FLEW: Fully Emulated WiFi

Hsun-Wei Cho and Kang G. Shin
University of Michigan

ABSTRACT

WiFi is the *de facto* standard for providing wireless access to the Internet using the 2.4GHz ISM band. Tens of billions of WiFi devices were shipped worldwide and WiFi Access Points (APs) are ubiquitous in public, enterprise and personal environments. We have also witnessed the fast growth of IoT (Internet of Things) devices. With more stringent board-space and power requirements, many IoT devices use more power-efficient, lower-cost and smaller wireless chips, such as Bluetooth or proprietary wireless chips. Due to the mismatch of different wireless technologies, these devices access the Internet *indirectly* via far less ubiquitous IoT gateways. Bluetooth and most proprietary wireless chips are based on FSK (Frequency-Shift Keying) modulation since FSK can be implemented with extremely simple and low-power FM (Frequency Modulation) circuits.

In this paper, we present FLEW (Fully Emulated WiFi), which uses a single FSK chip to fully emulate **both transmission and reception of WiFi signals**. Using FLEW, FSK-equipped IoT or mobile/wearable devices can directly communicate with **unmodified** WiFi APs, just like any WiFi device. FLEW combines the best of both technologies: extremely simple and low-power hardware and ubiquitous Internet access.

We evaluate FLEW extensively with chips from all major chip makers. At 20 meters, FLEW can sustain 708kbps uplink and 857kbps downlink at the transport layer.

CCS CONCEPTS

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

KEYWORDS

Cross-Technology Communication, WiFi, FSK, Bluetooth

ACM Reference Format:
Hsun-Wei Cho and Kang G. Shin. 2022. FLEW: Fully Emulated WiFi. In *The 28th Annual International Conference On Mobile Computing And Networking (ACM MobiCom '22), October 24–28, 2022, Sydney, NSW, Australia.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3495243.3517030

1 INTRODUCTION

WiFi is the *de facto* standard for tens of billions of devices [1] to access the Internet. Designed to support all types of Internet applications, WiFi chips require a considerable amount of DSP circuitry for processing various WiFi waveforms, including high-throughput waveforms, leading to physically larger chips, higher energy consumption and higher chip costs.

Typical IoT applications only require a low data rate but energy consumption and chip size/cost are of paramount importance. Owing to these requirements, many IoT devices are built with FSK chips, also commonly known as *2.4 GHz proprietary wireless chips*. FSK chips are also the hardware of Bluetooth/BLE. In fact, many Bluetooth/BLE chips are essentially FSK chips with a Bluetooth software stack. FSK is arguably the simplest modulation scheme that offers decent enough throughput and noise immunity. Using FM circuits to transmit/receive digital waveforms, FSK chips are extremely energy-efficient and low-cost as well as occupy less board-space than WiFi chips. For example, we find the smallest WiFi chip available is Silabs' WF200 [2], whereas the smallest FSK chip available is TI's CC2500 [3]. Table 1 shows a comparison between these two chips. Although they differ in many ways [1], FSK chips are, in general, much smaller and cheaper, and consume less power than WiFi chips.

Table 1: Comparison of WiFi and FSK chips.

	Tech.	Package	Tx current (mA)	Rx current (mA)	Price (USD)
WF200	WiFi	QFN32	108 (PA) + 44.6 (BB)	41.6	3.28
CC2500	FSK	QFN20	100 (PA) + 21.5 (BB)	19.6	1.18

However, using protocols other than the ubiquitous WiFi means that IoT devices cannot use the WiFi infrastructure for Internet connectivity. Instead, they must rely on additional IoT gateways to relay the data to/from, and access the Internet indirectly. This gateway reliance implies that, to use any IoT device (with FSK or protocols other than WiFi), their corresponding IoT gateways must be installed in the environment. This hampers the adoption of IoT since using these IoT devices requires not only buying the devices but also investing/deploying/managing the IoT gateways. This "gateway problem" [5] is worsened by the non-existence of a universal protocol that dominates the market, and hence different IoT devices may require separate gateways.

This poses an important question: *What if we can still use the small, simple, low-power FSK chips on the IoT devices, but we somehow allow FSK chips to directly communicate with WiFi infrastructures?* If such communication is possible/enabled, these FSK IoT devices can leverage existing WiFi infrastructures and eliminate the need

ACM MobiCom '22, October 24–28, 2022, Sydney, NSW, Australia
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9181-8/22/10…$15.00
https://doi.org/10.1145/3495243.3517030

[1]WF200 is much newer than CC2500 and therefore allows lower supply voltage (1.8V) for baseband and Rx circuits compared to CC2500 (3.0V). CC2500 still consumes less power after factoring in this difference. Newer FSK chips have even lower power consumption and can use lower supply voltage. For example, CC2650 [4] allows 1.8V operation and consumes only 6.1mA (Tx, BB) and 5.9mA (Rx) at 3.0V. An end-to-end power measurement with the same power condition is present in Sec. 4.9.

for IoT gateways. They can stay connected wherever there is WiFi coverage!

