DREW: Double-Throughput Emulated WiFi

Hsun-Wei Cho and Kang G. Shin
University of Michigan
{hsunweic,kgshin}@umich.edu

ABSTRACT

Bidirectional communication between BLE/FSK devices and WiFi access points (APs) combines the benefits of long battery life, low device cost, and ubiquitous Internet access. However, prior cross-technology communication (CTC) solutions require *transmission mixers* inside FSK chips, thus not applicable to newer ultra-low-power (ULP) BLE chips, which removes these mixers to conserve power. Furthermore, throughputs of prior CTC solutions are limited to 1Mbps.

We present DREW that fundamentally overcomes these limitations. It is designed to effectively transmit WiFi packets by only controlling the power amplifier (PA), and is thus applicable to mixer-less ULP BLE chips. We also propose an innovative use of BLE's IQ sampling capability to receive standard WiFi packets. We design efficient algorithms with SIMD (Single Instruction Multiple Data) acceleration to detect, synchronize and demodulate WiFi packets from IQ samples in real time. DREW also implements WiFi's CSMA/CA and timing, thus adding direct WiFi connectivity to ULP BLE devices. Unlike prior work, DREW uniquely supports QPSK and therefore doubles the downlink throughput. This 2x throughput increase is crucial for new applications that prior work cannot support. In particular, DREW can stream **lossless, HiFi-quality audio** from WiFi to ULP BLE chips. Since stereo audio requires a throughput of 1.411Mbps, no prior work can support this important application due to their 1Mbps limitation.

CCS CONCEPTS

• **Networks** → **Wireless local area networks**.

KEYWORDS

Cross-Technology Communication, WiFi, Bluetooth, BLE

ACM Reference Format:
Hsun-Wei Cho and Kang G. Shin. 2024. DREW: Double-Throughput Emulated WiFi. In *The 30th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '24), September 30-October 4, 2024, Washington D.C., DC, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3636534.3649388

1 INTRODUCTION

Cross-technology communication (CTC) enables direct communication between heterogeneous wireless devices, such as between WiFi and Bluetooth devices. With the ubiquitous usage and wide deployment of WiFi and Bluetooth around the world, the WiFi–Bluetooth CTC creates highly useful connectivity applicable to tens of billions of devices. The state of the art (and DREW) enable end-to-end WiFi connectivity by transforming Bluetooth chips to fully-operational WiFi chips.

One practical application of this new connectivity is for the Internet of Things (IoT). For IoT, each device should be low-cost and highly energy-efficient in order to facilitate massive deployment and support ultra-long battery life. To meet these requirements, Bluetooth Low Energy (BLE) is a popular, widely-deployed technology. BLE chips cost as low as $0.99 apiece [1, 2]. In addition, the latest BLE chips use ultra-low-power (ULP) designs that consume as little as 6mA [3, 4], and 10 years of battery life can be achieved with a single coin-cell battery [5]. However, the BLE protocol alone cannot provide Internet connectivity, and additional gateways are required to bridge the communication and forward the packets to/from the Internet. These IoT gateways incur additional costs to users, which are substantially higher than the price of individual IoT devices. Moreover, unlike the WiFi infrastructures (such as WiFi APs and routers), these IoT gateways have a much smaller installed base and do not have the same global adoption and coverage as WiFi. The necessity to use IoT gateways is thus a major obstacle to the widespread adoption of IoT [6, 7].

WiFi–Bluetooth CTC eliminates the need for gateways and thus effectively overcomes this obstacle. The state-of-the-art CTC solutions, FLEW [8] and Unify [9], achieve bidirectional communication between FSK (BLE modulation) and WiFi. With these solutions, IoT devices can use low-cost and energy-efficient FSK/BLE chips while having direct Internet access and global routability via the ubiquitous WiFi infrastructures.

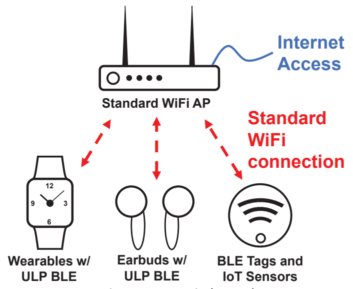


Figure 1: DREW **is a end-to-end (e2e) system for ULP BLE devices to have standard WiFi connection.**

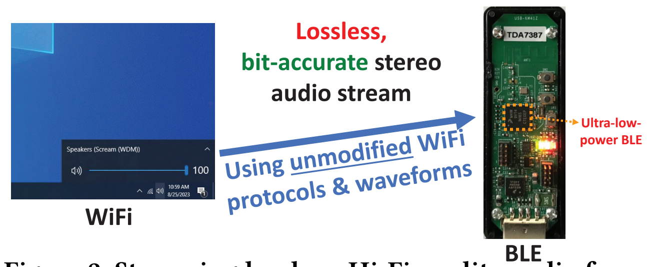

Lossless, bit-accurate stereo audio stream
Using unmodified WiFi protocols & waveforms
WiFi
BLE
Ultra-low-power BLE

Figure 2: Streaming lossless, Hi-Fi quality audio from WiFi to ultra-low-power BLE chips

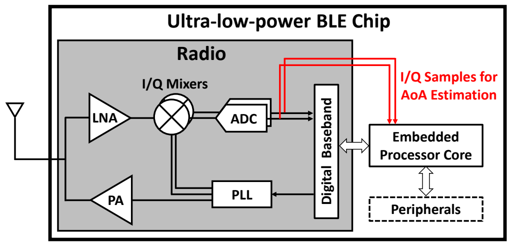


Figure 3: Architecture of ULP BLE chips [3, 4].

However, FLEW and Unify are both based on older FSK chips, which have several key differences from the latest ultra-low-power (ULP) BLE chips. Specifically, older FSK chips use transmission mixers to modulate signals. Mixers have high insertion loss (passive mixers) or have high power consumption (active mixers). Newer BLE chips instead use **direct modulation** [3, 4, 10], where the signal is directly modulated by the PLL (phase-locked loop) and then amplified by the power amplifier (PA) (Fig. 3). Eliminating mixers between the PLL and PA is crucial for enabling ultra-low power consumption with more than 10 years of battery life. Such ULP chips only consume 6.1mA at 3.6V (KW41Z [3]) or 6.1mA at 3V (CC2650 [4]), whereas the CC2541 [11] chip used in Unify consumes ~12mA at 3.6V. On the other hand, FLEW and Unify require using mixers to flip the phase of the carrier for implementing the phase-shift keying modulation of WiFi. Therefore, they are not applicable to ULP BLE chips.

Another limitation of prior WiFi–Bluetooth CTC solutions is the 1Mbps throughput limitation. In particular, these WiFi–Bluetooth CTC works [8, 9] rely on the similarity between the WiFi BPSK and 1Mbps FSK waveforms in order to demodulate standard BPSK waveforms using FSK hardware. However, this similarity does not exist in WiFi's QPSK waveforms. In particular, QPSK is *not* BPSK clocked at twice the speed. In fact, QPSK is two orthogonal BPSK transmissions while keeping the same symbol rate. It is much more challenging for FSK hardware to receive WiFi's QPSK modulation, since QPSK cannot be demodulated by clocking the receiver twice faster. As a result, the state-of-the-art CTCs [8, 9, 12–16] focus exclusively on BPSK waveforms, which limits their maximum physical-layer throughput to 1Mbps.

In this paper, we propose DREW (**D**ouble-th**R**oughput **E**mulated **W**iFi), which enables conventional WiFi connectivity on newer, ULP BLE chips. Because their hardware is significantly different, DREW addresses new challenges while exploring and leveraging new opportunities. DREW is an end-to-end WiFi system containing several key technical innovations: Specifically, we

- Propose and demonstrate new use of BLE's IQ sampling to receive standard, unmodified WiFi packets;
- Devise efficient algorithms, with SIMD parallel processing, for detecting and demodulating WiFi packets;
- Double the downlink throughput (compared to prior work) by uniquely supporting QPSK demodulation;
- Transmit WiFi packets by carefully controlling the PA on mixer-less BLE chips, which consumes ultra-low power; and
- Coordinate Tx/Rx to emulate WiFi's CSMA/CA and timing, specifically on ULP BLE chips.

The key design requirements – QPSK demodulation and mixer-less transmission – lead to brand-new designs for both WiFi-to-BLE and BLE-to-WiFi communications, which are very different from all prior work. Our WiFi-to-BLE design is based on an innovative use of the IQ sampling feature on modern BLE chips. Specifically, Bluetooth *localization* is a key feature on modern BLE chips, and the localization relies on processing the Angle-of-Arrival (AoA) of the incoming signal. The AoA is estimated by processing the IQ samples from the receiver. Therefore, modern BLE chips are capable of *IQ sampling* where the IQ samples are exposed to the processor core for post-processing. This feature has been incorporated into the standard since Bluetooth 5.1 [17]. For DREW, the key idea is that we can use this feature for data communication (instead of localization) to capture standard WiFi signals.

The IQ samples collected are relatively narrow-band and we design special algorithms to detect, synchronize and demodulate standard WiFi packets. Furthermore, we aim to process the IQ samples in "real time" with an ARM Cortex-M0 core at 48MHz, since Cortex-M0 is the smallest ARM processor [18] and its nominal speed is 48MHz. To meet this goal, we come up with a special design that leverages the SIMD parallel processing. We also design algorithms to support QPSK and double the downlink throughput.

This real-time processing of IQ samples is important for DREW to behave and coexist like a typical WiFi chip. If the processing is not done in real time, the demodulation lags behind the waveform coming from the antennas, which results in processing delays at the end of a WiFi packet. Any such delays directly decrease transport-layer throughputs. More importantly, since the WiFi standard requires immediately sending an acknowledgement packet after receiving a packet, the processing delay prevents sending ACKs within the time limit, which makes such a design incompatible with off-the-shelf WiFi devices.

The challenge in the opposite direction is enabling BLE-to-WiFi communication with ULP BLE chips. Since ULP BLE chips do not have transmission mixers, we must generate the waveforms using only the PLL or the PA. We overcome this challenge by proposing a novel way of controlling the PA for BLE-to-WiFi communication. It is applicable to ULP BLE chips as it only relies on changing PA's power level.

In order for BLE chips to interoperate with unmodified WiFi devices, DREW must follow the timing and medium access mechanism of the WiFi standard. DREW also coordinates packet transmission and reception, and overcomes the timing challenges of ULP BLE chips.

