Combating BLE Weak Links with Adaptive Symbol Extension and DNN-based Demodulation

Yeming Li, Jiamei Lv, Hailong Lin, Yi Gao, and Wei Dong
College of Computer Science, Zhejiang University
{liymemnets,lvjm,linhl,gaoyi,dongw}@zju.edu.cn

ABSTRACT

Bluetooth Low Energy (BLE) is one of the most popular wireless protocols for building IoT applications because of its low energy, low cost, and wide compatibility nature. However, BLE communication performance can be easily affected by interference and blockages because of its low transmission power. This paper presents *BLEW*, a technique to improve the BLE communication performance over weak links by exploiting adaptive symbol extension and DNN-based demodulator to combat channel interference and maximize network throughput. First, we propose a phase peak clustering-based preamble detection method that coherently adds up the phase difference of preambles to combat the interference. We then propose a multi-domain DNN-based demodulator to fully extracts the temporal and spectrum features of the signal and enhance the demodulation performance. Finally, we model the throughput of Commercial Off-The-Shelf (COTS) BLE chips transmitting extended packets, which can be used to optimize the symbol length in an adaptive manner. We implement *BLEW* with USRP B210 and COTS nRF52840 platform. Experiments show that *BLEW* can increase throughput by up to 119.55 Kb/s compared with native BLE over typical weak links. Compared with existing approaches, *BLEW* has up to 25.70% higher preamble detection rate and up to 3.37 dB demodulation gain.

CCS CONCEPTS

• **Networks** → **Wide area networks**; **Mobile networks**; • **Computer systems organization** → **Sensor networks**.

KEYWORDS

BLE, Weak links, Symbol extension, DNN-based demodulator

ACM Reference Format:
Yeming Li, Jiamei Lv, Hailong Lin, Yi Gao, and Wei Dong. 2024. Combating BLE Weak Links with Adaptive Symbol Extension and DNN-based Demodulation. In *ACM Conference on Embedded Networked Sensor Systems (SenSys '24), November 4–7, 2024, Hangzhou, China.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3666025.3699362

1 INTRODUCTION

Bluetooth Low Energy (BLE) is one of the most popular wireless protocols because of its low-power nature and wide adoption in consumer devices such as smartphones and wearables [6, 48]. BLE demonstrates remarkable efficiency in connecting devices within short ranges, providing convenient network access. However, BLE is primarily used indoors, where there are often numerous obstructions and WiFi interference. Additionally, to reduce energy consumption, BLE prefers to use low signal transmission power. Therefore, the performance of BLE is unsatisfactory over weak links, which poses obstacles for BLE applications.

It is promising if BLE can work over weak links. For example, in a health monitoring scenario, an old man wears a smartwatch that reports his heart rate data to the gateway via BLE. If the BLE supports communication over weak links, the smartwatch can still normally transmit emergency messages when the user and the gateway are in different rooms. Another example is that users use smartphones via BLE to remotely control the horns of cars in an underground garage to help the users find the car. The numerous obstacles in a garage (e.g., vehicles and walls) can cause severe BLE signal attenuation. Users can remotely control the horns of cars at a much longer distance, making it easier to find the car.

The existing solution is deploying multi-hop Bluetooth Mesh networks [7]. However, this approach heavily relies on the network infrastructure, which brings significant deployment and maintenance costs. Z. Li *et al.* propose Symphony [34], which achieves long-range communication on Commercial Off-The-Shelf (COTS) BLE devices by demodulation of an extended symbol (i.e., continuously repeating a native BLE symbol) and uses a LoRa receiver to demodulate the extended signal. However, the existing work still cannot ensure BLE transmission performance under weak link conditions. Specifically, they face the following two limitations.

Inaccurate preamble detection at low SNR: BLE uses Gaussian Frequency Shift Keying (GFSK) modulation and each BLE packet starts with a 1-byte preamble (0x10101010). Existing works [31, 34] achieve preamble detection by exploiting the repeating frequency peaks in the Short-Time Fourier Transform (STFT) results. However, the STFT windows near the symbol boundaries contain the sample points from two continuous bits, and the boundaries of symbols in the STFT spectrogram are wide and blurry. Therefore, they often fail to find the actual starting point of the preamble. If the distance between the detected and the actual preamble is larger than half a symbol, the receiver fails to demodulate the following symbols. The error of STFT-based preamble detection is more significant for BLE due to the relatively short symbols.

High demodulation error over weak links: Over the long-range link, even if the receiver can accurately detect the position of each symbol, the symbol can still be corrupted due to high noise. The existing works [8, 31, 38] propose DNN models that filter out the noise in the spectrogram to enhance the LoRa demodulation performance. However, BLE has a much shorter symbol duration than LoRa, i.e., 1 μs for BLE and over 1 ms for LoRa. Besides, the frequency of BLE is

SenSys '24, November 4–7, 2024, Hangzhou, China

ACM ISBN 979-8-4007-0697-4/24/11…$15.00
https://doi.org/10.1145/3666025.3699362

constant during each symbol, whereas the frequency of each LoRa symbol varies linearly across the entire bandwidth. Therefore, short BLE symbols, which carry minimal information, can be efficiently utilized by DNN.

In order to address these limitations, this paper proposes *BLEW*, a novel system to improve the performance of BLE over weak links. The key idea of *BLEW* is to adaptively extend the symbol length to enable demodulation with low SNR and possible interferences. On the receiver side, *BLEW* employs two key techniques. First, to achieve accurate preamble detection, *BLEW* proposes the *phase peak clustering-based preamble detection*. By exploiting the phase difference between consecutive sample points, a *BLEW* receiver coherently adds up the phase difference caused by preamble bits to form a cluster with seven high phase peaks. Second, to effectively decode the BLE symbols even at low SNR levels, we propose *multi-domain DNN-based demodulator*. It simultaneously extracts features of BLE signals from both the frequency and time domains for demodulation. Furthermore, to maintain high throughput for long-range communication, *BLEW* establishes an *channel adaptive throughput optimization* for finding appropriate values for the parameters of the communication system including the symbol extending factor and the packet payload length. *BLEW* requires only software modifications at the transmitters and physical layer modifications at the receivers (e.g., the BLE gateway). Therefore, *BLEW* can be easily applied to a wide range of existing BLE devices with a specially designed *BLEW* gateway.

We implement *BLEW* with USRP B210 and the popular Nordic nRF52840 BLE chip. The experiment results show that *BLEW* can increase up to 119.55 Kb/s throughput than native BLE over weak links. *BLEW* has up to 25.70% higher preamble detection rate than the existing STFT-based method and the DNN-based demodulator can bring 0.71-3.37 dB demodulation gain. The contribution of our work is three-fold:

- We propose *BLEW* to boost BLE performance over weak links. It only performs software modifications at the transmitters and can be seamlessly implemented across a wide range of existing BLE devices with a specially designed *BLEW* gateway.
- We propose two novel techniques: phase peak clustering-based preamble detection, and a multi-domain DNN-based demodulator, resulting in higher preamble detection accuracy and lower symbol demodulation errors.
- We implement *BLEW* with USRP B210 and the Nordic nRF52840. Extensive experiments are conducted and results show the effectiveness of *BLEW* for communications over weak links.

2 PRELIMINARY AND BACKGROUND

2.1 BLE Physical Layer

BLE works on the 2.4GHz ISM band. It has 40 channels and each has 2 MHz bandwidth. BLE utilizes GFSK modulation [6] and each symbol contains one-bit data. The GFSK is a form of continuous phase modulation and each data bit in a physical layer packet is modulated into the baseband signals:

$$I(t) = \cos(\phi(t)),\ Q(t) = \sin(\phi(t)), \tag{1}$$

where the $I(t)$ and $Q(t)$ are the in-phase and quadrature parts of the BLE signal in the time domain, respectively. The GFSK modulation changes the signal frequency by calculating the continuous phase $\phi(t)$ at time t. Before modulation, the data bits are translated into Non-Return Zero (NRZ) code, where the bit 1 and bit 0 are presented as 1 and -1, respectively. The NRZ code of i-th data bit in the physical layer packet is denoted as d_i. The signal phase at time t is the cumulation of all phase changes caused by data bits before, and it is calculated by:

$$\phi(t) = 2\pi h \int_0^t \sum_{i=0}^{\infty} d_i g(\tau - iT_\mathrm{b})\mathrm{d}\tau, \tag{2}$$

where h is the modulation index, which is specified as 0.5 in the Bluetooth specification [6]. T_b is the duration of each data bit. $g(\cdot)$ is the Gauss function for pulse shaping, and the center is located at iT_b. Since the d_i is the NRZ code, the phase increases on transmitting bit 1 and decreases on bit 0. The center frequency of the baseband signal is 0, and bit 1 refers to the positive frequency and bit 0 to the negative frequency. After the GFSK modulation, the I/Q baseband signal in Eq. (1) will be loaded on a carrier wave with center frequency f_c for transmission, and the receiver can remove it to get the baseband signal.

To demodulate the BLE signals, an ideal method is utilizing Fast Fourier Transform (FFT) to estimate the signal frequency [27, 33]. However, it requires specific hardware to accelerate FFT computation. Therefore, the COTS BLE chips utilize phase change to demodulate symbols [57]. Specifically, a phase increase represents bit 1, and vice versa.

2.2 Basic Idea

BLE's short communication range can be attributed to two main factors. Firstly, BLE typically operates at low transmission power due to strict power consumption constraints imposed by most IoT applications. Secondly, the short duration of each BLE symbol makes it susceptible to noise interference. BLE supports two physical layers: 1 Mbps and 2 Mbps, with symbol lengths of only 1 μs and 0.5 μs, respectively. In contrast, certain Low-Power Wide-Area Network (LPWAN) protocols employ longer symbol lengths to enhance resistance against channel interference. For instance, LoRa adopts a 1,024 μs symbol length (SF=7, BW=125 KHz). The basic idea of *BLEW* is to extend the BLE communication range by extending its symbol length. The length of the extended symbol is f_ext times the length of one symbol in the 1 Mbps physical layer. We call the f_ext as *extension factor*. The question at hand is how to efficiently extend symbols with a fixed extension factor.