To answer this question, we propose FLEW, which turns FSK chips into WiFi chips and enables them to directly communicate with unmodified WiFi APs.

Although several CTC (cross technology communication) efforts have explored communications *from* WiFi devices, FLEW represents a very different design philosophy and targets different use-cases. In particular, prior WiFi CTC work is *WiFi-centric* whereas FLEW is *FSK-centric*. Specifically, prior work enables one-way communication from modified WiFi APs/devices to unmodified FSK devices (e.g., Bluetooth), whereas FLEW focuses on enabling **bi-directional** communication between unmodified WiFi APs/devices and modified FSK devices. FLEW complements existing CTC work well by covering scenarios where users are not permitted to modify the firmware of WiFi APs/devices (e.g., WiFi APs in public or enterprises), or where one FSK device may connect to many APs (e.g., roaming) arbitrarily without needing to modify the firmware of every single AP. Furthermore, many IoT applications have uplink (i.e., FSK to WiFi) traffic, which is not possible with the prior WiFi-to-FSK work.

Even though several CTC studies have demonstrated the communication from WiFi devices to Bluetooth devices, their WiFi-to-FSK solutions are still not directly applicable because of more stringent system requirements. In particular, prior work requires modification of WiFi transmitters to generate FSK waveforms. However, since the goal of FLEW is for FSK chips to directly communicate with unmodified WiFi devices, it must be able to decode *any* standard WiFi packets at one data rate at least; not just WiFi packets with a magic payload (a payload that results in FSK-look-alike waveforms). To this end, we show how FSK hardware can be used to receive *any* WiFi DSSS waveforms. The difference can also be explained with the waveforms transmitted by the WiFi devices. Prior work modifies WiFi devices to transmit FSK-look-alike waveforms. In contrast, under FLEW, WiFi devices transmit standard 802.11b DSSS waveforms, like the conventional 802.11b WiFi operations, thus allowing the use of *unmodified* WiFi devices.

Furthermore, we show how FSK hardware can be effectively used to transmit 802.11b waveforms, thus enabling bi-directional communication between FSK and unmodified WiFi devices. FSK-to-WiFi communication is of great significance because any useful WiFi operation requires bi-directional communication at the physical layer. Even with a unilateral transport layer traffic, the physical layer requires the transmission of ACK packets in the opposite direction. Bi-directional communication is also needed for a client to join a WiFi network.

The different philosophy of FLEW also leads to different trade-offs. FLEW tries to modify FSK devices so that FSK devices work like WiFi devices as much as possible, whereas prior CTC work tries to modify WiFi devices so that they work like FSK/Bluetooth devices as much as possible. For this purpose, we utilize the underlying FSK hardware, instead of the full BLE/Bluetooth stack, to ensure a FLEW device behaves as close to a WiFi device as possible, since a full BLE/Bluetooth operation imposes unnecessary software limits. To ensure maximum compatibility with different unmodified WiFi devices, a FLEW terminal is designed to directly appear like a WiFi device, not a WiFi and Bluetooth device, during FLEW operation. This does not imply that new hardware is needed, however, as we have shown that FLEW can be implemented on existing FSK devices and chips. While prior CTC works are compliant with both Bluetooth and WiFi waveforms, they cannot support full operations of either standards. For example, the (COTS) WiFi chips in prior CTC work cannot receive Bluetooth packets and the Bluetooth chips cannot receive arbitrary WiFi packets. Although FLEW directly uses conventional WiFi waveforms and therefore does not have the "dual-compliance" during FLEW operations, we feel that this is a worthy trade-off because using conventional WiFi waveforms achieves the goal of enabling full WiFi operations and is the only way to ensure maximum compatibility with unmodified WiFi devices.

On the technical side, FLEW leverages the insights that a) at its core, WiFi (802.11b DSSS) encodes the information in the form of PSK (Phase Shift Keying), b) PSK modulation is similar to, and therefore can be demodulated by FSK receivers with a frequency shift, and c) PSK with DSSS signals can be transmitted by directly connecting digital waveforms to the mixer. These high-level principles are relatively simple, but are extremely powerful, and can bridge the gap between DSSS and FSK modulations.