As a complete system, DREW follows the WiFi standard, including probing, authentication, association and encryption. DREW uses the 4-way handshake and WPA2-CCMP (AES encryption), and is secured against eavesdropping.

DREW uses WiFi DSSS waveforms because DSSS provides the highest reliability and robustness. Since no other WiFi waveforms match the same reliability and because of WiFi's backward compatibility, DSSS is an important baseline modulation for current and future WiFi standards and the latest WiFi devices are still required to support it.

DREW's novel algorithm and use of IQ sampling achieve a physical-layer downlink bit rate of 2Mbps (= 2x the rate of FLEW and Unify). This enables new applications and benefits a wide variety of BLE systems, including IoT, wearable, and other ULP devices. For example, DREW can be used for delivering audio and short video messages to wearable devices. With the new transmission design, DREW (908kbps) also provides about 26% higher uplink throughput than FLEW (721kbps), which is particularly useful for IoT applications.

Another important application is high-quality audio streaming. We demonstrate that DREW can directly stream lossless stereo audio from WiFi to BLE devices (Fig. 2). This standard audio stream has a bit rate of 1.411Mbps, which well exceeds the throughputs of FLEW and Unify. In fact, even conventional Bluetooth headphones cannot stream lossless, uncompressed audio because Bluetooth can only support compressed audio streams (e.g., SBC, AAC, aptX), which degrade audio quality (typically 4x compression) and introduce latencies. In contrast, DREW leverages audio over IP (i.e., WiFi), and thus audio is not compressed by Bluetooth transmitters. Without compression, DREW offers a bit-accurate, wire-equivalent audio experience with ultra-low power consumption of BLE chips.

2 SYSTEM DESIGN

2.1 WiFi to BLE

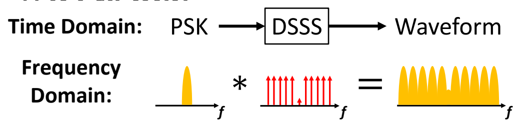
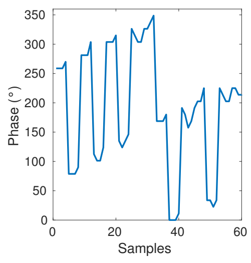


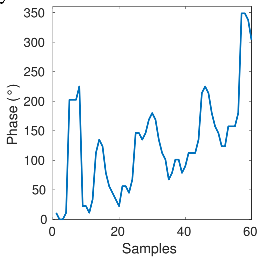
Figure 4: The DSSS modulation process in WiFi.

2.1.1 Observation. To design the WiFi-to-BLE communication, we start by looking into a key block in the WiFi transmitters. For an 802.11 waveform, a WiFi packet (after scrambling and differential coding) is converted to a BPSK or QPSK bitstream with a symbol rate of 1MSym/s. Each PSK symbol is then multiplied by the 11-chip Barker sequence to generate the waveform, as shown in Fig. 4. The spectrum of a periodic 11-chip Barker sequence is 11 impulses located at $\pm1, \pm2, \ldots$MHz. Since multiplication in the time domain corresponds to convolution in the frequency domain, the result of the DSSS process is replication of the original PSK spectrum at $\pm1, \pm2, \ldots$MHz. If we capture the waveform at -1MHz and calculate its phase, we can recover the PSK symbols. By leveraging the close relationship between BPSK and FSK, prior work [8] subsequently recovers the BPSK symbols with an FSK demodulator operating at an additional frequency shift. However, since there is no similar relationship between QPSK and FSK, this design cannot be extended to QPSK demodulation.

Our key idea is to leverage the IQ sampling capability on modern BLE chips to capture the PSK spectrum. By processing the IQ samples, both BPSK and QPSK demodulations become possible, and thus we can double the throughput. Since BLE chips are designed to receive FSK waveforms at 1MSym/s, the IQ sampling works well for receiving BPSK/QPSK waveforms at 1MSym/s.



(a) BPSK



(b) QPSK

Figure 5: Utilizing BLE's IQ sampling to receive standard WiFi waveforms.

As a real example, we use an off-the-shelf BLE chip to perform IQ sampling on BPSK and QPSK WiFi packets. The

WiFi packets are sent at 2452MHz and the BLE chip operates at 2451MHz. Figs. 5(a) and 5(b) show the phase of the IQ capture. The IQ sampling runs at 4MHz, and thus each WiFi symbol is represented by 4 data points. In Fig. 5a, the phases change by either 0° or 180° in groups of four. Similarly for QPSK, the phases change by 0°, 90°, 180° or 270° in Fig. 5b. The figure also illustrates we should ideally select the phases near the center of each symbol for accurate sampling.

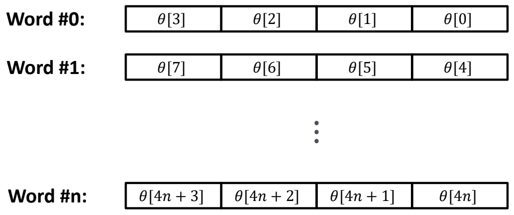

Word #0:
$\theta[3]$ $\theta[2]$ $\theta[1]$ $\theta[0]$
Word #1:
$\theta[7]$ $\theta[6]$ $\theta[5]$ $\theta[4]$
Word #n:
$\theta[4n+3]$ $\theta[4n+2]$ $\theta[4n+1]$ $\theta[4n]$

Figure 6: Packing phases into 32-bit words.

2.1.2 Processing Phases. To process the collected phases, we first pack them into 32-bit words, as illustrated in Fig. 6. We use 32-bit words because they are the intrinsic unit for most arithmetic and logical operations on ARM microcontrollers. Since each WiFi symbol spans 4 phases and WiFi uses differential coding, we calculate the phase difference between WiFi symbols. Let the phases be $\theta[0], \theta[1], \theta[2], \cdots$. We calculate 4 sets:

- $\{\theta[4n] - \theta[4n-4], n \in \mathbb{N}\}$
- $\{\theta[4n+1] - \theta[4n+1-4], n \in \mathbb{N}\}$
- $\{\theta[4n+2] - \theta[4n+2-4], n \in \mathbb{N}\}$
- $\{\theta[4n+3] - \theta[4n+3-4], n \in \mathbb{N}\}$.

These sets represent the phase differences sampled at slightly different time instants within WiFi symbols, and the one with the optimal sampling instant will have the best estimates. Fig. 7 shows an example of using IQ sampling to capture an actual QPSK WiFi packet (with a BPSK header and a QPSK payload). In Fig. 7, $\{\theta[4n+2] - \theta[4n+2-4]\}$ has the best estimates. For the BPSK header ($n < 127$), the phase differences between symbols are 0° (bit '0') or 180° (bit '1'). For the QPSK payload ($n \geq 127$), the phase differences are 0° (bits '00'), 90° (bits '01'), 180° (bits '11') or 270° (bits '10').

For each phase within a word, we use 5 bits to represent the range $[0, 360)$. The overflow and underflow properties of integers map nicely to the fact that the phase wraps around every 360°. That is, if we add or subtract two phases, the lower 5 bits will represent the principal angle of the result (even when an overflow or underflow occurs).

By packing multiple phases in a 32-bit word, we can use one instruction to perform the same arithmetic operation on 4 phases simultaneously. This type of parallel computation is known as *Single Instruction Multiple Data* (SIMD). Although ARM has an official SIMD implementation ("NEON" [19]), it is usually not available on low-end ARM microcontrollers. However, we can still apply SIMD with only conventional instructions (such as `ADD` or `SUB`). In Fig. 6, if we calculate

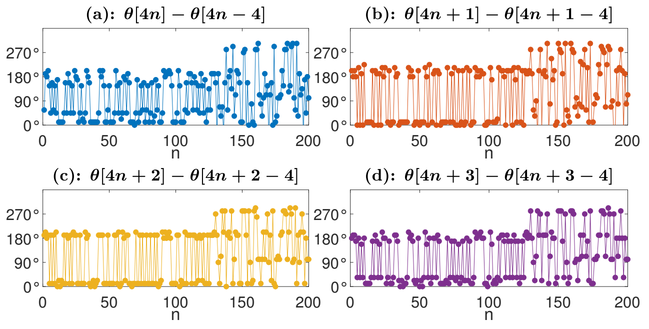


Figure 7: Phase changes between WiFi symbols.

`Word[1]−Word[0]`, the upper 8 bits will be $\theta[7] - \theta[3]$ and the next 8 bits will be $\theta[6] - \theta[2]$, etc. However, if $\theta[6] < \theta[2]$, the upper 8 bits will be $\theta[7] - \theta[3] - 1$ because of borrowing. To account for this, we can instead calculate `Word[1]+0x80808080−Word[0]` so that the borrowing will never occur across byte boundaries and we will take the lower 5-bits of each byte as results.

2.1.3 BPSK Demodulation. For BPSK demodulation, we need to determine whether the phase difference is 0° or 180°. Taking the first half of Fig. 7(c) as an example, we can make a bit decision by first adding a constant 90° to the phase difference and then determining whether the result has a principal angle in $[0°, 180°)$ or in $[180°, 360°)$.

This process can be implemented very efficiently using SIMD and leveraging the property of binary representation. With the phase format we use, 360° corresponds to a value of 32, and thus adding 90° to each phase difference is adding 8 to each byte. We use SIMD and add 0x08080808 to each 32-bit word. We can further merge two additions and simply calculate `Word[i]−Word[i-1]+0x88888888`. Within this 32-bit result, the principal angles are the lower 5 bits of each byte and the most significant bit (of these 5 bits) will indicate whether the angle is in $[0°, 180°)$ or $[180°, 360°)$. Therefore, we can recover the WiFi BPSK bitstream by continuously calculating `Word[i]−Word[i-1] +0x88888888` and extracting that bit from every 32-bit result.

