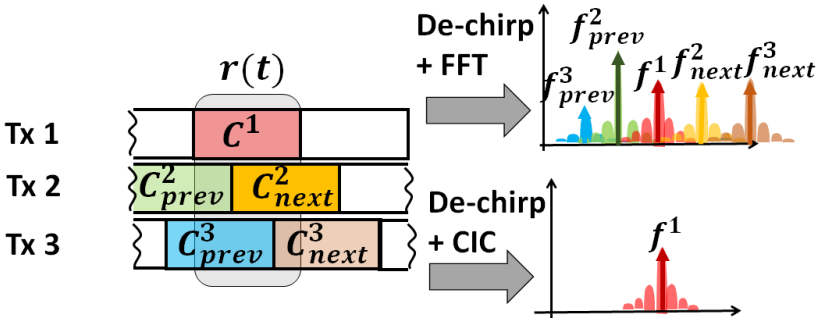


Fig. 4.2: Standard LoRa decoding under collisions results in multiple frequency peaks causing confusion. CIC removes these interfering frequency peaks.

illustrates the decoding of symbol C^1 of a collision-free transmission (Tx 1), whose de-chirping generates a sinusoid of frequency f^1. However, in a multi-packet collision, the received signal is a superposition of several interfering transmissions. De-chirping the received symbol $r(t)$, results in a superposition of multiple sinusiods (instead of one), each with a frequency corresponding to one of the interfering symbols. An FFT of this received symbol then results in a clutter of multiple frequencies. The standard LoRa receiver is unable to determine which of these frequencies corresponds to that of the symbol being decoded. Figure 4.2 illustrates this for a 3 packet collision scneario.

To handle such packet collisions in large scale LoRa networks, *in this chapter, I present a novel demodulation technique, Concurrent Interference Cancellation (CIC), that enables decoding of multiple colliding packets from COTS LoRa devices.* In the following section, we explore state-of-the-art techniques for resolving packet collisions in LoRa networks. Section 4.1 highlights the novel approach introduced by CIC for disentangling packet collisions and finally, we implement, evaluate and compare CIC against existing collision resolution techniques.

State-of-the-art techniques to resolve LoRa packet collisions

The approach taken by all existing works has been to match and group all discovered symbols to their corresponding transmitters based on certain features common to them. Choir [15] is a pioneering work in LoRa collision resolution with the goal of improving network throughput. It leverages the inherent hardware imperfections in the radio of LoRa transmitters and distinguishes colliding packets by uniquely mapping their imperfections to the transmitters. mLoRa [17] and CoLoRa [18], group symbols based on the similarity in their received power levels. FTrack [16] generates time-frequency tracks of the various symbols in the received signal by sliding a fixed window to generate Short Term Fourier Transforms (STFT) on the de-chirped signal. NScale [19] focuses on collision reso-

lution in low-SNR conditions using non-stationary scaling to match data symbols to packets. NScale achieves the symbol error rate of FTrack over a range of SNRs. While prior works focus exclusively on either time-domain or frequency-domain features to resolve LoRa packet collisions, CIC introduces a novel approach that combines the strengths of both, leveraging joint time-frequency analysis to achieve more effective and robust collision resolution.

4.1 LoRa Decoding Under Collisions

In this section we extend the discussion in Chapter 2 to a multi-packet collision scenario.

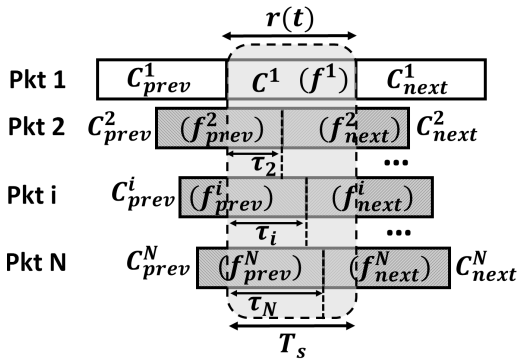


Fig. 4.3: Collision

LoRa Demodulation during a collision. During a packet collision, the received signal is a superposition of multiple LoRa receptions, each starting at a different time, with a different symbol boundary and at a different received power level. Fig. 4.3 illustrates the demodulation of a LoRa symbol C^1 from transmitter 1 during an N packet collision. Since symbol boundaries of colliding transmissions are not aligned, C^1 overlaps partially with two consecutive symbols from each interfering transmissions – C^i_{prev} and C^i_{next} from transmitters $2 \leqslant i \leqslant N$ corresponding to de-chirped frequencies f^i_{prev} and f^i_{next} respectively. τ_i depicts the time difference of symbol boundaries between 1^{st} and i^{th} transmission. We assume that the receiver has accurately determined the symbol boundaries as well as the CFO for transmitter 1 using preamble detection (as described in Section 4.2.8) δ^i represents the difference of the i^{th} transmit-

ter's CFO from that of transmitter 1. The received signal symbol $r(t)$ (with $t = 0$ at the start of C^1), during the collision is given by,

$$r(t) = A_1 C^1_{\phi(\tau)}(t) + I(t) \tag{4.1}$$

$$I(t) = \sum_{i=2}^{i=N} A_i e^{j2\pi\delta^i t} \begin{bmatrix} C^i_{prev}(t + T_s - \tau_i) W(\frac{t}{\tau_i}) \\ + C^i_{next}(t - \tau_i) W(\frac{t-\tau_i}{T_s-\tau_i}) \end{bmatrix} \tag{4.2}$$

$$W(t) = \begin{cases} 1 & \text{if} \quad 0 \leqslant t \leqslant 1 \\ 0 & otherwise \end{cases} \tag{4.3}$$

In Eqn 4.1, $I(t)$ is the interference. The window functions $W(\frac{t}{\tau_i})$ and $W(\frac{t-\tau_i}{T_s-\tau_i})$ represent the partial overlap regions of C^i_{prev} and C^i_{next} with C^1 and have widths $T_s - \tau_i$ and τ_i (Fig. 4.3) respectively. A_i is received signal amplitude from i^{th} transmitter. Demodulating $r(t)$ by de-chirping and FFT, results in $2(N-1)$ peaks corresponding to the $2(N-1)$ partially overlapping interfering symbols, C^i_{prev} and C^i_{next} and one peak corresponding to symbol C^1.

Fig. 4.10 depicts LoRa demodulation of 6 colliding transmissions (at SF=8) using COTS LoRa devices. Red circles indicate interfering transmissions while the green circle indicates the true peak. In Fig. 4.10, there are 3 peaks higher than the true peak corresponding to the symbol being decoded. This occurs because, the received powers of the interfering transmissions can be stronger than the one being decoded.

4.2 CIC

As described in Section 4.1, in the event of an N packet collision, the received signal of the symbol being decoded, $r(t)$ (Eqn 4.1), comprises a superposition of $2(N-1)$ interfering symbols in addition to the symbol of interest (C^1). Consequently, the de-chirped signal comprises $2N-1$ frequencies instead of one. CIC exploits temporal variations in spectral content of the symbol as interfering transmissions transition through their

respective symbols. *The key observation that drives the design of CIC is that none of the interfering symbols span the entire symbol duration while* C^1 *does.*

CIC selects a specific set of sub-symbols (parts of the symbol being decoded) such that none of the interfering symbols is common across all these sub-symbols. Such a set of sub-symbols is deemed the Interference Cancelling Sub-Symbol Set (ICSS). This means that the only common frequency across all sub-symbols in an ICSS will be C^1, as it is present in all sub-symbols of $r(t)$. By estimating spectra for all sub-symbols in the ICSS and extracting the common frequency, CIC removes all interfering symbols while retaining the frequency f^1 corresponding to C^1.

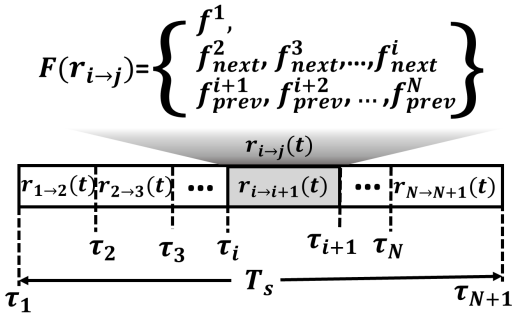


Fig. 4.4: CIC - Sub-Symbol Sections

Sub-Symbols. A sub-symbol $r_{i\to j}(t)$ is a part of the received symbol $r(t)$ that start at $t = \tau_i$ and ends at $t = \tau_j$. Recall that $t = \tau_i$ is the symbol boundary of the i^{th} transmission when it transitions from C^i_{prev} to C^i_{next} (Figure 4.3).

$$r_{i\to j}(t) = \left\{ \begin{array}{cc} r(t) & \text{if} \quad \tau_i < t < \tau_j \\ 0 & otherwise \end{array} \right| \tau_1 = 0, \tau_{N+1} = T_s \qquad (4.4)$$

As an example, Figure 4.4 depicts the set of sub-symbols $r_{i\to i+1}(t)$. The key property common to these sub-symbols is that each of the sub-symbols comprises exactly
$N-1$ interfering symbols $-\{C^2_{next},\cdots,C^i_{next}\}\cup\{C^{i+1}_{prev},\cdots,C^N_{prev}\}$. This is because, transmissions $2 \leqslant k \leqslant i$, have already transitioned from C^k_{prev} to C^k_{next} prior to $t = \tau_i$ and thus all of f^k_{prev} are absent and all f^k_{next} are present. On the other hand, transmissions $i < k \leqslant N$ have

not yet transitioned to transmitting the symbol C^k_{next}, thus for all these transmissions, all f^k_{next} are absent and f^k_{prev} are present.

Thus, the spectrum of their de-chirped versions $\Phi(r_{i\rightarrow i+1}(t))$ (Eqn 2.4), comprises a set of exactly the N frequencies $F(r_{i\rightarrow i+1}) = \{f^1\} \cup \{f^2_{next}, \cdots, f^i_{next}\} \cup \{f^{i+1}_{prev}, \cdots, f^N_{prev}\}$ (Figure 4.4). f^1, *the frequency of the symbol to be decoded will be present in all sub-symbols since it is present throughout* $r(t)$.

Interference Cancelling Sub-Symbol Set (ICSS). Consider the set of sub-symbols $\{r_{1\rightarrow 2}(t), r_{N\rightarrow N+1}(t)\}$. $F(r_{1\rightarrow 2}(t))$ comprises of the frequencies $\{f^1\} \cup \{f^1_{prev}, \cdots, f^N_{prev}\}$ as none of the symbols have transitioned to their next symbol. $F(r_{N\rightarrow N+1}(t))$ comprises of the frequencies $\{f^1\} \cup \{f^1_{next}, \cdots, f^N_{next}\}$ as all of the symbols have transitioned to their next symbol. Thus, no interfering symbol frequencies are common across this set of frequencies. Consequently, $\{r_{1\rightarrow 2}(t), r_{N\rightarrow N+1}(t)\}$ is an example of ICSS.

A Strawman-CIC. To provide an intuition into how CIC works, we consider a Strawman-CIC that uses the ICSS $\{r_{1\rightarrow 2}(t), r_{N\rightarrow N+1}(t)\}$. The only constituent frequency common to these two sections is f^1. Thus, f^1 can be extracted by estimating all the frequencies in $F(r_{1\rightarrow 2}(t))$ and $F(r_{N\rightarrow N+1}(t))$ and finding $F(r_{1\rightarrow 2}(t)) \cap F(r_{N\rightarrow N+1}(t))$ (Figure 4.5). As we discuss later in this section, Strawman-CIC's performance is adversely affected by loss of spectral resolution as dictated by Hiesenberg's Time-Frequency uncertainty principle.

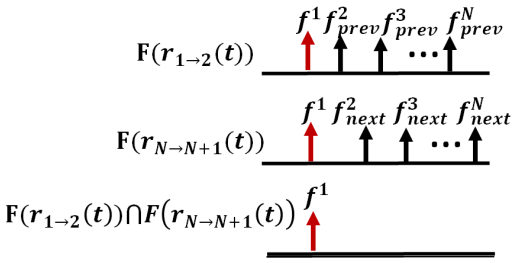


Fig. 4.5: Strawman CIC

4.2.1 Time-Frequency Uncertainty

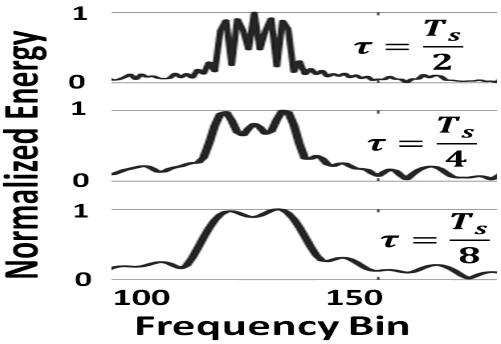


Fig. 4.6: Heisenberg's Principle

Heisenberg's time-frequency uncertainty principle states that estimating frequencies over signals with short time-spans result in poor frequency resolution. Specifically, estimating the frequency spectrum using a sub-symbol with a smaller time-span $\frac{T_s}{K}$, will reduce the spectral resolution to $\frac{B}{K}$ (B is the bandwidth of the signal and K is a constant), making it difficult to distinguish between two symbols whose frequencies are less than $\frac{B}{K}$ apart.

This is illustrated in Figure 4.6, which shows the spectrum estimated for a signal with 5 interfering symbols using progressively smaller sized time-spans τ. As seen in Figure 4.6, when τ is large ($T_s/2$), all the five peaks corresponding to the five symbols are distinct. However, as τ decreases, the frequency resolution of the spectrum decreases and the peaks merge into a single peak at $\tau = \frac{T_s}{8}$. *This loss of resolution adversely effects CIC, as the peaks in spectral estimates merge into one another when using sub-symbols with a short time-span.*

4.2.2 Spectral Intersection

CIC attempts to find the common frequency across all sub-symbols in an ICSS, each estimated with a different frequency resolution since their time-spans may be different. The process of extracting frequencies in spectra is achieved by detecting peaks in the spectra; it requires careful choice of thresholds to reject false peaks arising out of noise, without missing the right ones, and can be error prone. Finding constituent frequency peaks separately for each of the spectra in an ICSS and then set intersection results in errors incurred at each step to accumulate, leading to poor cancellation.

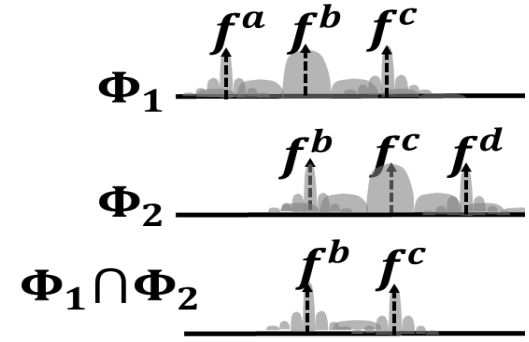


Fig. 4.7: Illustration of Property P2

Rather than extract peaks for each of the spectra separately, CIC first computes a Spectral Intersection by computing *the minimum energy across all the spectra at each frequency*. Spectral intersection is illustrated for Strawman-CIC in Figure 4.7, where, after taking the minimum energy in the spectra of Φ_1 and Φ_2 at each frequency, only peaks that are located at common frequencies f^b and f^c remain. Now peak detection needs to be performed only once to extract the common frequencies, making the process less error prone. Note that, prior to computing the intersection, all estimated spectra must be normalized to have unit energy to eliminate scaling effects due to different sized windows. In this paper, we shall use $\Phi_1 \cap \Phi_2$ to denote the spectral intersection operation using minimum.

Properties of Spectral Intersection. Spectral intersection has some key properties that CIC exploits in its design.

P1 : it is commutative and associative, as inherited from the Minimum operation.

P2 : when two spectra have different frequency resolutions, the operation preserves the higher resolution at each constituent frequency. The illustration in Figure 4.7 provides an intuition into property P2. In Figure 4.7, Φ_1 has a lower resolution estimate on frequency f^b but a higher resolution estimate for f^c while the vice-versa is true for Φ_2. Computing the spectral intersection takes the minimum at each frequency and hence preserves the best resolution for both frequencies.

4.2.3 Effect of Poor Frequency Resolution on Strawman-CIC

In order to motivate our design of CIC we start by showing how time-frequency uncertainty adversely effects Strawman-CIC. The time-spans of $r_{1\to2}(t)$ and $r_{N\to N+1}(t)$ are τ_1 and $T_s - \tau_N$ respectively. Assuming that the symbol start times of all colliding transmissions are uniformly distributed in the interval $(0, T_s)$, the expected values of τ_1 and $T_s - \tau_N$ are both $\frac{T_s}{N}$ *. Consequently, the corresponding spectral estimates have a frequency resolution of $\frac{B}{N}$, making it hard to separate two constituent frequencies that are within this resolution. Figure 4.8 illustrates the effect of this lower frequency resolution estimates on CIC, where energy from each frequency spills over in its neighborhood. Instead of a clean sharp peak as illustrated in Figure 4.5, the resulting spectrum comprises multiple wide interfering peaks in Figure 4.8, rendering cancellation ineffective.