In contrast to the conventional IoT topology where gateways and devices employ similar radio circuitry and chips, the hardware of WiFi APs and FSK devices in FLEW is highly asymmetrical. This asymmetry provides an opportunity to use simple and energy-efficient FSK chips while still providing good performance by leveraging powerful PAs and LNAs in WiFi APs. In addition, WiFi's DSSS modulation at 1Mbps has a higher coding gain than Zigbee or Bluetooth, and is intrinsically robust. Finally, to support higher data rates in newer 802.11 standards, many APs come with multiple antennas and advanced MIMO signal processing can further enhance the performance in both directions. Specifically, WiFi APs are allowed to transmit at high power and some APs support transmit beamforming for 802.11b [6], which enhances signal strength and overall mixed-client throughput. Also, many APs use diversity or MIMO processing (e.g., RAKE or MRC for 802.11b) to boost reception performance. For example, for 1Mbps, modern WiFi APs has a sensitivity as low as -102dBm [7], which outperforms the latest Zigbee offerings from TI (-100dBm [8]) and Microchip/Atmel (-101dBm [9]) even though WiFi is 4x faster than Zigbee (250kbps). Compared to Bluetooth/BLE chips, the difference is even greater (TI: -97dBm [10], Qualcomm/CSR: -95dBm [11], Broadcom/Cypress: -96dBm [12]).

Without WiFi encryption, FLEW is as secure as existing FSK protocols. If an existing protocol encrypts the payload, FLEW can simply transmit the encrypted payload over open WiFi networks. On the other hand, since it allows devices to directly communicate via WiFi, FLEW can provide stronger, enterprise-grade security protection by directly using the tried-and-true WiFi security framework on which billions of devices currently rely.

We implement FLEW with COTS FSK chips. With FLEW, these FSK chips are emulated as WiFi chips and can communicate with conventional, unmodified WiFi devices/APs. We extensively evaluate the performance of FLEW with multiple WiFi devices equipped with many different widely-adopted chips from all major WiFi chipmakers. We note that FSK hardware is the foundation of Classic Bluetooth and BLE (Bluetooth Low Energy). Therefore, with FLEW, it is possible to simultaneously support Bluetooth, BLE and WiFi using a single FSK chip!

FLEW enables connectivity and use-cases that were previously deemed impossible. FLEW allows IoT devices to directly connect to already-deployed WiFi APs and eliminates IoT gateways altogether. Alternatively, since FSK chips are cheaper, smaller and more energy-efficient than WiFi chips, FLEW can provide general Internet access for mobile devices where cost, area and/or power are of great importance. As an example, we showcase using FLEW for steaming high-quality music or streaming 720p YouTube videos in real time.

In FLEW, we focus on transmitting/receiving data at 1Mbps, which is on par with BLE 4 and 4x faster than Zigbee, and is sufficient for IoT operations. For WiFi, the 1Mbps data rate has a special significance. 1Mbps has the most robust performance among all possible WiFi modulations and many APs use 1Mbps for management (beacon, association, authentication, etc.) frames regardless of the data rates of data frames. In a multi-rate environment (which is almost always the case for typical WiFi networks), the transmit data rate is controlled by the rate adaptation algorithm (RAA), which will reduce the transmit data rates (i.e., use more robust modulation) if transmitted packets are not acknowledged. APs will try 1Mbps modulation if transmitting with higher data rates is unsuccessful. Therefore, implementing 1Mbps ensures that the WiFi-FSK connection will converge to a steady state using 1Mbps. If only a higher data rate is supported instead, then a connection may not be able to be established because of the packet loss of management frames. In addition, even if the higher data rate is negotiated, any transient behavior in the network may cause two devices to diverge from the agreed-on data rate and thus cause disconnection.

For the WiFi AP, a FLEW terminal will appear as a device that needs the most robust modulation and only 1Mbps modulation can get through. Such a scenario can legitimately happen with a conventional WiFi terminal (e.g., with a weak signal or with strong interference). Therefore, rate adaptation algorithms should always support FLEW operations regardless of their actual implementation.

The techniques used in FLEW may also help develop future low-power WiFi transceivers. For example, we show that instead of using a full-blown multi-rate PSK receiver with complicated phase synchronization, the relatively simple FM/FSK demodulator can be used to demodulate WiFi waveforms at 1Mbps very effectively. Since 1Mbps is frequently used to transmit management frames, the receiver can be completely turned off and only use low-power FM circuits to monitor the management traffic. The FM circuit can be used to wake up the main WiFi receiver after certain management frames (e.g., those containing traffic indication map (TIM)) are received. We note that Bluetooth is also using FM circuits, so it is even possible to use a Bluetooth receiver to wake up WiFi.