Table 1: Processing with and without SIMD

DREW	Without SIMD
SUB r1, r0, r1	SUB r2, r0, r1
ADD r1, r1, r2	ADD r3, r2, #0x8; first byte
	LSR r0, r0, #8
	LSR r1, r1, #8
	SUB r2, r0, r1
	ADD r4, r2, #0x8; second byte
	LSR r0, r0, #8
	LSR r1, r1, #8
	SUB r2, r0, r1
	ADD r5, r2, #0x8; third byte
	LSR r0, r0, #8
	LSR r1, r1, #8
	SUB r2, r0, r1
	ADD r6, r2, #0x8; fourth byte

SIMD is a critical part of DREW to enable real-time IQ processing. Furthermore, the SUB and ADD instructions used in our SIMD method are generic 32-bit subtract and addition. Thus, this method is not specific to ARM and is thus applicable to 32-bit processors in general. Using BPSK demodulation as an example, Table 1 compares processing phases with and without SIMD. DREW takes 2 cycles whereas processing without SIMD takes 14 cycles. With a processor running at 48MHz, new sets of IQ samples are generated every 48 cycles, and other processing steps (loading phases, extracting bits, pattern matching) take about 39 cycles. Therefore, without SIMD, DREW cannot process the IQ samples in real time. Also, processing without SIMD requires using additional registers (r3∼r6), and additional cycles are needed for managing these lower registers of the ARM processor.

2.1.4 QPSK Demodulation. We extend the above process to demodulating QPSK symbols. The major difference is that the phase change of QPSK symbols can be 0°, 90°, 180° or 270°. In Fig. 7(c), QPSK demodulation can be achieved by first adding a constant 45° to the phase difference and then determining the quadrant of the phase difference.

To demodulate QPSK bits, we calculate `Word[i]−Word[i−1]+0x84848484`. (Note that the value '4' corresponds to 45°.) The quadrant of the phase difference is naturally represented by the upper 2 bits of the lower 5 bits of each byte. Finally, since WiFi uses Gray code for the QPSK symbol mapping, we convert the "dibits" from binary code to Gray code. We build a simple lookup table for this conversion.

2.1.5 Packet Detection and Time Synchronization. To receive an entire WiFi packet, we need to locate the beginning of an incoming WiFi packet. This process is known as *packet detection* in which the receiver constantly monitors the incoming waveform and triggers the packet decoding when a valid packet arrives. Packet detection relies on the principle of pattern matching. The WiFi standard defines special BPSK bit patterns (i.e., PLCP preamble) that the transmitter should use, and the receiver searches for the bit pattern to detect the start of a valid WiFi packet.

At the start of a standard packet, a WiFi transmitter sends the PLCP preamble. However, since WiFi transmitters later apply bit-scrambling to the entire packet, the over-the-air packet contains the scrambled PLCP preamble. Thus, a possible implementation is to first descramble the demodulated BPSK bits and then search for the original PLCP preamble.

Prior work [8, 9] point out that the WiFi standard [20] defines the exact scrambler seed all transmitters should use for BPSK packets, and hence the descrambling step can be eliminated by directly searching for the scrambled PLCP preamble among the demodulated bits.

Different from prior work, however, DREW must achieve a precise synchronization suitable for QPSK. Furthermore, this synchronization should be at the IQ-sample level, since the FSK demodulator (that prior work used) cannot decode QPSK payloads. At a finer time granularity, we sample each WiFi BPSK symbol with 4 phase samples, and thus there are 4 possible sampling time offsets, as shown in Fig. 7. Equivalently, each 32-bit word contains 4 bytes and provides 4 possible BPSK estimates at slightly different sampling time instants. If the optimal sampling time instant is known, we can directly extract the BPSK bit from one of the four possibilities. However, there is no intrinsic time synchronization between a transmitter and a receiver, and a WiFi waveform could start at $\theta[0]$, or at $\theta[1], \cdots$. Thus, in addition to packet detection, we need to establish precise time synchronization when a valid packet arrives.

We design an algorithm that solves packet detection and time synchronization simultaneously. We form 4 independent bitstreams from the 4 BPSK estimates of every word and search for the scrambled PLCP preamble from each bitstream. We use one register as one 32-bit FIFO for each bitstream and the BPSK estimates are constantly shifted into the FIFO. This can be efficiently implemented by shifting a demodulated BPSK bit into the carry bit with `LSR` (logical shift right) and shifting the carry bit into the corresponding FIFO with `LSL` (logical shift left) and `ADC` (add with carry). FIFO #0 contains BPSK bitstream estimated from $\{\theta[4n] - \theta[4n-4]\}$; FIFO #1 contains BPSK bitstream estimated from $\{\theta[4n+1]-\theta[4n+1-4]\}$, etc. Next, we compare each FIFO with the bit pattern of the scrambled PLCP preamble using `CMP` (compare). An exact match indicates that the start of a packet is detected at a fine-grain sampling time instant.

Since there are 4 BPSK estimates for every WiFi symbol, one valid PLCP will ideally trigger multiple exact matches. In Fig. 6, if a WiFi packet starts at $\theta[0]$, exact matches will be triggered at $\theta[4k], \theta[4k+1], \theta[4k+2]$ and $\theta[4k+3]$ (for some k). In such a case, we will subsequently use the phase samples near the center of symbols (e.g., $\{\theta[4n+2], n \in \mathbb{N}\}$) for header and payload decoding.

If a WiFi packet starts at $\theta[2]$, exact matches will be triggered at $\theta[4k+2], \theta[4k+3], \theta[4k+4]$ and $\theta[4k+5]$ and we will select $\{\theta[4n+4], n \in \mathbb{N}\}$. In the general case, if any exact match occurs at $\theta[4k], \theta[4k+1], \theta[4k+2]$ or $\theta[4k+3]$, we consider a valid packet is detected and will perform the demodulation and the FIFO matching for one extra round to see if there are more matches at $\theta[4k+4], \theta[4k+5]$ or $\theta[4k+6]$. Based on whether the preamble is detected or not at $\theta[4k \sim 4k+6]$, we can establish fine-grain time synchronization and select an appropriate sampling time offset l such that $\{\theta[4n+l], n \in \mathbb{N}\}$ are near the center of symbols. After the synchronization, only $\{\theta[4n+l], n \in \mathbb{N}\}$ will be used for decoding.

In our design, the result of preamble matching at $\theta[4k \sim 4k+6]$ is stored as an integer with one-hot encoding. We build a lookup table that directly converts this encoding to the best sampling time offset l. In our implementation, this sampling time offset l is stored in the form of the number of right shifts to be applied after performing SIMD.

2.1.6 *Long Preamble and Short Preamble.*

The WiFi standard defines two possible PLCP preambles, so a transmitter may use long (144 bits) or short (72 bits) preambles. However, the long preamble is always used for BPSK payloads. Since prior work focused on BPSK only, their packet detection is for long PLCP preamble only.

For `DREW`, a transmitter may use either the long or short preamble for QPSK payloads. Technically, the short preamble is an optional (and typically configurable) feature and a receiver is not strictly required to support it. However, we find that some off-the-shelf WiFi APs use the short preamble by default. To ensure the best compatibility, we aim to support both long and short preambles simultaneously.

Since both the long and short PLCP preambles use BPSK modulation, the packet detection and time synchronization are applicable to both. However, the short preamble is a different bit pattern scrambled with a different seed. Two types of preambles thus have different bit patterns, and `DREW` has to search for both simultaneously. This can be implemented efficiently since we already have the demodulated BPSK bits in the FIFOs. We simply add 4 extra compares (`CMP`) to search for the short preamble from 4 independent bitstreams.

The pattern that the `CMP` instruction tries to match can be any 32-bit sub-sequence of the scrambled (long or short) PLCP preamble. We use `0x78869b04` (for the long preamble) and `0xfa51c63f` (for the short preamble), which are the scrambled bit patterns near the end of the preambles. With such selections, `DREW` can detect a packet as long as the last few bytes of the PLCP preamble are received, even when the receiver starts sometime later than the beginning of the packet.

2.1.7 *Supporting Realtek's Non-standard Preamble.*

Prior work [8, 9] observed that Realtek has a wrong WiFi implementation that produces incorrect long preambles. Consequently, Realtek's packets also have a different (scrambled) bit pattern near the end of the preamble. To detect Realtek's packets, we instead search for `0x1e21a6a5`, which is a 32-bit subsequence using Realtek's PLCP format.

A unique feature of `DREW` is supporting QPSK packets, which may use the short preamble format (unlike BPSK packets that always use the long preamble). We find that Realtek's short preamble is also wrong. Using simulation and analyzing waveform captures, we have traced the issue down to the scrambler seed. Sec. 16.2.3.9 of the WiFi standard [20] specifies that a transmitter should initialize the scrambler with `0b0011011` for all short PLCP packets. In contrast, Realtek always uses `0b0110010` as the scrambler seed. The scrambled short PLCP preamble is thus a different BPSK pattern. We solve this issue by using `0xda1503e9` (the last few bytes of this pattern) to detect such packets.

2.1.8 *Parsing PLCP Header and Bit Descrambling.*

The PLCP header follows the PLCP preamble and contains the duration and modulation of the payload. The header uses BPSK (for the long PLCP) or QPSK (for the short PLCP). We demodulate BPSK or QPSK bits using the time-synchronized phases ($\{\theta[4n+l]\}$). We then descramble the bitstream to recover the actual header bytes. We also convert the duration of the payload to the number of bytes of the payload.

2.1.9 *Decoding Payload and Checking CRC.*

The WiFi payload begins after the PLCP header. Depending on the "SIGNAL" byte in the header, `DREW` dynamically selects BPSK or QPSK demodulation. Payload is recovered after demodulation and descrambling. The payload is then stored in the memory.

The last 4 bytes in the payload are the preceding bytes' CRC32 values. When receiving the PLCP header, we initialize a 32-bit register to hold the CRC32 result. After recovering each WiFi payload byte, we calculate the updated CRC32 using a lookup-table-based algorithm. The updated CRC32 is a function of the current CRC32 value and the WiFi byte received. At the end of packet reception, we check the CRC32 value and discard any packet with a mismatched CRC.

2.2 BLE to WiFi

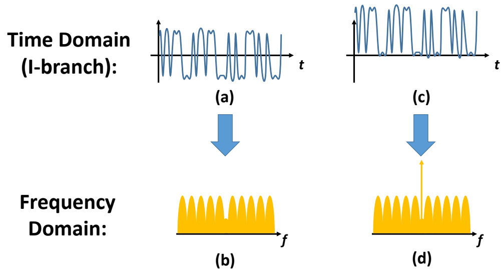


Figure 8: Comparing BPSK and BPSK with a DC offset.