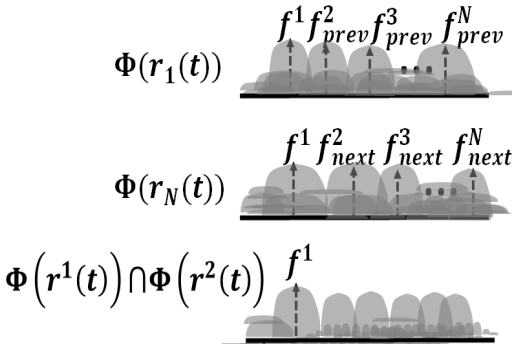


Fig. 4.8: Effect of poor resolution on Strawman-CIC

4.2.4 Design of CIC

CIC aims to pick an ICSS that best preserves the frequency resolution for each of the constituent frequencies. This choice not only preserves the maximum resolution for f^1, making it easy to extract its peak, but also aids in sharply canceling the other interfering peaks.

How to cancel a specific interfering transmission at the best possible resolution. An interfering frequency f^i_{prev} is present throughout the sec-

*The expectation of the minimum of N uniform random variables

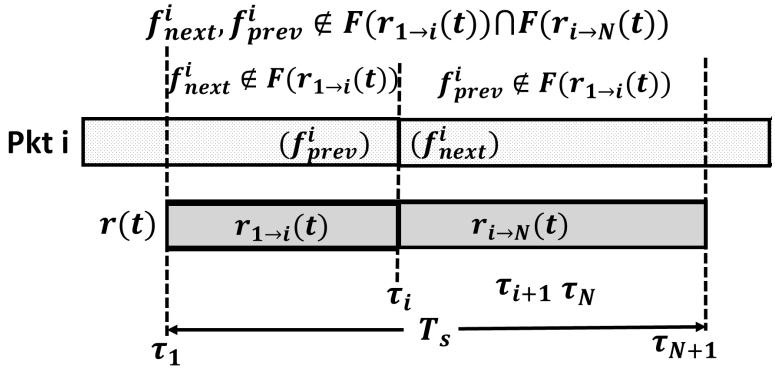


Fig. 4.9: Canceling a single interferer in CIC

tion $r_{1\to i}(t)$ - a time-span of τ_i. Thus, the maximum achievable frequency resolution for f^i_{prev} can be obtained in the spectrum estimate $\Phi(r_{1\to i}(t))$ is given by $B\frac{\tau_i}{T_s}$. Similarly, the best frequency resolution for f^i_{next} is achieved from the spectrum estimate $\Phi(r_{i\to N+1}(t))$ is given by $B\frac{T_s-\tau_i}{T_s}$.

As depicted in Figure 4.9, $F(r_{1\to i}(t))$ does not have f^i_{next} since the transmission only switches to f^i_{next} for $t > \tau_i$. Similarly, $F(r_{i\to N+1}(t))$ does not have f^i_{prev} since the the transmission has already switched to f^i_{next} at $t = \tau_i$. Thus, $F(r_{1\to i}(t)) \cap F(r_{i\to N+1}(t))$ will not have the frequencies f^i_{prev} and f^i_{next}. Further, $\Phi(r_{1\to i}(t))$ and $\Phi(r_{i\to N}(t))$ will also have the highest possible frequency resolutions for f^i_{prev} and f^i_{next}, following property P2 of spectral intersection, $\Phi(r_{1\to i}(t)) \cap \Phi(r_{i\to N+1}(t))$ *will remove* f^i_{prev} *and* f^i_{next} *at their respective maximum possible frequency resolutions*.

The Optimal Choice for ICSS. CIC constructs ICSS with $2N-1$ sub-symbols, comprising all pairs $r_{1\to i}(t), r_{i\to N+1}(t)$ for $2 \leqslant i \leqslant N$ and finally $r(t)$ itself. Each pair, $r_{1\to i}(t), r_{i\to N+1}(t)$ cancels the i^{th} transmission at their corresponding highest possible frequency resolutions. Following properties P1 and P2 of spectral intersection, computing a spectral intersection over the ICSS will cancel all frequencies at their highest possible frequency resolutions. Finally, the inclusion of $r(t)$ ensures that f^1 is recovered at the highest possible resolution. Thus, CIC computes the spectral intersection

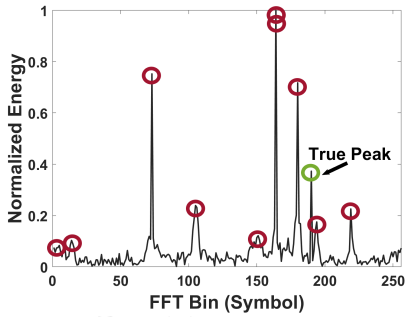


Fig. 4.10: Collision

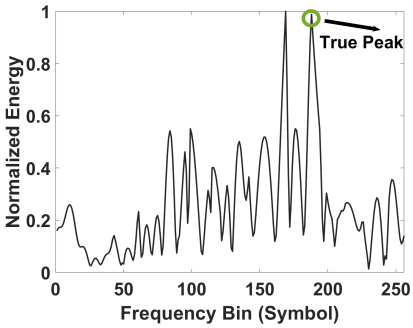


Fig. 4.11: Effect of loss of resolution in Strawman CIC

as,

$$\Phi_{CIC}(r(t)) = \left[\bigcap_{i=1}^{i=N} (\Phi(r_{1\to i}(t)) \cap \Phi(r_{i\to N+1}(t))\right] \cap \Phi(r(t)) \tag{4.5}$$

To provide an intuition into how CIC removes interfering symbols, Figures 4.10, 4.11,and 4.12 depicts demodulation of 6 colliding transmissions (at SF=8) using COTS LoRa devices using Standard LoRa, Strawman-CIC, and CIC respectively. Red circles indicate interfering transmissions while the green circle indicates the true peak. In Figure 4.10, four interfering peaks have higher energy than the true peak corresponding to the symbol being decoded. This occurs because, the received powers of the interfering transmissions can be stronger than the one being decoded. While Strawman-CIC does eliminate some of interfering symbols, it suffers due to lack of frequency resolution (Figure 4.11). However, CIC is able to extract the correct frequency while preserving the

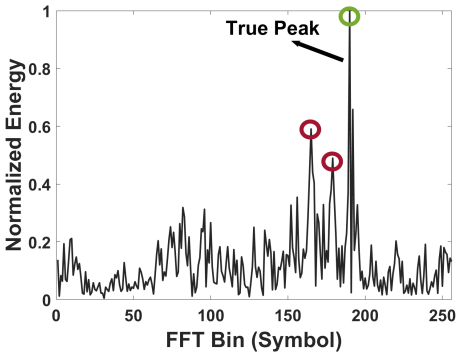


Fig. 4.12: Interference Cancellation in CIC

highest frequency resolution.

4.2.5 Extent of CIC Cancellation

In this section we try and answer the question, "How much cancellation can CIC offer and what factors does it depend on?" The conjunction of two events make cancellation hard. First, an interfering symbol overlaps to a large extent with the symbol of interest ($Min(\tau_i, T_s - \tau_i)$ is small). Second, the symbols' chirp start frequencies are very close. Thus, the combination of both close time proximity $\Delta\tau$ and frequency proximity Δf of an interfering symbol adversely effects cancellation. For a symbol with start chirp frequency f1 We define $\Delta\tau$ and Δf, from a symbol of frequency f as,

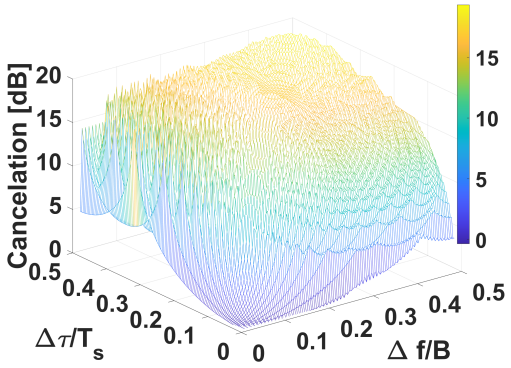


Fig. 4.13: Cancellation in CIC

$$\Delta\tau = \begin{matrix} \tau_i & \text{for} & C^i_{next} \\ T_s - \tau_i & \text{for} & C^i_{prev} \end{matrix} \tag{4.6}$$

$$\Delta f = \begin{matrix} |f^1 - f^i_{prev}| & \text{for} & C^i_{prev} \\ |f^1 - f^i_{next}| & \text{for} & C^i_{next} \end{matrix} \tag{4.7}$$

The extent of cancellation for CIC can be analytically computed, however, we avoid presenting it due to lack of space. Figure 4.13 depicts the extent to which CIC can cancel a particular symbol as a function of $\Delta\tau$ and Δf for $SF = 8$. As seen from Figure 4.13, while the cancellation can be as high as 20dB when $\frac{\Delta\tau}{T_s} = \frac{\Delta_f}{B} = 0.5$, there is almost no cancellation when both these values are close to 0. The cancellation increases to 5dB by the time these

values reach 0.1. This shows that even after CIC, one or more peaks may only be partially cancelled and in case of a strong interfering transmission, they may be hard to discard.

4.2.6 Spectral Edge Difference (SED)

CIC may only be able to cancel certain symbols partially when $\Delta\tau$ and Δf are small. This can be seen in Figure 4.12, where while the peak corresponding to f^1 is the significant highest peak, some of the interfering peaks with high transmit powers (indicated by red circles) remain. In such a case, CIC has more than one potential candidate and CIC must pick one among them. For this, CIC computes the Spectral Edge Difference (SED), the absolute difference in energies of the candidate frequency spectra between left and right halves of $r(t)$, for each candidate frequency. It then picks the frequency with the least value of SED. The key intuition behind computing SED is that, SED will be zero only for f^1 since unlike interfering symbols, this frequency exists uniformly across the entire symbol.

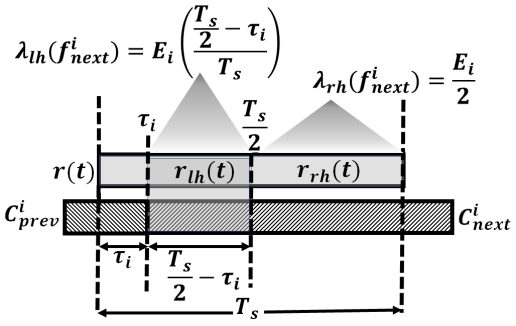


Fig. 4.14: SED Illustration

We illustrate the intuition into SED in Figure 4.14 for the symbol C^i_{next}. Let E_i be the total energy per symbol in the i^{th} packet (based on the received signal strength). $r_{lh}(t)$ and $r_{rh}(t)$ represent the left and right halves of $r(t)$. Since the total duration that C^i_{next} is present in $r_{lh(t)}$ is $\frac{T_s}{2} - \tau_i$, the energy of the peak corresponding to f^i_{next} denoted by $\lambda_{lh}(f^i_{next}) = E_i\left(\frac{\frac{T_s}{2}-\tau_i}{T_s}\right) = \frac{1}{2}E_i\left(1-\frac{\tau_i}{T_s}\right)$. f^i_{next} is continuously present in the entire right half $r_{rh}(t)$, consequently, the energy of the peak in this half, $\lambda_{rh}(f^i_{next}) = \frac{E_i}{2}$. The SED $\Lambda(f^i_{next})$ for this frequency is given by

$|\lambda_{rh}(f^i_{next}) - \lambda_{lh}(f^i_{next})| = \frac{1}{2}E_i\frac{\tau_i}{T_s}$. Since, f^1 is present in both halves completely, $\lambda_{rh}(f^1) = \lambda_{lh}(f^1) = \frac{1}{2}E^1$ and $\Lambda(f^1) = 0$. SED exploits this difference and picks the candidate with the least $\Lambda(f)$.

To make the SED estimate robust, instead of relying only on a single pair (left and right), in practice, CIC uses multiple sliding windows of span ($\frac{T_s}{2}$) over the signal (10 in our implementation) from the left and right ends of the symbol and computes their spectral intersection.

$$\Lambda(f) = |\lambda_{rh}(f) - \lambda_{lh}(f)| \tag{4.8}$$

$$\lambda_{lh}(f) = \bigcap_{i=1}^{i=n} \Phi\left(r(t).W\left(\frac{2(t - i\epsilon)}{T_s}\right)\right) \tag{4.9}$$

$$\lambda_{rh}(f) = \bigcap_{i=1}^{i=n} \Phi\left(r(t).W\left(\frac{2(t + i\epsilon - \frac{T_s}{2})}{T_s}\right)\right) \tag{4.10}$$

In Eqn 4.9 and 4.10, $W(t)$ is the rectangular window function as defined in Eqn 4.3.

4.2.7 Using Additional Features in CIC

Prior work has exploited the uniqueness of features such as CFO [15] and received power [18] for each packet to group symbols. These features can also be used to pick out the appropriate symbol in the event of multiple candidates. We estimate CFO and RSSI from the preambles (as described in Section 4.2.8) and use these as additional features to filter out partially cancelled interfering symbols. In our implementation, we use the technique in Choir [15] to estimate the fractional CFO for each of the candidate symbols and eliminate symbols with fractional-CFO-error more than $\frac{B}{4*SF}$. We use a size 16× FFT instead of 256× FFT since we find that is more computationally efficient without sacrificing on performance. Similarly, we also filter symbols whose received power deviates by more than 3dB from the estimated value in the preamble. In our evaluations, we examine

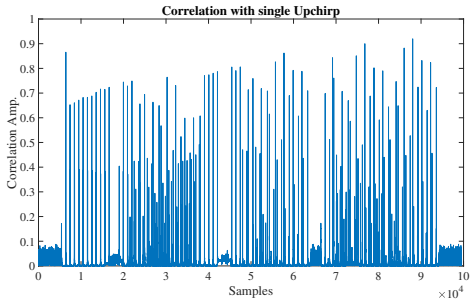


Fig. 4.15: Upchirp based preamble detection

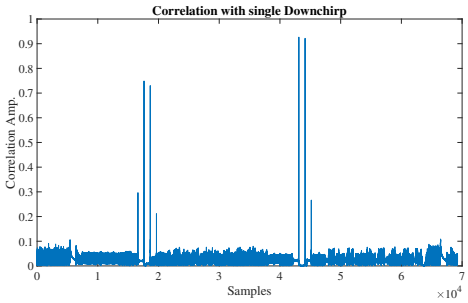


Fig. 4.16: DownChirp based preamble detection

the improvements due to each of these features compared to basic CIC.

4.2.8 Down-Chirp Based Preamble Detection

As described in Chapter 2, preamble detection in LoRa exploits its repetitive structure and searches for a sequence of 8 consecutive C_0 symbols. A key challenge in employing this scheme in the event of collisions, arises from the fact that all data symbols, as well as SYNC symbols in LoRa are merely frequency shifted versions of C_0. Consequently, data symbols from ongoing concurrent transmissions interfere with preambles, creating a clutter of peaks resulting in preamble detection errors. To provide an insight into this clutter, Figure 4.15 depicts the clutter of peaks during preamble detection of a new packet due to ongoing concurrent transmissions.

In our implementation, we take a different approach of searching for the two down-chirps in the preamble instead. The key insight that drives this choice, is that down-chirps do not correlate with C_0 and consequently do not correlate with data symbols in ongoing concurrent transmissions. Thus, to detect preambles we correlate with the down-chirp C_0^* instead and look for two consecutive peaks. Figure 4.16 depicts the peaks as detected by using down-chirps. Comparing Figures 4.15 and 4.16, using down-chirps significantly clears the clutter of peaks. Having found the two down-chirps, to confirm the preamble, we detect a preceding sequence

of 8 C_0s and two SYNC words by employing an up-chirp as with standard preamble detection. This approach besides improving preamble detection compared to existing methods, also reduces the computational complexity of considering all peaks arising out of data symbols from ongoing interfering transmissions.

Estimation of CFO and Received Power. As discussed in Section 4.2.7, performance of CIC improves by using CFO and received power to filter partially cancelled symbols. In order to enable this, for each detected preamble, we estimate CFO (as in Choir [15]) by averaging over all the preamble up-chirp symbols for a robust estimate and maintain a list of CFOs for all ongoing transmissions. Similarly, we also estimate the FFT peak height for each preamble up-chirp symbol and average across them for a robust estimate. These peak heights are also maintained in a similar manner as CFOs.

4.3 Implementation

CIC is implemented at the LoRa gateways and does not require any changes to the COTS LoRa sensor devices. We envision CIC as either being co-located with a Software Defined Radio-based gateway at the edge or as a virtual gateway in the cloud in case of a C-RAN [38] architecture. In general, a LoRa receiver comprises three separable parts – a radio front end, a demodulator, and a decoder as depicted in Figure 4.21. The radio front end receives radio waves and converts them into raw digital baseband samples.

