BYOG : Multi-Channel, Real-time LoRaWAN Gateway testbed using general-purpose Software Defined Radio

MUHAMMAD OSAMA SHAHID, University of Wisconsin-Madison, USA
BHUVANA KRISHNASWAMY, University of Wisconsin-Madison, USA

Adaptive Data Rate (ADR) is used by multi-channel LoRaWANs to meet the demanding capacity needs of LoRa networks. The network server running ADR in each channel determines the optimum data rate and assigns the appropriate spreading factor for each LoRa device to maximize the network throughput. This in turn requires the gateway to be capable of receiving LoRa packets of all possible spreading factors. Existing gateways achieve this by using multiple RF front ends, increasing the overall cost and complexity. In this work, we propose BYOG (Bring Your Own Gateway), a LoRaWAN receiver that can receive and decode 10 channels simultaneously in real-time. Towards this pipeline, we develop *self-dechirping*, an SF-agnostic packet detection algorithm that also detects the spreading factor of the packet. This computationally lightweight algorithm can be implemented on any general-purpose software-defined radio, bringing down the cost and ease of LoRaWAN gateway implementations. BYOG will enable research and development in LoRaWAN ADR. Using experimental, real-world datasets, we show that the proposed algorithm can detect the spreading factor accurately and operate over a wide range of SNRs using three different SDRs (RTL-SDR, HackRF One, USRP B210). BYOG performs as well as a high-end LoRaWAN gateway in terms of network throughput.

CCS Concepts: • **Networks** → **Link-layer protocols**.

Additional Key Words and Phrases: LoRa, Spreading Factor (SF), Self-Dechirping, BYOG

ACM Reference Format:
Muhammad Osama Shahid and Bhuvana Krishnaswamy. 2024. BYOG : Multi-Channel, Real-time LoRaWAN Gateway testbed using general-purpose Software Defined Radio. *Proc. ACM Netw.* 2, CoNEXT2, Article 10 (June 2024), 17 pages. https://doi.org/10.1145/3656299

1 INTRODUCTION

LoRa has been developed to address the growing needs of low-power wide area networks (LPWAN) that can reach long distances and have long battery life. The distributed nature of LoRa has rendered itself to be one of the most adopted LPWAN technologies for outdoor as well as indoor deployments [1–3]. This wide adoption has resulted in dense deployments with many low-power wireless devices connecting to a LoRa gateway [4–6]. In order to manage the increasing number of devices in the last-mile, LoRaWAN has been proposed as a medium access control built on top of LoRa. As illustrated in Figure 1 [7], LoRaWAN network architecture consists of LoRa end nodes communicating to a LoRaWAN gateway (GW in Fig. 1), which then connects to LoRa Network Server through traditional backhaul [8]. The network server in turn decodes the messages and has the option to send acknowledgments and other control messages back to the LoRa end devices.

Authors' Contact Information: Muhammad Osama Shahid, University of Wisconsin-Madison, Madison, USA, mshahid2@wisc.edu; Bhuvana Krishnaswamy, University of Wisconsin-Madison, Madison, USA.