2.2.1 *Observation.*

To design the BLE-to-WiFi communication, we look into the standard WiFi waveform. Fig. 8(a) shows a typical WiFi BPSK waveform. We plot the time-domain waveform in the baseband (IQ form) and only the I-branch is shown because the Q-branch is always 0 for BPSK. We can see that the time-domain waveform is a PSK modulation (the I-branch is either 1 or -1) and each WiFi bit is represented by a Barker sequence. Therefore, each WiFi bit ultimately becomes either $\{1, -1, 1, 1, -1, 1, 1, 1, -1, -1, -1\}$ or $\{-1, 1, -1, -1, 1, -1, -1, -1, 1, 1, 1\}$. The Barker sequence results in a corresponding spectrum (Fig. 8(b)).

Fig. 8 also shows a waveform related to (a). When a constant offset is added, (a) becomes (c). In the spectrum, this

DC is manifested as an arrow at $f = 0$, as shown in (d). Note that spectrum (b) and (d) are identical except at DC.

The DC ($f = 0$) in the baseband corresponds to the carrier frequency (e.g., $f = 2412$MHz) in the passband. A constant DC level (or a constant carrier) does not carry information and receivers commonly remove DC prior to demodulation because the RF circuitry can produce unwanted DC residue and cause an uncertain DC level in the baseband. For example, the bias voltage at which the mixers or ADCs operate becomes DC in the baseband. Also, the LO leakage in zero-IF receivers will introduce an uncertain amount of DC in the baseband. Specifically, the LO signal (e.g., $f = 2412$MHz) will have some leakage that couples (along with the desired signal) into the RF port, and such a leakage becomes a DC uncertainty in the baseband after downconversion.

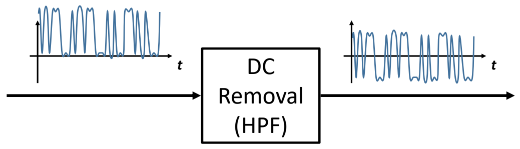


Figure 9: DC removal in receivers.

Suppose a WiFi device receives the waveform in Fig. 8(c), the DC will be removed in the baseband before demodulation. As illustrated in Fig. 9, Fig. 8(c) becomes Fig. 8(a) after DC removal. Therefore, if we substitute Fig. 8(a) with Fig. 8(c) at the transmitter, the receiver still sees the same baseband signal after DC removal and can therefore decode the packet correctly.

This observation is key to the design of BLE-to-WiFi communication. Fig. 8(a) is difficult to implement on mixer-less BLE chips as it requires precisely inverting the phase of the carrier at a relatively high speed. In contrast, Fig. 8(c) can be implemented by switching the carrier on and off.

2.2.2 Sending Bits. This switching can be realized by turning on and off any component along the transmission signal path. A straightforward and effective design is to use assembly to control the power level of the PA. Since the STR (store) instruction takes 2 cycles in ARM microprocessors, we update the power level of the PA at 8MHz, which gives enough headroom for microprocessors running at 16MHz or higher (which is typical for BLE chips). The Barker sequence also scales well at 8MHz (and becomes $\{1, 0, 1, 0, 1, 1, 0, 0\}$). To send a WiFi bit, the power levels are $\{1, 0, 1, 0, 1, 1, 0, 0\}$ for phase $0°$ and $\{0, 1, 0, 1, 0, 0, 1, 1\}$ for phase $180°$, assuming that '1' is the maximum power level of the PA.

2.2.3 Sending Packets. By repeating the same process for every bit in a packet, we can send a complete WiFi packet. Instead of spelling out all power levels of a WiFi packet with thousands of instructions, we design a compact routine that sends a WiFi packet with less than 20 instructions. Specifically, the sketch of the transmission routine is:

```
txloop:
   R1 → [PA power register]
   R0 → [PA power register]
   R1 → [PA power register]
   R0 → [PA power register]
   R1 → [PA power register]
   R1 ← Next WiFi bit (from memory)
   R0 → [PA power register]
   R0 ← $\overline{R1}$
   Jump to txloop if there are more bits
```

The idea is that since '10101100' (or '01010011') contains consecutive duplicates, instead of setting the power level twice, we can use those cycles to get the phase of the next WiFi bit and prepare the power levels. Note that R0 ← $\overline{\text{R1}}$ is not a memory operation, and thus we can insert a conditional jump afterwards without affecting the timing. In our implementation, we also insert NOP at appropriate locations so that each txloop takes exactly 1μs.

2.2.4 Zero-Wait Packet Transmission. The above assembly routine works well for sending WiFi packets. However, it requires copying the entire packet to the memory before transmission, which causes delays and decreases throughputs. This issue is manifested on FLEW [8] and Unify [9].

We design another assembly routine that eliminates this waiting time, thus increasing throughputs. We start transmission as soon as the first byte is ready. During the cycles where the power level stays the same, we collect the WiFi bits (from UART), update R0 and R1 accordingly, and copy the WiFi bits to memory. We still copy the WiFi packet to memory so that we can run the routine in Sec. 2.2.3 to retransmit the packet in case of unsuccessful transmissions.

2.3 MAC Layer

DREW is to make a ULP BLE chip *indistinguishable* from a WiFi chip (and thus ensuring compatibility). To achieve this goal, the ULP BLE chip should emulate WiFi's MAC, including CSMA/CA, Tx/Rx switching (timings) and ACK/CTS handling. Since the emulated MAC layer of prior work [8, 9] is shown to be effective and highly compatible across WiFi vendors, DREW's MAC layer follows the same design principles with several differences to accommodate the entirely different physical-layer design and QPSK packets.

To emulate CSMA/CA, DREW uses RSSI from the radio to perform Clear Channel Assessment (CCA) before transmission, similar to prior work. Although DREW's IQ sampling could, in theory, alternatively be used to implement CSMA/CA, we chose to directly read the RSSI register because it is simple and effective. Random backoff is also implemented.

Applying the same design principle, we create DREW's finite-state machine, shown in Fig. 10. The FSM governs the state transitions and associated actions. The actions performed after receiving ACK, RTS or unicast packets closely follow the WiFi standard. However, the FSM has several differences from prior work. First, because of the *zero-wait packet transmission*, the FSM is actually simpler and has higher transmission throughputs. DREW can directly go to the the Tx state once the channel is clear (without copying the entire packet first), whereas prior work have several wait states and flags in order to reuse part of the downlink (Rx) airtime to copy uplink (Tx) data. DREW's design especially benefits highly asymmetrical traffic (e.g., UDP uplink). There is still one flag associated with the Tx path to ensure that each unicast packet is properly acknowledged before a new packet is used in the Tx state. In Tx, DREW will only fetch a new packet if flag is 0.

Second, since DREW supports QPSK, the Rx state searches for both long and short preambles simultaneously, which is efficiently implemented by adding 4 compare instructions. Also, the FSM handles 3 combinations (long-BPSK, long-QPSK, short-QPSK) that an incoming packet may have.

As prior work pointed out, WiFi's Tx/Rx timings can be particularly tricky for BLE chips to emulate. Surprisingly, we found that using the IQ sampling actually helps relax some of the timing requirements, such as the Tx–Rx turnaround. In particular, direct processing of IQ samples eliminates any delay or timing uncertainty that the FSK demodulation logic might have. More importantly, IQ sampling allows a greater flexibility in selecting the bit patterns for packet detection. Prior work has more restrictions in selecting the bit patterns because the FSK demodulator has to be stabilized with the "1010101" pattern first. For DREW, we can use the bit patterns near the very end of a WiFi preamble and thus relax the Tx–Rx turnaround requirement. We can invoke the normal Rx warm-up sequence and still satisfy WiFi timing.

Because DREW only relies on the PA to transmit packets, we can leverage the power override idea (proposed in [9]) to satisfy the Rx–Tx turnaround. Specifically, for ACK or CTS transmission, DREW keeps IQ sampling running (so that the PLL is still locked) while powering on the PA (and signal buffers along the Tx path). The carrier is modulated by controlling the power level. After ACK or CTS, the power overrides are turned off and another Rx warm-up is invoked.

2.4 Implementation

We have implemented DREW on COTS BLE chips. We use the Freescale (now NXP) Kinetis KW41Z ultra-low-power [3] BLE chip as the hardware platform because its reference manual [21] and register maps are publicly available. The KW41Z is a single-chip SoC that has an ARM Cortex-M0+ core [22] running up to 48MHz. In the evaluations, we use the USB-KW41Z development board [23] from NXP. This

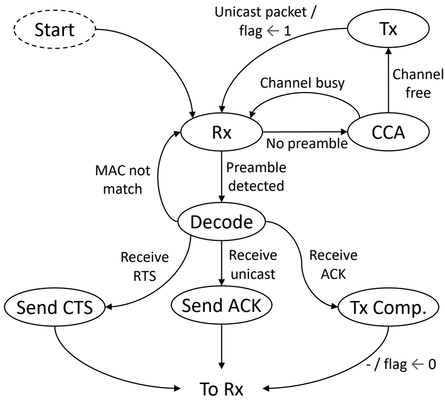


Figure 10: Finite state machine.

KW41Z SoC can also be found in industrial IoT modules made by Panasonic Industry [24] and u-blox [25].

The firmware is written in C and is developed in the MCUXpresso IDE [26] with the GNU ARM Embedded Toolchain [27]. Time-sensitive tasks (such as packet detection, demodulation or transmission) are written in ARM assembly (with the Thumb instructions). The USB-KW41Z development board has an on-board debugger, which is a separate microcontroller running the OpenSDA [28] debugging functions. We also use this microcontroller as the USB-UART bridge. We rewrote the USB-UART bridge firmware ourselves because we found that SEGGER's UART implementation does not work properly at higher (>1MBaud) speeds.

2.4.1 IQ Sampling. The reference manual of KW41Z [21] provides detailed instructions of IQ sampling using DMA. We configure KW41Z to convert IQ to phases and pack them in the 4-byte format as shown in Fig. 6. We also enable the channel filter that runs a 2-sample moving mean.

Instead of using the DMA, we use the ARM core to fetch the phases and process them directly. A new 32-bit word (containing 4 phases) is fetched every $1\mu s$. To help the ARM core distinguish a new word from the old one, we slightly increase (about 0.1MHz) the RF frequency, ensuring the phases change slightly each time and consecutive samples are different. This frequency bump is later compensated for when calculating the phase differences.

