Beyond Sensing: A High-Performance Software-Defined LoRa Gateway

Vincenz Mechler
TU Darmstadt, Germany
vmechler@seemoo.tu-darmstadt.de

Matthias Hollick
TU Darmstadt, Germany
mhollick@seemoo.tu-darmstadt.de

Bastian Bloessl
TU Darmstadt, Germany
bbloessl@seemoo.tu-darmstadt.de

Abstract

LoRa is about to become the standard for Low-Power Wide-Area Networks (LPWANs), being well suited for many Internet of Things (IoT) and sensor network applications. The success of the technology sparked interest to adopt LoRa for other, more challenging use-cases in industrial automation or control of cyber-physical systems. Regular LoRa gateways are, however, limited in their number of parallel demodulation paths and restricted to a single network. To overcome this limitation, we implement a software-defined, multichannel LoRa receiver that is able to receive all common spreading factors and uplink channel combinations of the LoRaWAN EU868 frequency plan. In addition, our receiver does not rely on hard-coded sync words, enabling a mechanism similar to monitor mode in WLAN networks. Experimental evaluation using Software-Defined Radios (SDRs) confirms superior performance compared to commercial LoRa gateways.

CCS Concepts

• **Networks** → **Network experimentation**.

Keywords

LoRa Gateway, LoRaWAN, Software-Defined Radio

1 Introduction

LoRa is a proprietary Low-Power Wide-Area Network (LPWAN) physical layer technology that employs Chirp Spread Spectrum (CSS) modulation, which is very robust against noise and interference [7]. Due to the computationally inexpensive modulation and narrow-band characteristics, LoRa transceivers are cheap and exhibit exceptionally low power consumption. This makes them ideal for battery-powered wireless sensor nodes distributed over large areas.

The Spreading Factor (SF), a central parameter of the LoRa physical layer, determines the symbol duration and robustness of the encoding. Higher values yield exponentially higher symbol duration but also increased receiver sensitivity. While multiple transmissions using different SFs are not perfectly orthogonal to each other, they interfere very little at similar Signal to Noise Ratios (SNRs) [3].

These unique characteristics of the LoRa physical layer led to rapidly expanding adoption and growing interest in the research community. Its success motivated a number of use cases that adopt the technology for applications that go beyond sensing, including industrial automation [8] and emergency communication [1, 15].

LoRa end devices are usually connected in a LoRaWAN network, with one or multiple gateways covering a specific geographical area and connecting to the end devices in a star topology. The gateways are interconnected via a central server or cloud infrastructure. A regional frequency plan defines recommended transmission channels, which can be combined with the available semi-orthogonal SFs to enable many collision-free transmissions in parallel.

Commercial LoRa gateways are, however, limited in their number of parallel demodulation paths [10], whereas Software-Defined Radio (SDR)-based LoRa receivers usually cover only a single channel and SF. They are, therefore, not suited to experimentation or deployment in scenarios where the required data rates or number of end devices necessitate the simultaneous use of many channels and SFs.

To overcome this limitation, we present a software-defined LoRa gateway capable of receiving on multiple channels and SFs simultaneously, achieving a very high Packet Reception Rate (PRR) even under high channel utilization, and reaching a higher performance than previous software-defined LoRa transceivers. In addition, we add a mechanism to receive all packets independently from their sync word value (a field in the preamble used for network partitioning), similar to WLAN's monitor mode. Our implementation is published as Open Source software as part of FutureSDR.[1]

ACM MobiCom '24, November 18–22,2024, Washington D.C., DC, USA

ACM ISBN 979-8-4007-0489-5/24/11
https://doi.org/10.1145/3636534.3697317

[1] https://github.com/futuresdr/futuresdr

2 Background and Related Work

In this work, we investigate the performance of LoRa gateways under high channel utilization and compare our software-defined multichannel receiver against a state-of-the-art commercial hardware gateway.

While the regional parameters vary depending on the area of deployment, we focus on the eight 125 kHz uplink channels of the EU868 frequency plan. Our results are, however, transferable to all common regional parameters.

2.1 Scalability

As the number of end devices per network and the number of networks deployed in an area increase, the channel utilization rises. Additionally, networks and deployments can also be designed with the coexistence of different channels and SFs in mind. This opens up new use cases for which the capacity of a single LoRa link would not be sufficient.

Therefore, the scalability of LoRa networks has come into the focus of researchers in recent years. Employing simulations of very large networks, these studies highlight the need for sophisticated network planning to avoid or at least minimize packet collisions, i.e., concurrent transmissions on the same channel and SF [5, 8, 11, 13].

However, it has also been noted that the limited number of demodulation paths in a commercial gateway can be a major limiting factor under high channel utilization since it limits the number of concurrently receivable packets far below the theoretical maximum of available channel and SF combinations [6]. While this limitation is clearly documented in the specification of commercial gateways (e.g., Semtech SX1302 [10]), relevant implementation details like prioritization of packets of different SFs arriving in parallel are not known.