ACM 2834-5509/2024/6-ART10
https://doi.org/10.1145/3656299

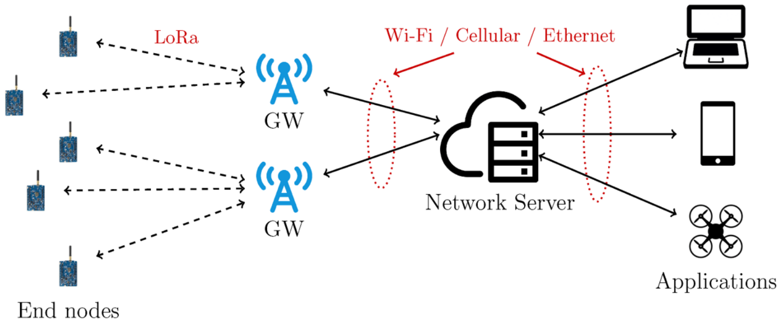


Fig. 1. LoRaWAN network architecture

To meet the growing capacity needs of LoRaWAN, Adaptive Data Rate (ADR) has been proposed [9]. ADR improves network capacity by allowing multiple LoRa transmitters to send at different datarates within the same network. In a LoRaWAN, where devices are spread around the gateway, ADR enables devices closer to the gateway to operate at a higher datarate and those farther away at a lower datarate. Such a mechanism would improve the overall network capacity since devices are not competing with each other for resources. The datarate of a LoRa transmitter is determined by the two parameters viz., Spreading Factor (SF) and Bandwidth (BW). In a fixed datarate network, all the transmitters and the gateway use the same SF and BW. In a network using ADR, the transmitters can choose the most appropriate SF and therefore multiple SFs are supported in a network. Unlike protocols like WiFi, where the packet header is always sent at a known datarate and the datarate of the rest of the packet is mentioned in the header, LoRa transmitter uses a predetermined SF to send the entire packet, including the header. Therefore, the receiver/gateway and the transmitter must agree on the SF and BW used for the communication. Due to this, in order to implement ADR, a LoRaWAN gateway must be capable of processing multiple SFs in parallel.

This need to receive multiple SFs parallelly is one of the key reasons for the limited capacity of LoRaWAN gateways. For every LoRa channel, the gateway continuously searches for a preamble corresponding to all possible SFs (SF7,8,..12) and hence the number of parallel preamble searches increases by 6X [10]. Despite the availability of 64 LoRa channels with 125 kHz and 8 channels with 500 kHz, most LoRaWAN gateways offer 8 channels [11–13] due to the high computational complexity of running ADR (with some exceptions of 64 gateways costing over $3000 [14]). Utilizing all the channels would help improve the network throughput significantly. Additionally, the computational complexity restricts researchers from implementing real-time ADR on software-defined radios (SDR), in turn, posing a hindrance to research and development in LoRa ADR [15].

In this work, we develop BYOG (Bring Your Own Gateway), a software, real-time LoRaWAN receiver that can run 10 channels simultaneously on general-purpose SDRs. We propose a lightweight, SF-agnostic preamble-search algorithm that reduces the complexity by 6 times in a single channel as that of the standard preamble-search algorithm.

The software implementation of single-channel LoRa physical layer led to major breakthroughs in LoRa packet collision resolution [16–18], machine learning-assisted LoRa receiver [19, 20], among others [21–23]. Software implementation of multi-channel LoRaWAN gateway can kickstart significant progress in ADR. However, the major bottleneck towards a software, multi-channel gateway is the lack of hardware-software platforms that can leverage the entire band of 64 channels. This is in turn caused by the computational complexity of searching for LoRa packets (preambles) of all SFs [10]. To address this limitation, we propose **self-dechirping**, an SF estimation algorithm that detects the presence of a LoRa preamble without prior knowledge of the transmitter's SF. We leverage the repeated upchirps in a LoRa packet header and the relationship between the chirps of varying SFs to determine the presence of a LoRa preamble as well as estimate the SF of the packet.

We design the software pipeline that performs channelization and SF estimation in real-time on RF samples streamed using off-the-shelf low-cost (RTL-SDR [24]), medium-cost (HackRF [25]), and high-cost (USRP B210 [26]) SDRs. Any SDR that can stream raw RF samples to a computer can be used to build our own LoRaWAN gateway with ADR capabilities. To the best of our knowledge, **this is the first multi-channel LoRaWAN gateway that is programmable, can operate in real-time, and implemented on low-cost SDRs**. We believe that the platform built here will enable significant breakthroughs in maximizing LoRaWAN network capacity.