The insights gained from designing FLEW are also useful for understanding and mitigating the interference between FSK (such as Bluetooth) and WiFi. For example, Bluetooth devices should avoid using certain bit sequences (e.g., 0x05AE4701) as their access codes, since legitimate WiFi waveforms can cause accidental packet detection on Bluetooth receivers with such access codes.

2 SYSTEM DESIGN

2.1 Primer

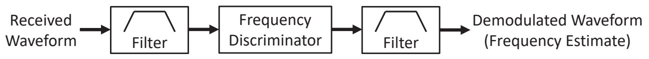


Figure 1: General model of FM/FSK receivers.

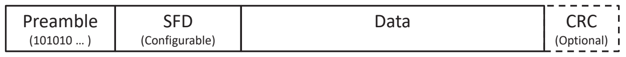


Figure 2: FSK packet format.

2.1.1 FSK. At the bit level, FSK is simply FM with digital data. In other words, FSK sends digital (high or low) data into an FM modulator. The instantaneous frequency of an FSK waveform depends on the input level. For bit '1', the frequency is higher than the center frequency by one frequency deviation. For bit '0', the frequency is lower than the center frequency by one frequency deviation. An FSK receiver uses an FM demodulator to recover the bits. The FM demodulator tries to estimate the received signal's instantaneous frequency, which corresponds to the original digital data.

Fig. 1 shows the general model of FM/FSK receivers. The first filter is used to extract signals near the receive frequency. The filter after the frequency discriminator is also essential since the frequency discriminator is nonlinear and the filtering on the signal itself does not guarantee that its instantaneous frequency is properly filtered. Fig. 1 is the general model, regardless of whether the filters or demodulator are implemented in analog or digital domain, or whether there is down-conversion between these components (e.g., zero-IF or low-IF receivers).

Fig. 2 shows the packet format of FSK/proprietary protocols. The preamble consists of alternating 1's and 0's so that the demodulator in the receiver can be stabilized. The preamble is followed by a configurable SFD,[2] which signifies the receiver that it should expect and start collecting actual data after it receives SFD. If CRC is enabled, the CRC sequence is appended to the data field for detecting bit errors.

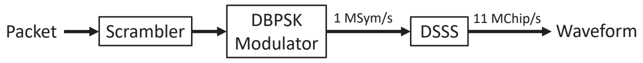


Figure 3: 802.11b modulation process.

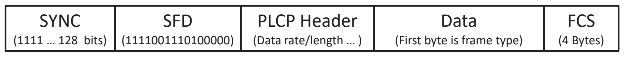

SYNC (1111 ... 128 bits)
SFD (1111001110100000)
PLCP Header (Data rate/length ...)
Data (First byte is frame type)
FCS (4 Bytes)

Figure 4: 802.11b packet format.

2.1.2 802.11b. Fig. 3 shows the modulation process of 802.11b. The incoming bitstream is first scrambled with a different scrambler from other 802.11 standards. The scrambled bits are modulated by differential binary PSK, which either rotates the phase of the carrier by π for bit '1' or keeps the phase unchanged for bit '0'. This waveform is then modulated using DSSS, which further toggles the phase within each bit duration using the 11-chip Barker sequence.

Fig. 4 shows the packet format of 802.11b. The SYNC field consists of 128 bits of '1', which are used to stabilize the receiver. A constant SFD follows the SYNC field and is used by the receiver to detect the start of a WiFi packet. The PLCP header contains vital information about the modulation and the total length of subsequent fields. The PLCP header also contains a 16-bit CRC for detecting errors in the header. Upper-layer packets are put into the data field. The FCS

[2]To avoid confusion, here we use the WiFi nomenclature. This field is commonly referred to as the sync word in FSK protocols.

(Frame Check Sequence) uses CRC32 and is calculated over the data field. A WiFi transmitter constructs a physical-layer packet this way before sending the entire packet into the scrambler.

2.2 Overview

For any useful WiFi operations, bi-directional communication is required. Therefore, Secs. 2.3 and 2.4, respectively, show how the FSK hardware can be used to receive and transmit WiFi signals with arbitrary payloads. In addition, since WiFi devices work in a half-duplex manner, the FSK chip must switch between transmission and reception appropriately and timely to avoid missed reception or transmission collisions. These MAC layer issues are addressed in Sec. 2.5.