With our controlled experiments on a Semtech SX1302-based LoRa gateway, we demonstrate this limitation and provide first insights into the packet-dropping behavior of commercial gateways under high channel utilization.

Furthermore, this limit of commercial gateways is merely the result of a cost-benefit analysis since massively concurrent transmissions were rare in traditional and sparsely deployed LoRa networks. With the recent developments in the deployment and new use cases of LoRa networks, it might be beneficial to relax this limitation, even if it incurs higher hardware costs for the gateways.

While simulations can give first insights into this matter, the research community lacks the tools necessary for practical experimentation and verification. All commercial gateways are limited in their number of parallel demodulation paths, with the 16 paths of the SX1302 chip already being the highest number available. With our software-defined gateway implementation, we enable the reception of parallel transmissions on all channel and SF combinations (8 channels × 6 SFs = 48 in case of the EU868 frequency plan), providing unprecedented receive capabilities.

2.2 Software-Defined LoRa Transceivers

While we are the first to propose an efficient software-defined LoRa gateway capable of decoding all channels and SFs simultaneously, there has been much previous work on providing software defined LoRa transceiver capabilities.

Robyns et al. released the first fully reverse-engineered software defined LoRa transceiver for the popular GNU Radio SDR framework in 2017 [9]. Frame detection and synchronization is performed using a modified version of the Schmidl-Cox algorithm. However, the receiver sensitivity is comparably low, achieving 100 % PRR only at 20 dB SNR, as opposed to the negative SNR possible with commercial LoRa receivers.

Based on this, Tapparel et al. developed an improved receiver implementation employing a different synchronization algorithm to achieve receiver sensitivity comparable to commercial LoRa devices [12]. The synchronization algorithm, originally proposed by Xhonneux et al. [14], iteratively estimates sampling time and frequency offsets, achieving a good approximation at low overall complexity. This is generally considered the state-of-the-art implementation for LoRa on SDR due to its superior performance, together with the high accessibility due to the integration into the GNU Radio framework.

A recent implementation by Busacca et al. [2] also provides full transceiver capabilities and integrates an automatic repeat request mechanism to improve reliability. Extensive experimentation using the Colosseum real-time channel emulator evaluates the receiver sensitivity under noise and packet collisions and explores additional mechanisms like interference cancellation and end device localization. However, the receiver can only monitor a single channel and SF, and scalability is not addressed. Furthermore, the implementation uses the Python programming language, which is not intended for compute intensive real-time processing, and does not integrate directly with any SDR framework.

We, in turn, based our implementation on the LoRa transceiver by Tapparel et al. [12]. We extend the synchronization procedure to no longer require prior knowledge of the sync word and implement an efficient multichannel receiver covering all channels and SFs. Our software defined LoRa gateway can decode transmissions on all channels and SFs simultaneously, even at a channel utilization of 100 %. Furthermore, our implementation uses the Rust-based FutureSDR[2] framework instead of GNU Radio, bringing improved portability, fine-grained control over the scheduler, and improved processing speed, as we demonstrate in our evaluation.

[2] https://www.futuresdr.org/

3 SDR-based LoRa Node

Our software defined LoRa transmitter and receiver are based on the work by Tapparel et al. [12]. We ported the GNU Radio implementation to our Rust-based FutureSDR framework, improved the synchronization algorithm, and optimized the execution speed.

3.1 LoRa Packet Structure

Figure 1 shows the general structure of a LoRa packet. The preamble starts with unmodulated upchirps, encoding the value 0. The sync-word, which is essentially a sub-network ID encoded in the preamble to isolate networks from each other, is encoded into two consecutive symbols. The preamble is delimited by 2.25 downchirps, which mark the beginning of the header. Header and payload symbols are encoded as modulated upchirps.

3.2 General Receiver Design

Our software-designed receiver is split into blocks performing the sequential operations necessary to decode symbols, as can be seen in Figure 2.

The FrameSync block performs packet detection by continually demodulating windows with stride and length of one symbol duration. Once the necessary number of consecutive symbols encoding the same value has been observed, a packet is detected, and the block performs synchronization of time and frequency offsets.

The FFT-Demod block demodulates the now-aligned windows of samples into coded and interleaved symbols. The following blocks reverse the various coding steps of the transmitter, namely gray mapping, interleaving, and hamming coding. The header decoder intercepts the beginning of the frame, calculates the coding rate, payload length, and other parameters encoded in the header, and informs the FrameSync block of the expected packet length. It further

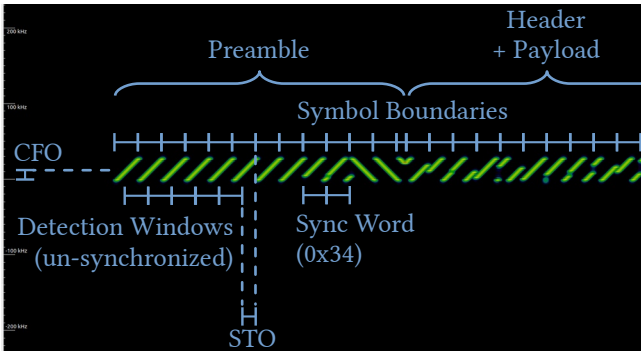

Figure 1: LoRa packet with annotated symbol boundaries and receiver offsets. Detection is performed without synchronization in time or frequency.