Previous studies have explored link-layer symbol extension methods. For instance, Heinzer *et al.* [26] and Li *et al* [32] employ Direct Sequence Spread Spectrum (DSSS) to enhance communication range. DSSS maps a few data bits to a long sequence consisting of 0s and 1s. Similarly, the BLE-coded channel [6] utilizes pattern mapping to extend one bit up to four bits. More specifically, bit 1 extends to 0b1100 and bit 0 to 0b0011. However, despite these methods, each bit in the long sequence is still 1 μs, making it susceptible to noise interference. Fig.1(top) illustrates BLE signal examples that encode the bit1 as 0b01010101. The BLE physical layer operates in 1 Mbps mode. Under high SNR conditions, all symbols are clear.

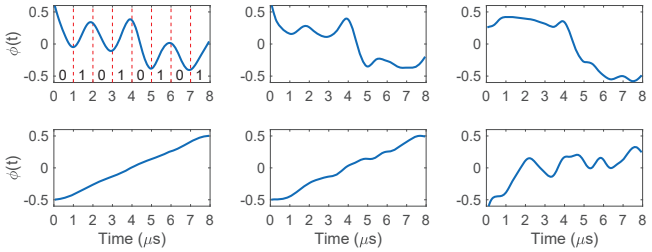


Figure 1: Normalized phase difference while transmitting an extended symbol (1 to 8). By repeating symbols several times, the cumulated phase difference is still distinguishable even with a low-SNR level

However, as the SNR decreases, the entire sequence can become corrupted (Fig. 1(c)(top)).

To address this issue, we propose ***BLEW***, a method that repeats each data bit f_{ext} times to transmit an extended BLE symbol. In this approach, all symbols within one extended symbol collaborate to maximize the phase difference caused by the extended symbol. For example, Fig. 1(bottom) provides examples of BLE transmitting eight consecutive bit 1s to represent an extended bit 1. Even in low-SNR scenarios, we can still observe an increasing phase trend, enabling the demodulation of bit 1. This behavior holds true for FFT-based demodulation as well. The resolution of FFT increases with the sampling time, allowing energy at target frequency bins to be coherently accumulated [46]. Meanwhile, random noise can be mutually canceled out [52].

2.3 Understanding the Problems

To improve BLE performance over weak links, two challenges need to be solved. First, the receiver needs to accurately detect the incoming BLE packets. Second, the receiver needs to correctly demodulate symbols with low SNR levels.

preamble detection: Preamble detection involves two key steps: preamble identification and symbol synchronization. In BLE, the preamble of each packet is represented by the pattern 0xAA (0x55 over the air, LSB first). Preamble identification verifies the presence of the preamble in the raw signal. Symbol synchronization ensures that the demodulation windows are properly synchronized with each symbol, enabling subsequent accurate demodulation of the raw signal. Native BLE receivers (e.g., BTLE [29]) adopt an alternative approach. They directly decode the raw signal and examine the decoded data for the presence of preambles. The matched filter is a state-of-the-art technique used for detecting signals with known shapes. In the case of BLE, the matched filter calculates the correlation between the received signal and the reference signal of the BLE preamble. However, BLE has a relatively large Carrier Frequency Offset (CFO) tolerance (150 KHz [6]), and the CFO can also be affected by factors like temperature variations and receiver hardware imperfections [23]. To handle CFO, this method needs to generate multiple reference signals with different CFOs, which can slow down the preamble detection process.

Fig. 2 provides a simple illustration of preamble detection. There are 14 sample points in total, out of which 10 sample points correspond to a 2-bit preamble. However, this can lead to misalignment between the transmission and demodulation windows. In Fig. 2,

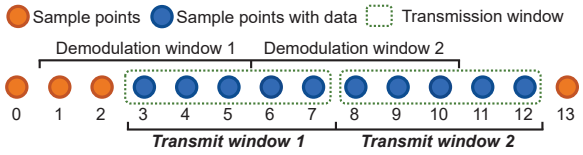


Figure 2: Preamble detection needs to identify the preamble and conduct symbol synchronization. Directly demodulating signals to find the preamble can lead to misalignment of the demodulation window.

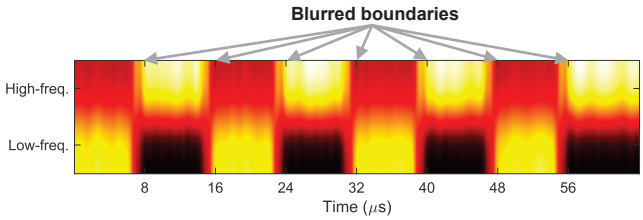


Figure 3: The preamble spectrogram's blurred symbol boundaries can cause symbol synchronization errors.

for example, each demodulation window contains three sample points of the desired symbol, resulting in successful demodulation but failure to accurately symbol synchronization. Consequently, preamble detection becomes a limitation for data demodulation.

Symphony [34] proposes an STFT-based preamble detection method, which identifies preambles based on repeating peaks in the frequency domain. Specifically, by applying STFT to the raw signal, there are energy peaks periodically shifting between high and low-frequency bands over time. While the STFT performs well under low SNR conditions, it can result in a blurred symbol boundary issue. Specifically, the preamble is alternative negative and positive frequency, when the STFT window is near the boundaries of one symbol, it simultaneously contains both the negative and positive frequency. Therefore, the STFT results near the preamble symbol boundaries get blurred. Fig. 3 illustrates an example of wide and blurred boundaries, with an extension factor of 8 and a STFT window size of 2 μs. Symphony uses long symbols (i.e., 160 μs) and the blurred boundary issue can be ignored. However, this issue significantly impacts symbol synchronization accuracy if symbols are short.

Noise-resistent Demodulation: Over weak links, the BLE signal experiences attenuation and is highly susceptible to noise and interference, which significantly impact demodulation performance. Several works [36, 38] have proposed physical layer designs for LoRa based on DNN models. One notable work is NELoRa [31], which introduces a DNN-based noise filter that takes symbol spectrograms as input and reconstructs clean spectrograms. However, the existing DNN-based modulation approach is not applicable to BLE. This method is effective for LoRa due to its longer symbol duration (over 1 ms) and utilization of CSS modulation. The spectrogram of LoRa symbols contains abundant information suitable for the DNN model. In contrast, BLE symbols are much shorter in duration compared to LoRa symbols, and the frequency remains constant within each BLE symbol. Consequently, the spectrograms of BLE symbols contain less information for accurate demodulation.

Throughput Optimization: At low signal-to-noise ratio (SNR) levels, achieving high throughput can be challenging. There is a

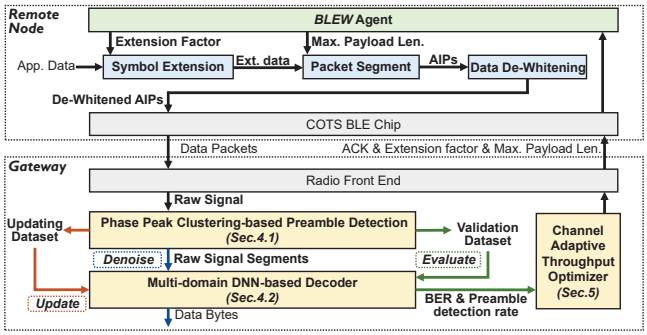


Figure 4: System overview

trade-off that needs to be considered in order to find the optimal parameters. Specifically, the symbol extension can significantly improve the preamble detection and demodulation performance. However, the long symbol extension can reduce the physical-layer symbol rate. Besides, the extended symbols have to be encapsulated into multiple BLE Air Interface Packets (AIPs), which include transmission overhead caused by mandatory fields such as headers and the Inter-Frame Size (IFS).

3 *BLEW* OVERVIEW

Fig. 4 shows an overview of the architecture of *BLEW*, in which the remote node continuously uploads data to the gateway.

Remote node: The remote node is built on the COTS BLE chips and includes three key modules: symbol extension, packet segment, and data de-whitening. *BLEW* specifies the maximum payload length and extension factor for each remote node. To transmit data, the remote node first extends the data bits with the extension factor f_{ext}. Specifically, each bit is repeated f_{ext} times to extend the corresponding symbol f_{ext} times. Then, the extended data is segmented, so it can be encapsulated into AIPs and transmitted through the BLE physical layer. Before transmission, data de-whitening is applied to counteract the disruptive effect of data whitening performed by COTS BLE chips. This enables control over the COTS BLE chip to transmit the extended symbols while ensuring compatibility with the existing hardware configuration.

Gateway: Compared with the remote node, the gateway is typically wall-powered and equipped with relatively powerful computing resources, enabling it to successfully demodulate signals even in challenging low SNR conditions. Additionally, it dynamically selects the optimal communication parameters, i.e., the maximum payload length and extension factor, to maximize the system throughput. To achieve these, three modules are required:

(1) Phase peak clustering-based preamble detection: This module is responsible for detecting data packets by identifying the preamble. Subsequently, each detected packet is sliced into symbols, which are then inputted into the multi-domain DNN-based demodulator. Furthermore, this module maintains an updating dataset and validation dataset from the known fields to update the demodulator and evaluate its demodulation performance in practice. The known fields consist of the data bits obtained from the preamble, access address, and even manually added known bits. The details of this module are presented in Sec. 4.1.

(2) Multi-domain DNN-based demodulator: This module incorporates a deep learning model designed to decode BLE symbols, even under challenging low SNR conditions. The demodulator leverages both time-domain and frequency-domain information to efficiently decode data with short BLE symbols. To accommodate varying channel conditions, the DNN model can be updated using the updating dataset provided by the preamble detection module. The details of this module are presented in Sec. 4.2.

(3) Channel adaptive throughput optimizer: The goal of *BLEW* is to maintain high throughput under varying SNR levels. This module is responsible for determining the optimal symbol length. It assesses the performance of the preamble detector and DNN-based demodulator using the validation dataset and gets the preamble detection rate and Bit Error Rate (BER) of different symbol lengths under the current channel condition. Then, we develop a model that captures the behavior of COTS BLE chips when transmitting extended data packets, enabling *BLEW* to find the optimal parameters. This module is extensively discussed in Sec. 5.