2.3 WiFi to FSK

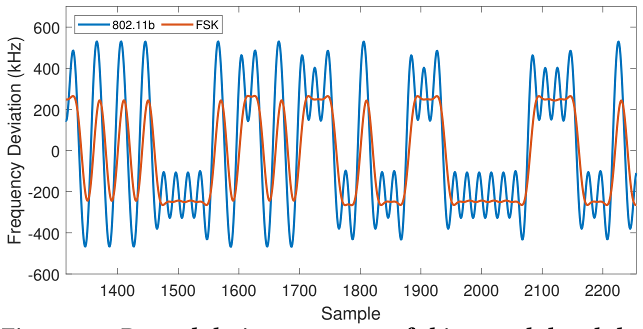


Figure 5: Demodulation outputs of bits modulated by 802.11b and FSK.

2.3.1 Bit Level. We devise a key method that enables FLEW to use FSK hardware to receive WiFi frames with arbitrary payloads. Its underlying insight is: **With an appropriate frequency shift, conventional FM/FSK receivers can work as a DSSS *plus* DBPSK demodulator.**

The implication of this insight is significant. With this method, instead of using a conventional PSK demodulator (which involves much more complicated phase synchronization), a simple, low-power FM demodulator can be used. Furthermore, the addition of DSSS requires conventional demodulators to run at a significantly higher speed (e.g., 11MHz). In FLEW, the FM demodulator can run at a much lower speed (1MHz). Therefore, this method allows FLEW to demodulate conventional WiFi frames in a much simpler and more power-efficient fashion.

The intuition behind this method is that the DSSS modulation process is essentially, in the frequency domain, convoluting the PSK spectrum with the spectrum of a repeated 11-length barker sequence. Note that an 11-length barker code has a white spectrum and the spectrum of a repeated 11-length barker sequence is only non-zero at ±1, ±2, ±3, ±4, ±5 MHz. The convolution process simply copies the PSK spectrum and places the replicas at these frequencies. Therefore, if we employ a relatively narrow filter to the DSSS waveform near one of the frequencies, the result is approximately the main lobe of the PSK spectrum.

Differential PSK is somewhat similar to FSK. Conceptually, the difference of phase is frequency. One major difference is that, in FSK, the phase is constantly increasing or decreasing for the duration of each bit, whereas in DPSK, the phase remains constant and only changes *between* bits. In FSK receivers, this difference can be mitigated by the filter after the frequency discriminator since the filter smoothens the step-like phase waveform of PSK modulation to become a constantly increasing or decreasing phase waveform.

Finally, we also need a small frequency shift in addition to, say 1 MHz, because an FSK receiver expects the frequency deviation of each bit to be either positive or negative. However, with DBPSK waveforms, the frequency deviation is either zero (no phase change) or non-zero (phase change). This can be corrected using a small amount of frequency shift, which equivalently adds a constant bias to the frequency deviation of each bit, and converts the frequency waveform to be non-return-to-zero.

Applying frequency shifts requires no additional hardware. In practice, we can simply change the center frequency of the receiver.

Let us illustrate this method with an example. In Fig. 5, we show the output waveforms of a FSK receiver demodulating an 802.11b waveform and a standard FSK waveform. Specifically, we generate an 802.11b waveform and use the general model of FSK receivers to demodulate the waveform. The receiver has a frequency offset of 1.22MHz and two low-pass filters are used. The first trace shows an 802.11b waveform segment demodulated using the FSK model. The segment corresponds to bit 67 to bit 113 of DBPSK bits, which are 1010101000001011010111001000111000000001111000010. The second trace shows the FSK demodulation results when these bits are instead modulated with FSK (using exactly the same modulation parameters as BLE), and demodulated without the frequency offset. Although the first trace tends to have higher overshoot, the correct bit sequence is still clearly visible, indicating that FSK receivers can indeed be used as WiFi receivers.

With the FSK hardware replacing DSSS and DBPSK demodulator, the only step left for recovering the WiFi bits is descrambling the bit stream. The descrambling process is extremely simple. Furthermore, the descrambling can be done in batch to each byte or word, and does not require extracting/reassembling bits to process them one-by-one. Specifically, the descrambling in 802.11b can be simplified as XOR'ing the input with two shifted versions of the input. With least-significant-bit-first ordering, the descrambling process involves only 4 lines of code:

```
reg = (descrambling_in<<8) | lastbyte;
reg2 = reg ^ (reg>>3) ^ (reg>>7);
descrambling_out = 0xFF & (reg2);
lastbyte = descrambling_in;
```

2.3.2 Packet level. Even with bit-level communication from WiFi to FSK hardware, we still need to address the differences in packet formats in order to successfully receive a WiFi packet.