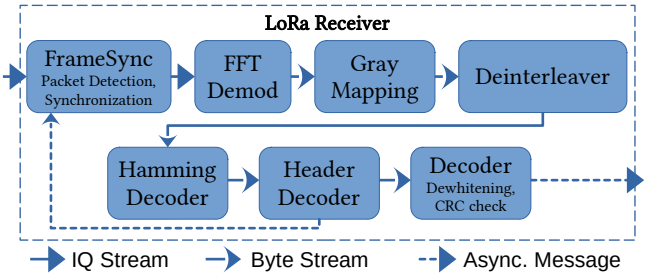

Figure 2: Software-defined LoRa receiver structure, adopted from [12].

checks the header checksum, and initiates a reset if the check fails. After this, it forwards the appropriate number of payload symbols, before expecting the next header.

Finally, the payload symbols are de-whitened and delivered to the application.

3.3 Packet Detection and Synchronization

The detection window is not necessarily aligned to the symbol boundaries, both in time and frequency. It may experience Sampling Time Offset (STO) and Carrier Frequency Offset (CFO), as illustrated in Figure 1. Additionally, Sampling Frequency Offset (SFO) might cause misalignment over time.

The CSS modulation encodes values by shifting the base chirp in frequency or, due to its linear nature and resistance to aliasing, equivalently in time. Therefore, each potentially misaligned detection window during this part of the preamble will contain a modulated upchirp encoding the same value. Furthermore, this decoded value can be used as a first estimate of the STO, which is measured in sample durations: an STO of one shifts the observed symbol value by one and can be compensated simply by re-aligning the demodulation window by one sample.

The synchronization algorithm by Xhonneux et al. [14] is used to estimate and compensate the STO, CFO, and SFO values iteratively. This algorithm is a low-cost approximation since the offsets are all interrelated, and joint optimization is too costly for a real-time receiver. Therefore, the offsets are individually estimated one after the other, using the early rough estimate of the STO to achieve a better estimation of the frequency offsets.

3.4 Additional Sampling Time Offset Correction

Finally, Tapparel et al. [12] introduce an additional step of correcting the STO in case it is misaligned after the iterative offset correction.

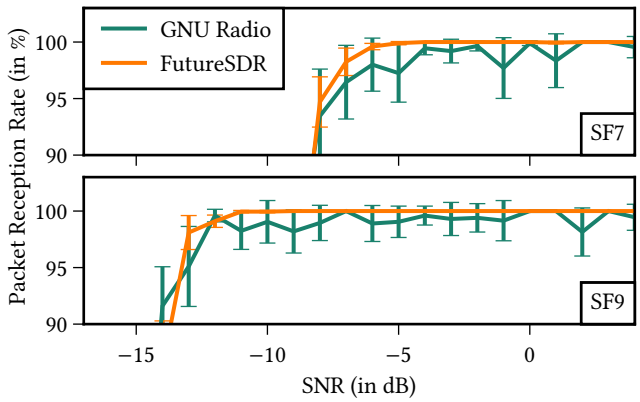

Figure 3: Receiver sensitivity without prior knowledge of the sync word, with (FutureSDR) and without (GNU Radio) improved sampling time offset correction. For SF7 and SF9, simulated over 2000 packets per sampling point with random sampling time offset.

3.4.1 Sync Word Re-Synchronization. For this, they compare the expected value of the sync word with the observed value after the initial offset correction.

The sync word is an 8 bit value split into two 4 bit nibbles, each represented by a symbol in the preamble. For robustness, each nibble is left-shifted by 3 bit prior to modulation since the preamble employs none of the error correction techniques used for the payload.

Due to the potentially inaccurate iterative offset estimation, the demodulation of the sync word might be affected by a small remaining STO. If both sync word symbols deviate from the expected values by the same margin and not more than ±2, the STO is adjusted by this amount.

3.4.2 Sync Word Independent Re-Synchronization. While Tapparel et al. [12] rely on the knowledge of the sync word value, we instead exploit the syntactical structure of the encoded sync word to perform this action.

The left shift spaces all possible values as multiples of 8, thus making it possible to identify a remaining STO of up to ±3 without prior knowledge of the sync word encoded in the preamble. In practice, correct primary synchronization will yield a remaining STO of not more than ±1. This makes it possible to perform this action without knowing the actual value of the encoded sync word.