2.4.2 Clocking. We found that setting the CPU and peripheral clocks on the KW41Z chip requires special attention. This is because KW41Z uses a legacy clock module (MCG) from the Motorola (later Freescale) 68HCS08 family [29]. The KW41Z has two major clock sources: a 32MHz oscillator and a 32.768kHz RTC (real-time clock). The BLE radio circuitry always uses the 32MHz clock, whereas the ARM core must use the FLL (frequency-locked loop) inside the MCG to reach the advertised speed of 48MHz.

We design the receive codes to run at 48MHz. For transmission, however, we find that the 48MHz clock from the FLL has too many jitters for the transmission routine. Broadcom and Ralink WiFi chips can actually tolerate this waveform jitters. However, the jitters have a noticeable performance impact with Atheros chips.

We solve this issue by dynamically switching the clock source so that the transmission routine is clocked from the 32MHz oscillator. Because WiFi only allows a very short time between Tx and Rx, clock switching also has to be swift. We find that this is possible when the clocks are *coherent*. Therefore, we use the FLL to generate a 48MHz clock from the 32MHz clock and use it for the Rx routine. For the Tx routines (including sending ACK and CTS), we directly switch on the PA, LO and various signal buffers with the `OVRD1` and `OVRD3` registers. Since both the ARM core and the PA use the same clock source during transmission, we set `GASKET_BYPASS` to 1 to bypass unnecessary clock synchronization.

2.4.3 Driver. We write a custom driver and `DREW` is directly compatible with the Linux kernel and the `mac80211` module. In addition to normal Tx/Rx paths, monitor mode and packet injection also work properly.

3 EVALUATION

3.1 Experimental Setup

We first run microbenchmarks on the PHY layer. The microbenchmarks evaluate the performance of communication between BLE and COTS WiFi chips. We measure the packet error rate (PER) at different distances in both WiFi-to-BLE and BLE-to-WiFi directions. We use WiFi cards from major chip-makers that are widely used in WiFi infrastructures (APs), including Atheros (now Qualcomm), Broadcom, Ralink (now Mediatek) and Realtek. All chipsets are unmodified in terms of hardware, firmware or driver. We used the default firmware and driver provided by Ubuntu 20.04. We use WiFi channel 9 to avoid background interference.

In the microbenchmarks, we put a WiFi card (or `DREW`) in monitor mode and injected 4096 packets in each test case. Each packet has 1508 bytes of payload and has a unique sequence number. The receiving WiFi card (or `DREW`) is also in monitor mode and we collect the packets in Wireshark. CRC check is enforced and the sequence numbers of correctly-received packets are put into a set. The PER is derived by calculating the percentage of the sequence numbers missing out of 4096. To showcase `DREW`'s unique ability to receive QPSK packets, we perform PER microbenchmarks of BPSK and QPSK demodulation. For Atheros, Broadcom and Ralink, the injected packets are standard WiFi packets with an 802.11 frame type of 0x08. For Realtek, we found that injection of QPSK data (0x08) packets does not work with the logic of the default driver (rtlwifi). We solve this issue by injecting packets with the frame type 0x00, which forces the driver to use the correct modulation specified in the radiotap header.

Table 2: WiFi APs and chipsets used in evaluations.

Chipset Maker	Router	Chipset	Preamble
Atheros	ASUS RT-AC55U	QCA9557	Long
Broadcom	ASUS RT-AC66U	BCM4331	Short
Ralink/ Mediatek	TP-Link TL-WR841N	MT7628NN	Long
Realtek	Edimax BR-6478AC	RTL8192CE	Long
Marvell	Linksys EA3500	88W8366	Short

To evaluate end-to-end performance, we directly connect `DREW` to various off-the-shelf WiFi APs. We measure system-oriented metrics, such as TCP/UDP throughputs and round-trip time (RTT). Table 2 summarizes the WiFi APs and the chipsets inside. All APs are unmodified and WiFi encryption (WPA2) is enabled. Table 2 also lists the default preamble setting. This preamble setting only applies to QPSK packets since BPSK packets always use the long preamble. We use the AP's setup webpage to select WiFi channel 9, and in Realtek's webpage we select QPSK. We use iperf3 [30] to measure throughputs. Specifically, a Windows laptop running an iperf3 server is connected via LAN to the AP. Then, we use `DREW` and run the iperf3 client on the BLE side. We further set the UDP bitrate to 2Mbps for downlink since `DREW` exceeds iperf3's default of 1Mbps.

3.2 Microbenchmark

Table 3: Packet Error Rate (WiFi to BLE) (%)

Chipset	5m		10m		20m	
	BPSK	QPSK	BPSK	QPSK	BPSK	QPSK
Atheros AR9462	0.02	0.00	0.00	0.05	0.07	0.05
Broadcom BCM4313	0.10	0.76	0.07	1.32	0.63	1.83
Ralink RT3290	0.00	0.00	0.00	0.27	0.12	0.49
Realtek RTL8188CE	0.05	0.46	0.05	1.42	0.15	1.44

Table 3 shows `DREW`'s excellent performance in receiving standard WiFi packets. The PER with Atheros is very close to 0.00% across all distances. Thanks to `DREW`'s good reception, the overall packet error comes from very occasional background interferences (e.g., 0.07% PER is 3 packet misses out of 4096 packets) and the performance difference between BPSK and QPSK is insignificant. Ralink also has good performance, achieving 0.00% PER for both BPSK and QPSK at 5m. The PER increases with distance, and QPSK has a higher PER than BPSK because QPSK has smaller decision regions. Broadcom and Realtek have similar performance numbers and are slightly worse than Atheros and Ralink. Even so, all PERs are less than 2% at 20m, corroborating the effectiveness of our algorithms and using IQ sampling. Several variables can impact PER. Actual WiFi cards may have a slight frequency deviation from the ideal WiFi channel, which affects the optimal decision boundaries. Additionally, IQ imbalance and circuit non-linearity can cause slight variations in PER.

Table 4: Packet Error Rate (BLE to WiFi) (%)

Chipset	5m	10m	20m
Atheros AR9462	4.86	7.30	10.03
Broadcom BCM4313	1.42	2.56	7.50
Ralink RT3290	4.57	6.23	12.21
Realtek RTL8188CE	10.86	11.65	10.21

Table 4 shows the PER in the BLE-to-WiFi direction. Because the BLE chip does not have an external PA, its lower transmit power incurs a higher PER. Broadcom shows the best performance. From experiments, we found that Broadcom's chips are capable of correcting even substantial timing jitters and LO leakage. These qualities translate into better PER in realistic settings. Atheros and Ralink show comparable trends, with Ralink having higher PER at greater distances. Realtek shows the worst PER, which is consistent with the prior report [8] of an inferior performance. To investigate Realtek's chip further, we have tested sending packets at longer intervals but the results are consistent. There might be other issues associated with Realtek's monitor mode or its driver, such as unintended WiFi scan.

3.3 TCP and UDP Throughputs

Table 5 shows DREW's transport-layer throughputs. There are multiple factors, besides the PHY performance, that can impact the overall system goodput. The 802.11 standard allows different implementation options and variations, particularly in the MAC layer. For example, devices can choose different CCA methods and they are all 802.11-compliant. Furthermore, the standard does not specify the details of rate adaptation, which dynamically changes the modulation of transmission. WiFi APs may use different or proprietary *rate adaptation algorithms*, leading to throughput variations.

For uplink, the zero-wait packet transmission design provides high throughputs. Atheros and Broadcom perform very similarly with TCP and with UDP. We found that the MAC layer implementation of Ralink works really well when consecutively receiving a series of packets. Combined with our zero-wait packet transmission, Ralink achieves even better performance than Atheros and Broadcom. Realtek's throughputs are mostly comparable but exhibit a more significant decrease at 20 meters due to its worse PHY layer. DREW's throughputs are considerably higher than prior work (since they must copy the entire packet into the memory before transmission). (In general, the difference is ~100kbps for TCP and ~200kbps for UDP.) The Marvell AP is found more sensitive to the residue DC, thus having a lower uplink throughput than others. However, DREW's zero-wait packet transmission still yields higher throughputs at 5m than FLEW.

For downlink, QPSK demodulation doubles the throughput. Ralink performs best and Broadcom also performs well. Broadcom devices exhibit good compatibility and stability but with slightly lower throughputs. Since TCP requires transport-layer ACKs and UDP requires PHY-layer ACK packets, the BLE-to-WiFi performance can affect downlink throughputs. So, Realtek and Marvell have good throughputs at shorter distances but their throughputs decrease as the distance increases. Finally, the Atheros AP we use is based on Atheros' LSDK, and LSDK is found to have a peculiar behavior: RTS/CTS is always used regardless of the configuration. This extra overhead lowers the throughputs.

3.4 RTT

We measure the round-trip time (RTT) through LAN and WAN using `ping`. We ping the AP (LAN) or 8.8.8.8 (WAN) 10 times and calculate the mean (μ) and standard deviation (σ), as shown in Table 6. The ethernet connection between each AP and 8.8.8.8. has an RTT of 6.68ms.

Table 6 shows that DREW has good RTTs of 3~6ms for LAN and 11~13ms for WAN. The RTT is affected by how fast the AP responds and forwards packets and it is dependent on the processing speed of each AP. The Broadcom AP has the fastest processor (1GHz) and the lowest RTT. The Ralink AP has comparatively higher RTT because it uses the slowest processor (575MHz).

3.5 Coexistence

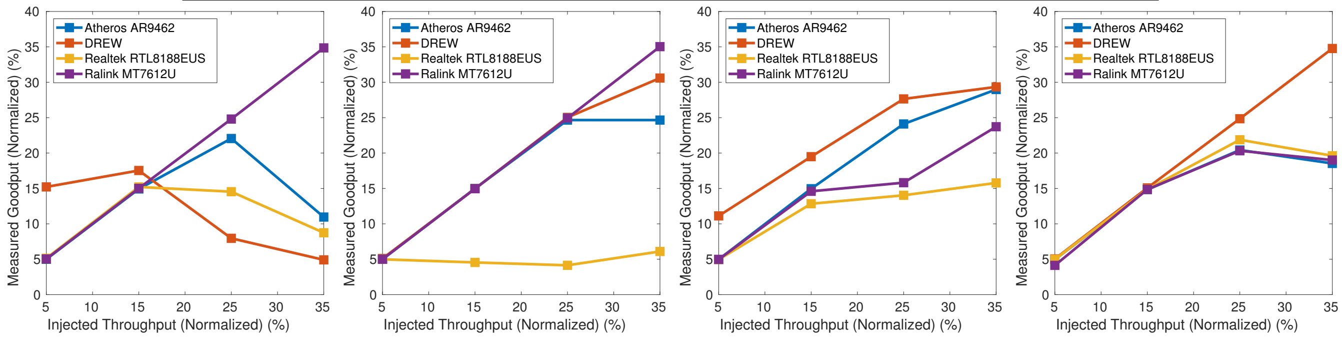
To demonstrate DREW's ability to coexist with other devices like a normal WiFi device, we measure the throughputs of concurrent connections with multiple devices.