The 802.11 standard [13] (Sec. 17.2.3.2) explicitly specifies the constant seed that an 802.11b transmitter should use. Because the scrambler seed is always the same, the scrambled SYNC sequence is always the same bit sequence. We verify that this is also the behavior of actual WiFi chips. For all WiFi devices we tested, including (Qualcomm) Atheros, Broadcom, Intel, Marvell, Mediatek/Ralink and Realtek chips, their (scrambled) SYNC is exactly the same bit sequence.

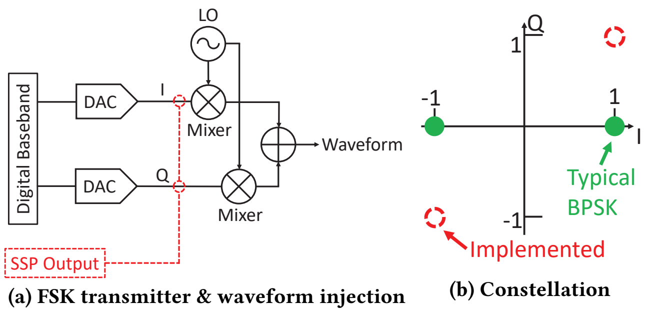
During reception, conventional WiFi chips detect a WiFi packet by matching the SFD pattern (which follows the SYNC field) in the descrambled DBPSK sequence.

In FLEW, the FSK demodulator outputs the scrambled WiFi DBPSK sequence. Because the (scrambled) WiFi SYNC field is always the same sequence, FLEW detects WiFi packets by directly matching a pattern in the scrambled sequence instead of the descrambled sequence. This design allows use of the SFD matching hardware in FSK chips, which is significantly more efficient. Since the matching circuit in FSK chips expects alternating 1's and 0's preceding the SFD to stabilize the demodulator, FLEW matches a bit sequence within the scrambled WiFi SYNC field instead of matching the scrambled WiFi SFD. Specifically, as illustrated in Fig. 5, bit 67 to bit 73 of the DBPSK bit stream is [1,0,1,0,1,0,1] and bit 74 to bit 105 is 0x05AE4701. Therefore, we configure FSK chips to search for 0x05AE4701. Once this sequence is detected, FSK chips continuously put subsequent bytes into the receive FIFO. We can thus periodically retrieve these bytes and descramble them to recover the WiFi packet. The reception is terminated once the number of bytes received reaches the length specified in the PLCP header.

While testing FLEW, we unexpectedly discovered a major bug in Realtek chips: their WiFi SYNC field is 2 bits shorter (126 bits of 1 instead of 128 bits of 1) than required. This is a clear deviation from the 802.11 standard. Even so, we further devise a simple and effective solution so that FLEW uses the same routine for communicating with both Realtek and non-Realtek transmitters. FLEW dynamically detects and fixes the bug for the former. Specifically, after descrambling, FLEW checks the last byte of the WiFi SYNC field, which should be 0xFF, as specified in the standard. If, instead, it is 0x3F, this indicates that the SYNC field is 2 bits shorter and 2 bits of SFD are shifted to this byte (since WiFi transmits least significant bits first). If the bug is detected, we simply apply a shift of 2 bits to all subsequence bytes.

The tail of each WiFi packet is 4 bytes of FCS, which is the CRC32 of the data field. FCS is used to check the integrity of the received packet and if the calculated FCS does not match the received FCS, the receiver should not acknowledge this packet and the transmitter will re-transmit the packet. The implementation of FCS in FLEW is straightforward. We add a few optimizations, such as using table-based calculation and updating the CRC immediately after receiving each byte.

2.4 FSK to WiFi



(a) FSK transmitter & waveform injection (b) Constellation

Figure 6: FSK to WiFi Design

In a conventional PSK system, the PSK receiver is usually complicated due to phase synchronization, but the PSK transmitter is extremely simple. In fact, when a digital bit stream is directly fed into a mixer, the output is the PSK waveform, since bit '0' has a negative voltage and inverts the carrier, while bit '1' has a positive voltage and leaves the carrier unchanged.

By convention, BPSK constellations are (1,0) and (-1,0), as illustrated in Fig. 6b. When these constellations are fed into the IQ modulator, we essentially feed the digital bit stream (that swings to either 1 or -1) into the I-branch mixer and turn off the Q-branch mixer. Note that (1,0) and (-1,0) are 180° apart. However, this constellation involves three voltages (-1,0,1). A simpler implementation is actually tying the I-branch and Q-branch together. The constellations become (1,1) and (-1,-1), which are still 180° apart but only involve two voltages. Furthermore, the output gets 3dB stronger using both branches.