While it is possible to deactivate this re-synchronization procedure entirely in the GNU Radio implementation from [12], without this step the iterative offset approximation algorithm incurs occasional synchronization failures even at relatively high SNR values, as can be seen in Figure 3. In this and the following plots, the error bars indicate the confidence interval of the mean for a confidence level of 95 %. Our improved algorithm, on the other hand, does not rely on prior knowledge of the sync-word value. Therefore, it can reliably decode all packets that arrive with an SNR value within the receiver sensitivity, even if the sync word values are not known beforehand.

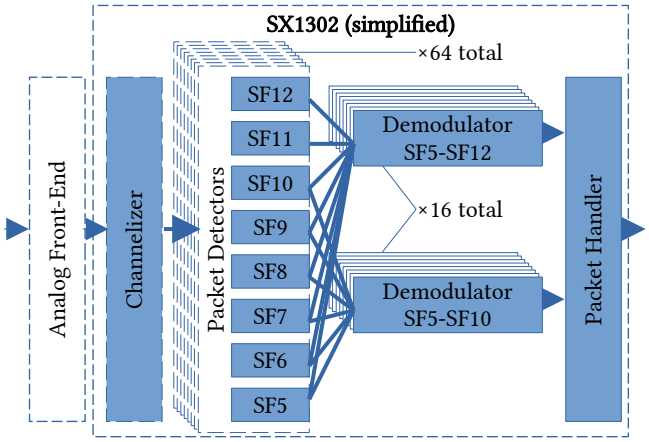

Figure 4: Simplified architecture of the Semtech SX1302 LoRa gateway baseband processor derived from [10]. 64 packet detectors for the eight supported SFs and up to eight configurable channels. 16 parallel demodulation paths in total, eight supporting SF5 to SF10, eight supporting all eight SFs.

4 Commercial LoRa Gateways

Commercial LoRa gateways are available from different vendors, but all are based on the Semtech concentrator chip series. The gateways support three independent decoding chains: (1) a Frequency Shift Keying (FSK) receiver, (2) a high-speed LoRa decoder that supports up to 500 kHz channels, and (3) a multichannel LoRa receiver that can decode all SFs on eight 125 kHz channels in parallel. While the first two types of links support higher throughput and are mainly intended for communication between gateways, the multichannel receiver is designed for communication with LoRa end devices.

We focus on the multichannel receiver since SDR implementations for FSK and single-channel LoRa are readily available, and the heavily parallelized multichannel, multi-SF receiver presents a challenge. Here, there previously were no SDR implementations available, and the Semtech chips have limitations, degrading performance in scenarios with higher network load.

A schematic overview of the receiver is depicted in Figure 4. It channelizes the incoming signal into eight 125 kHz channels, each with packet detectors for all SFs. Once a packet is detected, it is forwarded to a decoder. While there are packet detectors for all channel and SF combinations, the number of decoders is limited.

In this paper, we examine the Semtech SX1302, which is a state-of-the-art model that offers 16 decoders, the highest number available on the market. Eight decoders support all SFs, while the other eight are limited to SFs five to ten. This number should be considered with regard to the maximum number of 64 possible parallel receptions (eight channels, eight SFs). These demodulation paths are shared between all packet detectors and present a bottleneck when a large number of packets arrive in parallel.

4.1 Gateway Saturation

To demonstrate this effect, we connect a LoRa gateway based on a Semtech SX1302 concentrator chip to an Ettus Research B200mini SDR. Figure 5 shows the setup. The devices are connected via cable to improve reproducibility, minimize interference, and comply with regulatory restrictions regarding spectrum utilization.

To evaluate the receive performance of the gateway, we synthesize complex baseband traces with varying spectrum utilization and transmit them with the SDR. To this end, we generate 30 s traces that contain frames on eight channels each with SFs from seven to twelve. The payload size of the transmitted packets is fixed to 64 Byte, with explicit header mode and coding rate 7/8. Per channel and SF, we calculate the number of frames required for a given channel utilization and distribute them randomly but without overlap in the time window. Note that frames with different SFs may well overlap on the same channel while still being perfectly decodable.

The results are shown in Figure 6, where we plot the PRR of the LoRa gateway for different channel loads, with equal utilization on all channels and SFs. At around 20 % channel load, packet bursts start overloading the gateway occasionally, leading to packet loss and a decreasing reception rate.