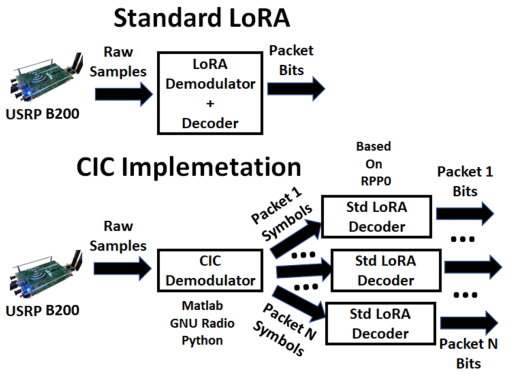
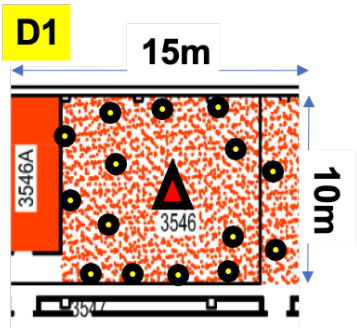
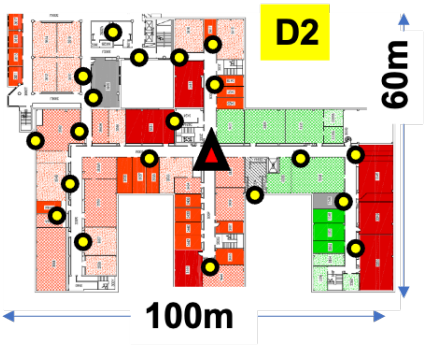


Fig. 4.21: Our Implementation





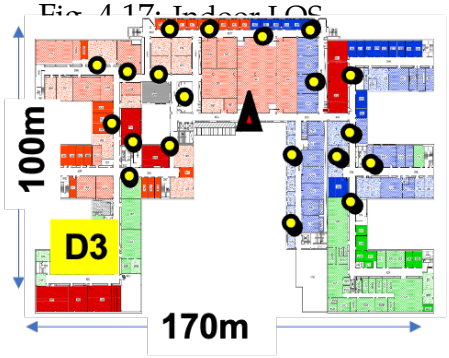


Fig. 4.19: Indoor NLOS(larger)

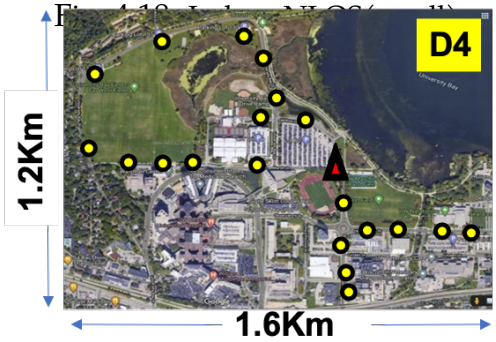


Fig. 4.20: Outdoor

The demodulator is responsible for preamble detection and converting the raw samples into LoRa symbols. Finally, the decoder maps the obtained LoRa symbols to bits based on the LoRa standard specification for deinterleaving, Forward Error Correction(FEC), and Cyclic Redundancy Check(CRC).

The Radio Front End. We used USRP B200 [39] as our radio front end at 2MHz bandwidth. Typically the received signal is oversampled i.e the sampling rate employed is significantly higher than the required Nyquist rate (4-10×) to allow better averaging. Since our COTS devices were configured at 250 KHz bandwidth, we have an oversampling of 8×. The received samples are then input to the demodulator.

Need for Distinct Demodulator and Decoder Modules. CIC replaces the standard LoRa demodulator for preamble detection and converting samples to symbols. Since standard LoRa demodulators expect to receive only

one packet at a time, the demodulator and decoder are usually integrated into a single implementation. Unlike standard LoRa, CIC however, can process concurrent transmissions and thus, generate multiple streams of symbols, one for each packet simultaneously. This means that multiple decoders might be needed to concurrently process the output of a single sample stream. Consequently, we provide separate implementations for a demodulator and a decoder.

Demodulator. We have implemented CIC demodulator in three different environments – Matlab, GNU Radio [40] and Python. Matlab is often the first choice for a large number of communication researchers as it allows quick trials, modifications, simulations, and experimentation to gain experience. For experimental deployments and trials, GNU Radio is a popular choice, as it allows for quick configuration of the receiver through a GUI. We have implemented CIC demodulator as a GNU Radio block. Finally, our python implementation is useful for practical deployments in the cloud as a C-RAN module or at the gateway edge. We also provide data sets collected in our experiments for testing and verification.

Decoder. Since the decoder needs to be LoRa compliant, we modified rppo/gr-lora [41], a popular, open-source GNURadio block for LoRa reception. Since demodulation and decoding are integrated in rpp0, we extracted the decoder C++ code and created a separate GNU Radio module for the decoder that takes symbols as input, and outputs bits. This allows researchers to mix and match decoders with different demodulators and decode multiple packets concurrently.

LoRa Devices.We used the commercially available LoRa transmitters – Adafruit Feather M0 with RFM 95 [42]. These devices allow us to configure various transmission parameters such as Spreading Factor (SF), Bandwidth (BW), Coding Rate using Arduino Library RadioHead [43].

4.4 Evaluation

In this section we evaluate and demonstrate the efficacy of CIC. We answer the following questions.

- How does CIC improve network capacity compared to standard LoRa as well as state of the art techniques?
- How does CIC perform in various deployments including indoor/outdoor, low/high noise, and Los/NLoS scenarios?
- How does employing down-chirp based preamble detection improve packet detection over conventional preamble detection techniques?
- How do various additional discriminating features such as CFO, and Received Power improve CIC performance?
- How does temporal proximity of packet collisions effect CIC's ability to cancel inteference?

4.4.1 Deployments and Experimental Setup

To test CIC under varying conditions such as high/low SNR, indoor/outdoor, LoS/NLoS, we evaluated CIC in four different test deployments as described below. Each deployment comprised 20 LoRa devices (depicted as circles) and a gateway (depicted as a triangle).

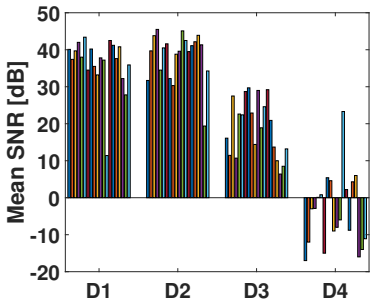


Fig. 4.22: SNR Distribution for each of the deployments

D1: Small Indoor Space - High SNR, LoS All state of the art techniques perform their best in High SNR

and LoS scenarios. To provide the best benefit to state of the art, we deployed 20 LoRa nodes within a large laboratory (Fig. 4.17,Fig. 4.23). The received SNR from the devices was approximately 30-40dB as seen in Fig. 4.22 and they were all in Line of Sight.

D2: Small Floor Space - High SNR, NLoS Next we evaluate CIC across the floor of an indoor space (Fig. 4.18). The nodes were stuck 6 feet high on the walls, some inside rooms and others outside. Here the received SNR was also between 30-40dB (Fig. 4.22), however many of the devices did not enjoy a direct line of sight to the gateway.

D3: Large Floor Space - Low SNR, NLoS While many state of the art techniques flounder in low SNR scenarios, CIC continues to perform well. In this deployment we chose a large floor space (Fig. 4.19). There was significant variation in received SNRs across the various devices ranging between 5dB to 30dB (Fig. 4.22) and most devices had no line of sight. This deployment is representative of a realistic large scale indoor deployment.

Fig. 4.23: Nodes in Indoor Deployments

D4: Outdoor Wide Area Deployment - SubNoise, NLoS In order to test the performance of CIC in a practical wide area outdoor scenario, motivated by the smart street lighting application, we deployed our LoRa devices on street lights over an area of 2 Sq. Km in an urban environment, as depicted in Fig. 4.20. Most of our packet receptions in this deployment were below the noise floor and signal strength fluctuations were common as pedestrians and traffic passed by (Fig. 4.22). Consequently, CIC was tested to its utmost in this deployment.

Traffic Generation and Experimental Methodology. IoT traffic is gen-

erated in response to unpredictable random physical events e.g. cars arriving at a parking lot often modeled as Poisson [44, 45] arrivals. Consequently, in our experiments, devices were configured to generate packets with exponentially distributed intervals. Each sensor node generates an exponentially distributed random variable ΔT ($pdf(\Delta T) = \mu e^{-\mu \Delta T}$) to determine the time interval for transmitting the next packet. In order to generate Poisson traffic in the network with an aggregate rate of R packets/second, we choose $\lambda = \frac{R}{20}$ since we had 20 nodes in each of our deployments. To record the actual number of packets transmitted, we recorded the transmissions at each node. Finally, the number of correctly received packets (based on all bits being correct) measures the network throughput.

Each device is configured to transmit packet with 28 bytes of a randomly generated data packet at SF=8, BW=250KHz, 4/5 coding rate, lasting a duration of 45ms. These choices are anecdotally, the most popular in LoRa deployments. Thus, a single device could transmit a maximum of 22 packets each second back-to-back. We increased the aggregate rate R from 5 Pkts/sec to 100 Pkts/sec by changing λ from 0.25 Pkts/sec to 5 Pkts/sec to measure network capacity in each experiment. A USRP B200 was used to collect received samples at 2 MHz sampling rate providing us $8\times$ oversampling. In each deployment, packets were transmitted at each rate for a duration of 1 minute. Thus, at the highest rate of 100 Pks/sec a total of 6000 packets were transmitted, while at the lowest rate of 5 packets/sec, 300 were transmitted.

Comparison with State of the art. We compared the performance of CIC with standard LoRa as a baseline, and two popular state of the art in research – Choir [15] and FTrack [16]. Choir is probably the first significant effort towards multi-packet collision decoding in LoRa. To the best of our knowledge, FTrack has the best performance of all existing literature. We thank the authors of FTrack for providing us their implementation and

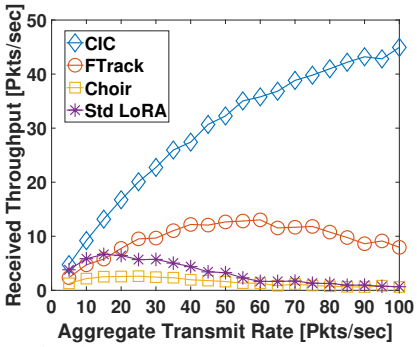


Fig. 4.24: Network Capacity for D1 (High SNR, LoS)

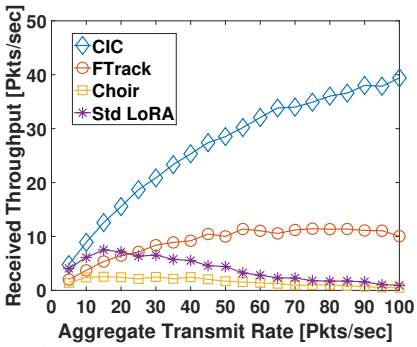


Fig. 4.25: Network Capacity for D2 (High SNR, NLoS)

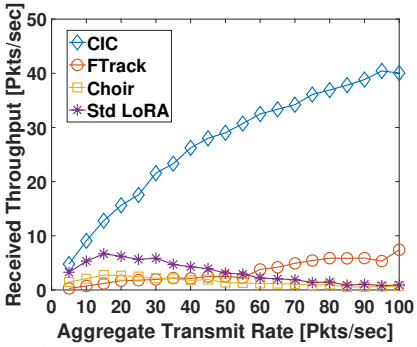


Fig. 4.26: Network Capacity for D3 (Low SNR, NLoS)

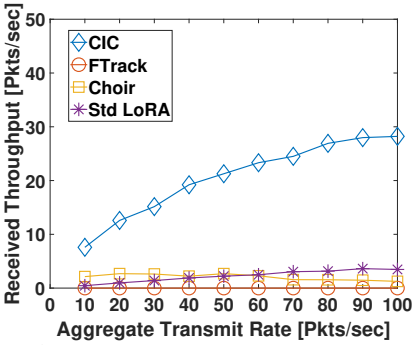


Fig. 4.27: Network Capacity for D4 (Outdoor, SubNoise SNR, NLoS)

supporting us. We implemented Choir based on the description in the paper.

4.4.2 Network Throughput

Figs. 4.24, 4.25, 4.26, 4.27 depict the number of successfully received packets per second as the aggregate network traffic increases from 5 Pkts/sec to 100 Pks/sec. Note that since each packet lasts 45ms in our deployment allowing for a maximum of 22 Pkts/sec, if a single node were transmitting back-to-back, the maximum rate of 100 Pkts/sec is 5× greater than what any single node could transmit.

D1 : High SNR, LoS. This scenario gives the best benefit of doubt to all

schemes and establishes the limits of their performance. As seen from Fig. 4.24, CIC significantly outperforms FTrack (by 4×), standard LoRa (by 5×) and Choir. CIC is able to decode 45 Pkts/sec, 2× greater than 22 Pkts/sec (the maximum possible for any single node) when the aggregate network traffic is 100 Pkts/sec. Standard LoRa is able to achieve a highest throughput of about 8 Pkts/sec, roughly one third of 22 Pkts/sec (the maximum). FTrack outperforms Standard LoRa, achieving about 12 Pkts/sec when the aggregate rate is 50 Pkts/sec (25% of the offered load). However, at aggregate network rates greater than 50 Pkts/sec, its performance degrades; this is due to the increase in collisions, which in turn leads to higher chance of collisions where majority of the packets overlap. In such scenarios, FTRack fails to distinguish between the corresponding frequency tracks due to its poor frequency resolution.

D2,D3 : High/Low SNR, NLoS. In this scenario as well, CIC significantly outperforms FTrack, LoRa as well as Choir by 4×. CIC is able to receive about 40 Pkts/sec, slightly less than the high SNR, LoS scenario. Standard LoRa's performance is consistent and almost the same as that of high SNR scenario as it successfully captures the higher SNR packets in case of a collision. As the authors of FTrack themselves claim, Ftrack fails to detect packets with low SNR, especially in the presence of stronger transmitters. Consequently, FTrack's performance degrades in the low SNR scenario.

D4 : Wide Area Deployment - SubNoise, NLoS. This deployment stress tests every scheme the most as the received SNRs are below noise levels. CIC's performance really shines in this regime providing almost 10× the throughput of standard LoRa. As expected FTrack is unable to decode at these SNRs and completely fails. Choir and Standard LoRa suffer heavy packet losses due to low SNR as well as collisions. Curiously, the net throughput of LoRa increases slightly at higher aggregate rates. This is because, the gateway successfully captures more packets with higher signal strengths and most of these packets are from a small subset of

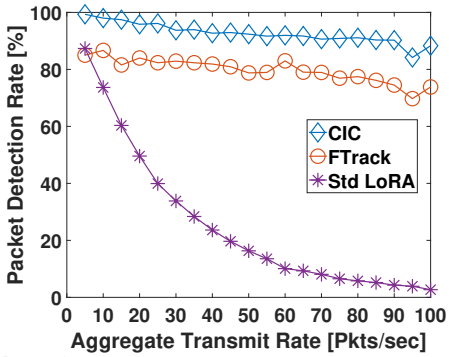


Fig. 4.28: Packet Detection : D1 (High SNR, LoS)

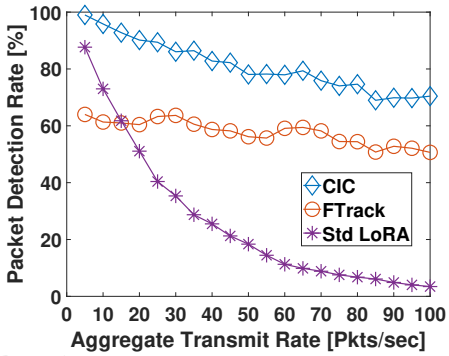


Fig. 4.29: Packet Detection : D2 (High SNR, NLoS)

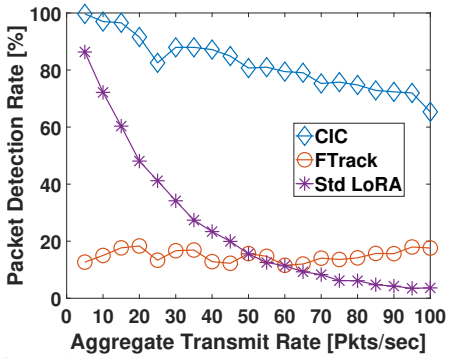


Fig. 4.30: Packet Detection : D3 (Low SNR, NLoS)

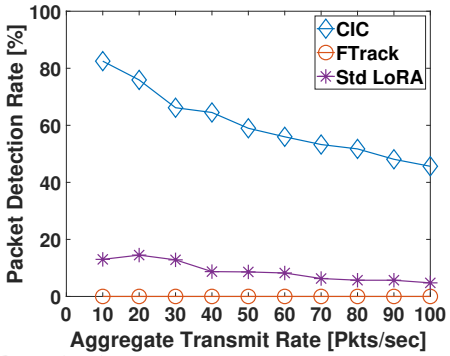


Fig. 4.31: Packet Detection : D4 (Outdoor, SubNoise SNR, NLoS)

transmitters whose aggregate rate also increases proportionally.

Conclusion. As seen from the above experiments, CIC outperforms FTrack as well as other schemes significantly ($4\times$), especially in wide area deployments where received signals strengths can be below noise levels, where it achieves about $10\times$ gains. This superior performance of CIC stems from its interference cancellation mechanism guided by the Heisenberg's Time-Frequency uncertainty principle.