To transmit PSK waveforms using FSK hardware, we turn off the digital baseband and DAC completely and directly inject the signals into the mixers, which can be achieved using the analog pins on FSK chips.

802.11 uses DSSS after PSK modulation. A PSK signal with DSSS is still a PSK signal, only faster. Conceptually, before DSSS, we are injecting either 1 or -1 into the mixer every 1μs. After DSSS, we are injecting either 10110111000 or 01001000111 every 1μs, which translates to a chip rate of 11MChip/s.

To generate the bit stream at 11Mbps, the serial interface (such as SPI, SSP, USART or even I2S) in microcontrollers can be used. These serial interfaces are also commonly double-buffered, ensuring that bits are transmitted continuously and precisely. In fact, on the microcontroller we use, the SSP has 8 transmit buffers.

At the packet level, FLEW assembles packets according to the 802.11b format.

2.5 FSM and MAC Layer

2.5.1 CSMA/CA. In WiFi, the transmission and reception of signals operate in half-duplex. A WiFi device should avoid transmitting signals when other devices are transmitting. WiFi uses CSMA/CA in the MAC layer, which senses the spectrum before transmission and waits if a wireless carrier is present. Typical FSK chips are also capable of sensing the spectrum. In particular, when in receive mode, FSK chips give the RSSI estimates. Combined with the timing design, presented in Sec. 2.5.3, CSMA/CA can be thus implemented.

2.5.2 Packet Handling. In typical WiFi systems, most of packet handling is implemented in the driver or software layers. Two exceptions are ACKs and RTS-CTS, which are subject to a very tight timing constraint, and hence usually handled by hardware.

Except for special packets, normal unicast packets in WiFi need to be acknowledged immediately. Failure of acknowledging a packet results in the sender constantly re-transmitting the same packet, which severely decreases the goodput. Furthermore, when a client tries to join a network, it must acknowledge the association response sent by the AP. Otherwise, the AP de-authenticates the client and a connection cannot be established.

According to the 802.11 standard, an ACK frame should be transmitted by the receiver one SIFS after it receives a packet that passes the FCS check. 802.11b has a very short SIFS (10μs). The standard also specifies the ACKTimeout, which is SIFS (10μs)+aSlotTime (20μs)+Preamble/Header (192μs). This timeout value is measured with respect to the end of the header of the ACK packet. Thus, when measured with respect to the start of the preamble, the time interval between a packet and its ACK packet should ideally be less than 30μs.

According to our testing, the Rx-to-Tx turnaround time of the FSK chip we use is around $30\sim40\mu$s. We found that for chips of industry leaders (Atheros, Broadcom, etc.), the ACK packets sent can be successfully detected without any further design or modification. No workaround is needed for connecting to APs with Atheros, Broadcom and Marvell chips.

On the other hand, certain chips are sensitive to the ACK timing. An extreme example is the Intel chips. We found Intel chips can only detect ACK frames transmitted within a very short time (ideally just less than 10μs). The timeout specified in the standard is determined by the reception of the ACK header, not by the start of the ACK preamble. So, we can, in theory, start the preamble late but transmit less scrambled 1's to meet the deadline. However, such a design has little effect with Intel chips. Without detecting the ACK, the Intel chip re-transmits the same packet over and over again, essentially reducing the goodput to 0.

We, therefore, design a general solution that allows FLEW to meet the timing requirement of all WiFi products we tested. Our solution leverages the facts that a) the tail of WiFi packets is the 4-byte FCS and not the actual payload, b) higher layer either has additional error checking (e.g., TCP, even UDP, has checksums), or naturally anticipates occasional errors. Specifically, we terminate the reception after receiving 1 byte of FCS, thus reserving more time for the FSK chip to transition to transmit mode for ACK. The 1 byte of FCS is still used to check the integrity of the packet and determine whether an ACK packet should be transmitted.

An alternative design is only acknowledging the retransmitted packets and performing a full FCS check on the first packet. However, this will decrease the throughput by half due to re-transmissions.

Since ACK packets are always the same for a given (source) MAC address, instead of generating the ACK on the fly, we pre-generate the ACK bits before sending the authentication packet to the AP and re-use those ACK bits for all subsequent packets. We choose this time instance since the authentication packet signifies an FSK chip's intent to join the AP's network, and the FSK chip is expected to acknowledge traffic from the AP afterwards.