3.5.1 Coexistence with WiFi Devices. To evaluate DREW's coexistence with others, we use the Broadcom AP and maintain 4 active connections with Atheros, DREW, Ralink, Realtek WiFi cards. The AP's LAN port is connected to a laptop running 4 iperf3 servers listening on different ports. We first measure the maximum throughputs of individual devices, and then simultaneously run 4 iperf3 clients and inject 5, 15, 25 or 35% of the throughputs per device into the network.

We tested TCP and UDP in both directions and plotted the measurement results in Fig. 11. For each test-case, the network was not saturated when injecting 5% or 15% of throughputs per link. The network is at maximum capacity when injecting 25% per link. The network is oversubscribed and spectrum contention is high when injecting 35% per link. As shown in Fig. 11(a), the devices coexist nicely with each other when the network is not saturated and each throughput matches the injected throughput. The throughput of DREW is actually higher because of the TCP packet size iperf3 uses. For 25% per link, the network is saturated and the throughputs are lower than the injected throughputs, and it is more

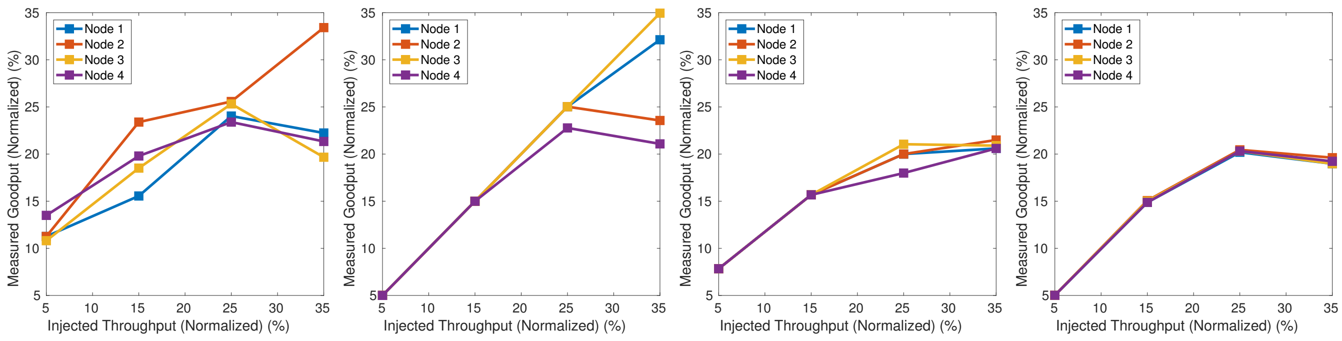
Table 5: TCP/UDP Throughputs (kbps)

Direction	Uplink (BLE to WiFi AP)						Downlink (WiFi AP to BLE)					
Transport	TCP			UDP			TCP			UDP		
Distance	5m	10m	20m	5m	10m	20m	5m	10m	20m	5m	10m	20m
Atheros	757	746	639	891	881	753	1234	1246	1010	1388	1399	1260
Broadcom	770	739	660	872	856	758	1332	1324	1231	1515	1516	1454
Ralink	785	752	672	908	892	765	1344	1337	1258	1608	1607	1509
Realtek	767	735	617	865	843	708	1300	1233	1147	1520	1502	1336
Marvell	693	543	515	779	582	569	1325	1187	1140	1535	1455	1403



(a) TCP UL (b) UDP UL (c) TCP DL (d) UDP DL

Figure 11: Coexistence with COTS WiFi devices



(a) TCP UL (b) UDP UL (c) TCP DL (d) UDP DL

Figure 12: Coexistence with DREW devices

Table 6: Round-trip Time (ms)

	LAN		WAN	
	μ	σ	μ	σ
Atheros	4.56	0.36	11.02	0.15
Broadcom	2.94	0.37	10.68	0.89
Ralink	6.35	0.84	12.62	0.70
Realtek	3.26	0.52	10.65	0.34
Marvell	3.38	0.82	11.38	0.99

so for the oversubscribed case. Of all the clients, the uplink throughput of Ralink is least affected. This is consistent with the findings in Sec. 3.3 where the MAC layer of Ralink utilizes and accesses the spectrum proactively. With a more dynamic protocol like TCP, we observe the throughput of DREW is affected more under saturation. However, it does not completely drop to zero and still allows the data to be transmitted successfully. In the UDP case where the WiFi traffic is more unilateral, Fig. 11(b) shows that the throughputs of Ralink, DREW and Atheros coexist nicely, although they react differently in an oversubscribed environment. Realtek performs badly in this test and cannot access the spectrum efficiently in the presence of other unilateral UDP traffic. For downlink, the data packets are mostly sent by the AP, and thus the results are different from uplink. Figs. 11(c) and (d) show that DREW is least affected in saturated conditions under AP's arbitration.

3.5.2 Coexistence with DREW Devices. To evaluate the coexistence performance of multiple active DREW devices, we use 4 DREW clients and repeat the coexistence evaluation. Fig. 12 shows a similar trend for all nodes since they have the same timing and MAC-layer implementation. The throughputs increase linearly before saturation and taper off at 25% and 35% injections per node. There are some variations in throughputs due to the randomness in the WiFi MAC layer. For uplink, the 4 nodes contend for the spectrum at the same time, so the uplink throughputs vary more than the downlink (since the AP transmits most of the traffic for downlink).

Table 7: Throughputs with active BT connections

	TCP UL	UDP UL	TCP DL	UDP DL
No BT	780 kbps	896 kbps	1359 kbps	1552 kbps
1 BT	786 kbps	896 kbps	1324 kbps	1566 kbps
2 BT	792 kbps	898 kbps	1314 kbps	1586 kbps

3.5.3 Coexistence with Bluetooth Devices. We measure the throughputs of DREW when simultaneously using Airpod earbuds and Sennheiser headphones. Table 7 shows that these active Bluetooth connections do not have significant effect on the throughputs. Furthermore, we do not observe any audio stuttering or discernible defects with these Bluetooth headphones when DREW is running the tests. This good coexistence performance comes from the fact that Bluetooth devices will minimize interference by avoiding the channels used by nearby WiFi APs.

3.6 End-to-End Applications

DREW enables new applications that no prior work could support. Specifically, the bit rate of a real-time, stereo, Hi-Fi quality audio is 1.411Mbps ($= 44100 \cdot 16 \cdot 2$). DREW's unique support of QPSK (2Mbps) is crucial for streaming audio, since the throughputs of prior work (BPSK) are insufficient (FLEW: 857kbps, Unify: 655kbps). DREW (~1.6Mbps transport-layer throughput) can directly stream uncompressed, bit-accurate audio, which guarantees lossless, wired-equivalent audio quality and completely eliminates algorithmic latencies (>100ms for SBC) of audio compressions. In fact, even existing Bluetooth headphones ("Bluetooth Classic") are not capable of streaming such an audio stream, since the Bluetooth standard **only** supports lossy audio formats (e.g., SBC, AAC and aptX) with heavy (~4x) compressions. Furthermore, BLE is not compatible with Bluetooth Classic. Streaming audio over BLE has even higher compression (4~8x) due to BLE's lower throughput and is not supported on most (Bluetooth 5.1 or older) devices.

The direct WiFi connectivity enables DREW to leverage **audio over IP** solutions, thus circumventing the limitations of the Bluetooth stack. Of the many audio over IP solutions available, we choose Scream [31] to demonstrate audio streaming. Scream is a simple Windows driver that creates a virtual sound card and streams uncompressed sound samples over UDP. We use the unicast mode and the UDP packets are sent over WiFi. On the BLE side, bit-accurate audio samples are outputted to the speakers using Scream's Linux receiver. The bit rate is measured at 1.44Mbps. Interestingly, the BLE chip we used (KW41Z) only supports Bluetooth 4.2[1] and does not support Bluetooth Classic or BLE audio. With DREW, however, we can directly stream uncompressed audio via WiFi and offer higher audio quality than Bluetooth.

This throughput advantage also enables fetching high-quality multimedia contents directly from the Internet. We have used DREW to watch 720p (and some 1080p) Youtube videos without buffering. We also used DREW to stream Netflix at 720p, the maximum resolution under DRM restrictions.

With WiFi's global routability, DREW is also well-suited for IoT applications. With DREW, IoT devices can directly retrieve information from various websites or upload sensor readings to the cloud. Furthermore, DREW's direct interoperability with WiFi allows ULP BLE devices to use normal web services without requiring custom packet translation or gateways. For example, we have used DREW to send prompts and receive responses to/from the ChatGPT server. We also use DREW to perform Google Voice Search. These use-cases are particularly useful for adding valuable features to ULP wearable devices, such as smartwatches.

3.7 Power Consumption

Table 8: Power-consumption measurements

	Tx (A)	BPSK Rx (A)	QPSK Rx (A)
Atheros AR9271	0.49	0.07	0.07
Ralink RT3572	0.42	0.15	0.15
Ralink MT7612U	0.42	0.15	0.15
Realtek RTL8811AU	0.28	0.08	0.08
FLEW [8]	0.16	0.11	Not supported
Unify [9]	0.04	0.04	Not supported
DREW	0.07	0.07	0.07
DREW w/ K22F sleep	0.05	0.05	0.05

The KW41Z chip used in our implementation has ultra-low power consumption. According to its datasheet, the active currents (at 3.6V) are as low as 6.1mA (Tx) and 6.8mA (Rx), which are considerably lower than those of a typical WiFi chip. For example, a QCA9377 module consumes 538mA (Tx) and 155mA (Rx) at 3.3V [32].