4 *BLEW* DEMODULATION

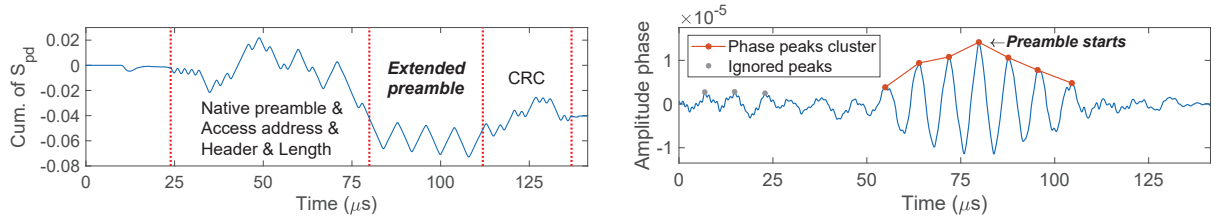
To achieve successful demodulation of symbols in low SNR environments, we first introduce a phase peak clustering-based method for accurate preamble detection. This approach effectively identifies incoming BLE packets and segments the raw signal into symbol-specific segments. Additionally, to enhance the symbol demodulation performance under low-SNR conditions, we propose a multi-domain DNN-based demodulator that utilizes both the time-domain and frequency-domain information of the raw signal to achieve performant demodulation.

4.1 Phase peak clustering-based preamble detection

The preamble detection technique leverages phase differences between successive sample points for precise packet identification and symbol synchronization. Initially, *BLEW* derives the phase difference sequence, denoted as S_{pd}, from the raw signal. Each element in S_{pd} represents the phase difference between the i-th and $(i+1)$-th sample points. A conventional approach involves calculating the phase of each sample point by calculating $\arctan(Q(t), I(t))$ to obtain the phase differences. However, this method faces two significant challenges: First, random noise can induce substantial phase fluctuations even with minor amplitude. Second, real-time preamble detection requires checking all raw signals, which entails substantial computational overhead due to trigonometric function calculations. To address these challenges, *BLEW* computes the amplitude-weighted phase difference, denoted as $A^2\Delta\phi$. Given the brief time interval between successive sample points and the small phase differences involved, $\Delta\phi \approx \sin(\Delta\phi)$. This approximation can be expressed as follows:

$$\begin{aligned}\sin(\Delta\phi) &= \sin(\phi(t+\Delta t) - \phi(t)) \\ &= \sin(\phi(t+\Delta t))\cos(\phi(t)) - \\ &\quad \cos(\phi(t+\Delta t))\sin(\phi(t)) \\ &= I(t)Q(t+\Delta t) - I(t+\Delta t)Q(t),\end{aligned} \tag{3}$$

where the Δt is the sample point interval. Since the in-phase and quadrature signal has the same frequency, we can assume that their



(a) Cumulation of phase difference sequence S_{pd}

(b) Peak-clustering result with channel noise

Figure 5: Preamble detection technique based on phase peak clustering. (a) The accumulation of the S_{pd} of the extended preamble. (b) After coherently adding up the phase change of the extended preamble, there are seven phase peaks centered around the preamble start point.

attenuation is identical. Consequently, the received in-phase and quadrature signals are given by:

$$I'(t) = A\cos(\phi(t)),\ Q'(t) = A\sin(\phi(t)), \tag{4}$$

where A is the attenuation of the signal. The phase difference calculated at the receiver side is $A^2\Delta\phi$. Notably, the calculation presented in Eq. 3 solely involves multiplication and subtraction operations, resulting in minimal computational overhead. Fig. 5a shows the accumulation of the S_{pd} in a high-SNR scenario with $f_{ext} = 4$. The extended preamble is encapsulated in the PDU field of the native BLE packets (detailed in Sec. 5.1). Each symbol is distinguishable, and the phase variations caused by weak noise are effectively suppressed (before 25 μs).

To detect the extended preamble, the intuitive way is to detect the repeating phase change, i.e., phase decreases first and then increases. However, at low-SNR levels, even with symbol extension, a single bit in the preamble can still be easily distorted by noise. To solve this issue, instead of detecting each extended symbol in the preamble individually, we sum up the phase differences across the entire preamble. This approach allows us to accumulate the phase difference caused by BLE symbols coherently while canceling out the random phase differences caused by noise. Consequently, this method generates prominent phase peaks corresponding to the preamble, even in low SNR conditions. Specifically, to detect the eight-bit preamble, *BLEW* employs a preamble kernel denoted as K and convolves it with the phase difference sequence. The length of the preamble kernel is $8 \times f_{ext} \times n_s$, where n_s represents the number of sample points in each native BLE symbol. Each item in K corresponds to a sample point in the extended preamble, and its value is the NRZ code of the data bit. Specifically, the first $f_{ext}n_s$ items in K are set to -1, the next $f_{ext}n_s$ items are 1, and so on.

The outcome of this convolution is a sequence representing the cumulative phase difference (S_c), which contains multiple phase peaks (Fig. 5b). Among all the phase peaks, some of them are caused by the preamble but the others are caused by channel noise. *BLEW* has to accurately find the start of the preamble and filter out those irrelevant phase peaks. The key idea for phase peak filtering is to cluster phase peaks according to their time differences and remove those clusters with a small number of peaks. Specifically, the repeating phase change can cause seven phase peaks, with a dominant peak at the center, three smaller peaks on the left side, and three on the right. From the center to two sides, the peaks mean the preamble kernel aligned with eight, six, four, and two bits in the preamble, respectively. The number of sample points between two peaks does not exceed $12 \times f_{ext} \times n_s$. In contrast, peaks generated by noise and other data bits are scattered in time. To filter out these irrelevant peaks, *BLEW* clusters the peaks detected in the S_c. The peak clusters associated with the preamble share two key characteristics: (1) their width is shorter or equal to $12 \times f_{ext} \times n_s$, and (2) they consist of up to seven peaks. Only peak clusters meeting these conditions are recognized as caused by a BLE preamble, while the others are discarded. Regarding symbol alignment, the central peak within the cluster indicates that the preamble kernel aligns entirely with the phase difference sequence of the preamble, and the corresponding sample point can be identified as the start of the preamble. Determining the receiving windows for subsequent symbols is also feasible, as their lengths are known. Since *BLEW* leverages phase differences across all sample points, it can precisely pinpoint the sample point at which the preamble begins. Compared to the existing STFT-based preamble detection method, our method is more efficient as it does not require STFT.

4.2 Multi-domain DNN-based Demodulator

After the preamble detection, *BLEW* slices the detected packet into symbols that are to be demodulated. There is information available for demodulation in both the time domain (i.e., phase difference) and the frequency domain (i.e., frequency deviation). A DNN model contains multiple layers to fully extract features from the input [31]. There are existing works that use DNN models to understand noise patterns for noise cancellation [58, 59]. However, in wireless communication, noise patterns can vary significantly, making it inefficient for DNN models to recognize and remove every noise pattern effectively. Fortunately, there are only two different BLE symbols (i.e., bits 0 and 1), allowing our DNN model to focus on recognizing the two signal patterns and demodulating the data bits. We introduce a multi-domain DNN-based demodulator that utilizes both time-domain and frequency-domain information. It addresses the limitation of the spectrogram of BLE symbols (i.e., contains little information), and enables high-performance demodulation, even in low SNR environments.

The architecture of our DNN-based demodulator is illustrated in Fig. 6 and comprises four modules: a symbol transform, temporal and spectrum feature extractors, and a demodulator.

(1) Symbol transform: To decode the data bits from the received BLE symbols, our model takes the dual-channel spectrogram and temporal signal as input. First, we normalize the raw signal and

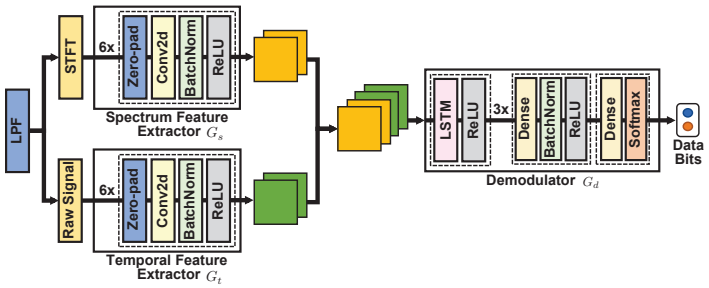


Figure 6: Architecture of multi-domain DNN-based demodulator.

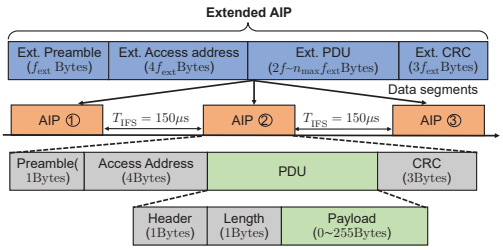


Figure 7: Extended AIP format and scheduling.

apply a digital Butterworth Low-Pass Filter (LPF) to remove high-frequency noises in the baseband signal. For the temporal signal input, we concatenate the real and imaginary parts of the raw signal $x[n]$ and the filtered signal $x_f[n]$ as follows:

$$z_t = [\mathcal{R}(x[n]), \mathcal{J}(x[n]), \mathcal{R}(x_f[n]), \mathcal{J}(x_f[n])]\,. \tag{5}$$

Then we reshape it to a four-channel 2-D array, mapping $x[n], x_f \in \mathbb{C}^{n_s \times f_{ext}}$ to $z_t \in \mathbb{R}^{4\times n_s \times f_{ext}}$. The collection of the temporal signal input is denoted as $X_t = \{z_t^i\}_{i=0}^M$, where the M represents the total number of samples. Intuitively, the DNN-based demodulator can just take the filtered raw signal as the input. However, the digital filter (i.e., Butterworth in our implementation) can cause some distortion to the raw signal, such as the transient response at the beginning and phase shift. We use both the filtered signal and the original signal as inputs so that the DNN-based demodulator can fully exploit the information in the temporal signal.

Then, we apply STFT to the filtered signal $x_f[n]$ and concatenate the real and imaginary parts to get a dual-channel 2D spectrogram:

$$\begin{aligned} STFT(m,\omega) &= \sum_{n=-\infty}^{+\infty} x_f[n]H[n-m]e^{-j\omega n}, \\ z_s &= [\mathcal{R}(STFT(m,\omega)), \mathcal{J}(STFT(m,\omega))]\,, \end{aligned} \tag{6}$$

where $H[n-m]$ is the *Hann* window whose center is m. $\mathcal{R}$ and $\mathcal{J}$ represent the real and imaginary part of the data. The step of the STFT is n_s. The STFT window size and the number of frequency bins are the same and denoted as $n_s \times n_w$, where n_w is a configurable parameter to control the STFT window size. We retain only the frequency bins within the range of -1~1 MHz, corresponding to $2 \times n_w$ frequency bins on either side of zero frequency. This maps the $x_f[n] \in \mathbb{C}^T$ to $X_s \in \mathbb{R}^{2\times 2n_w \times f_{ext}}$. The collection of spectrograms is denoted as $X_s = \{z_s^i\}_{i=0}^M$. We set the $n_w = n_s/2$ so that the size of temporal and spectrum inputs are the same.