4.4.3 Preamble Detection Accuracy

Packet detection using preambles is the first and most crucial step to decoding a packet. As discussed in Section 4.2.8, CIC modifies the commonly

used preamble detection using up-chirps to using down-chirps. In this section we ask the question "How does CIC's packet detection perform under packet collision scenarios and compare against the conventional approaches?" Figs. 4.28, 4.29, 4.30, 4.31, show the packet detection rate, the ratio of the number of packets detection (not necessarily correctly decoded) to the total number of packets transmitted, for each of the deployments and aggregate transmit rates. We compare the detection performance of CIC to that of FTrack and Standard LoRa. We are unable to compare the preamble detection of Choir since the authors do not describe their preamble detection in their paper; in the implementation of Choir, we therefore assume standard LoRa-based packet detection for Choir.

D1: High SNR, LoS. As seen in Fig. 4.28, CIC outperforms FTrack by a margin of about 20% steadily as the aggregate network traffic (hence packet collision rate) increases. Standard LoRa's packet detection quickly suffers and degrades with aggregate packets transmitted in the network.

D2: High SNR, NLoS. As SNR decreases, the packet detection rates of both CIC and FTrack suffer, however, CIC still performs better with a margin of over 20%.

D3: Low SNR, NLoS. As SNR decreases further, the packet detection rates of FTrack completely flounders and in fact falls below that of standard LoRa for high aggregate network traffic scenarios. CIC however, offers close to 80% detection even at very high aggregate network traffic scenarios.

D4: SubNoise SNR, NLoS. In this deployment, FTrack is simply unable to detect packets while standard LoRa has a detection rate of about 5%. While CIC's preamble detection performance decreases, it still offers up to 80% in low traffic and 50% in very high traffic scenarios.

Conclusions. Based on this analysis, using down-chirps as the first step for packet detection significantly improves packet detection rate under collisions and especially in low and sub-noise SNR scenarios that are common to both indoor and outdoor LoRa deployments in practice.

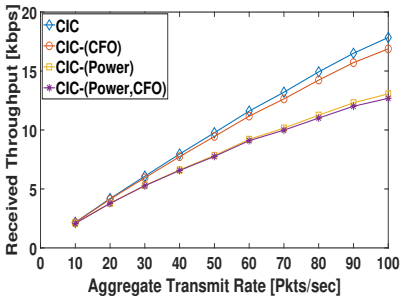


Fig. 4.32: Effect of Removing Various Features from CIC for D1

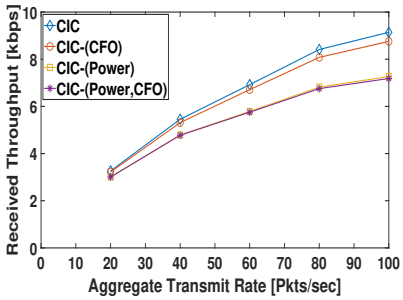


Fig. 4.33: Effect of Removing Various Features from CIC for D4

4.4.4 Effect of the additional features of CIC

As discussed in Section 4, existing works relies on grouping symbols from the same transmitter by exploiting various discriminating features that are distinct to transmitters such as received power and CFO. As discussed in Section 4.2.7, CIC makes use of two additional features – Received Power and Carrier Frequency Offset (CFO), to filter out candidate frequencies, when CIC is unable to cancel them sufficiently. In this section we ask the question, *How much do these additional features contribute to the overall performance of CIC?* To this end, we use four different versions of CIC.

- *CIC* : Implementation of CIC with both Received Power and CFO included as discriminating features.
- *CIC-(CFO)* :Implementation of CIC with only Received Power i.e. without CFO as a discriminating feature.
- *CIC-(Power)* : Implementation of CIC with only CFO i.e without Received Power as additional feature.
- *CIC-(Power, CFO)* : Implementation of CIC without either Received Power or CFO as discriminating features.

Figs. 4.32 and 4.33 depicts the aggregate network throughput obtained for each of these options in deployments D1 and D4 respectively. D1

(High SNR, LoS) represents the easiest scenario for CIC and D4 (Outdoor, SubNoise SNR, NLoS) the hardest. Thus, these two represent the two extreme cases.

D1. As seen in Fig. 4.32, CIC gains by about 20% using both Received Power and CFO as discriminating features. However, most of these gains are due to the Received Power feature rather than CFO. Using CFO helps CIC marginally, about 2%, whereas using Received Power helps CIC by almost 18%.

D4. As seen from Fig. 4.33, even though the net achieved throughput is lower, the relative gains (in %) due to each of these features is almost the same.

Conclusion: Received power used as a feature helps CIC the most and provides close to 18% gains. While CFO also assists CIC, it does so rather modestly by about 1-2%.

4.4.5 Effect of Temporally Close Collisions

As discussed in Section 4.2.5, CIC can effectively cancel transmissions whose symbol boundaries are far apart. In this section, we try and understand how CIC is affected by the proximity of the interfering symbol boundaries. Since synchronizing two COTS LoRa devices to transmit within sample level accuracies is hard, we rely on simulations.

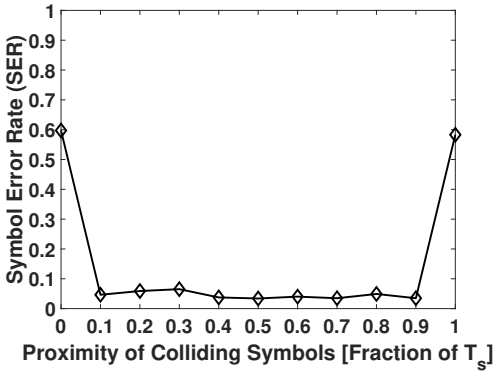


Fig. 4.34: Simulation study of CIC as two packets collide closer in time.

In our simulations, we generate packets with random bits and generate raw signals as a LoRa transmitter would. Then we superimpose two such packets with varying sub-

symbol time offsets (< 1ms). We increase time offsets in increments of 10% of the symbol ($\approx 100\mu$s). Then we use our implementation of CIC to recover all the symbols in each packet and measure the symbol error rate. In order to avoid effects due to noise, the signals are generated at 30dB SNR. Fig. 4.34 depicts the dependence of symbol error rate (SER) as a function of inter-symbol separation $\Delta\tau$ as a fraction of the symbol time T_s. As seen in the figure, CIC is able to cancel efficiently for $\frac{\Delta\tau}{T_s} > 0.1$. As the two colliding packets are closer than 10% of the symbol time, CIC starts to experience high SER.

5 DESIGNING A PRACTICAL CLOUD RADIO ACCESS NETWORK (CRAN) FOR LORA

While CIC gives a significant boost in the throughput, its real-world deployment faces a major hindrance. Commercially available LoRa gateways have a very hardware-centric design. The inflexible receiver chains baked into their ASICs can only run standard LoRa demodulation and decoding. As a result, Several promising PHY-layer demodulation techniques [16, 19, 15, 46, 47, 21, 48, 49] that show an order of magnitude improvement in throughput or range, remain largely confined to research and are not adopted in practice. Moreover, another approach to address the challenge of capacity scaling in dense areas is adding more number of radio channels, allowing devices to operate on less congested frequencies. However, COTS LoRa gateways support a fixed and limited number of channels, typically 4 or 8. As capacity needs increase, traditional LoRa gateways need to be physically upgraded to high-end gateways with parallel receiver chains to support more channels which is not economically viable for large scale LoRa networks.

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. 5.1). CRANs applied to LoRa, offer a departure from the rigid, hardware-centric design of COTS gateways. By offloading physical-layer processing to the cloud, CRANs enable rapid deployment and A/B testing of promising PHY-layer demodulation techniques like CIC as virtual software receivers. Moreover, CRAN gateways capture a wide radio spectrum and allow for the potential to dynamically scale the number of channels in the cloud depending on network demands. Additionally,

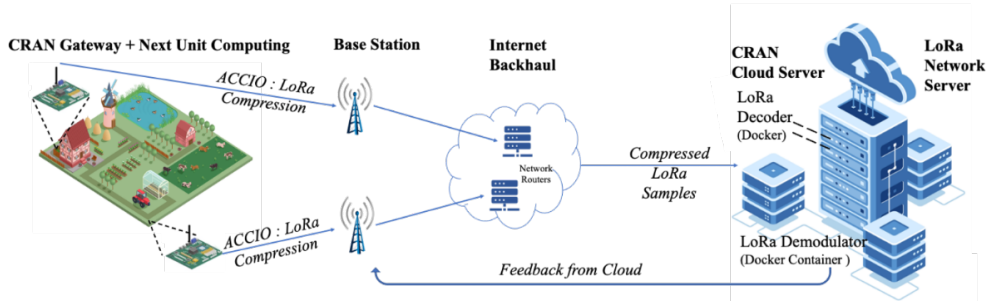


Fig. 5.1: An Illustration of CRAN and Cloud-LoRa

since CRANs offload gateway processing to the cloud, in the cloud their is an opportunity to coherently combine weak signals across multiple gateways to improve SNR and therefore, improve overall range of communication. However, deploying a practical CRAN for LoRa presents several real-world challenges, which I address in this chapter. I propose **Cloud-LoRa**, the first end-to-end practically-deployable LoRa CRAN for urban and rural deployments. In the following sections, I outline the key challenges faced in real-world deployments, present a design to overcome these challenges, and discuss the implementation and evaluation of our Cloud-LoRa system.

Variable Backhaul Bandwidths: 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* to the cloud. A CRAN gateway designed to match an 8 channel off-the-shelf LoRa gateway [50] will generate a continuous 128 Mbps sample stream. While access to broadband connectivity has been expanding, available bandwidths still vary vastly across the country [51, 52]. FCC 2020 reports on broadband access determined that potentially over 50% of rural Americans lack broadband access [53, 54] 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 [55, 56, 57]. Addi-

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

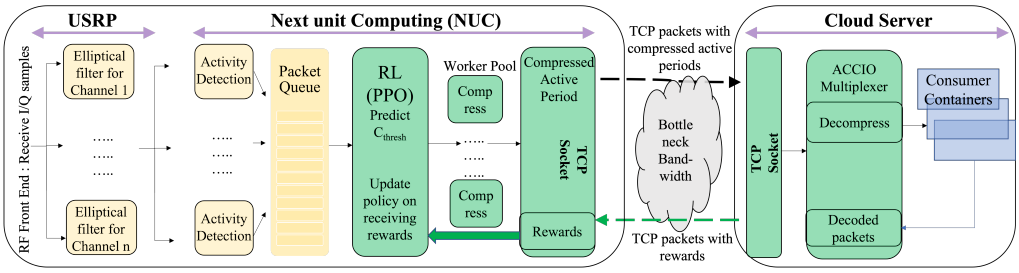


Fig. 5.2: Components of Cloud-LoRa : CRAN gateway (USRP) performs channelization, followed by activity detection at the NUC. ACCIO then compresses active periods using the RL agent and streams to the cloud server. The cloud server decodes the received packets and provides reward feedback to the ACCIO RL agent.

tionally, 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 [58, 59]. Urban uplink speeds vary widely between 10 Mbps and 100 Mbps, still insufficient to stream raw radio data from 8-channel gateways operating at 128 Mbps.

Recent works on CRAN-based LoRa: As discussed in Chapter 3, Charm [23], Nephalai [25], and OPR [24], SparSDR [26] and CharIoT [27] 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.

5.1 Cloud LoRa

Towards a practical, real-time LoRa CRAN, our Cloud-LoRa framework consists of three components, illustrated in Fig. 5.2:

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 5.2) : 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.

5.1.1 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. 5.2) [60]. 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 [26]. 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 [61, 62]. 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. 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.

Figure 5.3 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.

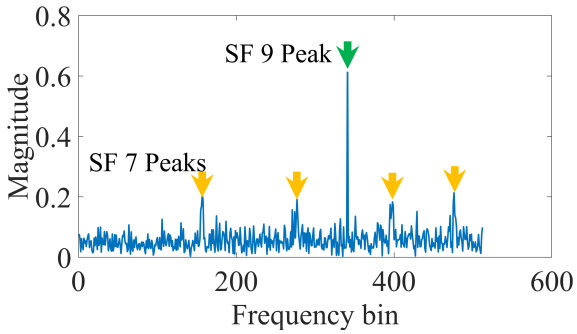


Fig. 5.3: FFT of signal dechirped with *superDC*

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 [†]) 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

$$\begin{aligned} superDC_{low}(t) \;&=\; \sum_{m=0}^{3} C(t - m\,T_{SF7},\, 7) \\ &+ \sum_{m=0}^{1} C(t - m\,T_{SF8},\, 8) \;+\; C(t,\, 9), \quad 0 \leqslant t \leqslant T_{SF9}, \end{aligned} \tag{5.1}$$

where $T_{SF} = \dfrac{2^{SF}}{BW}$ and $C(t, i)$ is the downchirp of SFi. We can similarly

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

define $\mathrm{superDC}_{\mathtt{high}}$. 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 $\mathrm{superDC}_{low}$ and $\mathrm{superDC}_{\mathtt{high}}$ 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 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 ($\mathrm{superDC}_{\mathrm{low}}$ and $\mathrm{superDC}_{\mathrm{high}}$) 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.

5.1.2 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 [25], a recent work on LoRa CRAN, utilizes downsampling and compressive sensing for compression.

But, novel demodulators leverage oversampling to resolve packet collisions [46, 16, 47, 15], and hence compression without downsampling is necessary. Moreover, LoRa's chirp spread-spectrum (CSS) modulation renders dictionary-based lossless compression methods ineffective. We propose BYOG , 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) [63] 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 ($\mathcal{O}(N)$, where N is the length of the signal) makes it light-weight, allowing it to compress in real-time.

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.

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 [25, 26] 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.

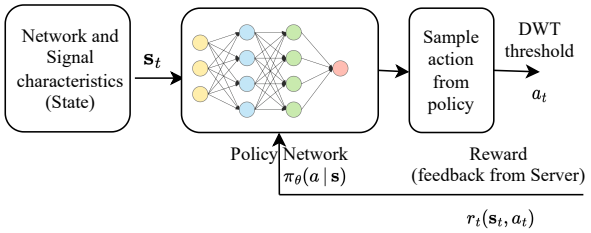


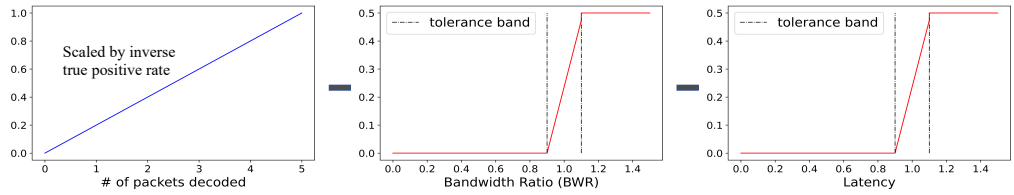
Fig. 5.4: Overview of the RL algorithm of BYOG

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 [64]. 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 [65]. 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 from the target environment, and still learn a good stable policy (see § 5.3.6). 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. 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 [66, 67], 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 [68] 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.



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

Fig. 5.5: Components of the reward function : (a) $u_{dec}(,a)$, (b) $u_{band}(,a)$, (c) $u_{lat}(,a)$.

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 $\pi(a)$ 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 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 5.1.

Reward Function. A well-designed reward function is a crucial part of the RL design. As discussed earlier, the goal of ACCIOis 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

Table 5.1: 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.

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

reward term $u_{dec}(,a)$ that is a weighted count of the number of packets decoded correctly (Fig. 5.5 a); **ii)** a negative penalty term $u_{band}(,a)$ that strongly discourages the bandwidth utilized from getting very close to the available bandwidth (Fig. 5.5 b); **iii)** a negative penalty term $u_{lat}(,a)$ that strongly discourages the overall latency (from client-side processing and network delays) from getting very close to a preset limit (2 seconds) (Fig. 5.5 c); and **iv)** a strong negative penalty $u_{over}(,a)$ (equal to -10) that prevents the packet queue from filling up close to its limit (dictated by the hardware). The last penalty term $u_{over}(,a)$ 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,

$$r(,a) = u_{dec}(,a) - u_{band}(,a) - u_{lat}(,a) - u_{over}(,a),$$

Details of our PPO implementation is given in Sec. 5.2.

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 $u_{dec}(,a)$, 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.

5.1.3 CRAN Cloud Server

The active LoRa signals compressed using ACCIOare 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 [23] atop our implementation. Each Cloud-LoRa packet contains time-stamps for coarse time synchronization between the base stations.

5.2 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 [39] 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 $\mathrm{superDC}_{low}$ and $\mathrm{superDC}_{high}$ 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 5.1 lists the state variables used by the RL agent.

The RL agent was trained using the PPO algorithm [68], whose implementation is provided in the Keras and TensorFlow libraries [69]. 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 light-weight 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 [70], 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 DWT level sizes (needed for Inverse DWT).

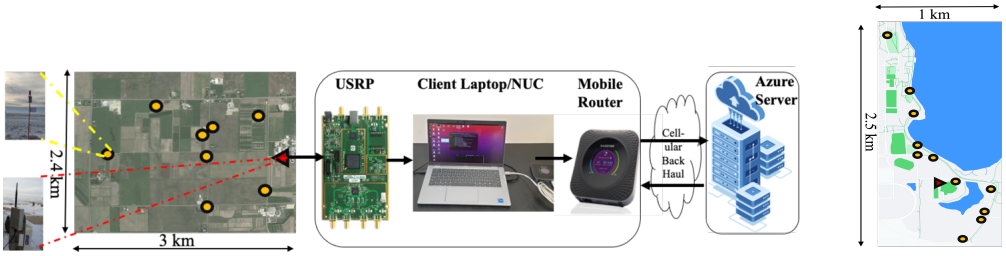
We utilize BBR as the TCP variant; it provides the estimated network bandwidth to the client. We use the socket statistics tool to obtain the bottleneck bandwidth, delivery rate estimated by BBR, and the link's

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 [71], 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 [72] 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.