Table 8 shows the power consumption of DREW, various common WiFi dongles, and prior work. We measure the USB (5V) current during active transmission or reception (and every device uses the same modulation format). DREW has the lowest power consumption among chips that support QPSK. Also, the USB power on the USB-KW41Z board is shared by the BLE chip and the debugger. The debugger itself is a powerful Cortex-M4 K22F microcontroller running at 120MHz, which contributes to the overall power consumption. We further tried downclocking the K22F and using sleep modes (with the `WFI` instruction) whenever possible, and the power consumption is reduced to 0.05A for all cases. Further power saving may be possible by enabling KW41Z's buck converter

[1]KW41Z's stack was not certified for all Bluetooth 5 features but the hardware supports IQ sampling. The IQ sampling was incorporated in Bluetooth 5.1, and thus newer chips' hardware support it. The standard does not specify circuit implementation and ULP designs commonly eliminate mixers.

or by removing the debugger completely. The KW41Z is operated at 3.5dBm. Since the PA is switched on 50% of the time, the transmit power is approximately $3.5 - 3 = 0.5$dBm, which is within the power range specified in the WiFi standard. If a higher power is needed, an external PA can be used. Assuming a nominal efficiency of 33%, the PA will add $(0.1\text{W} - 0.001\text{W})/0.33/5\text{V} = 0.06\text{A}$ to the transmit current.

4 DISCUSSION

4.1 Advantage over using WiFi 64-QAM

Although a prior work [33] indicates that utilizing WiFi's 64-QAM modulation ***at the same transmit power*** may consume less energy than BLE during packet transmission, the goal of DREW is to enable WiFi connectivity on **existing** hardware that comes with simple, low-cost hardware and without high-speed DSP or wideband RF ADCs/DACs. Furthermore, a typical BLE chip has a sensitivity of -97dBm [4], which is 19dB better than -78dBm [34] of MCS7 (64-QAM). Therefore, without affecting the range, the transmit power of BLE should be 19dB lower than MCS7, which would have resulted in much lower power consumption for BLE. (Also, DSSS waveforms can achieve an even better sensitivity at -102dBm [34] thanks to the advantage of PSK over FSK.) DREW is significant in that a ultra-low-power WiFi connectivity is possible by reusing the BLE's hardware, since BLE chips have ultra-low continuous currents (Tx: 6.1mA, Rx: 6.8mA) with low-power ARM processors. (In contrast, the WiFi chip used in [33] is known for its high current draw in the industry (DSSS Tx: 240mA, DSSS Rx: 95mA) and the chip consumes more than 20mA even with WiFi disabled [35].)

4.2 IQ Sampling

IQ sampling is available on a wide range of new BLE chips to support localization and AoA features defined in the Bluetooth standard. Different BLE chips have different procedures for enabling the IQ sampling. We set up IQ sampling by following the *transceiver DMA* procedure described in NXP's manual [21]. Other parts of the Bluetooth firmware and software stack are not utilized in DREW. By directly fetching IQ samples with the ARM core (Sec. 2.4.1), the length of IQ capture need not be known in advance and is not limited by the buffer size or by the DMA configuration. However, this design requires real-time IQ processing, thus making SIMD a critical component. Finally, we have tested using IQ sampling with OFDM waveforms, but the challenge is that the IQ sampling is relatively narrow-band and is insufficient for decoding a packet without errors.

4.3 Spectrum

In terms of spectrum, DREW is at least as efficient as prior CTC designs, and is twice as efficient when QPSK is used. Furthermore, typical WiFi systems still commonly use DSSS for critical traffic, such as management packets, beacons, RTS/CTS, and ACKs. Spectrum efficiency can also be improved by spatial diversity and power control between devices.

4.4 Packet Detection

The packet detection of DREW is optimized for both speed and memory. We directly use ARM registers as 32-bit FIFOs for the fastest access. Updating 4 FIFOs (for 4 IQ samples) takes 12 cycles and comparing these 4 FIFOs with both long and short preambles takes 16 cycles. In contrast, the correlation-based method requires 128 ($= 32\,\mu\text{s} \cdot 4$ samples$/\mu$s) complex multiplications and additions for every IQ sample received. Additional loads and stores for memory access are also needed. Such a method exceeds the capability of typical ARM microcontrollers.

5 RELATED WORK

Earlier CTC works [36–39] use one packet to represent one or a few modulation symbols for a heterogeneous device to receive. These designs have lower throughputs and they need to modify both Tx and Rx. Recent CTC works directly transmit compatible waveforms at the PHY layer and achieve higher throughputs. These works include WiFi-to-Bluetooth [12–16], WiFi-to-Zigbee [40–45], and WiFi-to-LoRa [46, 47]. However, they only enable one-way communication and require modifying the transmitter. Furthermore, they focus on generating 1Mbps FSK [12–16], 250kbps Zigbee [40–45] or slower waveforms. Some other CTCs [48, 49] modify WiFi devices to receive non-WiFi waveforms. Other CTCs focus on Bluetooth–Zigbee [50, 51] or LTE [52–55]. CTC is also related to concurrent-communication systems [56–58].

FLEW [8] and Unify [9] achieve bidirectional WiFi–FSK communication without any modification on the WiFi side. However, they require older FSK chips that have mixers, and the PHY throughput is capped at 1Mbps. Other software-defined-radio-based [59] and simulation-only [60] systems have also been proposed but are not directly applicable to COTS WiFi systems.

Many prior work [61–69] use BLE's AoA feature and propose various post-processing algorithms for localization. To the best of our knowledge, DREW is the first that uses the AoA feature for data communication (specifically WiFi communication) instead of localization.

6 CONCLUSION

DREW enables direct communication between ULP BLE chips and WiFi APs. We proposed innovative use of the PA and IQ sampling, and devised various algorithms with SIMD acceleration. DREW is applicable to mixer-less BLE chips and uniquely supports QPSK. It is shown to have good coexistence, low power consumption and much higher throughputs.

REFERENCES

[1] Silicon Labs. Silicon Labs Launches Blue Gecko Bluetooth Smart Solutions. https://news.silabs.com/2015-02-23-Silicon-Labs-Launches-Blue-Gecko-Bluetooth-Smart-Solutions, Feb 2015.

[2] Silicon Labs. Six Hidden Costs in a 99 Cent Wireless SoC. https://www.rs-online.com/designspark/rel-assets/ds-assets/uploads/knowledge-items/application-notes-for-the-internet-of-things/Six-hidden-costs-of-a-99-cent-soc.pdf, 2015.

[3] NXP. MKW41Z/31Z/21Z Data Sheet. https://www.nxp.com/docs/en/data-sheet/MKW41Z512.pdf, Mar 2018.

[4] Texas Instruments. CC2650 SimpleLink™ Multistandard Wireless MCU. https://www.ti.com/lit/ds/symlink/cc2650.pdf, Feb 2015.

[5] Texas Instruments. Low-Power Carbon Monoxide Detector With BLE and 10-Year Coin Cell Battery Life Reference Design. https://www.tij.co.jp/lit/ug/tidubx8c/tidubx8c.pdf, Nov 2017.

[6] Thomas Zachariah, Noah Klugman, Bradford Campbell, Joshua Adkins, Neal Jackson, and Prabal Dutta. The Internet of Things Has a Gateway Problem. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile '15, page 27–32, New York, NY, USA, 2015. Association for Computing Machinery.

[7] Thomas Zachariah, Neal Jackson, and Prabal Dutta. The Internet of Things Still Has a Gateway Problem. In *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications*, HotMobile '22, page 109–115, New York, NY, USA, 2022. Association for Computing Machinery.

[8] Hsun-Wei Cho and Kang G. Shin. FLEW: Fully Emulated Wifi. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, MobiCom '22, page 29–41, New York, NY, USA, 2022. Association for Computing Machinery.

[9] Hsun-Wei Cho and Kang G. Shin. Unify: Turning BLE/FSK SoC into WiFi SoC. In *Proceedings of the 29th Annual International Conference on Mobile Computing And Networking*, MobiCom '23, New York, NY, USA, 2023. Association for Computing Machinery.

[10] Hadi Givehchian, Nishant Bhaskar, Eliana Rodriguez Herrera, Héctor Rodrigo López Soto, Christian Dameff, Dinesh Bharadia, and Aaron Schulman. Evaluating Physical-Layer BLE Location Tracking Attacks on Mobile Devices. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1690–1704, 2022.

[11] Texas Instruments. 2.4-GHz Bluetooth low energy and Proprietary System-on-Chip. https://www.ti.com/lit/ds/symlink/cc2541.pdf, Jan 2012.

[12] Lingang Li, Yongrui Chen, and Zhijun Li. Poster Abstract: Physical-layer Cross-Technology Communication with Narrow-Band Decoding. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–2, 2019.

[13] Lingang Li, Yongrui Chen, and Zhijun Li. WiBle: Physical-Layer Cross-Technology Communication with Symbol Transition Mapping. In *2021 18th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9, 2021.

[14] Hsun-Wei Cho and Kang G. Shin. BlueFi: Bluetooth over WiFi. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 475–487, New York, NY, USA, 2021. Association for Computing Machinery.

[15] Ruofeng Liu, Zhimeng Yin, Wenchao Jiang, and Tian He. WiBeacon: Expanding BLE Location-Based Services via Wifi. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, MobiCom '21, page 83–96, New York, NY, USA, 2021. Association for Computing Machinery.

[16] Ruirong Chen and Wei Gao. TransFi: Emulating Custom Wireless Physical Layer from Commodity Wifi. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, MobiSys '22, pages 357–370, New York, NY, USA, 2022. Association for Computing Machinery.

[17] Bluetooth SIG. Core specification v5.1. , Jan 2019.

[18] STMicroelectronics. Arm Cortex-M0 in a nutshell. https://www.st.com/content/st_com/en/arm-32-bit-microcontrollers/arm-cortex-m0.html.

[19] ARM. Neon. https://developer.arm.com/Architectures/Neon, 2022.

[20] IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pages 1–3534, 2016.

[21] NXP. MKW41Z/31Z/21Z Reference Manual. https://www.nxp.com/files-static/32bit/doc/ref_manual/MKW41Z512RM.pdf, Oct 2016.

[22] ARM. Arm Cortex-M0+ Processor Datasheet. https://documentation-service.arm.com/static/620545c494e7af28dd7c9cbc, 2020.

[23] NXP. Bluetooth Low Energy/IEEE 802.15.4 Packet Sniffer/USB Dongle. https://www.nxp.com/design/development-boards/freedom-development-boards/wireless-connectivy/bluetooth-low-energy-ieee-802-15-4-packet-sniffer-usb-dongle:USB-KW41Z, Mar 2018.