In summary, we make the following contributions in this paper:

- We build a multi-channel programmable LoRaWAN gateway that can capture 10 LoRa channels and decode packets in each of them in real time. Any Software Defined Radio that can stream RF samples to a computer can be used as a LoRaWAN gateway.
- We design and implement **self-dechirping**, an SF-agnostic algorithm that can detect the presence of a LoRa preamble and the corresponding SF without demodulating the packet. This in turn allows us to scale to multiple channels.
- We implement **self-dechirping** and the receiver pipeline using Python on three different SDRs - RTL-SDR, HackRF, and USRP, connected to a laptop. We show that we can detect all the packets detected by standard LoRa and estimate their SFs with 100% accuracy.
- We perform extensive real-world experiments and show that BYOG can receive from 10 LoRa channels using low-cost SDRs. Our datasets and source code are made publicly available[1].

2 BACKGROUND AND MOTIVATION

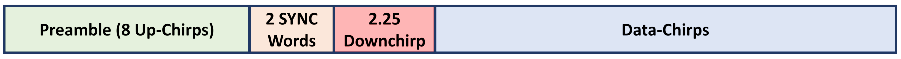


Fig. 2. LoRa packet format : every symbol is transmitted with a predetermined SF

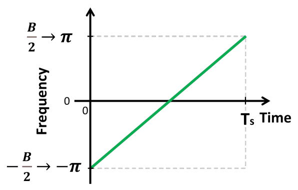


Fig. 3. Time-Frequency variation of base upchirp $C_0(t)$

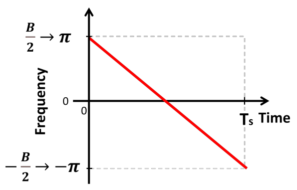


Fig. 4. Time-Frequency variation of downchirp $C_0^*(t)$

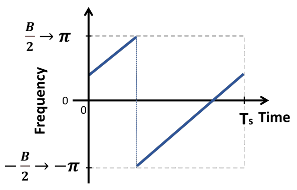


Fig. 5. Time-Frequency variation of Datachirp $C_\phi(t)$

LoRa Modulation. 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. 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 3. A downchrip is the complex conjugate of an upchirp (Figure. 4). LoRa modulates a data symbol by changing the starting frequency of the upchirp (Figure 5). The upchirp and data chirp can be represented mathematically as below.

[1] https://github.com/UW-CONNECT/BYOG.git

$$C_\phi(t) = C_0(t) \cdot e^{j2\pi f_\phi t}; \tag{1}$$

$$C_0(t) = e^{j2\pi\left(\frac{B^2}{2\times 2^{SF}}t^2 - \frac{B}{2}t\right)}, 0 \le t \le T_s \tag{2}$$

$$f[n] = j2\pi\left(\frac{B}{N}n - \frac{B}{2}\right), 0 \le t \le N, N = 2^{SF} \tag{3}$$

In Eqn 1, $\phi \in \{0, 1, \cdots, 2^{SF} - 1\}$ and $T_s = \frac{2^{SF}}{B}$. The spreading factor, SF that can take values $\in \{7, 8, 9, 10, 11, 12\}$ is fixed for each LoRa packet and dictates the transmission data rate.

LoRa Demodulation. LoRa demodulates by continuously de-chirping the received samples. Dechirping is the process of multiplying a window of I/Q samples with a downchirp, followed by Fourier Transform (FFT). Multiplying with a downchirp results in a sinusoid of frequency equaling the starting frequency of the chirp, which indicates the data symbol. Thus, performing FFT accumulates the energy from the entire window into one FFT bin corresponding to the starting frequency of the data chirp; we see a peak at that frequency in FFT. The length of the dechirping window must be equal to that of the downchirp, to maximally accumulate energy.

LoRa receiver infers the start of a packet on detecting 8 consecutive peaks at the same frequency, corresponding to 8 upchirps in the packet header. Therefore, in order to detect the start of the packet, the receiver continuously dechirps with the known SF. SYNC words then locate the symbol boundary and confirm the onset of a new packet.

LoRaWAN. LoRaWAN architecture, illustrated in Figure 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 [9]. 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** [10], 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.