(2) Temporal/spectrum feature extractor (G_t, G_s): We employ two six-layer DNNs to extract features from the temporal and spectral inputs, respectively. Both of these feature extractors utilize 2-D convolutional layers to capture temporal and spectral features. The outputs from these two feature extractors are concatenated and fed into the demodulator.

(3) Demodulator G_d: The demodulator is a DNN-based binary classifier that demodulate data bits. It uses a LSTM to further extract temporal features and uses four dense layers to demodulate data.

To train the DNN-based demodulator, we first let the BLE chip transmit known data and collect the signal at a high SNR level as the ground truth Y. Then, we enhance the generalization capabilities of our DNN model by training it with a variety of synthesized BLE symbols, encompassing different SNR levels. Specifically, we generate random Gaussian white noise and achieve fine-grained SNR control by adjusting the amplitudes of the collected raw signal and the added noise. It is worth noting that Gaussian white noise is sufficient for simulating cross-technology interferences. This is because wireless protocols are engineered to transmit Pseudo-Noise (PN) signals. For example, the DSSS used in WiFi convolutes the baseband signal with a PN sequence, mapping the 1 MHz baseband signal to an 11 MHz pseudo-noise signal [13]. Finally, we train the DNN-based demodulator using the synthesized signal traces employing the Binary Cross-Entropy (BCE) loss function:

$$\mathcal{L} = \mathrm{BCE}\left(G_d(G_s(X_s) \oplus G_t(X_t)), Y\right), \tag{7}$$

5 CHANNEL ADAPTIVE THROUGHPUT OPTIMIZATION

BLEW extends the physical layer symbols to combat channel noise, but this also significantly reduces data throughput. Therefore, a throughput optimization method is required. In this section, we first present the details of *BLEW* packet format and then model the throughput for *BLEW*. We finally propose an optimization method to find the best extension factor.

5.1 *BLEW* Packet Format

Fig. 7 shows the *BLEW* AIP format. In the following paper, we call the AIP in Bluetooth specification as native AIP and the packet after symbol extension as extended AIP. The extended AIP shares the same format as the native one. Specifically, each AIP has four fields: `preamble`, `access address`, `protocol data unit` (PDU), and `CRC`. If $f_{ext} = 1$, the *BLEW* AIP is the same as the native AIP, and any COTS BLE chip can receive it. For $f_{ext} > 1$, each bit is repeated f_{ext} times, and the extended AIP should be segmented and encapsulated into multiple native AIPs.

However, not all fields in the native AIP can be used to encapsulate extended symbols. The `preamble` field can only be set to a few options that are mentioned in the Bluetooth specifications, and the `CRC` is calculated and appended by the hardware. The `access address` can be set by software, but it is a crucial field to achieve channel multiplexing. Specifically, the COTS BLE chip will only listen to a specific `access address`. Once a packet is detected, the COTS BLE checks the `access address` and discards packets with non-target `access addresses`. To prevent nearby COTS BLE devices from overhearing packets transmitted by *BLEW* devices, the `access address` field should be retained and adhere to the

Bluetooth specification. In the PDU field, a 1-byte `header` and 1-byte `length` field are required. For example, the `length` field is encapsulated in the packet and used to inform the RF chip how many data bytes need to be transmitted. Therefore, with symbol extension, the entire extended AIP can only be encapsulated in the `payload` field, whose maximum length is 255 bytes. We call the fields except for `payload` as mandatory fields. The length of the mandatory fields in a native AIP is $N_f = 10$ bytes. An extended AIP should be segmented and loaded into multiple AIPs.

5.2 Throughput Modeling

The minimum time between two continuous AIPs is IFS (T_{IFS}), which is 150μs according to Bluetooth specification [6].

We denote the maximum payload length in each extended AIP as n_{max} bytes, which is an integer number between 0 and 255. Since the extended AIP shares the same packet format with the native BLE, the total length of the extended AIP including the N_f bytes mandatory fields in bytes is:

$$n_{total} = f_{ext} \times (n_{max} + N_f). \tag{8}$$

To ensure that extended symbols are continuous, the transmission of each extended symbol should be completed within a single AIP. The maximum number of bits that can be loaded in one AIP is:

$$n_{bit} = \lfloor 255 \times 8 / f_{ext} \rfloor . \tag{9}$$

The number of native AIPs that are needed to transmit the extended AIP is:

$$n_{AIP} = \lceil (n_{max} + N_f) \times 8 / n_{bit} \rceil . \tag{10}$$

If $f_{ext} = 1$, the *BLEW* transmitter can reuse all the mandatory fields in native AIP and all the payload can be transmitted in one native AIP. The corresponding maximum throughput is:

$$MT(f_{ext} = 1, n_{max}) = \frac{n_{max}}{n_{max} + N_f + \frac{T_{IFS}}{8}} \times 1\text{Mbps}, \tag{11}$$

where the $\frac{T_{IFS}}{8}$ is the overhead of IFS (i.e., the max number of bytes that can be transmitted in IFS). If a larger extension factor is used, the mandatory fields in native AIP are transmitted n_{AIP} times and the length of mandatory fields in the extended AIP is $f_{ext} \times N_f$. Therefore, the throughput for $f_{ext} > 1$ can be formulated as follows:

$$MT(f_{ext} > 1, n_{max}) = \frac{n_{max}}{n_{total} + (N_f + \frac{T_{IFS}}{8}) \times n_{AIP}} \times 1\text{Mbps}. \tag{12}$$

The actual throughput $T(f_{ext}, n_{max})$ needs to consider the preamble detection rate with different f_{ext}, which can be presented as:

$$T(f_{ext}) = (1 - BER(f_{ext})) \times P_d(f_{ext}) \times MT(f_{ext}, n_{max}), \tag{13}$$

where $P_d(f_{ext})$ is the preamble detection rate with f_{ext}.

5.3 Throughput Optimization

Our goal is to find the optimal f_{ext} that maximizes the $MT(f_{ext})$ in Eq. 13. To calculate the throughput, *BLEW* first needs to obtain the $P_d(f_{ext})$ and $BER(f_{ext})$ in the current channel condition. For $P_d(f_{ext})$, the *BLEW* transmitter can rapidly transmit certain number of packets with different f_{ext}. Since these packets do not need to include PDU bytes, the interval between consecutive packets can be small and the time for the measurement is fast. For the BER, the transmitter can transmit some packets with a large f_{ext} (i.e., 64) and each long symbol can be segmented into multiple short symbols. For example, a f_{ext}=64 symbol can be segmented as four $f_{ext} = 16$ symbols and 64 $f_{ext} = 1$ symbols. Therefore, *BLEW* can fully get the $BER(f_{ext})$. In summary, the steps of channel adaptation can be outlined as follows: (i) The transmitter first rapidly transmits probing packets to assess the preamble detection rates with different extension factors. These probing packets have only mandatory fields without payload (i.e., n_{max}=0); (ii) The transmitter transmits packets with known bytes and the longest extension factor for BER evaluation; (iii) *BLEW* finally decides the optimal extension factor. A more convenient approach is to pre-obtain the SNR-BER relationship for different f_{ext}. Like the traditional demodulator, the DNN-based demodulator also has a transition band in the SNR-BER relationship (Fig. 8), and the relationship can be fitted with the Piecewise Linear Function (PLF). However, different environments may affect the BER-SNR relationship and a calibration is required. We reserve it as the future work.

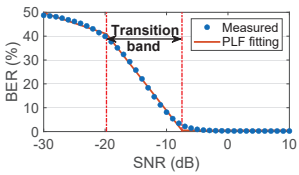


Figure 8: The SNR-BER relationship of $f_{ext} = 4$.

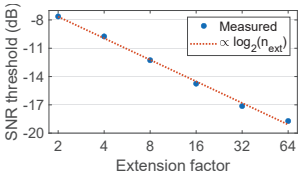


Figure 9: The SNR threshold has a near linear correlation with $\log_2(f_{ext})$

However, the extension factor can be set to an arbitrary integer between 1 and 255×8, it would be very time-consuming to evaluate the preamble detection rates and BERs with all the possible extension factors. Fortunately, we find not all values are practically necessary. Specifically, as the symbol extension increases, the efficiency gain from further extension diminishes rapidly. We define the SNR threshold as the point at which the SNR-BER relationship first reaches a 10% BER. Fig. 9 illustrates the SNR threshold for different extension factors, showing a nearly logarithmic relationship with the extension factor. Consequently, we limit our search to extension factors of 2^i, where i ranges from 0 to 6, resulting in seven potential choices. Meanwhile, large extension factors can significantly reduce the symbol rate and throughput. *BLEW* sets the maximum extension factor to 64, as extension factors larger than 64 can result in extremely low data rates, rendering the network impractical. Therefore, *BLEW* only uses seven extension factors (1 to 64). To make *BLEW* more practical, we can set a preamble detection rate restriction (e.g., 30%) that *BLEW* should choose an extension factor that ensures this preamble detection rate. The reason is that the BLE advertising interval should be at least 7.5 ms, frequent retransmission can lead to high transmission delay.

After measurement of channel conditions, *BLEW* calculates the throughput with respect to each extension factor $T(f_{ext})$. Specifically, using Eq. 11 for f_{ext}=1 and Eq. 12 for f_{ext}>1. The overhead of this throughput calculation is relatively small since the equations only contain basic operations. Besides, Eq. 12 is repeated six times, we transform the f_{ext}, T, and P_d into vectors to accelerate the calculation with matrix operations. After calculating all the seven throughput values, *BLEW* chooses the f_{ext} with the highest throughput while meeting the preamble detection rate constraint.

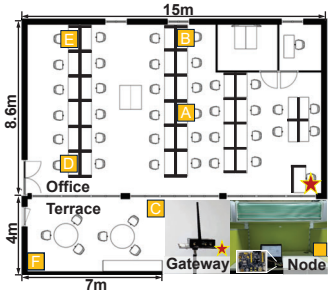


Figure 10: Experiment setup.