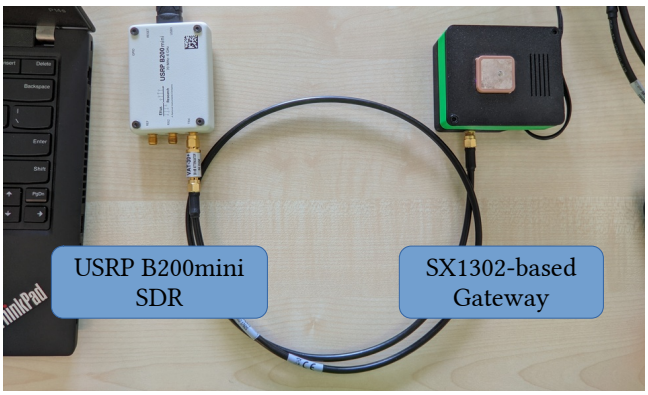

Figure 5: Evaluation setup for the hardware gateway, comprised of a Semtech SX1302 chip integrated into a LoRa hat on a Raspberry Pi Zero, connected to a USRP B200mini SDR.

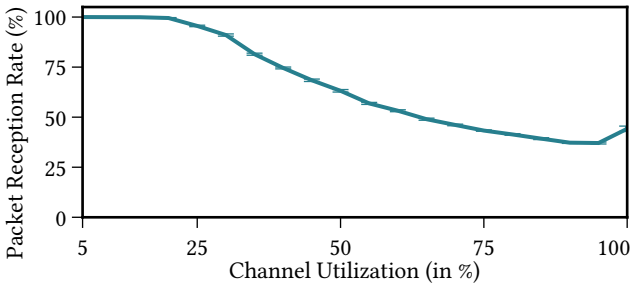

Figure 6: PRR of the hardware gateway over channel load. Average over 20 traces of 30 s each.

With higher loads, this effect becomes more pronounced with a drop in the reception rate to around 33 % as the gateway is overloaded more frequently. This number can be explained by the 16 available packet decoders for up to 48 concurrent transmissions (eight channels with six SFs). Approaching a channel load of 100 %, we can, furthermore, see a slight increase again, which was unexpected to us at first.

To better understand this effect, we evaluated the utilization of the decoders per SF, i.e., the time the decoders are occupied with packets of a given SF. The results are shown in Figure 7, where we plot the normalized air time of frames over the channel load. Since we send frames with six SFs on eight channels in parallel, the normalized transmitted air time approaches 48. However, as we only transmit complete packets, the actual air time for each channel and SF might be slightly below the target channel load.

Again, the results show the saturation of the 16 decoders for higher channel loads, but they also indicate the time spent on decoding packets with a given SF. At lower channel utilization, the gateway distributes the available resources roughly equally between the SFs. However, since SFs eleven and twelve can only be processed by eight of the 16 decoders, these receive slightly fewer resources. We can see that shorter frames gain precedence when approaching a fully loaded channel with frames back-to-back. While implementation details of the gateway are unknown, we believe that this is because frames with lower SFs have shorter symbols (a factor of 32 between SFs seven and twelve) and can trigger frame detection much faster. Therefore, a decoder that becomes free is more likely to pick up a frame with a lower SF. This behavior explains the increased reception rate at high loads in Figure 6, since spending proportionally more time with shorter frames increases the number of received frames. The time spent to decode a single frame with SF12 could be used to decode 18 frames with SF7, for example. (While the symbols of frames with SF7 are 32 times shorter, each individual symbol decodes fewer bits, resulting in a factor of $\approx$18 in terms of throughput.)

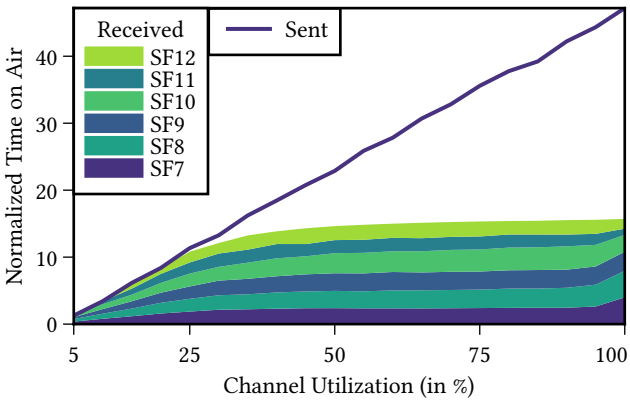

Figure 7: Normalized reception time per SF over channel load for hardware gateway. Averaged over 20 traces of 30 s for each channel utilization. At a channel utilization of 100 % on all 48 channel–SF combinations, the transmitted time-on-air approaches 48.

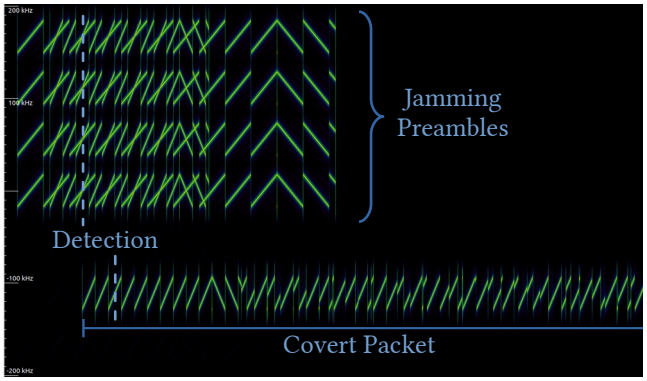

Figure 8: Spectrogram of an RF trace of a covert LoRa transmission. Eight jamming packets are transmitted on four adjacent channels. An additional transmission on a different channel starts after the gateway has detected the jamming packets.