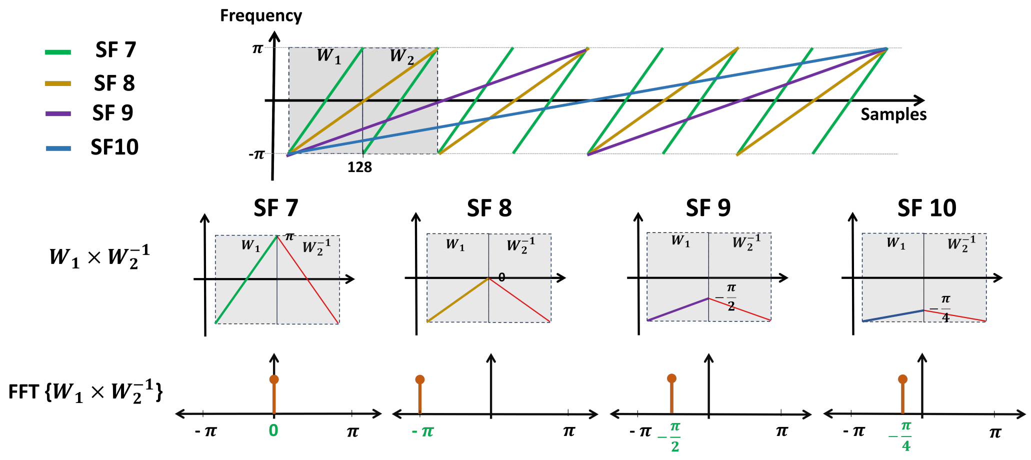


Fig. 6. Self-Dechirping: Top row shows the upchirps of various SFs. The gradient of each upchirp is a scaled function of SF7. Middle row shows our choice of two windows W_1 and W_2 with a fixed size $N_0 = 128$. W_1 and W_2 capture the entire upchirp of SF7, half of SF8, quarter of SF9, and 1/8 th of 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

3 PROPOSED WORK

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

3.1 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. 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, 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 3. 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 4 given sampling rate is equal to LoRa bandwidth. Similarly, Equation 5 gives the time-frequency variation of W_2^{-1} (conjugate of W_2).

$$f_{SF}[n] = j\frac{2\pi}{2^{SF}}n - j\pi, \tag{4}$$

$$f_{SF}^{-1}[n-128] = -j\frac{2\pi}{2^{SF}}n + j2\pi\frac{128}{2^{SF}} + j\pi, 1 \leq n \leq N_0, N_0 = 128 \tag{5}$$

Upon performing self-dechirping ($W_1 \times \mathrm{W}_2^{-1}$), the phase in the exponent of the chirp equations gets subtracted, therefore the frequency of the dechirped signal is given by Equation 6 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.

$$f_{SF}[n] - f_{SF}^{-1}[n-128] = -j\frac{2\pi}{2^{SF-7}} \tag{6}$$

Based on Equation 6, 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.

3.2 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 7 and 8.

$$f_{SF}[n+\tau] = j\frac{2\pi}{2^{SF}}n + j2\pi\frac{\tau}{2^{SF}} - j\pi, \tag{7}$$

$$f_{SF}^{-1}[n-128+\tau] = -j\frac{2\pi}{2^{SF}}n + j2\pi\frac{128}{2^{SF}} + j2\pi\frac{\tau}{2^{SF}} + j\pi, \tag{8}$$

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 stays unaffected. Therefore, the second challenge of aligning with the start of the packet is solved by self-dechirping, as misalignment is cancelled out.

3.3 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.