Position	*BLEW*	Symphony	Native BLE
A	894.89	13.46	894.88
B	699.58	9.43	676.17
C	128.42	13.10	24.46
D	66.86	6.66	0.00
E	367.68	12.97	248.13
F	824.22	12.50	812.53

Table 1: *BLEW* goodput in NLOS scenario (Kb/s).

Position	n_{ext}	Detection rate	BER
A	1	98.58%	0.01%
B	1	78.33%	0.62%
C	4	62.08%	2.36%
D	8	63.75%	2.17%
E	2	89.17%	0.35%
F	1	92.08%	0.40%

Table 2: *BLEW* extension factor and performance.

6 IMPLEMENTATION

***BLEW* receiver:** We implement the *BLEW* transmitter on the popular Nordic nRF52840 platform [2] based on the RIOT OS [3] and NimBLE protocol stack [1]. BLE operates at a physical layer rate of 1 Mbps. On the receiver side, we employed the USRP B210 to capture BLE signals. The USRP B210 delivers the raw signal to a host device, which in this case is a laptop equipped with an i7-10750H CPU and 1650Ti GPU. The sampling rate is set at 8 MHz. To generate the STFT input for the demodulator, we perform STFT on the raw signals utilizing a 4μs STFT window with a 1μs step. It is important to note that the signal demodulation is independent of the hardware, making it adaptable to other gateways.

***BLEW* transmitter with COTS BLE chip:** We implement the symbol extension, packet segmentation, and data de-whitening functions as a system module within RIOT OS. To transmit long extended AIPs, we modify the ble_phy_tx() function so that it can keep the radio in transmission mode as needed after completing the transmission of each native AIP. To de-white the data, we perform XOR at the link layer between the data and the corresponding PN sequence before the hardware data whitening. It is important to note that no modifications were made to the BLE chip itself, and *BLEW* transmitter can be implemented on other COTS BLE chips.

Channel adaptation: To choose the optimal f_{ext}, *BLEW* needs to assess the preamble detection rate and BER of each f_{ext}. *BLEW* first uses a 10ms interval to quickly transmit 1,024 packets with only mandatory fields to assess the preamble detection rates. This consumes 71.68s. Then, *BLEW* transmits 240 packets with f_{ext}=64 and $n_{\max}$=255 to assess the BER for all f_{ext}. Since the packet has a payload, the packet interval is 200ms and the total time is 48s.

Signal dataset: We train our DNN-based demodulator with synthesized BLE symbols. Specifically, we collect BLE symbols at high SNR levels. Then, we generate various random Gaussian white noise and precisely change the amplitude of the BLE signals to achieve fine-grained SNR control. Finally, we add the white noise on the BLE signal to generate the BLE signals with various SNR levels [31, 53]. We also collect a small number of data in real-world Line-of-Sight scenarios for DNN model update.

Comparison: We compare *BLEW* with Symphony [34] (STFT-based) and the native BLE protocol for evaluation. Symphony employs a fixed symbol extension factor of 160, while we adjusted it to 64 to match the maximum extension factor used in *BLEW* for a fair comparison. All data packets have 255 bytes payloads.

7 EVALUATION

7.1 Overall Performance over Weak Links

We use a case study in the real world to illustrate the performance of *BLEW*. The experiment is conducted in an office with many seats and an adjacent terrace (Fig. 10). Each seat is equipped with a 0.86 m high partition over the desktop, which can block the wireless signal and create a None-Line-of-Sight (NLOS) environment. Meanwhile, the BLE transmitter uses a low transmission power (i.e., -8 dBm) to create weak links. To avoid the high detection error of the STFT-based method and ensure fairness, we use our phase-peak clustering-based method for related works to evaluate the throughput. Since DNN-based demodulator has similar performance as traditional methods with short symbols (details in Sec. 7.2), *BLEW* uses the STFT-based demodulation method for f_{ext}=1.

We evaluate *BLEW*'s communication performance in the NLOS environment and the experiment results are shown in Tab. 1. The goodput of *BLEW* has 60.20 Kb/s-881.43 Kb/s higher than Symphony and 0.01 Kb/s-119.55 Kb/s higher than the native BLE. For positions that are relatively close to the gateway (i.e., Position A and B) the native BLE works well and *BLEW* prefers not to extend the symbol to achieve a higher symbol rate and higher throughput. Although the native BLE and *BLEW* have the same symbol length (i.e., f_{ext} = 1), *BLEW* has a slightly higher throughput. The reason is *BLEW* uses the DNN-based demodulator ($f_{\text{ext}} \geq 2$) or STFT (f_{ext} <2) to demodulate the symbol, both have better performance than the phase difference-based demodulator used by the native BLE. We further evaluate the performance of different demodulators in Sec. 7.2. Symphony can maintain a relatively stable but low throughput at all positions since it uses a large extension factor (i.e., f_{ext}=64). Long symbols are resilient to channel noise but also significantly reduce the symbol rate and throughput. *BLEW* assesses the channel condition and adaptively chooses the optimal extension factor to maximize the throughput. Tab. 2 shows the extension factor f_{ext} and the corresponding preamble detection rate and BER at each position. *BLEW* maintains an acceptable preamble detection rate and BER while choosing a smaller extension factor to achieve a higher symbol rate to increase the throughput.

For Position D and E, the D has lower throughput than E although the E seems to have more blockages between it and the gateway. The possible reason is that Position D is near an opened door and part of the signal was emission to the outside of the office. The signal attenuation at Position F is minor, even though Position F is farther from the gateway than Position C. The reason is that

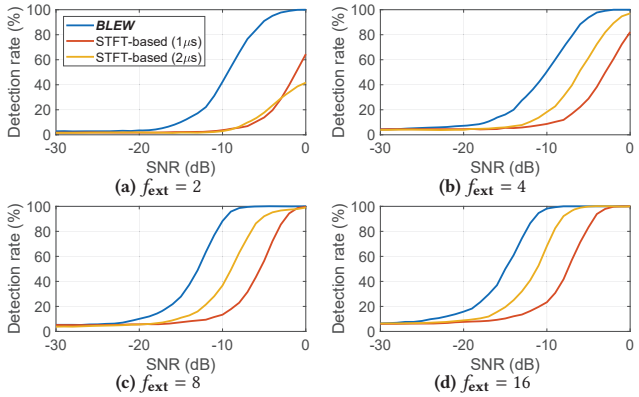


Figure 11: The preamble detection rate of *BLEW* and the STFT-based method with 1μs and 2μs STFT window.

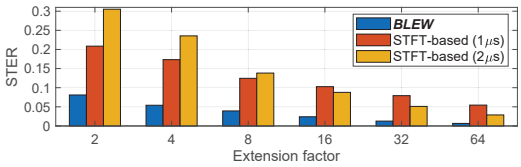


Figure 12: The average Symbol Timing Error Ratio (STER) of preamble detection with SNR from -15 dB to 0 dB.

Position C is close and lower than the window sill, and it is severely blocked by the wall. Position F is far from the window sill, and the signal can straightly reach the gateway through the window glass.

7.2 Impacts of Different Components

To illustrate the contribution of each module, we evaluate the performance of the preamble detector and the DNN-based demodulator of *BLEW*. Then, we evaluate the gain of symbol extension for preamble detection and demodulation.

(1) Accurate Preamble Detection

We evaluate the performance of the preamble detection mechanism. The ground truth is the preamble detection result when there is no noise. Then, the I and Q paths of the preamble signal are finely adjusted and added with the random Gaussian white noise to generate the preamble signal in the target SNR. A preamble is considered detected if it satisfies two conditions. Firstly, a phase peak cluster is detected. Secondly, the difference between the peak center and ground truth is less than half of the extended symbol length. We compare *BLEW* with the STFT-based preamble detection method with 1 μs and 2 μs STFT window. Specifically, we apply STFT to the raw signal and use an STFT preamble kernel to convolve with the STFT result to the frequency correlation. The preamble can cause a frequency peak in the frequency correlation, and we use our peak clustering-based preamble detection method to find preambles.

Fig. 11 shows the preamble detection rate at different SNR levels. For $f_{ext} \geq 2$ and SNR between -30 dB and 0 dB, *BLEW* has 21.92%-24.65% higher preamble detection rate than STFT-based method with 1 μs window and 11.02%-25.70% higher than STFT-based method with 2 μs window. In our experiment, we find that the STFT-based preamble detection method can be easily effected by the whitened native `length` field before the extended preamble

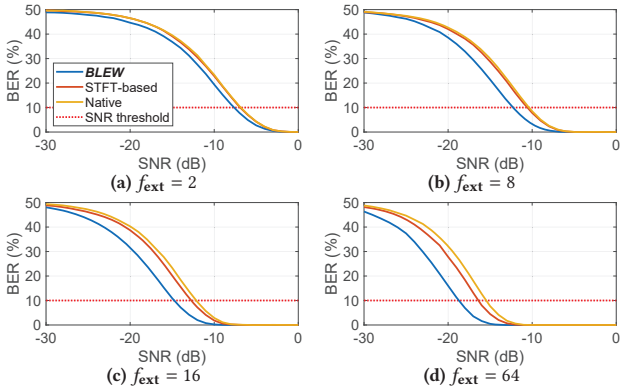


Figure 13: SNR-BER relationships.

because of the blurred boundary issue. Specifically, if the whitened bits before preamble are 0, the preamble detected by the STFT-based method tends to move a little before than the actual one, so it can have a longer low-frequency symbol at the beginning.

We use the Symbol Timing Error Ratio (STER) to indicate the preamble detection error, which is the ratio of the detection timing error and the extended symbol length. Fig. 12 shows the average STER with SNR from -15 dB to 0 dB. *BLEW* can reduce 0.0222-0.2242 STER compared with the STFT-based preamble detection method. STFT-based preamble detection method can use larger STFT window for long symbols to obtain higher frequency resolution, thereby increasing the detection rate. However, this can also lead to a higher detection error rate for short symbols.

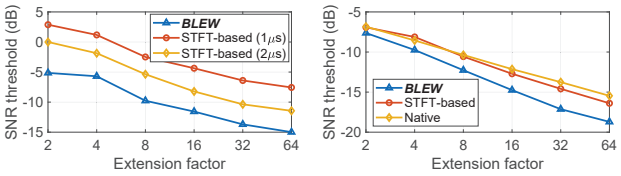
(2) Performant DNN-based demodulator

We evaluate the performance of the DNN-based demodulator. We precisely control the SNR of each symbol. Since individual symbols are short, we control the overall amplitude of the I/Q signal to generate a signal with the target SNR. We compared it with the STFT-based symbol demodulation method and the phase difference-based method used by the native BLE.