5.3 Evaluation

We have deployed the first LoRa CRAN operating in real-time, in two practical outdoor deployments/scenarios. In RURAL(Fig 5.6(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-



(a) (b)

Fig. 5.6: (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.

LoRa in this scenario averaged across multiple days. URBAN(Fig 5.6(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?

5.3.1 Real-world Deployment Settings

We describe our rural and urban deployments in detail below.
RURAL : Rural deployment scenario. As shown in Fig. 5.6(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. 5.6(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. 5.6(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 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 5.6(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) [73] 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. 5.3.6), LoRa channel quality(Sec. 5.3.7), network load(Sec. 5.3.5), backhaul latency(Sec. 5.3.7), 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** [25], which proposes a compressed-sensing-based static compression; 4) **SparSDR** [26] – 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.

5.3.2 Performance in a Rural, Bandwidth-Constrained Deployment

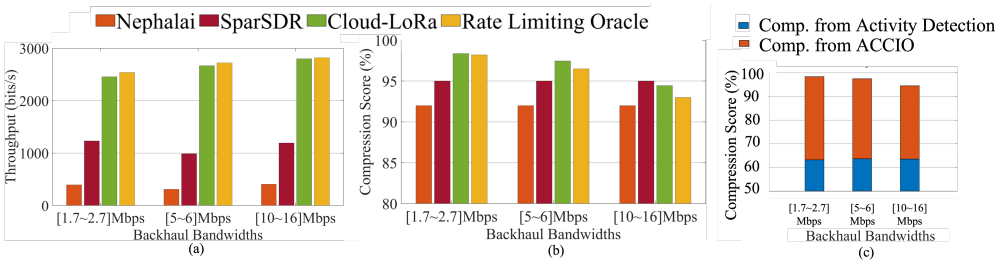


Fig. 5.7: (a) : Cloud-LoRa throughput performance across 8 parallel LoRa channels, averaged over multiple days in RURAL scenario. (b) Corresponding compression performance. (c) Ablation study on Compression.

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. 5.8. As seen from Fig. 5.8, 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. 5.6(a)) and simultaneously streams 8 channels. We plot the average LoRa throughput over all 8 channels (bits/second) achieved by Cloud-LoRa in Fig. 5.7(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).

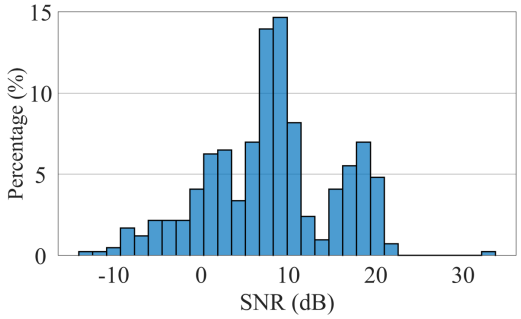


Fig. 5.8: Distribution of the received SNR at the Gateway in RURAL

In Fig. 5.7(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. 5.7(c). On average, the activity detection achieves a 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. 5.7(c).

5.3.3 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 [23] 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. 5.9, 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.

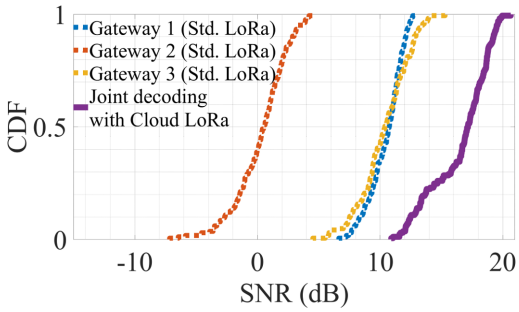


Fig. 5.9: SNR Gains from Joint Multi-Gateway Packet Decoding

5.3.4 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) [46] using Cloud-LoRa . CIC improves LoRa network throughput by decoding multi-packet collisions. In Fig. 5.10, we plot the network throughput with CIC as the demodulator in the cloud and compare it against Std. LoRa in the cloud.

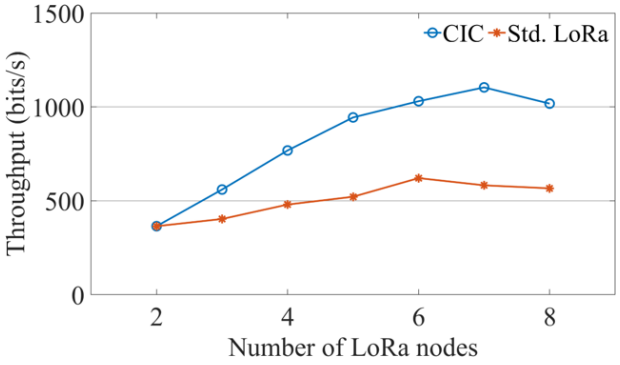


Fig. 5.10: Throughput Improvements due to Rapid Deployment of state-of-the-art

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.

5.3.5 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.

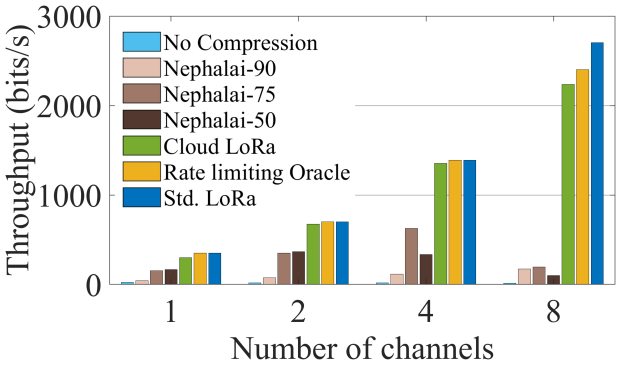


Fig. 5.11: Elastic Scaling to Multiple Channels - Throughput Vs # of channels

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. 5.11. 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.

5.3.6 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 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. 5.12, from 5 Mbps down to 1 Mbps,

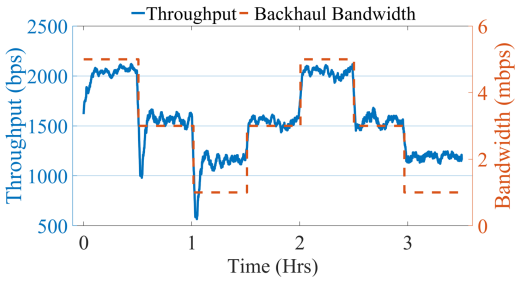


Fig. 5.12: ACCIO Adaption to Varying Backhaul Bandwidths

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. 5.12. 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 5.13 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.

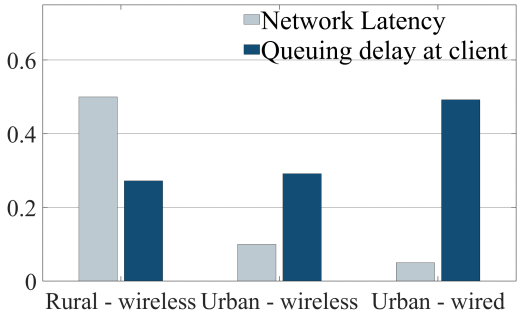


Fig. 5.13: ACCIO Adaption to Varying Backhaul Latency

5.3.7 Varying LoRa Channel Quality

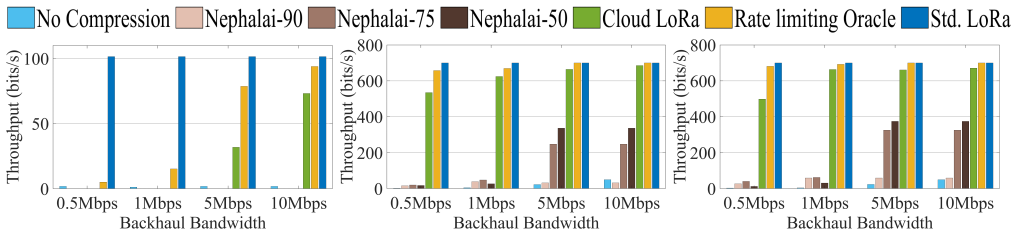


Fig. 5.14: ACCIO adaptation to varying LoRa Channel Quality (a) Low SNR (-20 to -5 dB) (b) Medium SNR (-5 to 10 dB) (c) High SNR (10 to 25 dB) LoRa signals.

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 5.14. Figure 5.14 (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 , 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. 5.14(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 BYOG 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. 5.14(c)) to the cloud with an average compression score of 98%.

5.3.8 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 5.15. 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

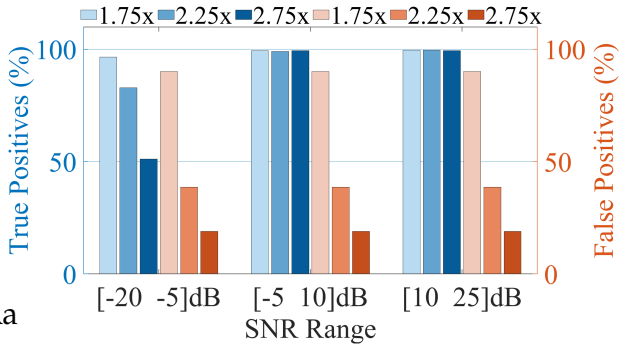


Fig. 5.15: Activity detection performance

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 A GENERAL-PURPOSE SOFTWARE DEFINED RADIO BASED MULTI-CHANNEL LORAWAN GATEWAY

Cloud-LoRa offers rapid integration of PHY-layer innovations. With CIC implemented in Cloud-LoRa, it delivers significant improvements, including boost in the network throughput 5.3.4, dynamic scaling to multiple LoRa channels 5.3.5, and doubled coverage for LoRa network 5.3.3. While Cloud-LoRa excels at enhancing weak links by coherently combining sub-noise signals, it may be overkill for high-SNR (Signal-to-Noise Ratio) packets. The end devices that are in proximity to the gateways can be reliably decoded locally. Compressing and relaying these signals to the cloud may not be the most efficient choice. This calls for a solution that allows local gateways to elastically scale to multiple radio channels to meet network demands. However, COTS gateways are hardware-centric and do not offer dynamic channel scaling. To overcome this, I propose BYOG in this chapter, a software, real-time LoRaWAN receiver that can run 10 channels simultaneously on general-purpose SDRs. Scaling to multiple channels on resource-constrained hardware is challenging, so we also introduce a lightweight packet decoding algorithm that reduces the complexity of standard LoRa decoding by a factor of 6 in a single channel.

6.1 BYOG

Our goal is to design a multi-channel LoRaWAN gateway that can receive a wide-band spectrum, channelize, demodulate, and decode LoRa packets in real-time on a software-defined radio. We propose BYOG that uses cascaded light-weight elliptical filters that can channelize more than 10 channels in real-time and propose and implement **self-dechirping**, an SF-agnostic preamble detection algorithm that has 6x fewer computations than that of a LoRaWAN gateway. It also estimates the SF of the preamble

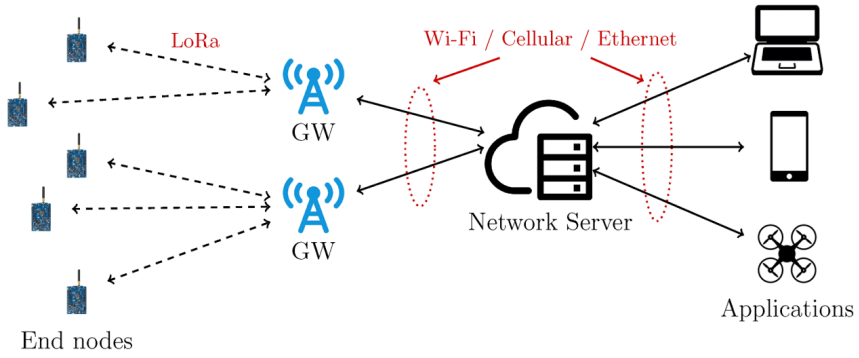


Fig. 6.1: LoRaWAN network architecture

detected, in turn avoiding the need for exhaustive SF search, as is the norm in LoRaWAN gateways.

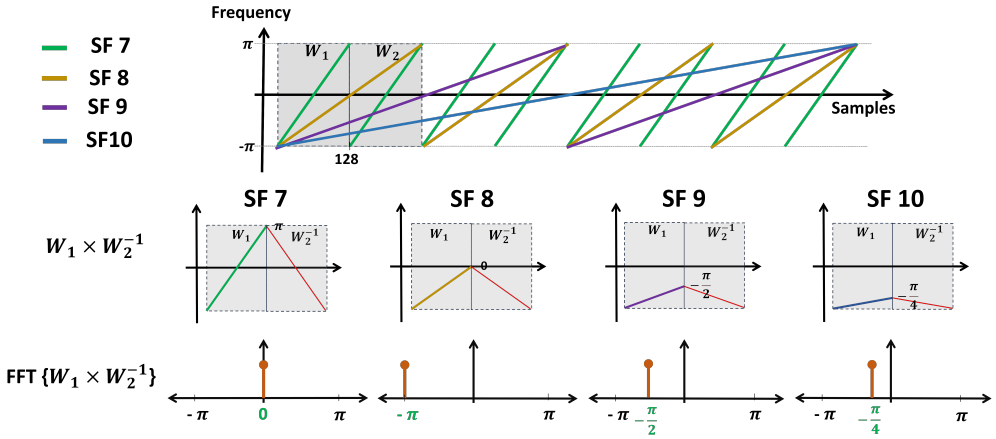


Fig. 6.2: Self-Dechirping: Top row shows the upchirps of various SFs. The gradient of each upchirp is a scaled function of SF7. Middle row shows our choice of two windows W_1 and W_2 with a fixed size $N_0 = 128$. W_1 and W_2 capture the entire upchirp of SF7, half of SF8, quarter of SF9, and 1/8 th of SF10. In the bottom row, the FFT of dechirping W_1 with W_2 shows that we can uniquely identify SF from the FFT of self-dechirping

6.1.1 Self-Dechirping

As described in Section 2, a standard LoRa demodulator dechirps the received signal - it multiplies the received window of samples with a

downchirp whose spreading factor (SF) equals the SF of the incoming signal. Such a dechirping process maximally accumulates energy into a single FFT bin. Since the packet detection block of standard LoRa also needs to dechirp preamble sequence composed of 8 base upchirps, off-the-shelf LoRaWAN gateway iteratively dechirps the preamble sequence with 6 different downchirps, each with a unique SF. Of these 6 dechirping processes, that which yields the maximum energy concentration into a single FFT bin for 8 consecutive windows is determined to be the correct SF, which is then used for packet demodulation. However, each dechirping requires multiplication followed by FFT, leading to a 6 fold increase in computations compared to a fixed-SF LoRa receiver.

We propose a single dechirping process that reveals both the SF of the received signals and flags the presence of preambles, thus saving up the resources to be used to scale to more channels. Our design of **self-dechirping** is built on the following insights.

- Downchirp is simply the complex conjugate of a base UpChirp and hence can be obtained from the received preamble.
- For a given bandwidth, downchirp of one SF is a scaled version of another downchirp.
- Data chirp is a scaled version of a linear upchirp.

Instead of locally creating downchirps of different SFs and then using it for dechirping, *BYOG uses the upchirp portion of the received preamble to generate the downchirp*. Consider the preamble of an SF7 packet in the top plot of Figure 6.2. If we look at the two consecutive windows W_1 and W_2, each of length $N_0 = 128$, the conjugate of the second window will yield a downchirp of SF7 as shown in the first plot of second row and if we multiply these 2 windows, this will dechirp the signal in the first window and an FFT will concentrate the energy into frequency index 0 as shown

in the first plot of bottom row. Therefore, we can dechirp without having to locally generate downchirp of SF7. We define this approach as **Self-Dechirping**, where the dechirping process is performed by leveraging the received signal rather than generating a new downchirp.

There are two challenges in generalizing this approach. First, in this case, W_1 and W_2 are aligned with the start of a LoRa symbol. Second, the dechirping window length of 128 matches with the chirp length of an SF7 symbol. However, aligning the windows requires accurate packet detection, which would lead to a chicken-egg problem. The same applies to choosing the appropriate dechirping window - which requires prior knowledge of SF.

We first address the choice of dechirping window length and then study the impact of aligning the windows in the next subsection. We propose to use the smallest dechriping window of length $N_0 = 128$, corresponding to SF7, regardless of the packet. We leverage our insight that a downchirp of one SF is a scaled version of another downchirp. Consider the received buffer containing a preamble of SF 8 packet in Figure 6.2, where a single SF 8 ($N = 256$) upchirp is split into two windows, each of length 128. The slope of an SF8 chirp is half as that of SF7 chirp and hence its frequency changes by π within a window of length 128, while that of SF7 changes by 2π (from $-\pi$ to $+\pi$). Similarly, if we maintain a window of length N_0 on chirps corresponding to all the spreading factors, their slope will keep reducing by a factor of $\frac{1}{2^{SF-7}}$ and the frequency change within N_0 window would be equal to $\frac{2\pi}{2^{SF-7}}$. Regardless of the SF of the packet in the receive buffer, taking conjugate of the second window creates a downchirp from the rest of the chirp segment. For the downchirp thus generated, the starting frequency equals the ending frequency of the chirp in the first window. Therefore, when we multiply the two windows, it results in the dechirped signal whose frequency equals the difference in the starting frequency of the chirps in the two windows. Therefore, an FFT yields a

peak at frequency equal to this difference given by $\frac{2\pi}{2^{SF-7}}$. Thus, not only can we dechirp the signal without prior knowledge of SF, we can also determine the SF of the signal using the fingerprint given by the unique peak position for each SF.