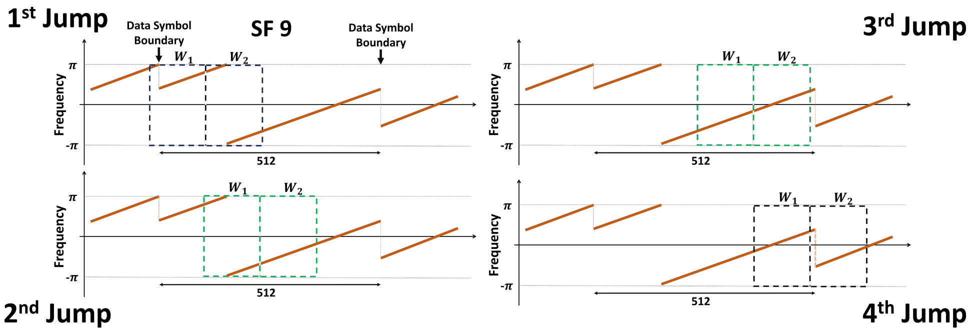
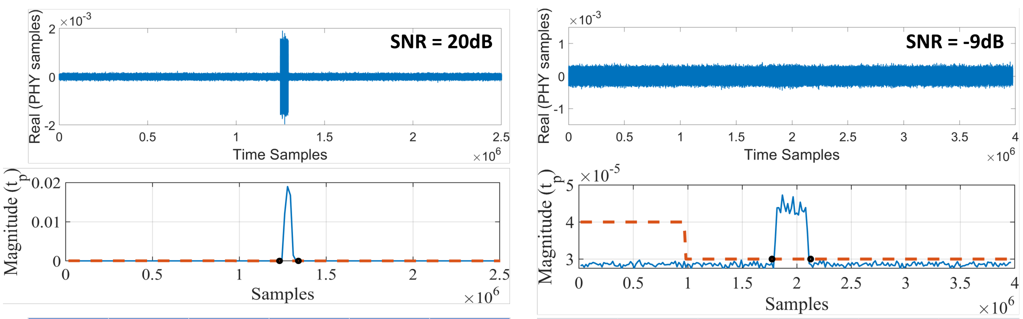


Fig. 7. Impact of self-dechirping on datachirps : Datachirp of an SF9 packet is shown here. In the first and fourth jump, W_1 and W_2 capture two different data chirps and do not yield a peak. In the second and third jump however, the windows capture the same datachirp and hence result in FFT peaks.

For example, in Figure 7, 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.

3.4 Putting It All Together : 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



$\omega = 0$	$\omega = -\pi$	$\omega = -\frac{\pi}{2}$	$\omega = -\frac{\pi}{4}$	$\omega = -\frac{\pi}{8}$	$\omega = -\frac{\pi}{16}$
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. 8. 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

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 8, 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 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

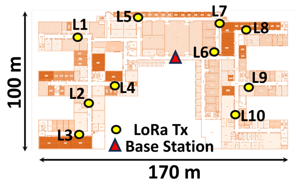


Fig. 9. Deployment Scenario

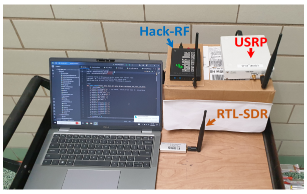


Fig. 10. 3 Software Defined Radios as Base Stations

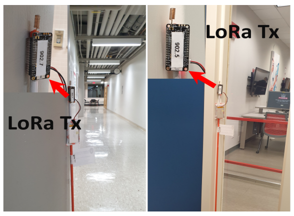


Fig. 11. LoRa Transmitters deployed in the building

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.

4 IMPLEMENTATION

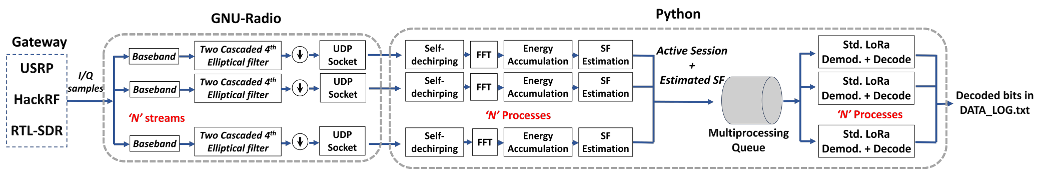


Fig. 12. BYOG implementation framework

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 [27]. We used 10 of these devices as shown in Figure 11, $T_i, i = 1, 2, ...10$ and using Arduino Library RadioHead [28] 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 10). BYOG is implemented on a 12-core, 16GB RAM, Intel i5 laptop. As shown in Figure. 12, 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 [24], 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 12, 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 × 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. 12, 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 [26] and 20 MSamples/s [25] 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, 12, since we initiate $N(whereN = \#ofchannels)$ 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. 9. 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.