[24] Panasonic Industry Europe GmbH. PAN4620 (KW41Z). https://industry.panasonic.eu/products/devices/wireless-connectivity/ieee-802154-modules/pan4620-kw41z, Feb 2020.

[25] ublox. R41Z Stand-alone Bluetooth 4.2 low energy and IEEE 802.15.4 module. https://content.u-blox.com/sites/default/files/R41Z_DataSheet_UBX-19033355.pdf, Jun 2022.

[26] NXP. MCUXpresso Integrated Development Environment (IDE). https://www.nxp.com/design/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE, Jan 2023.

[27] ARM. GNU Arm Embedded Toolchain Downloads. https://developer.arm.com/downloads/-/gnu-rm, Oct 2021.

[28] NXP. OpenSDA Serial and Debug Adapter. https://www.nxp.com/design/software/development-software/sensor-toolbox-sensor-development-ecosystem/opensda-serial-and-debug-adapter:OPENSDA, 2023.

[29] Freescale Semiconductor. M68HC08 to HCS08 Transition. https://www.nxp.com/docs/en/application-note/AN2717.pdf, Aug 2006.

[30] iPerf. iPerf - The ultimate speed test tool for TCP, UDP and SCTP. https://iperf.fr/, 2020.

[31] Tom Kistner. Scream - Virtual network sound card for Microsoft Windows. https://github.com/duncanthrax/scream, Sep 2022.

[32] SparkLAN Communications. WNFQ-158ACN. https://www.sparklan.com/product/wnfq-158acnbt-qca9377-5-m-2-industrial-module/, 2023.

[33] Ali Abedi, Omid Abari, and Tim Brecht. Wi-LE: Can WiFi Replace Bluetooth? HotNets '19, page 117–124, New York, NY, USA, 2019. Association for Computing Machinery.

[34] Cisco. Cisco Aironet 2700 Series Access Points Data Sheet. https://www.cisco.com/c/en/us/products/collateral/wireless/aironet-2700-series-access-point/datasheet-c78-730593.html, Jul 2020.

[35] Espressif Systems. ESP32 Series Datasheet. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf, Jul 2023.

[36] Kameswari Chebrolu and Ashutosh Dhekne. Esense: Communication through Energy Sensing. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*, MobiCom '09, pages 85–96, New York, NY, USA, 2009. Association for Computing Machinery.

[37] Song Min Kim and Tian He. FreeBee: Cross-Technology Communication via Free Side-Channel. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, page 317–330, New York, NY, USA, 2015. Association for

Computing Machinery.

[38] Zicheng Chi, Yan Li, Hongyu Sun, Yao Yao, Zheng Lu, and Ting Zhu. B2W2: N-Way Concurrent Communication for IoT Devices. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, SenSys '16, pages 245–258, New York, NY, USA, 2016. Association for Computing Machinery.

[39] Zhimeng Yin, Wenchao Jiang, Song Min Kim, and Tian He. C-Morse: Cross-technology communication with transparent Morse coding. pages 1–9, 2017.

[40] Zhijun Li and Tian He. WEBee: Physical-Layer Cross-Technology Communication via Emulation. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, page 2–14, New York, NY, USA, 2017. Association for Computing Machinery.

[41] Zhijun Li and Tian He. LongBee: Enabling Long-Range Cross-Technology Communication. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 162–170, 2018.

[42] Yongrui Chen, Shuai Wang, Zhijun Li, and Tian He. Reliable Physical-Layer Cross-Technology Communication With Emulation Error Correction. *IEEE/ACM Transactions on Networking*, 28(2):612–624, 2020.

[43] Xiuzhen Guo, Yuan He, Jia Zhang, and Haotian Jiang. WIDE: Physical-level CTC via Digital Emulation. In *2019 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 49–60, 2019.

[44] Jialiang Yan, Siyao Cheng, Zhijun Li, and Jie Liu. PCTC: Parallel Cross Technology Communication in Heterogeneous wireless systems. In *2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 67–78, 2022.

[45] Shuai Wang, Woojae Jeong, Jinhwan Jung, and Song Min Kim. X-MIMO: Cross-Technology Multi-User MIMO. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, SenSys '20, pages 218–231, New York, NY, USA, 2020. Association for Computing Machinery.

[46] Dan Xia, Xiaolong Zheng, Fu Yu, Liang Liu, and Huadong Ma. WiRa: Enabling Cross-Technology Communication from WiFi to LoRa with IEEE 802.11ax. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pages 430–439, 2022.

[47] Piotr Gawłowicz, Anatolij Zubow, and Falko Dressler. Wi-Lo: Emulation of LoRa using Commodity 802.11b WiFi Devices. In *IEEE International Conference on Communications (ICC 2022)*, Seoul, South Korea, 5 2022. IEEE.

[48] Xiuzhen Guo, Yuan He, Xiaolong Zheng, Zihao Yu, and Yunhao Liu. LEGO-Fi: Transmitter-Transparent CTC with Cross-Demapping. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 2125–2133, 2019.

[49] Ruofeng Liu, Zhimeng Yin, Wenchao Jiang, and Tian He. XFi: Cross-technology IoT Data Collection via Commodity WiFi. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–11, 2020.

[50] Wenchao Jiang, Song Min Kim, Zhijun Li, and Tian He. Achieving Receiver-Side Cross-Technology Communication with Cross-Decoding. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, page 639–652, New York, NY, USA, 2018. Association for Computing Machinery.

[51] Wenchao Jiang, Zhimeng Yin, Ruofeng Liu, Zhijun Li, Song Min Kim, and Tian He. BlueBee: A 10,000x Faster Cross-Technology Communication via PHY Emulation. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, New York, NY, USA, 2017. Association for Computing Machinery.

[52] Piotr Gawłowicz, Anatolij Zubow, Suzan Bayhan, and Adam Wolisz. OfdmFi: Enabling Cross-Technology Communication Between LTE-U/LAA and WiFi, 2019.

[53] Piotr Gawłowicz, Anatolij Zubow, and Suzan Bayhan. Demo Abstract: Cross-Technology Communication between LTE-U/LAA and WiFi. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1272–1273, 2020.

[54] Piotr Gawlowicz, Anatolij Zubow, Suzan Bayhan, and Adam Wolisz. Punched Cards over the Air: Cross-Technology Communication Between LTE-U/LAA and WiFi. In *2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 297–306, 2020.

[55] Ruofeng Liu, Zhimeng Yin, Wenchao Jiang, and Tian He. LTE2B: Time-Domain Cross-Technology Emulation under LTE Constraints. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, SenSys '19, page 179–191, New York, NY, USA, 2019. Association for Computing Machinery.

[56] Zicheng Chi, Yan Li, Xin Liu, Yao Yao, Yanchao Zhang, and Ting Zhu. Parallel Inclusive Communication for Connecting Heterogeneous IoT Devices at the Edge. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, SenSys '19, page 205–218, New York, NY, USA, 2019. Association for Computing Machinery.

[57] Zicheng Chi, Yan Li, Yao Yao, and Ting Zhu. PMC: Parallel multi-protocol communication to heterogeneous IoT radios within a single WiFi channel. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pages 1–10, 2017.

[58] Sihan Yu, Xiaonan Zhang, Pei Huang, and Linke Guo. Physical-Level Parallel Inclusive Communication for Heterogeneous IoT Devices. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pages 380–389, 2022.

[59] Zhijun Li and Yongrui Chen. BlueFi: Physical-layer Cross-Technology Communication from Bluetooth to WiFi. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 399–409, 2020.

[60] Yuanhe Shu, Jingwei Wang, Linghe Kong, Jiadi Yu, Guisong Yang, Yueping Cai, Zhen Wang, and Muhammad Khurram Khan. WiBWi: Encoding-based Bidirectional Physical-Layer Cross-Technology Communication between BLE and WiFi. In *2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 356–363, 2021.

[61] Xinyou Qiu, Bowen Wang, Jian Wang, and Yuan Shen. AOA-Based BLE Localization with Carrier Frequency Offset Mitigation. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–5, 2020.

[62] Cheng Huang, Yuan Zhuang, Hao Liu, Jianyu Li, and Wei Wang. A Performance Evaluation Framework for Direction Finding Using BLE AoA/AoD Receivers. *IEEE Internet of Things Journal*, 8(5):3331–3345, 2021.

[63] Marco Cominelli, Paul Patras, and Francesco Gringoli. Dead on Arrival: An Empirical Study of The Bluetooth 5.1 Positioning System. In *Proceedings of the 13th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, WiNTECH '19, page 13–20, New York, NY, USA, 2019. Association for Computing Machinery.

[64] Pooneh Mohaghegh, Alexis Boegli, and Yves Perriard. Bluetooth Low Energy Direction Finding Principle. In *2021 24th International Conference on Electrical Machines and Systems (ICEMS)*, pages 830–834, 2021.

[65] Da Sun, Yinong Zhang, Weiwei Xia, Zhiyuan Geng, Feng Yan, Lianfeng Shen, and Yingbin Gao. A BLE Indoor Positioning Algorithm based on Weighted Fingerprint Feature Matching Using AOA and RSSI. In *2021 13th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6, 2021.

[66] Wei He, Chenglin Huang, Zengshan Tian, Kaikai Liu, and Ze Li. Design and Implementation of Bluetooth Low Energy AoA Estimation System.

In *2022 IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting (AP-S/URSI)*, pages 311–312, 2022.

[67] Shuai He, Hang Long, and Wei Zhang. Multi-antenna Array-based AoA Estimation Using Bluetooth Low Energy for Indoor Positioning. In *2021 7th International Conference on Computer and Communications (ICCC)*, pages 2160–2164, 2021.

[68] Gaurav Kumar, Vrinda Gupta, and Rahul Tank. Phase-based Angle estimation approach in Indoor Localization system using Bluetooth Low Energy. In *2020 International Conference on Smart Electronics and Communication (ICOSEC)*, pages 904–912, 2020.

[69] Usman Raza, Aftab Khan, Roget Kou, Tim Farnham, Thajanee Premalal, Aleksandar Stanoev, and William Thompson. Dataset: Indoor Localization with Narrow-Band, Ultra-Wideband, and Motion Capture Systems. In *Proceedings of the 2nd Workshop on Data Acquisition To Analysis*, DATA'19, page 34–36, New York, NY, USA, 2019. Association for Computing Machinery.