4.2 Limitations of Commercial Gateways

With these experiments and detailed evaluations of commercial LoRa gateways, we have an unprecedented understanding of their implementation and performance, particularly in scenarios with high channel load. For traditional LoRa networks and lower penetration rates, these scenarios were rare, and the cost of over-provisioning the receiver hardware to cope with occasional bursts was likely not economical. However, following recent trends, these limitations might have a more significant impact in the future. Furthermore, the current generation of commercial gateways is not well suited to study and experimentally evaluate these new scenarios. To highlight this limitation, we discuss several implications of the current gateway design.

4.2.1 Denial of Service. First, an attacker could cause outages by simply keeping the decoders busy with bogus frames. This enables an efficient Denial-of-Service (DoS) attack since it is sufficient to occupy the available demodulation paths in the gateway rather than jamming all channels with the necessary power level. This was also found by simulation results, where the primary cause for packet loss under high-load scenarios was identified to be an overloaded receiver, rather than interference [6].

We, furthermore, found that it is enough to send a header to keep the decoders busy for the rest of the frame, similar to the truncate after preamble attack on Wi-Fi and Zigbee by Gvozdenovic et al. [4]. Due to LoRa's high symbol duration, one header can block a demodulation path for multiple seconds using a high SF, making the attack even more efficient.

4.2.2 Covert Transmissions. Looking at this from a slightly different angle, we can also use this behavior to establish a covert channel, hiding transmissions from a commercial gateway. We demonstrate this by generating signals with eight or 16 concurrent transmissions, depending on the number of available decoders for the SF of the frame that we want to hide, and send it with a slight delay on an unoccupied channel. The resulting spectrogram can be seen in Figure 8, which shows time on the x-axis and frequency on the y-axis. In this case, eight frames with SF11 and SF12 are sent on the upper four channels. Since there are only eight decoders available for these SFs, the delayed frame with SF11 is invisible to the gateway. However, a gateway that does not listen to the upper four channels or even a LoRa node that is limited to a single channel and SF could receive it if configured accordingly. This scheme is, furthermore, not limited to a single covert frame, as more channels and SF combinations are available.

In these experiments, we constructed the situations deliberately. Yet, with the increasing density of LoRa deployments and new applications, such a situation could also happen in the real world, leading to performance issues. Using a static, periodic channel access scheme could even lead to situations where nodes experience systematic outages, despite high SNR and no interference on the given channel.

5 Software-Defined LoRa Gateway

To overcome this issue and study LoRa networks with a high channel load in greater detail, we created a software-defined LoRa gateway that does not have the restrictions of commercial LoRa gateways. To this end, we use the single-channel, single-SF receiver to create a flexible multichannel version that can cover all common LoRaWAN configurations and receive packets on all channels and SFs simultaneously. Received packets can optionally be forwarded to a server using

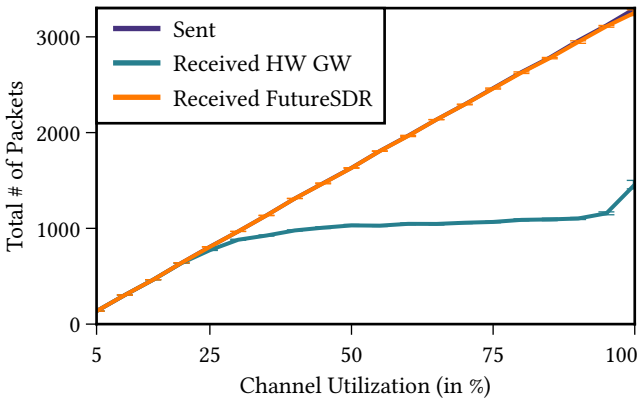

Figure 9: Total number of received packets over channel load for hardware and software gateway. Our software-defined implementation receives the vast majority of the sent packets, even at high duty cycles.

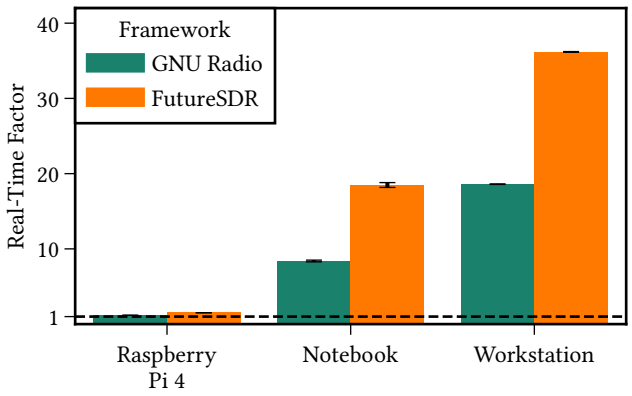

Figure 10: Execution speed of our software-defined multichannel gateway over multiple hardware platforms and in comparison to an equivalent GNU Radio implementation, as multiple of real-time. Measured over 100 iterations of 30 s transmissions.