5 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?

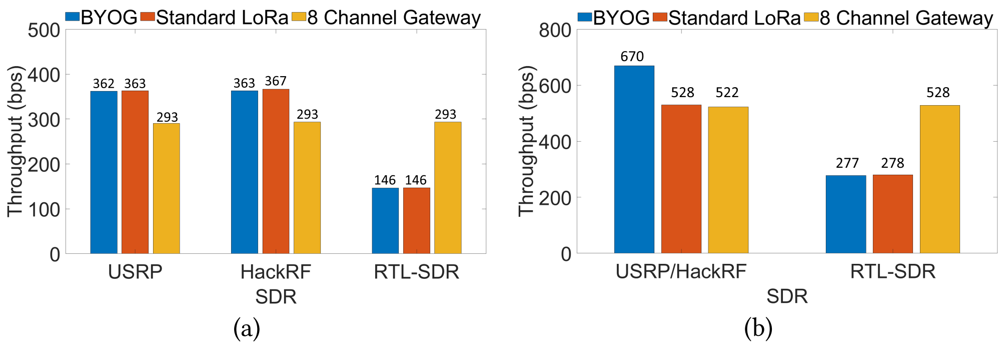


Fig. 13. Throughput of BYOG with different SDRs along with std. LoRa (a) 15% Duty Cycle, (b) 30% Duty Cycle (Emulated traffic)

5.1 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 13(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 13). 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 13).

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 Figure13(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 14. 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.

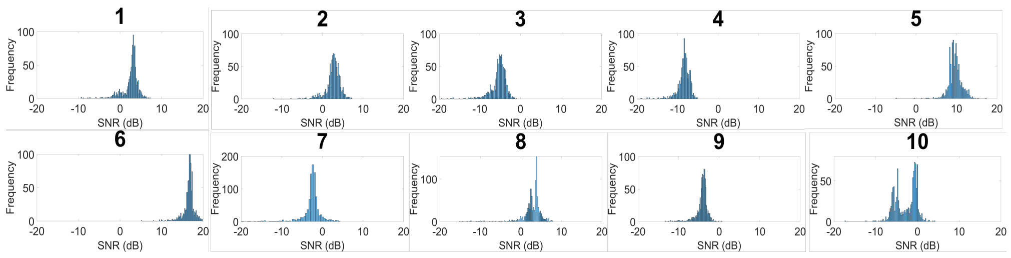


Fig. 14. SNR distribution of packets in each channel

5.2 Per-channel Throughput using BYOG

Following the overall network throughput, we delve deeper into per-channel throughput to understand fairness across channels in Figure 15. We plot the throughput in each of the channels for the three SDRs considered. We observe that the throughput remains relatively the same across channels,

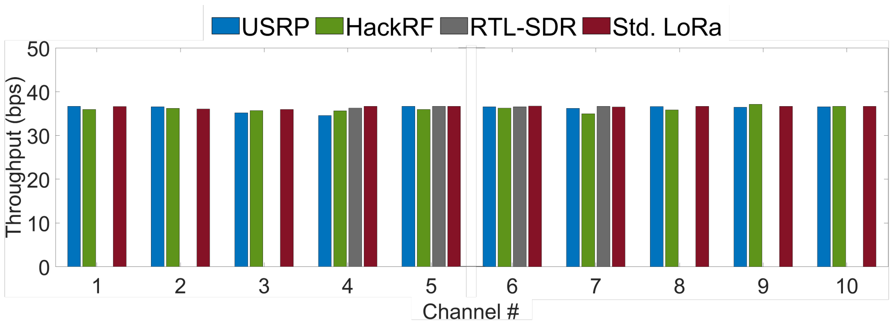


Fig. 15. Average per-channel throughput with BYOG compared against LoRaWAN

depsite the variabilities in the SNR range, as shown in Figure 14. 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.