Fig. 13 illustrates the SNR-BER relationship of BLE symbol demodulation. Overall, *BLEW* has 0.71-2.54 dB SNR gain compared with the STFT-based demodulation method, and 0.79-3.37 dB with the native BLE. We observe that the SNR gain of the DNN-based demodulator increases as the extension factor increases. Specifically, when $f_{ext} \leq 2$, the DNN-based demodulator has similar performance with the STFT-based and phase difference-based demodulator. The performance of the DNN-based decoder is significantly improved compared with traditional methods when $f_{ext} > 2$. The reason is, with a larger extension factor, the raw signal and spectrogram input into the DNN-based demodulator is larger and the DNN model can extract more information from the input for demodulation. Specifically, for $f_{ext} = 2$, the raw signal is only 16 sample points in length and the spectrogram size is 8×2. While for $f_{ext} = 64$, the raw signal contains 512 sample points and 8×64 size spectrogram. In practice, *BLEW* uses STFT-based demodulator when $f_{ext} = 1$ to reduce computational overhead and DNN-based demodulator when $f_{ext} \geq 2$ for higher demodulation performance.

(3) Gain of Symbol Extension

BLEW improves preamble detection and demodulation performance in low signal-to-noise ratio situations through symbol extension. In order to illustrate the gain of symbol extension, we evaluate



(a) Preamble detection rate (b) BER

Figure 14: SNR thresholds with different extension factors.

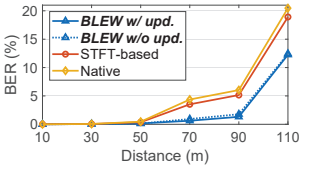


Figure 15: BER of f_{ext}=8 symbols in real-world LOS scenario.

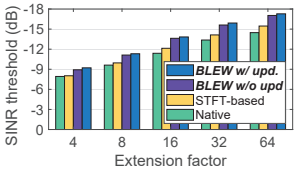


Figure 16: SINR threshold with WiFi interference.

the SNR threshold of preamble detection and symbol demodulation. The SNR threshold shows the SNR level of preamble detection when ensuring a 90% detection rate, and the SNR threshold of symbol demodulation is the SNR level with the 10% BER. To calculate the SNR thresholds, we measure the preamble detection rate and BER within the -30 to 10 dB SNR range with a 1 dB interval. If the SNR threshold occurs between two integer SNR values, we assume that the preamble detection rate and BER are linear within the 1dB range, thereby inferring the value of the SNR threshold.

Fig. 14 shows the experiment results. For preamble detection, extending the symbol length twice can bring a 1.97 dB gain on average. As for the SNR threshold of BER, the average gain of extending the symbol length twice is 2.22 dB. Compared with STFT-based and the native phase difference-based demodulation scheme, with extending symbol length twice, *BLEW* provides a BER gain that is 0.32 dB higher than STFT-based scheme and 0.49 dB higher than the native scheme. The gain is more significant when the extension factor is large. This is because the longer symbol contains more information and a DNN model can extract more features for demodulation.

7.3 Impact of Interference Types

The DNN-based demodulator is practical in real-world communication. We set the transmit power of the *BLEW* transmitter to maximum (i.e., 8 dBm) and collect the raw signal in a Line-of-Sight (LOS) road site. We also use a small partition of the dataset in the LOS scenario (i.e., 5%) to update the DNN-based demodulator. Fig. 15 shows the BER of demodulating the f_{ext}=8 symbols between 10 m and 110 m. The DNN-based demodulator has up to 6.66% lower BER than the STFT-based demodulation method and up to 8.22% lower than the native BLE. The DNN model update can reduce 0.18% to 0.46% BER within the 70 m to 110 m range. The gain of the DNN model update is relatively minor, since in the LOS scenario, the noise pattern is similar to the Gaussian white noise.

BLE often coexists with WiFi, and we evaluate the impact of WiFi interference. We deploy a WiFi AP that works in WiFi channel 1, which is totally overlapped with the BLE channel 4 (2412 MHz).

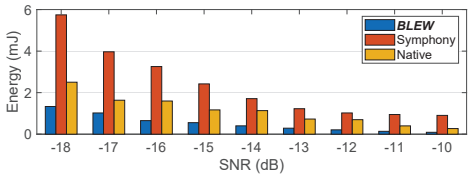


Figure 17: Energy consumption to transmit 100B data.

Then, we connect a laptop to the WiFi and use `iperf` to inject 40 Mb/s WiFi traffic. Fig. 16 shows the Signal to Interference plus Noise Ratio (SINR) threshold, which is the SINR value when BER is 10%. With heavy WiFi interference, *BLEW* can still bring 0.89-1.58 dB SINR gain compared with the STFT-based demodulation scheme and 0.99-2.57 dB with the native BLE. The reason is WiFi utilizes DSSS to transfer its signal to pseudo-white noise, the DNN trained with Gaussian white noise-synthesized signals adapt well to the WiFi interferences. The DNN-based demodulator can be updated with the known data bits. The updated DNN-based demodulator ("*BLEW* w/ upd." in Fig. 16) has a 1.19-1.82 dB SINR gain compared with STFT-based method and a 1.29-2.81 dB SINR gain with the native BLE. We find that when updating the DNN model, the optimal model only requires a few training steps (about 300-800 steps). Overtraining can cause the network overfit to the interference in the training set, leading to a decrease in performance.

7.4 System Overhead

***BLEW* transmitter energy consumption:** We first evaluate the energy consumption of *BLEW* to transmit 100B data. To derive the energy consumption, we first get the average throughput with the preamble detection rate and BER. After that, we can calculate the average transmission time of 100B data. According to the nRF52840 datasheet v1.7 [2], the typical power for its radio transmitting 0 dBm signal is 15 mW (3.0 mA for 5 V input DC/DC, REG0 out=3.3 V). Therefore, the energy consumption can be derived by multiplying the average transmission time and transmission power. We take the additive white Gaussian noise channel for example and let the *BLEW* decide the optimal extension factor. Fig. 17 shows the energy consumption for *BLEW* to transmit 100B data (without accounting for the high energy cost introduced by *BLEW*'s channel adaptation) with SNR between -18 dB and -10 dB. *BLEW* changes its extension factor from 16 to 2. *BLEW* has 74.24%-89.98% relatively lower energy consumption than Symphony (with 2 μs STFT window for preamble detection) and 37.51%-70.14% lower than native BLE. Although Symphony can maintain a relatively low BER on weak links, its throughput is low due to the extremely low symbol rate, resulting in higher energy consumption.

Packet extension overhead: To extend packets, *BLEW* transmitter should first copy each bit f_{ext} times to get the extended packet. Then segment the extended packet into multiple native AIPs and de-whiten each native AIP for transmission. We then evaluate the packet extension time on nRF52840. Specifically, we evaluate the total packet extension runtime by setting different maximum payload lengths (i.e., 32, 128, 255 Bytes) and all the seven extension factors. Fig. 18 shows the experiment results. The packet extension runtime is acceptable and in the worst-case scenario (255 Bytes payload and 64 extension factor), the extension time is 34.24ms.

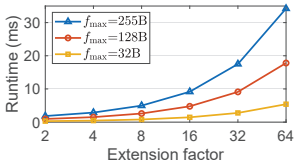


Figure 18: Packet extension runtime on nRF52840.

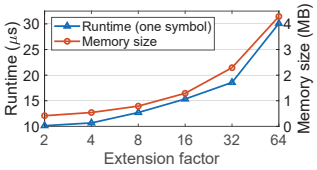


Figure 19: DNN-based demodulator overhead.

***BLEW* demodulation runtime:** We evaluate the DNN-based demodulator overhead used in the *BLEW* receiver. We set the batch size to 512 symbols to fully utilize the GPU. Fig. 19 shows the computational overhead and memory overhead of the DNN-based demodulator. *BLEW* uses DNN-based demodulator f_{ext}>=2, which cost 10.13 μs-30.08 μs time to demodulate one extended symbol. The memory usage for the DNN-based demodulator is 0.42 MB-4.29 MB.

8 RELATED WORK

Performance of BLE: For BLE, numerous works have been done to optimize the performance of neighbor discovery communication [30, 44], cross-technology communications [12], and indoor localization [4, 39]. The BLEach [49] proposed an IPv6-over-BLE stack and formulates the latency in the single connection scenario and a multi-hop version is further proposed in [45]. This greatly enhanced the interoperability of BLE. Due to the large number of existing BLE devices, it is common for one BLE gateway to connect with multiple remote nodes. The multi-connection scheduling is one of the interest points of BLE research [10, 16, 43]. Some works focus on BLE performance at low-SNR levels. Some BLE transmission performance models in noisy environments are proposed in [47, 48, 50, 51]. They ask the node to measure the average RTT of an application packet with maximum payload and infer the transmission latency under current link quality. Cross-Technology Communication (CTC) is one of the solutions to improve the communication range [35]. The Symphony [34] proposes a symbol emulation-based BLE long-range communication method. However, the symbol of Symphony is long and fixed, which makes the throughput very low. *BLEW* adaptively extends the BLE symbol length to achieve long-range communication while ensuring the network throughput.

Long-range communication: There has been extensive research on long-distance communication in recent years. One of the solutions is using link-layer coding, such as the Ratelss code [21], Direct Sequence Spread Spectrum (DSSS) [32]. At the physical layer, inspired by the distributed Multiple Input Multiple Output (MIMO) used in WiFi [25], recent works leverage multiple receivers to enhance the LoRa communication range [5, 19, 37]. However, the range of single-hop BLE communication is short and gateway deployment should be very dense, which significantly increases the system costs. In recent years, deep learning has been applied to wireless communication systems and proven that it can significantly improve the performance [20, 24, 41, 54]. Some of the works focus on leveraging deep learning to achieve channel estimation with fast time-varying channel conditions [28, 36]. DeepLoRa [38] relies on recognizing land-cover types along the link and uses a Bi-LSTM to develop a path loss model. Other works leverage deep learning to achieve physical layer symbol demodulation [56, 60]. NELoRa [31] develops a spectrogram-based DNN demodulator to extract the fine-grained information embedded inside the LoRa symbols for demodulation. However, NELoRa is designed for the LoRa signal, which has a much larger symbol length. There are two major differences. First, *BLEW*'s demodulator features two extractors while NELoRa has only one extractor. Therefore, *BLEW* can fully capture the time-domain and frequency-domain characteristics of the relatively short BLE signals. Second, *BLEW*'s demodulator uses a direct demodulation scheme while NELoRa requires an additional STFT denoising step. For shorter BLE signals, the STFT results contain less signal information, making the denoising gain from STFT results smaller. In *BLEW*, GFSK demodulation is treated as a binary classification problem, leading to a direct demodulation scheme.