Let us view the process of self-dechirping through the window of initial frequencies of a chirp, as shown in Equation **??**. Let's assume that our two self-dechirping windows of length $N_0 = 128$ are aligning with the start of preamble of the received packet of any SF. The received buffer may have a packet of any SF where $SF \in \{7, 8, 9, 10, 11, 12\}$. The time-frequency variation in W_1 of self-dechirping window depending on the SF of underlying packet is given by Equation 6.1 given sampling rate is equal to LoRa bandwidth. Similarly, Equation 6.2 gives the time-frequency variation of W_2^{-1} (conjugate of W_2).

$$f_{SF}[n] = j\frac{2\pi}{2^{SF}}n - j\pi, \tag{6.1}$$

$$f_{SF}^{-1}[n-128] = -j\frac{2\pi}{2^{SF}}n + j2\pi\frac{128}{2^{SF}} + j\pi, 1 \leqslant n \leqslant N_0, N_0 = 128 \tag{6.2}$$

Upon performing self-dechirping ($W_1 \times W_2^{-1}$), the phase in the exponent of the chirp equations gets subtracted, therefore the frequency of the dechirped signal is given by Equation 6.3 and the resultant frequency is wrapped into the range of $-\pi$ to $+\pi$ by adding or subtracting 2π in case if it exceeds the range. FFT of this dechirped signal gives a peak at the same frequency as in Equation 6.3.

$$f_{SF}[n] - f_{SF}^{-1}[n-128] = -j\frac{2\pi}{2^{SF-7}} \tag{6.3}$$

Based on Equation 6.3, if our received buffer contains SF7 preamble, the peak appears at 2π that is mapped to 0. Similarly, for SF8 the peak appears at frequency index $-\pi$, for SF9 the peak appears at $\frac{-\pi}{2}$, for SF10

the peak appears at $\frac{-\pi}{4}$, for SF11 the peak appears at $\frac{-\pi}{8}$, for SF12 the peak appears at $\frac{-\pi}{16}$. Therefore, we conclude that we can detect the presence of a packet by observing a peak at the end of self-dechirping and the index of the peak uniquely maps to the SF of the packet received. Our proposed SF and packet detection algorithm can be generalized to other chirp lengths as well. In other words, with simple tweaks in the choice of the window length, this algorithm can be generalized to a wider range of *integer* SFs, allowing it to be compatible to any future changes in SF set in LoRa standard.

6.1.2 Effect of Time Offsets on Self-Dechirping

We have addressed the first challenge of choosing the dechirping window; our next challenge is in aligning the window. So far we have assumed that window W_1 is aligning perfectly with the start of the packet. However, this may not be the case with real captures as we are continuously searching for packets. The received samples in the current buffer may not perfectly align with the start of the packet. However, since a LoRa packet contains 8 consecutive upchirps as the start of packet, if we dechirp windows W_1 and W_2 and jump these windows each of length N_0 on the received buffer for the next symbol, such that the two jumps are N_0 samples apart, our windows will definitely overlap with the preamble upchirps of the received packets regardless of the SF. In this section, we show that even if W_1 and W_2 do not align with the start of the packet, we still observe peaks after self-dechirping and can extract the SF fingerprint through these peak indices.

It can be noted that W_1 and W_2 are consecutive windows. Therefore, when W_1 is not aligned perfectly with the start of a packet, we perceive this as a time offset in W_1 compared to the perfect alignment case. Since W_1 and W_2 are consecutive windows, any time offset in the first window will also be translated to the second window. Due to the time-frequency linearity

of chirp signals, this time offset appears as a frequency offset in both the windows as shown in Equation 6.4 and 6.5.

$$f_{SF}[n+\tau] = j\frac{2\pi}{2^{SF}}n + j2\pi\frac{\tau}{2^{SF}} - j\pi, \quad (6.4)$$

$$f_{SF}^{-1}[n-128+\tau] = -j\frac{2\pi}{2^{SF}}n + j2\pi\frac{128}{2^{SF}} + j2\pi\frac{\tau}{2^{SF}} + j\pi, \quad (6.5)$$

Since the offset appears in both the windows, during self-dechirping, where the complex conjugate of the second window is used, this offset gets cancelled and the unique frequency fingerprint given by Equation 6.3 stays unaffected. Therefore, the second challenge of aligning with the start of the packet is solved by self-dechirping, as misalignment is cancelled out.

6.1.3 Retaining SF Sensitivity

BYOG thus can detect the start of any LoRa packet and estimate its SF accurately using self-dechirping. In a LoRaWAN network that uses ADR, devices closer to the gateway are assigned lower SFs and those far away are assigned higher SFs. LoRa uses higher SFs as they offer higher spreading. A higher SF symbol is longer than that of a lower SF packet. In other words, a transmitter spreads its signal over a wider band. At the receiver, more energy accumulation is possible. For example, an SF of 12 has a dechirping window that has 2^{SF-7} times more samples as compared to SF 7 packets. Therefore, for similar SNRs, higher SF packets offer higher SNR gains since the peaks in the FFT stand taller as opposed to lower SFs. Such SNR gains of higher SFs is lost once we fix the self-dechirping window length corresponding to the symbol length of smallest SF, i.e., 7. BYOG uses a pair of consecutive windows W_1 and W_2 each of length $N_0 = 128$ for dechirping which can only promise the SNR sensitivity corresponding to the smallest SF of value 7.

To retain the SF sensitivities over all Sfs, we accumulate energy across multiple self-dechirping windows. We observe that largest symbol corresponding to SF 12 symbol is 32 times longer than that of our self-dechirping window. Therefore, while jumping a pair of windows across the received buffer, at each jump we not only record the peak heights but also add up the energy in the tallest peak of the self-dechirped signal for 32 consecutive windows. If the underlying signal in the buffer is an SF 12 preamble, we will be able to detect the packet without compromising on SF sensitivity. Since all other SF symbols have lengths smaller than SF 12 symbol, energy accumulation across 32 self-dechirping windows retains their sensitivities as well.

Energy accumulation brings up a new challenge in self-dechirping. So far, we have assumed that we will see repeating peaks only when we parse through the preamble of a packet and that results in a unique frequency. This is true for an SF7 preamble upchirps for which we observe at least 6 peaks at frequency zero. As the self-dechirping window is smaller than the symbol length for higher SFs, we observe unique peaks not only for preambles, but also for data chirps.

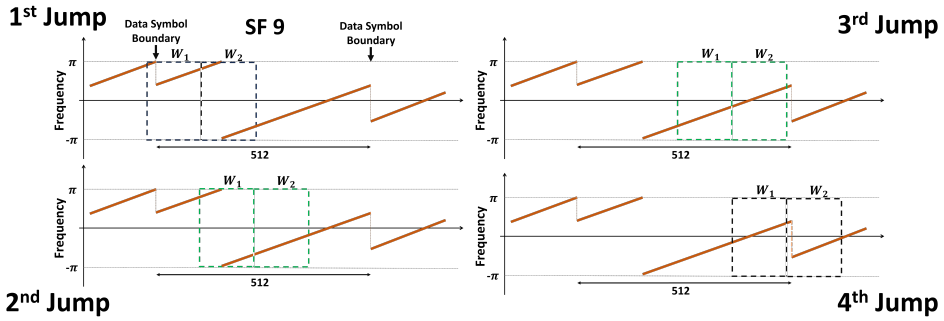


Fig. 6.3: Impact of self-dechirping on datachirps : Datachirp of an SF9 packet is shown here. In the first and fourth jump, W_1 and W_2 capture two different data chirps and do not yield a peak. In the second and third jump however, the windows capture the same datachirp and hence result in FFT peaks.

For example, in Figure 6.3, let us assume that while self-dechirping an

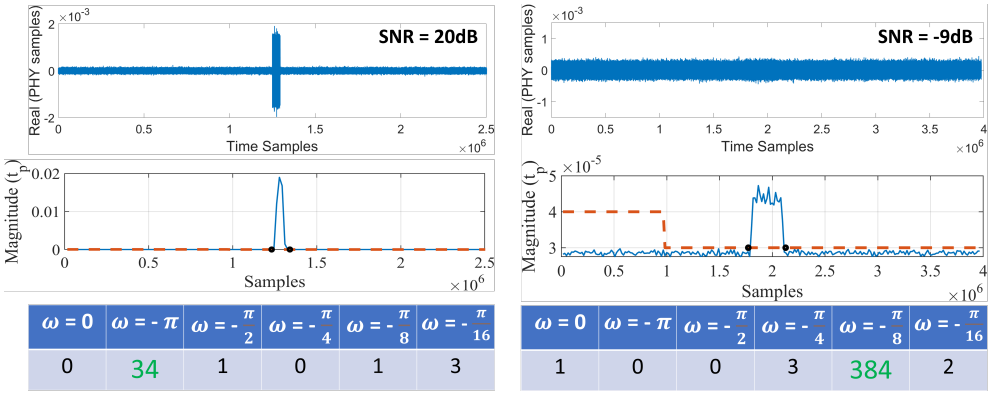
SF 9 packet, the windows W_1 and W_2 during the 1^{st} jump overlap with a datachirp such that either of the windows contains symbol transition i.e., frequency changes depending on the data to be sent in next symbol. In this scenario, we will be getting multiple peaks in FFT since the symbol boundary introduces a discontinuity in the linear frequency change of chirps. However, in the subsequent 2^{nd} jump, none of the windows aligns with the data symbol boundary because SF 9 symbol duration is longer than our fundamental window length of N_0 (SF 9 symbol length is 4 times N_0). Therefore, we get a fingerprint peak in FFT at the unique index corresponding to SF 9 at 2^{nd} and 3^{rd} jumps. Two out of four jumps yield SF fingerprint and accumulate energy. Whereas, 1^{st} and 4^{th} jump is obstructed by the data symbol boundaries. This observation can be extended to higher SFs as well. For SF 12, we get 30 SF fingerprints for every 32 jumps on datachirps as well while maintaining SF sensitivity by accumulating energy across these jumps. We take this into consideration in determining the SF, as detailed in the next subsection.

6.1.4 BYOG Algorithm

We put all these together and present the algorithm for BYOG which detects the SF of incoming packets and then demodulate them using standard LoRa. BYOG first filters the received I/Q samples and channelizes them. In each channel, it runs N_0 length of consecutive windows on the incoming raw I/Q samples. It jumps the window on the received buffer such that two jumps are N_0 samples apart. At each jump, it first performs self-dechirping, computes the FFT and then records the peak index and energy in the FFT. After the first 32 jumps, it accumulates the energies across all FFT peaks in a variable t_p to retain SF sensitivity while keeping the record of peak indices in a variable $index\_array$ for the same set of 32 windows. For the first set of 32 windows, when there is no transmission going on in the channel, BYOG uses the first value of t_p to estimate noise

floor and maintains a threshold, $thresh = 1.09 \times t_p$. From this point, it uses this threshold to detect any activity in the channel. Following this, sets of 32 windows are processed together. In each set, the accumulated energy t_p is compared against the noise floor $thresh$. If $t_p > thresh$, it then records that set to contain channel activity (records the sample #) and sets a flag flg indicating the possibility of a LoRa packet. Similarly, it looks for activity in the next set of 32 windows; if activity is found in 2 consecutive sets, it keeps the flg set, and appends the $index\_array$ of the two sets into another variable big_index_array. This is continued until activity is found in consecutive sets. As soon as t_p drops below threshold for any 32 window set, it clears the flg, records the sample # to determine the end of activity. Moreover, variable big_index_array so far contains the record of all the peak indices during activity period in the channel. It then counts the number of times each of the frequency $\omega \in \{0, -\pi, -\frac{\pi}{2}, -\frac{\pi}{4}, -\frac{\pi}{8}, -\frac{\pi}{16}\}$, appear in the big_index_array and records the frequency of these peak indices in 6 separate variables corresponding to each SF 7, 8, 9, 10, 11, 12. The highest frequency then denotes the SF fingerprint. This fingerprint maps to the correct SF of the packet, which is then passed to the demodulation block. A LoRa demodulator performs fixed SF demodulation on the received samples as recorded in the indices determined above.

In Figure 6.4, top plot shows the real part of SF 8 and 11 packets of SNR 20 dB and -9dB respectively, captured using a Software Defined Radio (SDR). When the above algorithm is run on these samples, we get middle-row plots that show energy accumulation t_p along with samples. The black circles on the plot denote the recorded start and end of channel activity. Since SF 11 packet has low SNR, we can see how the $thresh$ value gets estimated after first set of 32 windows. The bottom-most table shows the number of times each of the ω appears in big_index_array. Clearly, the most occurring frequency correspond to relevant SF, $\omega = -\pi$ for SF 8 and $\omega = \frac{-\pi}{8}$ for SF 11. The reason for having more number of $\frac{-\pi}{8}$ peaks as



$\omega = 0$	$\omega = -\pi$	$\omega = -\frac{\pi}{2}$	$\omega = -\frac{\pi}{4}$	$\omega = -\frac{\pi}{8}$	$\omega = -\frac{\pi}{16}$
0	34	1	0	1	3

$\omega = 0$	$\omega = -\pi$	$\omega = -\frac{\pi}{2}$	$\omega = -\frac{\pi}{4}$	$\omega = -\frac{\pi}{8}$	$\omega = -\frac{\pi}{16}$
1	0	0	3	384	2

Fig. 6.4: Self-Dechirping on real data : Top row shows the real value of the received samples for an SF 8 and SF 11 signals. Their corresponding energy accumulation is shown in the middle row. As can be seen, higher SF shows more energy accumulation. The bottom row shows the number of peak occurrences for each frequency and hence the SF. The left table has the highest entry corresponding to SF8 and the right table to that of SF11

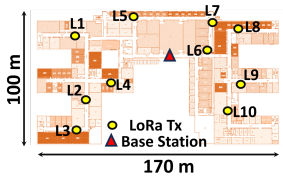


Fig. 6.5: Deployment Scenario

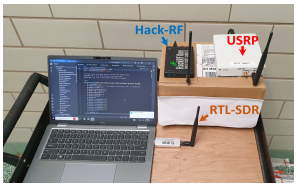


Fig. 6.6: 3 Software Defined Radios as Base Stations

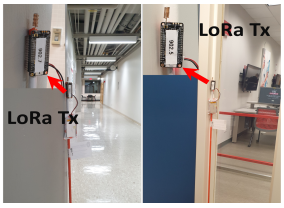


Fig. 6.7: LoRa Transmitters deployed in the building

opposed to $-\pi$ peaks is due to the fact that data symbols of SF 11 also give unique frequency fingerprint and higher SF packets are generally longer in time duration as well. Therefore, by applying BYOG on real data, we show how it can reveal the SF and start of underlying packet with a single dechirp run on the received signal. In the subsequent sections, we deploy BYOG in a real network setting and evaluate its accuracy.

6.2 Implementation

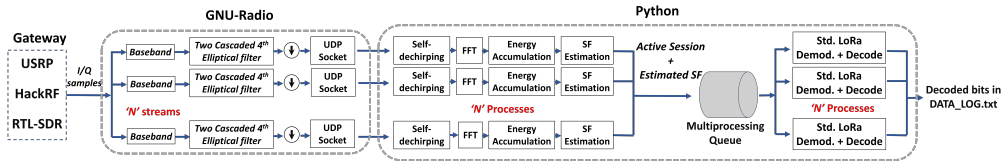


Fig. 6.8: BYOG implementation framework

BYOG is implemented on general-purpose SDRs to realize LoRaWAN gateway capable of performing ADR. Our implementation includes the following modules: off-the-shelf LoRa end devices as transmitters, LoRaWAN gateway implemented on SDRs, and a server computing unit implemented on a laptop. We describe each of these modules in detail below.

LoRa Devices. We use commercially available LoRa Devices - Adafruit Feather M0 with RFM 95 [42]. We used 10 of these devices as shown in Figure 6.7, $T_i, i = 1, 2, ...10$ and using Arduino Library RadioHead [43] configured each of these to transmit in separate channels in the range [902.3MHz, 904.1MHz] with the center frequencies 200 kHz apart. We use a control channel in 915 Mhz - during system setup, devices listen at SF8, 250kHz to coordinate experimental setup. We reserved an additional LoRa transmitter R that served as a coordinator and sent SF 8, 250kHz beacon packets in 915MHz which would specify the duty cycle, the duration for the experiment, and a start message. All T_i nodes upon hearing these beacons in order from R would copy the information and would switch to their respective channels and set their bandwidth to 125kHz. Due to LoRa regulations, we only use $SF \in \{7, 8, 9, 10\}$ in 125kHz BW. Each node then transmits packets of random SF to emulate ADR, random length to emulate various co-existing applications and random transmit power to emulate various distances, with uniformly distributed time intervals

such that the duty cycle follows the value defined by R. Experiments are performed for 20 minutes in one session and repeated over multiple sessions. This is due to the streaming limitations posed by the SDRs. In each session, all the nodes transmit LoRa packets at 15% duty cycle.