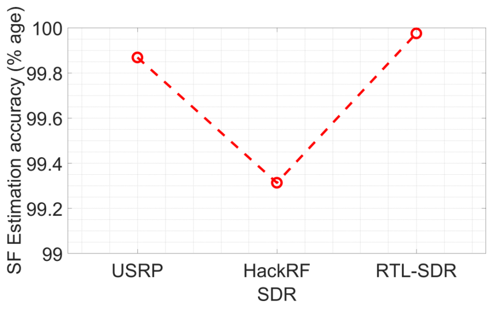


Fig. 16. Accuracy of SF Estimation with different SDRs as front-ends in Multi-Channel Experiment

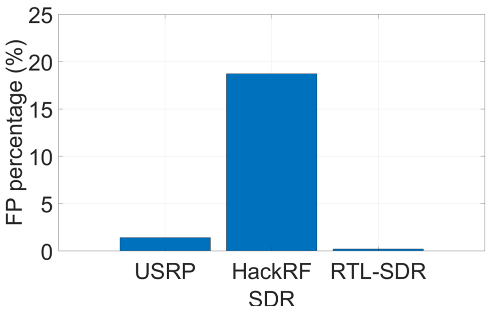


Fig. 17. Percentage of False Positives in Packet Detection in Multi-Channel experiment

5.3 Accuracy of Spreading Factor Estimation

In Figure 16, 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. 17. 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. 8 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.

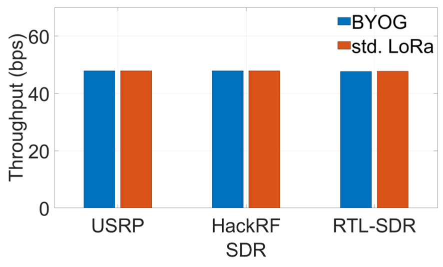


Fig. 18. Throughput in a single 500kHz channel with all SF

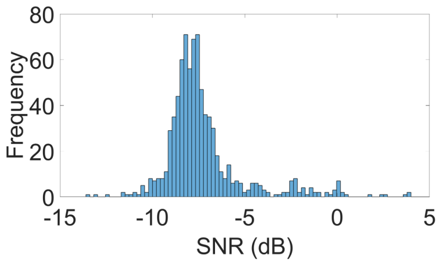


Fig. 19. SNR distribution in the single 500kHz channel

5.4 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. 9 that transmitted packets randomly of SF 7 though 12 at BW 500kHz. Figure 18 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. 19 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 RELATED WORK

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 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. [29], Koch et al. [30] and Cloud-LoRa [31] 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 DISCUSSIONS AND LIMITATIONS

BYOG proposes an SF agnostic, real-time LoRa demodulator and decoder implementation that can scale upto 10 channels. The current limit on the number of channels comes from the number of cores in the laptop running BYOG algorithm. We spawn a new process for each channel and stream upto 10 channels in real-time. We can scale to a maximum 64 channels using a NUC with more cores. Existing LoRa gateways cannot be modified to run other physical layer demodulation algorithms whereas our implementation is flexible to incorporate physical layer modifications that further improve the network scale. We have made our framework open source and we believe that our implementation will help researchers to rapidly deploy and test further improvements to LoRa's PHY layer as well as design new ADR algorithms for LoRaWAN. The real-time, multi-channel operation of BYOG in SDRs enables design and testing of such future algorithms. We believe that implementing BYOG using hardware radio with parallel receiver chains baked into its ASIC each running our SF-agnostic packet detection and decoding will significantly bring down the cost of LoRaWAN gateways; in turn realizing an affordable 64-channel LoRaWAN gateway.