9 DISCUSSION AND LIMITATIONS

(1) Compatibility with BLE protocol stack: To support connection-based communication, the gateway can work as the BLE Central, and the *BLEW* node works as the Peripheral. The Central starts a connection event by transmitting a native empty link-layer packet and the *BLEW* (Peripheral) returns an extended packet. The native ACK from the gateway is transmitted in the next connection event which is at least one connection interval later. The connection interval is between 7.5ms and 4s. The extended packet cannot be demodulated in time if the connection interval is short. Specifically, the time for demodulating packets with n_{max}=255B and f_{ext} between 2 and 64, the demodulation can cost 21.49ms-63.77ms. Fortunately, BLE devices typically avoid using extremely short connection intervals to save energy. For example, the connection interval of Android starts from 45 ms and some low-power devices can even utilize a connection interval of one second [15]. This allows the gateway to demodulate the extended packets and generate the ACK. Besides, *BLEW* can set a smaller n_{max} to ensure real-time performance of demodulation. For example, when setting n_{max} to 128B, the demodulation time will be reduced to 10.38ms-30.80ms.

Channel hopping is an important feature in BLE to combat interference [55], and it is still available for *BLEW*. The *BLEW* gateway and transmitter can periodically change channels based on a pre-agreed channel sequence. The DNN-based demodulator is trained with white noise. Theoretically, it is related to the pattern of interference and is independent of the specific channel number. *BLEW* can leverage native BLE's frequency hopping and channel blacklisting mechanisms to avoid interference-affected channels in order to achieve better performance.

BLEW is also compatible with 2M PHY. Compared with 1M PHY, the native symbol of 2M PHY is half of the native symbol in 1M PHY. Therefore, the 2M PHY requires sending twice the number of native symbols to ensure that the extended symbol length is the same as that in 1M PHY. The other difference is that the frequency deviation of 2M PHY is larger than 1M PHY, which can lead to a better performance. We will evaluate support for connection-based communication and channel hopping, as well as implementing a *BLEW* version compatible with the 2M PHY in future work.

(2) COTS BLE receiver: The direction finding proposed in Bluetooth 5.1 allows [6] COTS BLE chips to access the received I/Q signal. For example, the nRF5340 [42] supports 250 Ksps~8 Msps

I/Q sampling rate, while keeping a low current consumption of 3.7 mA. However, most of the modern COTS BLE chips cannot support the DNN-based demodulator because of their extremely low computational performance (e.g., 128 MHz CPU and 512 KB RAM). Therefore, COTS BLE devices can only use traditional demodulation schemes and rely on symbol extension to achieve performance improvement over weak links.

(3) *BLEW* over COTS smartphone: *BLEW* can be implemented in the application layer of smartphones with certain limitations. On smartphones, we cannot access the physical-layer APIs and cannot continuously transmit native AIPs like in Fig. 7. Alternatively, the smartphone can use advertising packets to transmit *BLEW* packets. Specifically, encapsulated the extended AIP into the 37B payload of a single advertising packet, which limits the extension factor and maximum payload length. As for receiving *BLEW*, it is hard to access the I/Q signal on smartphones, and they cannot function as *BLEW* receivers. With the growing adoption of the BLE directional finding, manufacturers could potentially enable I/Q sample reading through a firmware update in the future.

(4) DNN-based demodulator compression: The current design of the DNN-based demodulator has a relatively high cost, especially when deploying *BLEW* on resource-constrained gateways (e.g., Raspberry Pi, Nvidia Jetson). To address this problem, we can apply neural network pruning, which has been well-studied in the literature [17, 18, 22]. Specifically, in feature extractors (G_s and G_t) and the demodulator (G_d), there are CNN and dense layers which contains many filters for extracting features. Network pruning can identify the less important filters and remove them to save running time. Besides, LSTM in the demodulator G_d can be replaced by a lightweight RNN model (e.g., GRU layer [14]).

(5) Downlink communication: In this paper, we focus on the uplink. To make the communication bidirectional, long-range downlink transmission (i.e., from the gateway to the remote node) is also very important. In *BLEW*, the gateway has to update and transmit the optimal parameters to the remote node when the channel condition changes. Long-range downlink transmission can be achieved by using a higher transmission power (up to 20 dBm [6]), which is possible since the gateway is usually wall-powered. It is also possible to leverage other high-power wireless signals (e.g., WiFi) to achieve long-range CTC [9, 11] for the downlink. For extremely weak link conditions, *BLEW* gateway can use symbol extension to ensure the new parameters are transmitted to the *BLEW* node. Specifically, the *BLEW* node stays in the receiving mode with the longest extension factor. Due to the gateway's high transmission power and the use of long extension factor, the *BLEW* transmitter (e.g., nRF5340) can collect I/Q signal and uses computation-efficient demodulation methods (e.g., STFT-based and phase difference-based methods) to receive the message from the gateway. However, using the largest extension factor can lead to higher energy consumption. The evaluation under these conditions was not performed in this paper and is left as future work.

(6) Link-layer coding: In this paper, we focus on symbol extension at the physical layer. Link-layer coding techniques can be utilized to further enhance performance over weak links. For example, using the Forward Error Correction (FEC) code used by BLE-coded channel [6], and rateless codes (e.g., Luby-Transform code [40]). An interesting question is how to optimally allocate the redundancy to the physical layer and link layer, to achieve the best performance.

(7) Channel Adaptation Overhead: To choose the optimal f_ext, *BLEW* requires to evaluate the channel conditions by actively transmitting probing packets. In this paper, we only evaluate the performance after the channel adaptation. The overhead of the channel condition estimation is relatively high. Considering BLE nodes are often mobile, the channel adaptation should be conducted when the gateway finds the received SNR of the transmitter has significantly changed. Therefore, a fast channel adaptation method needs to be proposed. As we mentioned in Sec. 5.2, the SNR-BER relationship can be accurately fitted by PLF (Fig. 8), and the preamble detection rate shows a similar curve (Fig. 11). Therefore, a possible solution is to use PLF to model the relationship between SNR and BER, as well as the relationship between SNR and preamble detection rate. Once the channel condition changes, *BLEW* can quickly infer the preamble detection rate and BER, and then update the optimal f_ext. However, we find different channel conditions can change these two curves. The major challenge is how to accurately compensate the two relationships.

(8) Generalization of *BLEW*: Although *BLEW* is currently designed based on BLE, its principle can also be applied to other wireless low-power protocols. The existing work [34] proposes a method to let the COTS ZigBee chip transmit extended symbols with a stable frequency, which is similar to *BLEW*'s extended symbols. Therefore, our approach can be extended to ZigBee devices, which is a future direction of our work.

10 CONCLUSION

In this paper, we propose *BLEW* to improve BLE communication performance over weak links. The key idea of *BLEW* is to adaptively extend the BLE symbol to combat the channel noise. We propose two key techniques to improve preamble detection and signal demodulation performance. First, we propose a phase peak clustering preamble detection technique to achieve accurate preamble detection and symbol synchronization. Second, we propose a multi-domain DNN-based demodulator to extract time-frequency information from the BLE signal and effectively demodulate the BLE symbol even with a low SNR level. We implement *BLEW* with COTS nRF52840 chip as the remote transmitter and USRP B210 as the gateway. Our experiment results show that *BLEW* has up to 119.55 Kb/s higher throughput than native BLE over weak links. Compared with existing methods, our two key techniques can increase up to 25.70% preamble detection rate and bring 0.71 dB-3.37 dB symbol demodulation gain.

ACKNOWLEDGMENTS

We thank all the reviewers and the shepherd for their valuable comments and helpful suggestions. This work is supported by the National Natural Science Foundation of China under grant no. 62072396, the "Pioneer" and "Leading Goose" R&D Program of Zhejiang under grant No. 2023C01033, and the National Youth Talent Support Program. Jiamei Lv and Wei Dong are the corresponding authors.

REFERENCES

[1] 2024. Apache NimBLE. https://github.com/apache/mynewt-nimble.

[2] 2024. nRF52840 Objective Product Specification v1.7. https://docs.nordicsemi.com/bundle/nRF52840_PS_v1.7/resource/nRF52840_PS_v1.7.pdf.

[3] 2024. RIOT OS: The friendly Operating System for the Internet of Things. https://www.riot-os.org/.

[4] Roshan Ayyalasomayajula, Deepak Vasisht, and Dinesh Bharadia. 2018. BLoc: CSI-based accurate localization for BLE tags. In *Proc. of ACM CoNEXT*. 126–138.

[5] Artur Balanuta, Nuno Pereira, Swarun Kumar, and Anthony Rowe. 2020. A cloud-optimized link layer for low-power wide-area networks. In *Proc. of ACM MobiSys*. 247–259.

[6] Bluetooth SIG. 2023. Core Specification 5.4. https://www.bluetooth.com/specifications/specs/core54-html/.

[7] Bluetooth SIG. 2024. Bluetooth Mesh Profile. https://www.bluetooth.com/specifications/specs/mesh-profile-1-0-1/.

[8] Justin Chan, Anran Wang, Arvind Krishnamurthy, and Shyamnath Gollakota. 2019. Deepsense: Enabling carrier sense in low-power wide area networks using deep learning. *arXiv preprint arXiv:1904.10607* (2019).

[9] Kameswari Chebrolu and Ashutosh Dhekne. 2009. Esense: Communication through energy sensing. In *Proc. of ACM MobiCom*. 85–96.

[10] Jing-Ho Chen, Ya-Shu Chen, and Yu-Lin Jiang. 2017. Energy-efficient scheduling for multiple latency-sensitive Bluetooth Low Energy nodes. *IEEE Sensors Journal* 18, 2 (2017), 849–859.

[11] Zicheng Chi, Yan Li, Hongyu Sun, Yao Yao, Zheng Lu, and Ting Zhu. 2016. B2w2: N-way concurrent communication for iot devices. In *Proc. of ACM SenSys*. 245–258.