LoRaWAN gateway - SDR We use three general-purpose SDRs - USRP B200, Hack-RF, and RTL-SDR as our RF-front (Figure 6.6). BYOG is implemented on a 12-core, 16GB RAM, Intel i5 laptop. As shown in Figure. 6.8, using GNU-Radio's USRP, Hack-RF, and RTL-SDR source blocks, we capture raw I/Q samples at a center frequency of 903.2MHz which is exactly the midpoint of the 10 channels of nodes T_i. To capture 10 channels, we fixed the sampling rate of USRP and Hack-RF to 4 MSamples/s which is sufficient as per Nyquist rate requirements. Due to RTL-SDR's maximum sampling rate limit of 2.56 MSamples/s [74], we can capture only 4 channels. Therefore, we only use nodes $T_i, i = 4, 5, 6, 7$ while conducting experiments using RTL-SDR as the gateway. As shown in Figure 6.8, we first perform channelization bringing all channels down to the baseband followed by filtering. We use two cascaded elliptical filters for maximum out-of-band activity suppression. The low complexity of elliptical filters make them a popular choice for real-time filtering. We design 4^{th} order elliptical filter with 100dB passband to stopband attenuation. After filtering each of the 10 channels, BYOG downsamples the incoming stream from 4 MSamples/s to 500 kSamples/s for USRP and HackRF and from 2 MSamples/s to 500 kSamples/s for RTL-SDR. Each channel therefore outputs I/Q samples at a rate of 4 $\times$ LoRa BW of 125kHz. This oversampling factor helps BYOG to locate packets with better time resolution. The downsampled signal in each channel is dumped into a separate UDP socket.

LoRaWAN gateway - Server laptop In a configuration file 'client_config.py', users can specify the experiment settings such as the number of channels, along with the LoRa parameters such as the bandwidth of the signal

and the rate at which incoming packets are oversampled. As shown in Figure. 6.8, users first run Python implementation through 'main.py' script in which BYOG starts listening to each of the UDP sockets specified by the number of channels. After that, users run the gnu-radio sketch for available SDR (the sketches for each SDR can be found in the gnuflow folder). Upon receiving samples through each socket, BYOG initiates a process for each of the channel that parallelly runs self-dechirping, which detects the presence of a packet, Self-dechirping block takes 1s chunk of I/Q samples and runs self dechirping window. Since the received signal in each channel is 4x oversampled, the length of the dechirping window also becomes $4 \times N_0$. Each channel first estimates its $thresh$ value in the first second of SDR capture. It then performs self-dechirping followed by FFT to get peak fingerprints and accumulates peak energy across 32 window jumps. For the next incoming 1s chunk of I/Q samples, it appends $4 \times N_0$ tail samples of the previous chunk with the new chunk in order to preserve continuity. Whenever the presence of a LoRa packet is detected in any channel along with estimated SF, raw I/Q samples for the corresponding period along with the estimated SF value are appended to a shared multiprocessing queue. 10 parallel standard LoRa processes then consume the queue and use estimated SF to demodulate and decode the LoRa packets.

We may scale channels beyond 10 as well with the help of higher-end computing system. In order to go beyond 10 channels, our sampling rate needs to be greater than 4 MSamples/s. Even though both USRP and HackRF can support sample rates of upto 60 MSample/s [39] and 20 MSamples/s [75] respectively, the limiting factor is the RAM, memory, and the number of cores in the system. We tried to scale beyond 10 channels in real-time with the available laptop but started getting severe SDR overflows. Moreover, as shown in Figure, 6.8, since we initiate $N(where N = \#ofchannels)$ pool of parallel processes for channelization,

N processes for BYOG 's SF estimation block and N processes for standard LoRa's demodulation, as we scale beyond 10 channels, the cores in the system are not able to cater to all the parallel processes, which leads to SDR overflows. Hence, the entire pipeline fails to decode packets across all channels in real-time. With a system with more cores, faster CPU, and more memory, we may scale up to 16 channels using USRP and HackRF.
Deployment Scenario For evaluation, we deploy 10 nodes T_i at 10 different locations $L_1, L_2, ...L_{10}$ as shown in Figure. 6.5. These locations have different distances from the base station and are completely non-Line of Sight. The packets from nodes closer to base stations have higher SNRs, whereas packets received from distant nodes having concrete walls in between have very low SNRs. For each session of experiments, we run BYOG in real time with one SDR as a gateway (3 total sessions, each with a different SDR). While BYOG decodes packets across multiple channels in real time, we also save the received I/Q samples from each SDR locally on the system.
Baselines compared We compare BYOG against two baselines using the samples stored in the online experiments described above. We replay the samples and sequentially run standard LoRa on each channel offline to emulate an ideal 10-channel LoRa Gateway. Standard LoRa exhaustively searches for packet by iteratively dechirping the whole received buffer with all the SF downchirps. Through the stored file, we estimate the duration of the experiment (20mins) and then use $\frac{\textit{Total number of decoded packets across all channels}}{\textit{session duration}}$ to estimate the throughput. We compare the real time throughput of BYOG with the emulated throughput of the 10-channel LoRaWAN gateway. We also emulate the throughput of an 8-channel LoRa gateway with the assumption that it decodes all packets in a given session but only scales up to 8 channels.

6.3 Evaluation

We answer the following questions towards evaluating BYOG .

- What is the achievable network throughput when the number of channels goes beyond 8?
- How many LoRa channels can we support using SDR as LoRa Gateway?
- What percentage of preambles are detected by the self-dechirping algorithm? How accurate were the SF estimations?
- Can off-the-shelf low and medium-end SDRs operate as LoRaWAN ADR by accommodating the different settings of a LoRaWAN gateway?

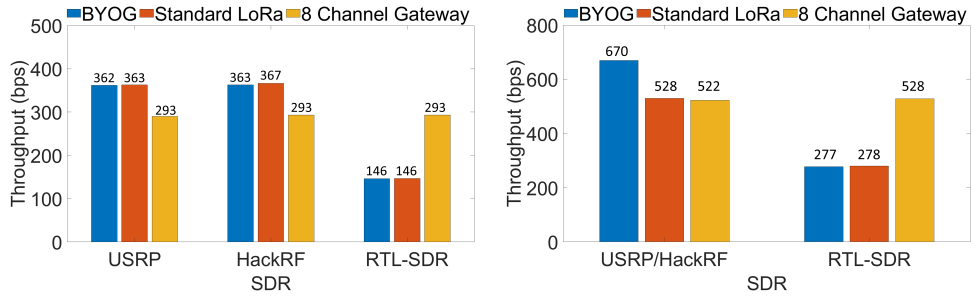


Fig. 6.9: Throughput of BYOG with different SDRs along with std. LoRa (a) 15% Duty Cycle, (b) 30% Duty Cycle (Emulated traffic)

6.3.1 Network Throughput of a Multi-Channel LoRaWAN

In this section, we evaluate the overall network throughput of a LoRaWAN with one transmitter per channel. To emulate ADR, each transmitter switches its SF randomly and transmits packets with 15% duty cycle. We

perform three sets of 20-minute experiments each using a different SDR and present the average network throughput in Figure 6.9(a) for different SDRs. The maximum network throughput depends on multiple factors including the sampling rate and in turn the number of channels supported by the SDR, the noise floor of the SDR's radio front end, preamble detection accuracy, and SF estimation accuracy. USRP and HackRF show the largest network throughput of 362 and 363 bps respectively as both can support a sampling rate of 4 Msamples/s which in turn can accommodate 10 channels each 125kHz. USRP's throughput is closest to std. LoRa and differs by just 1 bps. HackRF similarly has high throughput since it supports 10 channels; however, its throughput differs from std. LoRa by 4bps. HackRF has a higher noise floor than USRP resulting in lower received SNR. Due to lower SNR, HackRF lags behind USRP in approaching network throughput of std. LoRa. RTL-SDR on the other hand only offers 2 MSamples/s which only captures 4 channels. RTL-SDR also has a high noise floor similar to HackRF for a given BW. But due to the lower sampling rate, the noise floor stays low. Therefore, RTL-SDR approaches the network throughput of std. LoRa. We also plot the maximum throughput achieved by an 8 Channel LoRaWAN Gateway assuming it detects and decodes all the packets injected in the network. The number of channels that can be supported also depends on the computing unit (laptop in our experiments) used. We used a 12-core laptop where 1 core was dedicated to process each channel, one core to interface with the SDR, perform channelization, and one core for background applications, storing the samples among others. The number of channels that can be supported could be improved seamlessly by using a computing unit with more cores and/or better parallelism, which is part of our future work.

It must be noted that the throughput performance of BYOG is the same as that of the emulated high-end LoRaWAN gateway (orange bars in Figure 6.9). This shows that BYOG does not tradeoff performance for

computational complexity and can outperform an off-the-shelf 8-channel LoRaWAN gateway (yellow bars in Figure 6.9).

To evaluate our system in higher network traffic, we emulated data in MATLAB where each node transmitted packets at 30% duty cycle as shown in Figure6.9(b). We replayed data samples to BYOG in real time and ran standard LoRa as well. To emulate data from USRP and HackRF, our data had 10 channels and to emulate data from RTL-SDR our data had 4 channels, as dictated by the respective SDR's sampling rates. In higher network traffic, BYOG outperforms standard LoRa as well. This is because, for higher network traffic, standard LoRa sees more packets and runs preamble correlation corresponding to each SF before demodulation. This incurs a lot more computations and std. LoRa is not able to keep up real-time operations. Therefore, overall throughput decreases. With 4 channel SDR, since the number of channels is lesser, std. LoRa is able to keep up real-time processing.

SF was varied uniformly across packets in each channel. Therefore, all SFs have approximately same number of packets in the results presented. The received SNR cannot be tightly controlled and varies significantly across the channels. SNR distribution of all the packets in each channel is presented in Figure 6.10. We categorize packets in channel # 3, 4, 9 and 10 as low SNR distributed in the range of [-20 0]dB. Channel # 1, 2, 7 and 8 fall under medium SNR category distributed in the range [-10 10]dB. Channel # 5 and 6 are categorized as high SNR distributed in the range [0 20]dB. Our careful design of the channelization block ensures that inter-channel interference is maximally reduced.

6.3.2 Per-channel Throughput using BYOG

Following the overall network throughput, we delve deeper into per-channel throughput to understand fairness across channels in Figure 6.11. We plot the throughput in each of the channels for the three SDRs con-

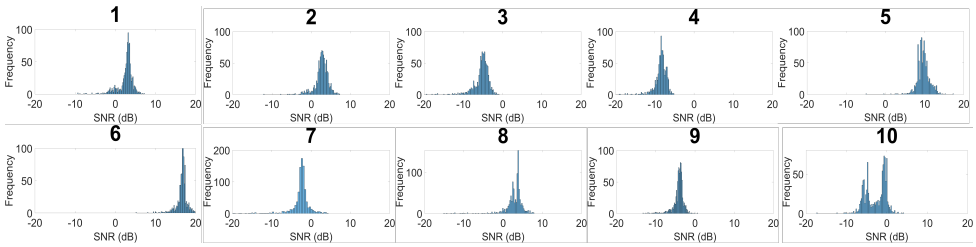


Fig. 6.10: SNR distribution of packets in each channel

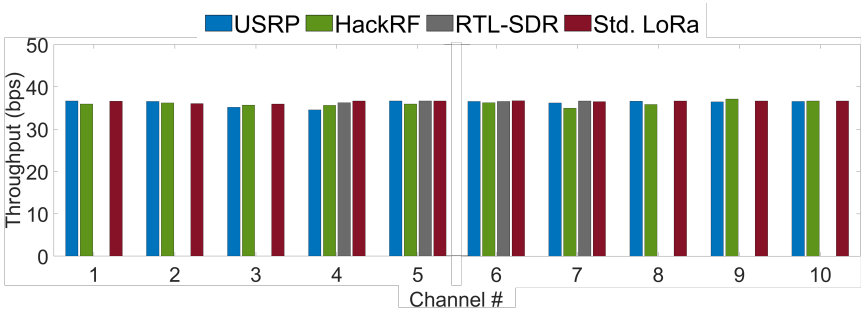


Fig. 6.11: Average per-channel throughput with BYOG compared against LoRaWAN

sidered. We observe that the throughput remains relatively the same across channels, depsite the variabilities in the SNR range, as shown in Figure 6.10. This shows that BYOG can operate throughout the range of SNRs supported by standard LoRa gateway. RTL-SDR only captures Channels 4 through 7. It must be noted that despite its low cost, the throughput performance is in par with USRP and Hack-RF. This shows that our proposed framework can be used with even low cost SDRs. The single channel performance of all the SDRs is comparable as it is not impacted by the sampling rate.

The per-channel throughput of BYOG suggests that we can scale to multiple channels without loss of generality using general-purpose SDRs and build a multi-channel LoRaWAN gateway.

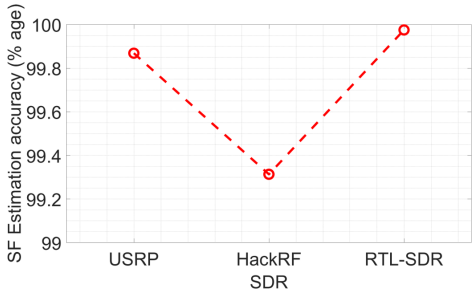


Fig. 6.12: Accuracy of SF Estimation with different SDRs as front-ends in Multi-Channel Experiment

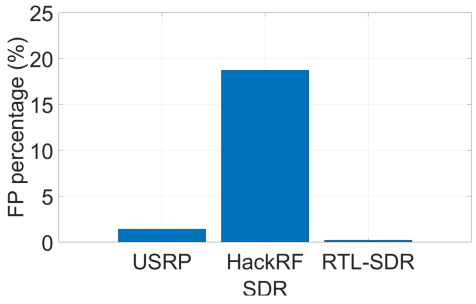


Fig. 6.13: Percentage of False Positives in Packet Detection in Multi-Channel experiment

6.3.3 Accuracy of Spreading Factor Estimation

In Figure 6.12, we plot the percentage of packets for which BYOG was able to detect SF accurately. As shown, with USRP and RTL-SDR as gateways, BYOG estimates the SF of incoming packets with 99.86% and 99.97% accuracy computed over almost 10000 packets. This suggests that SF estimation algorithm does not compromise accuracy for computational complexity. HackRF also achieves similar accuracy but slightly suffers as compared to other base stations due to higher noise floor that affects the peak position estimates in self-dechirping. We also present the % age of False Positives (FP) detected and summed over all channels in Figure. 6.13. False positive is defined as a session flagged by BYOG to contain a valid LoRa packet of an estimated SF but in reality it does not contain any LoRa packet. With USRP and RTL-SDR, we detect 1.41% and 0.2% of such FPs. This is made possible due to appropriate choice of threshold, i.e. $thresh = 1.09 \times t_p$ where t_p is sum of energy value in 32 consecutive self-dechirping FFT peaks. This threshold as shown by orange line in Figure. 6.4 is low enough to capture very low SNR packets and high enough to not capture random peaks as LoRa activity. HackRF has higher FP rate due to same reason of higher noise floor. Due to higher

noise floor, random peaks qualify the threshold and therefore increased FP rate is observed.

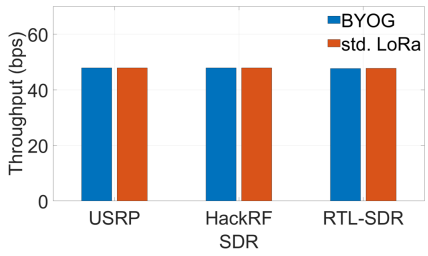


Fig. 6.14: Throughput in a single 500kHz channel with all SF

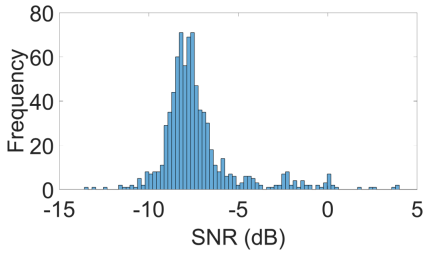


Fig. 6.15: SNR distribution in the single 500kHz channel

6.3.4 Throughput performance in a single 500 kHz channel

So far, we evaluated the 125kHz LoRa channels. Due to LoRa regulations, we only used SFs 7 through 10 in the 125 kHz band. LoRa also allows 500kHz band where all the SFs are permitted. We evaluate the throughput performance of each SDR operating in a single 500 kHz channel where the transmitters go through all possible SFs during the transmission. For the given sampling rate limitations, only one 500kHz (out of 8) is feasible. We deployed a LoRa Tx at location L2 in Figure. 6.5 that transmitted packets randomly of SF 7 though 12 at BW 500kHz. Figure 6.14 shows the throughput performance of BYOG for each of the SDR in the x-axis. For this single channel all SF case, the throughput achieved by BYOG is the same as that of standard LoRa. SNR distribution in Figure. 6.15 show that most of the packets were low SNR. Moreover, all the packets received in this experiment had 100% SF estimation accuracy and with 0% FP. Therefore, this experiment proves the ability of BYOG to estimate spreading factor of the entire range of SFs.