8 CONCLUSIONS

In this work, we design and deploy BYOG , a software programmable LoRaWAN gateway that can process 10 LoRa channels 125kHz each in using general-purpose SDRs. To the best of our knowledge, this is the first real-time implementation of LoRaWAN gateway on SDRs. Towards this design, we address the key limiting factor of preamble search at the gateways. We propose and implement self-dechirping, an SF-agnostic, lightweight algorithm for LoRa preamble detection and SF estimation. Self-dechirping is a novel packet detection algorithm that leverages the relationship between the downchirps of the different SFs to uniquely map the received signal to the appropriate SF. We implement BYOG using Python and perform real-world experiments using three different SDRs as LoRaWAN gateways. We show that they are capable of processing LoRa packets with any SF without any prior knowledge across multiple channels. This work does not raise any ethical issues.

9 ACKNOWLEDGEMENTS

We would like to thank our shepherd and anonymous reviewers for the valuable comments and helping us improve the paper. This work is supported by the National Science Foundation under following grants: CNS-2142978, CNS-2213688 and CCSS-2034415.

REFERENCES

[1] 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.

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

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

[4] Aloÿs Augustin, Jiazi Yi, Thomas Clausen, and William Mark Townsley. A study of lora. *Sensors*, 16(9):1466, 2016.

[5] 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 *MobiCom*, 2019.

[6] Martin C Bor, Utz Roedig, Thiemo Voigt, and Juan M Alonso. Do lora lpwans scale? In *Proceedings of the 19th ACM Int'l Conf. on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 59–67, 2016.

[7] Gaia Codeluppi, Antonio Cilfone, Luca Davoli, and Gianluigi Ferrari. LORAFARM: a LORAWAN-Based smart Farming Modular IoT architecture. *Sensors*, 20(7):2028, 4 2020.

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

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

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

[11] 8 Channel LoRa Gateway. https://www.adafruit.com/product/4327.

[12] LoRa Transceivers. https://www.thethingsnetwork.org/docs/lorawan/transceivers/.

[13] SX1276 DataSheet. https://www.semtech.com/products/wireless-rf/lora-connect/sx1276#documentation.

[14] Tektelic KONA Mega 64-Channel LoRaWAN Gateway. https://www.embeddedworks.net/sens652/.

[15] Pieter Robyns, Peter Quax, Wim Lamotte, and William Thenaers. A multi-channel software decoder for the lora modulation scheme. In *IoTBDS*, pages 41–51, 2018.

[16] Muhammad Osama Shahid, Millan Philipose, Krishna Chintalapudi, Suman Banerjee, and Bhuvana Krishnaswamy. Concurrent interference cancellation: Decoding multi-packet collisions in lora. In *Proceedings of ACM SIGCOMM 2021.*

[17] Xianjin Xia, Ningning Hou, Yuanqing Zheng, and Tao Gu. Pcube: scaling lora concurrent transmissions with reception diversities. *ACM Transactions on Sensor Networks*, 18(4):1–25, 2023.

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

[19] 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 Conf. on Embedded Networked Sensor Systems*, pages 56–68, 2021.

[20] Chenning Li, Xiuzhen Guo, Longfei Shangguan, Zhichao Cao, and Kyle Jamieson. {CurvingLoRa} to boost {LoRa} network throughput via concurrent transmission. In *USENIX NSDI 22.*

[21] 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 Int'l Conf. on Mobile Computing And Networking*, pages 676–689, 2022.

[22] Ningning Hou, Xianjin Xia, and Yuanqing Zheng. Don't miss weak packets: Boosting lora reception with antenna diversities. *ACM Transactions on Sensor Networks*, 19(2):1–25, 2023.

[23] Manan Mishra, Daniel Koch, Muhammad Osama Shahid, Bhuvana Krishnaswamy, Krishna Chintalapudi, and Suman Banerjee. OpenLoRa: Validating LoRa implementations through an extensible and open-sourced framework. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1165–1183, 2023.

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

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

[26] USRP B210. https://www.ettus.com/all-products/ub210-kit/.

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

[28] RadioHead Arduino Library. https://greatscottgadgets.com/hackrf/one/.

[29] 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.

[30] 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.

[31] 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).*

Received December 2023; revised January 2024; accepted March 2024