[12] Hsun-Wei Cho and Kang G Shin. 2021. BlueFi: Bluetooth over WiFi. In *Proc. of ACM SIGCOMM*. 475–487.

[13] Hsun-Wei Cho and Kang G Shin. 2022. FLEW: fully emulated wifi. In *Proc. of ACM MobiCom*. 29–41.

[14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[15] Jasper De Winkel, Haozhe Tang, and Przemysław Pawełczak. 2022. Intermittently-powered bluetooth that works. In *Proc. of ACM MobiSys*. 287–301.

[16] F John Dian and Reza Vahidnia. 2020. Formulation of BLE throughput based on node and link parameters. *Canadian Journal of Electrical and Computer Engineering* 43, 4 (2020), 261–272.

[17] Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan. 2019. Approximated oracle filter pruning for destructive cnn width optimization. In *Proc. of PMLR ICML*. PMLR, 1607–1616.

[18] Xuanyi Dong and Yi Yang. 2019. Network pruning via transformable architecture search. *Advances in Neural Information Processing Systems* 32 (2019).

[19] Adwait Dongare, Revathy Narayanan, Akshay Gadre, Anh Luong, Artur Balanuta, Swarun Kumar, Bob Iannucci, and Anthony Rowe. 2018. Charm: exploiting geographical diversity through coherent combining in low-power wide-area networks. In *Proc. of ACM/IEEE IPSN*. IEEE, 60–71.

[20] Sebastian Dörner, Sebastian Cammerer, Jakob Hoydis, and Stephan Ten Brink. 2017. Deep learning based communication over the air. *IEEE Journal of Selected Topics in Signal Processing* 12, 1 (2017), 132–143.

[21] Wan Du, Zhenjiang Li, Jansen Christian Liando, and Mo Li. 2014. From rateless to distanceless: Enabling sparse sensor network deployment in large areas. In *Proc. of ACM SenSys*. 134–147.

[22] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proc. of ACM MobiCom*. 115–127.

[23] Hadi Givehchian, Nishant Bhaskar, Eliana Rodriguez Herrera, Héctor Rodrigo López Soto, Christian Dameff, Dinesh Bharadia, and Aaron Schulman. 2022. Evaluating physical-layer ble location tracking attacks on mobile devices. In *Proc. of IEEEE S&P*. IEEE, 1690–1704.

[24] Tobias Gruber, Sebastian Cammerer, Jakob Hoydis, and Stephan ten Brink. 2017. On deep learning-based channel decoding. In *Proc. of IEEE CISS*. IEEE, 1–6.

[25] Ezzeldin Hamed, Hariharan Rahul, Mohammed A Abdelghany, and Dina Katabi. 2016. Real-time distributed MIMO systems. In *Proc. of ACM SIGCOMM*. 412–425.

[26] Pirmin Heinzer, Vincent Lenders, and Franck Legendre. 2012. Fast and accurate packet delivery estimation based on DSSS chip errors. In *Proc. of IEEE INFOCOM*. IEEE, 2916–2920.

[27] Ehsan Hosseini and Erik Perrins. 2013. Timing, carrier, and frame synchronization of burst-mode CPM. *IEEE Transactions on communications* 61, 12 (2013), 5125–5138.

[28] Qiang Hu, Feifei Gao, Hao Zhang, Shi Jin, and Geoffrey Ye Li. 2020. Deep learning for channel estimation: Interpretation, performance, and comparison. *IEEE Transactions on Wireless Communications* 20, 4 (2020), 2398–2412.

[29] Xianjun Jiao. 2024. BTLE. https://github.com/JiaoXianjun/BTLE.

[30] Christine Julien, Chenguang Liu, Amy L Murphy, and Gian Pietro Picco. 2017. Blend: practical continuous neighbor discovery for Bluetooth Low Energy. In *Proc. of ACM IPSN*. 105–116.

[31] Chenning Li, Hanqing Guo, Shuai Tong, Xiao Zeng, Zhichao Cao, Mi Zhang, Qiben Yan, Li Xiao, Jiliang Wang, and Yunhao Liu. 2021. Nelora: Towards ultra-low snr lora communication with neural-enhanced demodulation. In *Proc. of ACM SenSys*. 56–68.

[32] Songfan Li, Hui Zheng, Chong Zhang, Yihang Song, Shen Yang, Minghua Chen, Li Lu, and Mo Li. 2022. Passive DSSS: Empowering the Downlink Communication for Backscatter Systems. In *Proc. of USENIX NSDI*. 913–928.

[33] Yu Li, Ziqian Wang, Song Chen, and Fujiang Lin. 2017. A low-power digital GFSK receiver with mid-value filtering frequency offset estimator and soft anti-overlap slicer. In *2017 IEEE 12th International Conference on ASIC (ASICON)*. IEEE, 1109–1112.

[34] Zhijun Li and Yongrui Chen. 2019. Achieving universal low-power wide-area networks on existing wireless devices. In *Proc. of IEEE ICNP*. 1–11.

[35] Zhijun Li and Yongrui Chen. 2020. BLE2LoRa: Cross-technology communication from bluetooth to LoRa via chirp emulation. In *Proc. of IEEE SECON*. IEEE, 1–9.

[36] Yong Liao, Yuanxiao Hua, Xuewu Dai, Haimei Yao, and Xinyi Yang. 2019. ChanEstNet: A deep learning based channel estimation for high-speed scenarios. In *Proc. of IEEE ICC*. IEEE, 1–6.

[37] Jun Liu, Weitao Xu, Sanjay Jha, and Wen Hu. 2020. Nephalai: towards LPWAN C-RAN with physical layer compression. In *Proc. of ACM MobiCom*. 1–12.

[38] Li Liu, Yuguang Yao, Zhichao Cao, and Mi Zhang. 2021. Deeplora: Learning accurate path loss model for long distance links in lpwan. In *Proc. of IEEE INFOCOM*.

[39] Ruofeng Liu, Zhimeng Yin, Wenchao Jiang, and Tian He. 2021. WiBeacon: Expanding BLE location-based services via WiFi. In *Proc. of ACM MobiCom*. 83–96.

[40] Michael Luby. 2002. LT codes. In *Proc. of IEEE FOCS*. IEEE Computer Society, 271–271.

[41] Eliya Nachmani, Elad Marciano, Loren Lugosch, Warren J Gross, David Burshtein, and Yair Be'ery. 2018. Deep learning methods for improved decoding of linear codes. *IEEE Journal of Selected Topics in Signal Processing* 12, 1 (2018), 119–131.

[42] Nordic. 2024. nRF5340 Product Specification. https://docs.nordicsemi.com/bundle/ps_nrf5340/page/keyfeatures_html5.html.

[43] Eunjeong Park, Hyung-Sin Kim, and Saewoong Bahk. 2021. BLEX: Flexible Multi-Connection Scheduling for Bluetooth Low Energy. In *Proc. of ACM IPSN*. 268–282.

[44] Wonbin Park, Dokyun Ryoo, Changhee Joo, and Saewoong Bahk. 2021. BLESS: BLE-aided Swift Wi-Fi Scanning in Multi-protocol IoT Networks. In *Proc. of IEEE INFOCOM*. 1–10.

[45] Hauke Petersen, Thomas C Schmidt, and Matthias Wählisch. 2021. Mind the Gap: Multi-hop IPv6 over BLE in the IoT. In *Proc. of ACM CoNEXT*. 382–396.

[46] Sergio Rapuano and Fred J Harris. 2007. An introduction to FFT and time domain windows. *IEEE instrumentation & measurement magazine* 10, 6 (2007), 32–44.

[47] Raúl Rondón, Mikael Gidlund, and Krister Landernäs. 2017. Evaluating Bluetooth Low Energy suitability for time-critical industrial IOT applications. *International Journal of Wireless Information Networks* 24, 3 (2017), 278–290.

[48] Michael Spörk, Carlo Alberto Boano, and Kay Römer. 2019. Improving the Timeliness of Bluetooth Low Energy in Noisy RF Environments.. In *Proc. of EWSN*. 23–34.

[49] Michael Spörk, Carlo Alberto Boano, Marco Zimmerling, and Kay Römer. 2017. Bleach: Exploiting the full potential of ipv6 over ble in constrained embedded iot devices. In *Proc. of ACM SenSys*. 1–14.

[50] Michael Spörk, Markus Schuß, Carlo Alberto Boano, and Kay Römer. 2021. Ensuring End-to-End Dependability Requirements in Cloud-based Bluetooth Low Energy Applications.. In *Proc. of EWSN*. 55–66.

[51] Spörk, Michael and Boano, Carlo Alberto and Römer, Kay. 2020. Improving the timeliness of Bluetooth Low Energy in dynamic RF environments. *ACM Transactions on Internet of Things* 1, 2 (2020), 1–32.

[52] Shuai Tong, Zilin Shen, Yunhao Liu, and Jiliang Wang. 2021. Combating link dynamics for reliable lora connection in urban settings. In *Proc. of ACM MobiCom*. 642–655.

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

[54] Tianqi Wang, Chao-Kai Wen, Hanqing Wang, Feifei Gao, Tao Jiang, and Shi Jin. 2017. Deep learning for wireless physical layer: Opportunities and challenges. *China Communications* 14, 11 (2017), 92–111.

[55] Thomas Watteyne, Steven Lanzisera, Ankur Mehta, and Kristofer SJ Pister. 2010. Mitigating multipath fading through channel hopping in wireless sensor networks. In *Proc. of IEEE ICC*. IEEE, 1–5.

[56] Tian Wu. 2019. CNN and RNN-based deep learning methods for digital signal demodulation. In *Proc. of ACM IVSP*. 122–127.

[57] Bo Yu, Liuqing Yang, and Chia-Chin Chong. 2011. Optimized differential GFSK demodulator. *IEEE transactions on communications* 59, 6 (2011), 1497–1501.

[58] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. 2017. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE transactions on image processing* 26, 7 (2017), 3142–3155.

[59] Kai Zhang, Wangmeng Zuo, and Lei Zhang. 2018. FFDNet: Toward a fast and flexible solution for CNN-based image denoising. *IEEE Transactions on Image*

Processing 27, 9 (2018), 4608–4622.

[60] Lin Zhang, Haotian Zhang, Yuan Jiang, and Zhiqiang Wu. 2020. Intelligent and reliable deep learning LSTM neural networks-based OFDM-DCSK demodulation design. *IEEE Transactions on Vehicular Technology* 69, 12 (2020), 16163–16167.