6.4 State-of-the-art Activity Detection techniques for LoRa

Existing LoRaWAN Gateways are specialized hardware that run standard LoRa demodulator and decoder. Commonly available Gateways support either 4 or 8 LoRa channels. As discussed in Section 6.1.1, in order to support ADR, a LoRaWAN gateway must be capable of correlating the received buffer with multiple Spreading Factors (SFs) in parallel. As the SF increases, the computational cost of detecting SF increases. For higher SFs, we need to perform higher point correlation. This requires a set of compute resources dedicated for SF identification of incoming LoRa packets. We argue that if we could capture the SF of an incoming packet in a single shot correlation, the compute resources could be used for scaling to multiple channels. Hou et al. [28], Koch et al. [29] and Cloud-LoRa [76] propose algorithms to detect packets without correlating with all SFs in order to improve network scale. They propose two correlations of received buffer: one with the superposition of even SFs and second with the superposition of odd SFs. This technique saves computation but compromises on LoRa's sub-noise packet detection. Since, these works propose superposition of chirps of multiple SFs, this decreases the SINR of the correlation which results in the energy of the received signal being dominated by the superposition signal. Therefore, they fail to detect sub noise LoRa packets. As opposed to these works, we propose a lightweight, SF-agnostic packet detection algorithm that does not compromise on the SNR sensitivity of LoRa. We implement our proposed receiver design on multiple software defined radios, thus showing the flexibility of our framework as opposed to LoRa Gateways whose PHY layer cannot be modified to test other LoRa demodulators.

7 DISCUSSION

In this thesis, I have presented a set of practical, standard-compliant solutions aimed at improving the scalability and throughput of LoRa networks. A major design focus of my work has been to ensure compatibility with existing LoRa infrastructure. Specifically, no changes are proposed to the LoRa PHY-layer at the transmitter end, allowing off-the-shelf (OTS) LoRa radios to be used without modification. All proposed innovations are implemented at the gateway side using Commercial-Off-The-Shelf (COTS) software-defined radios (SDRs), maintaining cost-effectiveness and ease of deployment.

The three systems proposed CIC, Cloud-LoRa, and BYOG, described in Chapters 4, 5, and 6 respectively, are complementary rather than mutually exclusive. Together, they offer a layered strategy to enhance LoRa performance across physical-layer decoding, cloud-based scaling, and gateway-side flexibility.

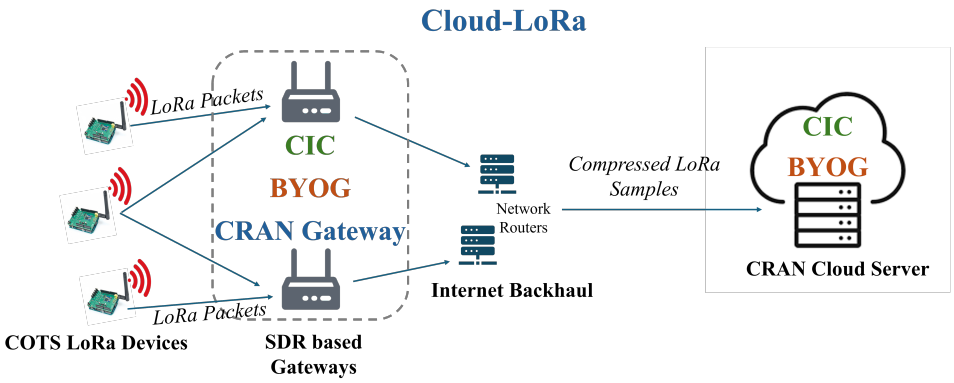


Fig. 7.1: Overall system proposed by thesis

Figure 7.1 illustrates the interoperability of these systems. SDR-based gateways are capable of running CIC, BYOG, or operating in CRAN mode as part of Cloud-LoRa. When operating in CRAN mode, gateways com-

press and forward I/Q samples to the cloud, where decoding algorithms such as CIC (for collision resolution) or BYOG (for lightweight, scalable multi-channel decoding) can be deployed dynamically.

Cloud-LoRa also aligns well with the higher layers of the network stack. In the cloud, once PHY-layer packets are reconstructed, network and application layer functions—such as packet forwarding, acknowledgments, and analytics—can be integrated into a unified processing pipeline. Looking forward, Cloud-LoRa has the potential to incorporate advanced features from CRAN architectures in cellular systems. For example, although this thesis utilizes TCP for reliable I/Q sample streaming, alternative transport protocols such as UDP may be explored in latency-tolerant scenarios to reduce overhead. Similarly, compression in Cloud-LoRa is based on Discrete Wavelet Transform (DWT). However, the compression framework is adaptable and can be extended to other signal families or protocols. Techniques from cellular CRANs, which offer lower loss and higher fidelity reconstruction, are promising candidates for future integration.

While Cloud-LoRa demonstrates strong performance in handling low-SNR links through joint cloud-side decoding, not all end devices experience similar channel conditions. In real-world deployments, devices are unevenly distributed, some in close proximity to gateways and others much farther away. The hybrid architecture proposed in this thesis allows gateways to make intelligent, link-aware decisions: packets received with high SNR can be decoded locally using CIC or BYOG, while weaker packets can be compressed and relayed to the cloud for joint decoding. This approach maximizes throughput while extending network coverage, making LoRa deployments more robust to heterogeneous link qualities and spatial diversity.

A key improvement introduced during the development of BYOG was the self-dechirping activity detection module, which achieves a significantly lower false positive rate (3%) compared to the 40% rate observed in

Cloud-LoRa's earlier superDC-based detection. The self-dechirping-based activity detection introduced in BYOG, developed after Cloud-LoRa, can seamlessly replace Cloud-LoRa's superDC-based detection module. By significantly reducing false positives, it enables more accurate identification of active periods, thereby improving compression efficiency and reducing unnecessary data transmission to the cloud.

Across all systems, network throughput has been the primary evaluation metric, as it directly reflects the system's ability to deliver user data to the cloud or local gateway. While throughput is especially well-suited for evaluating CIC, additional performance indicators are relevant for the other systems. For example, packet decode success rates at the gateway for BYOG and in the cloud for Cloud-LoRa offer deeper insight into decoding accuracy and system reliability across deployment scenarios.

The flexibility of the Cloud-LoRa framework also lays the foundation for my future research. It is capable of integrating decoding algorithms beyond CIC and BYOG, and while designed around LoRa, its architecture can be extended to other unlicensed LPWAN protocols, such as Sigfox. A key advantage of Cloud-LoRa is its decoupling of hardware and software: minimal, generic SDR-based gateways stream raw RF data, while all computation and intelligence are centralized in the cloud. This paradigm represents a significant shift for LPWANs—away from rigid, hardware-bound solutions and toward fully programmable, software-defined architectures. Similarly, BYOG's lightweight and flexible techniques can potentially be extended to other RF front ends as well.

Moving forward, I plan to improve Cloud-LoRa system, since, this framework lays the foundation for my next goals. In the future, the global IoT ecosystem is expected to expand to billions of devices[1], which will generate an immense amount of data. To connect such devices LPWANs are critical. However, a key barrier to the widespread adoption of LPWAN technologies in smart applications is the need for dedicated gateways for

each protocol, which drives up infrastructure costs. Moreover, the current tight coupling of hardware and software in LPWANs presents a major challenge, hindering the ability to efficiently mine this data and generate actionable insights across such a large scale of interconnected devices. The hardware-focused design also limits the physical infrastructure's ability to adapt to future technological advancements. I plan to integrate several tiers of LPWAN technologies in a cloud based CRAN architecture. I want to significantly reduce the cost of LPWAN infrastructure deployment. In the following sections, I present my future steps toward enabling smart and seamless connectivity for large-scale IoT.

Toward cloud based unified gateways

The Cloud-LoRa framework forms the basis for the next phase of my research. In the future, the global IoT ecosystem is expected to expand to billions of devices[1], which will generate an immense amount of data. To connect such devices LPWANs are critical. However, a key barrier to the widespread adoption of LPWAN technologies in smart applications is the need for dedicated gateways for each protocol, which drives up infrastructure costs. Moreover, the current tight coupling of hardware and software in LPWANs presents a major challenge, hindering the ability to efficiently mine this data and generate actionable insights across such a large scale of interconnected devices. The hardware-focused design also limits the physical infrastructure's ability to adapt to future technological advancements.

My future research aims to develop an intelligent, multi-protocol gateway that unifies different low-power wireless technologies using hardware-software co-design. Most existing works use software-defined radios (SDRs) to propose programmable wireless solutions. However, the limited capabilities of commercial-off-the-shelf (COTS) SDRs prevent them from effectively decoupling hardware and software for a unified protocol framework. To support such a unified framework, my future work will

explore a custom wideband radio front-end design capable of capturing all necessary channels. I also plan to explore highly programmable FPGA designs that provide enhanced channelization, adaptive filtering (particularly for weak signals near the noise floor), and dynamic ADCs to accommodate varying sensitivities for each protocol. Additionally, the front-end must be intelligent in order to dynamically configure hardware in real-time and adapt to varying operational conditions. Toward this, I will explore efficient learning algorithms deployed on both FPGAs and the cloud for adaptive resource allocation across multiple flexible gateways. Such a framework will open the door for cognitive connectivity, where end devices and gateways intelligently manage connection based upon available resources.

8 CONCLUSION

In this thesis, I have proposed novel solutions to improve the throughput and scalability of LoRa Networks while preserving the low power and long range features offered by LoRa modulation.

To improve network throughput at local gateways, I proposed, designed, and deployed CIC, a novel demodulation technique at the LoRa receiver to decode multiple colliding packets concurrently. CIC cancels out interfering symbols by selecting the optimal set of sub-symbols. It incorporates prior work on LoRa packet collisions as additional features. CIC implemented in real outdoor and indoor deployments using COTS devices shows significant improvements in network throughput.

To make LoRa networks more flexible and scalable, I also 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. We also developed ACCIO, an *RL-based adaptive compression* technique, whose compression threshold adapts to variations in backhaul characteristics and LoRa signal characteristics in real-time.

Finally, I propose lightweight LoRa decoding algorithm, that allows local gateways to scale to multiple channels. I designed, and deployed BYOG, a software-programmable LoRaWAN gateway capable of processing 10 LoRa channels simultaneously using OTS SDR hardware. We propose and implement self-dechirping, an SF-agnostic, lightweight algorithm for LoRa preamble detection and SF estimation. Through extensive real-world evaluation, we show that CIC, Cloud-LoRa and BYOG significantly enhance the performance of standard LoRa networks.

BIBLIOGRAPHY

[1] Cisco. Cisco and sas edge-to-enterprise iot analytics platform.

[2] LoRa. `https://www.semtech.com/lora`.

[3] 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.

[4] 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.

[5] 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.

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

[7] Fei Tao, Qinglin Qi, Ang Liu, and Andrew Kusiak. Data-driven smart manufacturing. *Journal of Manufacturing Systems*, 48:157–169, 2018.

[8] Harsha V Madhyastha and Chinedum Okwudire. Remotely controlled manufacturing: A new frontier for systems research. In *Proceedings of the 21st International Workshop on Mobile Computing Systems and Applications*, pages 62–67, 2020.

[9] Aloÿs Augustin, Jiazi Yi, Thomas Clausen, and William Mark Townsley. A study of lora: Long range & low power networks for the internet of things. *Sensors*, 16(9):1466, 2016.

[10] 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.

[11] Martin C Bor, Utz Roedig, Thiemo Voigt, and Juan M Alonso. Do lora low-power wide-area networks scale? In *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 59–67, 2016.

[12] Brecht Reynders and Sofie Pollin. Chirp spread spectrum as a modulation technique for long range communication. In *2016 Symposium on Communications and Vehicular Technologies (SCVT)*, pages 1–5. IEEE, 2016.

[13] LoRaWAN ADR. `https://lora-developers.semtech.com/documentation/tech-papers-and-guides/understanding-adr/`.

[14] Nicolas Sornin and Ludovic Champion. Signal concentrator device, October 17 2017. US Patent 9,794,095.

[15] 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*, SIGCOMM '17, page 309–321. Association for Computing Machinery, 2017.

[16] 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.

[17] Xiong Wang, Linghe Kong, Liang He, and Guihai Chen. mlora: A multi-packet reception protocol in lora networks. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2019.

[18] Shuai Tong, Zhenqiang Xu, and Jiliang Wang. Colora: Enabling multi-packet reception in lora. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 2303–2311. IEEE, 2020.

[19] 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.

[20] 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*. IEEE, 2021.

[21] 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.

[22] 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*, SenSys '21, page 56–68, New York, NY, USA, 2021. Association for Computing Machinery.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

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

[28] Yujun Hou, Zujun Liu, and Dechun Sun. A novel mac protocol exploiting concurrent transmissions for massive lora connectivity. *Journal of Communications and Networks*, 22(2):108–117, 2020.

[29] Daniel Jay Koch, Muhammad Osama Shahid, and Bhuvana Krishnaswamy. Spreading factor detection for low-cost adaptive data rate in lorawan gateways. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, SenSys '22, page 918–924, New York, NY, USA, 2023. Association for Computing Machinery.

[30] Dimitrios Zorbas, Khaled Abdelfadeel, Panayiotis Kotzanikolaou, and Dirk Pesch. Ts-lora: Time-slotted lorawan for the industrial internet of things. *Computer Communications*, 153:1–10, 2020.

[31] Khaled Q. Abdelfadeel, Dimitrios Zorbas, Victor Cionca, and Dirk Pesch. free —fine-grained scheduling for reliable and energy-efficient data collection in lorawan. *IEEE Internet of Things Journal*, 7(1):669–683, 2020.

[32] Aakriti Jain, Md Ashikul Haque, Abusayeed Saifullah, and Haibo Zhang. Burst-mac: A mac protocol for handling burst traffic in lora network. In *2024 IEEE Real-Time Systems Symposium (RTSS)*, pages 148–160, 2024.

[33] Gonglong Chen, Wei Dong, and Jiamei Lv. Lofi: Enabling 2.4ghz lora and wifi coexistence by detecting extremely weak signals. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, page 1–10. IEEE Press, 2021.

[34] 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*, MobiCom '20, New York, NY, USA, 2020. Association for Computing Machinery.

[35] Raghav Subbaraman, Yeswanth Guntupalli, Shruti Jain, Rohit Kumar, Krishna Chintalapudi, and Dinesh Bharadia. Bsma: scalable lora networks using full duplex gateways. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, MobiCom '22, page 676–689, New York, NY, USA, 2022. Association for Computing Machinery.

[36] Anna Triantafyllou, Panagiotis Sarigiannidis, Thomas Lagkas, Ioannis D. Moscholios, and Antonios Sarigiannidis. Leveraging fairness in lorawan: A novel scheduling scheme for collision avoidance. *Computer Networks*, 186:107735, 2021.

[37] Basem M. ElHalawany, Ahmed Gamal Abdellatif, Seham Ibrahem Abd Elkarim, Ola Mohammed Ali, M.M. Elsherbini, Sameh Zarif, and Mahmoud A. Shawky. Expert-driven multi-armed bandit approach for spreading factor allocation in lorawan. *Physical Communication*, page 102755, 2025.

[38] Aleksandra Checko, Henrik L Christiansen, Ying Yan, Lara Scolari, Georgios Kardaras, Michael S Berger, and Lars Dittmann. Cloud ran for mobile networks—a technology overview. *IEEE Communications surveys & tutorials*, 17(1):405–426, 2014.

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

[40] GNU Radio. `https://www.gnuradio.org/`.

[41] RPPO. `https://github.com/rpp0/gr-lora`.

[42] Adafruit Feather M0 with RFM95 LoRa Radio. `https://www.adafruit.com/product/3178`.

[43] RadioHead RFM95 Library. `https://www.airspayce.com/mikem/arduino/RadioHead/classRH_RF95.html`.

[44] Orestis Georgiou and Usman Raza. Low power wide area network analysis: Can lora scale? *IEEE Wireless Communications Letters*, 6(2):162–165, 2017.

[45] Jiangbin Lyu, Dan Yu, and Liqun Fu. Achieving max-min throughput in lora networks. In *2020 International Conference on Computing, Networking and Communications (ICNC)*, pages 471–476. IEEE, 2020.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 8 channel LoRa Gateway. `https://www.adafruit.com/product/4327`.

[51] 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.

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

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

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

[55] 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.

[56] 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.

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

[58] 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.

[59] 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.

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

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

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

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

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

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

[66] 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.

[67] 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.

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

[69] 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.

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

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

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

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

[74] RTL-SDR. https://www.rtl-sdr.com/.

[75] HackRF One. https://www.adafruit.com/product/3583.

[76] Muhammad Osama Shahid, Daniel Koch, Jayaram Raghuram, Bhuvana Krishnaswamy, Krishna Chintalapudi, and Suman Banerjee. Cloud-LoRa: Enabling cloud radio access LoRa networks using reinforcement learning based Bandwidth-Adaptive compression. In *USENIX Symp. on Networked Systems Design and Implementation* (*NSDI 24*).