Semtech's UDP-based protocol. At the moment, it is designed specifically for the EU868 frequency plan, comprising eight 125 kHz channels with a channel spacing of 200 kHz, each supporting SFs between seven and twelve.

The combination of bandwidth, channel spacing, and the fact that the receiver is designed with an integer oversampling factor makes channelization non-trivial. We tried various approaches but found a channelizer using a Polyphase Filter Bank (PFB) with an additional resampler for each channel to be the most efficient. The PFB channelizer is an efficient algorithm that separates a large number of channels with minimal runtime overhead. While this design is limited to evenly spaced channels with fixed bandwidth, this is common also for other LoRaWAN frequency plans.

The channelizer splits the signal into eight 200 kHz channels, and each channel is upsampled by a factor of 2.5 with a rational resampler, which is also based on a PFB. This results in a 500 kHz signal per channel, i.e., an oversampling factor of 4. For each channel, the signal is fed into a number of parallel packet detectors, one for each SF.

5.1 Software-Defined Gateway Performance

To demonstrate the advantage of our software-defined LoRa gateway, we conduct the same experiments as with the commercial hardware gateway and measure the number of received packets at different channel loads. To measure the performance of the SDR receiver, we connect the SDR via cable, as we did with the commercial LoRa gateway.

The results are depicted in Figure 9. They clearly show that our multichannel receiver does not experience overloading and instead achieves near-perfect reception even for a fully loaded channel. The few misses at higher loads occur randomly due to interference from frames with different SFs. While different SFs are nearly orthogonal to each other, concurrent frames on the same channel still contribute to the interference level. With interference from the five higher SFs, the least robust configuration might fail occasionally. This is, however, a normal effect of the system and not an issue of our receiver implementation. Even with this effect, our software-defined gateway can decode close to all packets with arbitrary channel access patterns, clearly outperforming the commercial gateway and overcoming the limitations described in Section 4.2.

5.2 Runtime Complexity

When starting the project, it was unclear to us if a software-defined LoRa gateway would be able to run in real-time on a standard desktop PC or even an embedded PC, like a Raspberry Pi. Furthermore, we were curious how efficiently such a complex receiver with many components (≈350 blocks) would work on the GNU Radio and FutureSDR frameworks. To this end, we also implemented an equivalent GNU Radio-based LoRa multichannel receiver. Both implementations use the exact same signal processing algorithms, the same parameters, and the same filter taps.

To measure the runtime complexity of the systems in a reproducible fashion, we generate traces of IQ data with a duration of 30 s for a channel load of 100 % and decode the files. With this approach, both implementations receive the same frames and trigger similar code paths. As long as such a trace can be processed in less than 30 s, the receiver is considered to be real-time capable.

We conducted the experiments on a Raspberry Pi 4, a Lenovo Thinkpad P14s, and a desktop PC with an AMD Ryzen 9 5900X CPU. For each measurement, we performed

100 runs and calculated the real-time factor, i.e., how fast the trace was processed with regards to its duration of 30 s.

The results are shown in Figure 10. They show that all software-defined receivers can run in real time, even though the GNU Radio implementation is barely above one on the Raspberry Pi 4. With this performance, other applications on the same system or interference from background tasks could become problematic in an actual deployment. FutureSDR, in turn, offers a speedup of 30 % on the Raspberry Pi, providing more headroom.

On the PCs, both implementations can run in real-time without problems, leaving ample resources for other tasks or to add a decoding path for FSK or LoRa channels with higher bandwidth, similar to the commercial gateway. On these platforms, the difference between FutureSDR and GNU Radio is more pronounced with a speedup of ≈1 (1.2 for the laptop, 0.95 for the desktop). Relying on FutureSDR, furthermore, benefits from the advantages provided by the framework (e.g., portability and the option to use custom CPU schedulers and hardware accelerators in future extensions of the receiver).

6 Conclusions

In this paper, we presented the first software-defined LoRa gateway, capable of decoding multiple channels and SFs in parallel. In contrast to commercial LoRa nodes and gateways, it does not run into overload situations for networks with high channel utilization. Furthermore, it can decode all frames, independent from the LoRa sync word, providing functionality similar to monitor mode in WLAN networks. Based on an existing single-channel and single-SF implementation, it provides state-of-the-art signal processing performance. We believe that our implementation can serve as an important tool for studying LoRa networks under high channel loads, which become increasingly relevant given the number of deployments and the interest in new use cases of LoRa networks. Our LoRa implementation is released as Open Source software together with examples that demonstrate how to connect it to LoRaWAN and Meshtastic networks.

References

[1] L. Baumgartner, P. Lieser, J. Zobel, B. Bloessl, R. Steinmetz, and M. Mezini. 2020. LoRAgent: A DTN-based Location-aware Communication System using LoRa. In *2020 IEEE Global Humanitarian Technology Conference (GHTC)*. IEEE, Seattle, WA, (Oct. 2020). DOI: 10.1109/GHTC46280.2020.9342886.

[2] F. Busacca, S. Mangione, S. Palazzo, F. Restuccia, and I. Tinnirello. 2024. SDR-LoRa, an open-source, full-fledged implementation of LoRa on Software-Defined-Radios: Design and potential exploitation. *Computer Networks*, 241, (Mar. 2024). DOI: 10.1016/j.comnet.2024.110194.

[3] D. Croce, M. Gucciardo, S. Mangione, G. Santaromita, and I. Tinnirello. 2018. Impact of LoRa Imperfect Orthogonality: Analysis of Link-Level Performance. *IEEE Communications Letters*, 22, 4, (Apr. 2018). DOI: 10.1109/LCOMM.2018.2797057.

[4] S. Gvozdenovic, J. K. Becker, J. Mikulskis, and D. Starobinski. 2020. Truncate after preamble: PHY-based starvation attacks on IoT networks. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (WiSec '20). ACM, Linz, Austria, (July 2020). DOI: 10.1145/3395351.3399356.

[5] A. Lavric. 2019. LoRa (Long-Range) High-Density Sensors for Internet of Things. *Journal of Sensors*, 2019, 1, (Feb. 2019). DOI: 10.1155/2019/3502987.

[6] I. Martinez, P. Tanguy, and F. Nouvel. 2019. On the performance evaluation of LoRaWAN under Jamming. In *2019 12th IFIP Wireless and Mobile Networking Conference (WMNC)*. IEEE, Paris, France, (Sept. 2019). DOI: 10.23919/WMNC.2019.8881830.

[7] B. Reynders and S. Pollin. 2016. Chirp spread spectrum as a modulation technique for long range communication. In *2016 Symposium on Communications and Vehicular Technologies (SCVT)*. IEEE, Mons, Belgium, (Nov. 2016). DOI: 10.1109/SCVT.2016.7797659.

[8] M. Rizzi, P. Ferrari, A. Flammini, E. Sisinni, and M. Gidlund. 2017. Using LoRa for industrial wireless networks. In *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*. IEEE, Trondheim, Norway, (May 2017). DOI: 10.1109/WFCS.2017.7991972.

[9] P. Robyns, P. Quax, W. Lamotte, and W. Thenaers. 2018. A Multi-Channel Software Decoder for the LoRa Modulation Scheme: in *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security*. SCITEPRESS - Science and Technology Publications, Funchal, Madeira, Portugal, (Mar. 2018). DOI: 10.5220/0006668400410051.

[10] Semtech. 2024. SX1302 LoRa Core Digital Baseband Chip. Semtech Corporation. Retrieved July 31, 2024 from https://www.semtech.com/products/wireless-rf/lora-core/sx1302.

[11] M. Slabicki, G. Premsankar, and M. Di Francesco. 2018. Adaptive configuration of LoRa networks for dense IoT deployments. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, Taipei, Taiwan, (Apr. 2018). DOI: 10.1109/NOMS.2018.8406255.

[12] J. Tapparel, O. Afisiadis, P. Mayoraz, A. Balatsoukas-Stimming, and A. Burg. 2020. An Open-Source LoRa Physical Layer Prototype on GNU Radio. In *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, (May 2020). DOI: 10.1109/SPAWC48557.2020.9154273.

[13] F. Van den Abeele, J. Haxhibeqiri, I. Moerman, and J. Hoebeke. 2017. Scalability Analysis of Large-Scale LoRaWAN Networks in ns-3. *IEEE Internet of Things Journal*, 4, 6, (Dec. 2017). DOI: 10.1109/JIOT.2017.2768498.

[14] M. Xhonneux, O. Afisiadis, D. Bol, and J. Louveaux. 2022. A Low-Complexity LoRa Synchronization Algorithm Robust to Sampling Time Offsets. *IEEE Internet of Things Journal*, 9, 5, (Mar. 2022). DOI: 10.1109/JIOT.2021.3101002.

[15] J. Zobel and R. Steinmetz. 2024. Enabling Information Transmission in Low-Throughput Wireless Channels for Aerial Disaster Monitoring Systems. *ISCRAM Proceedings*, 21, (May 2024).

This work has been co-funded by the LOEWE initiative (Hesse, Germany) within the emergenCITY [LOEWE/1/12/519/03/05.001(0016)/72] center, as well as the German Research Foundation (DFG) in the Collaborative Research Center (SFB) 1053 MAKI (Project-ID 210487104). The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the project "Open6GHub" (grant number: 16KISK014).