In the opposite direction, once FLEW sends a normal packet, AP should transmit an ACK packet. We use this packet to implement the re-transmit logic. Specifically, once a packet is sent, FLEW turns the FSK chip into receiving mode immediately. If a valid ACK is received, FLEW releases the transmit buffer, turns the chip into receiving mode and copies more data from the upper layer. If no packet is received after a timeout, FLEW re-transmits the packet. If a unicast (to FSK) packet is received, this indicates that the AP and the FSK chip might be transmitting simultaneously. In such a case, FLEW instructs the FSK chip to acknowledge the incoming packet, does not release the transmit buffer, and turns the chip into receiving mode for more incoming packets.

We noticed that the RTS-CTS mechanism is enabled by default on some APs. Thus, we also implement that mechanism. This is important for those APs since without receiving CTS, the AP will constantly re-send RTS without sending any data. The implementation of RTS-CTS is mostly the same as ACK handling, since CTS is also expected to be transmitted one SIFS after RTS.

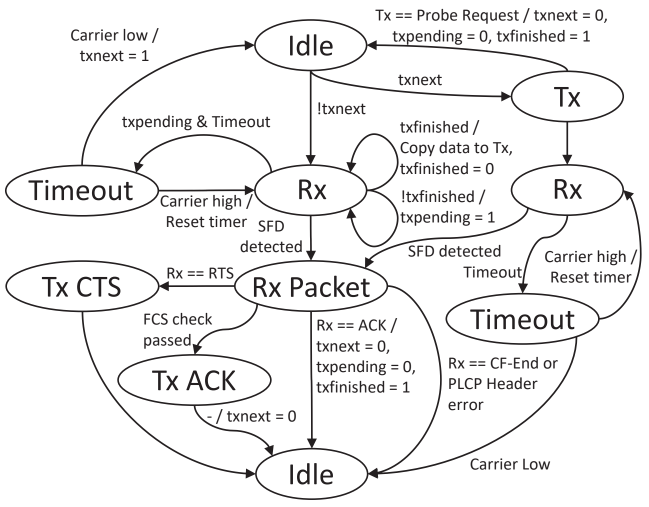

Idle
Tx == Probe Request / txnext = 0, txpending = 0, txfinished = 1
Carrier low / txnext = 1
txnext
Tx
!txnext
txpending & Timeout
txfinished / Copy data to Tx, txfinished = 0
Timeout
Rx
Carrier high / Reset timer
!txfinished / txpending = 1
Rx
SFD detected
SFD detected
Timeout
Carrier high / Reset timer
Tx CTS
Rx == RTS
Rx Packet
Timeout
FCS check passed
Rx == ACK / txnext = 0, txpending = 0, txfinished = 1
Rx == CF-End or PLCP Header error
Tx ACK
- / txnext = 0
Idle
Carrier Low

Figure 7: FSM

2.5.3 *Timing and FSM.* We can now put all components together and design the main control logic, which can be simplified as a Finite State Machine (FSM), shown in Fig. 7. Once in Idle state, FLEW transitions to either Tx or Rx immediately, depending on the value of `txnext`. Rx indicates that the FSK chip is in receive mode, but the SFD (0x05AE4701) is yet to be detected. The CSMA/CA is implemented in this FSM because the only way that `txnext` changes from 0 to 1 (thus initiating a new transmission), is that a) the FSK chip is in Rx and reaches timeout, b) no WiFi packet is detected, and c) no carrier is present in the medium. `txnext` turns to 0 or remains unchanged (e.g., during re-transmission) for all other paths.

The Rx Packet state indicates that a WiFi SYNC is detected and the FSK chip is actively collecting the data. Depending on the packet type, transmission of ACK or CTS may follow. In the case of packet errors, either in the PLCP header or the data field, FLEW goes to Idle and restarts the process.

3 IMPLEMENTATION

3.1 Hardware and Firmware

We use COTS FSK chips to implement FLEW. In particular, we use Ubertooth [14] as the underlying hardware. At its core, Ubertooth uses TI's CC2400 [15], which is a standard FSK chip designed for low-power, low-voltage applications. The CC2400 family is also widely used in wireless sensor networks (WSNs), which require considerably lower power consumption than WiFi networks. Ubertooth was originally conceived as a Bluetooth *and* BLE debugging tool. With an appropriate firmware, it can communicate with Bluetooth and BLE devices. For this purpose, an optional radio front-end is hard-wired to CC2400, although it is not necessary for the operation of CC2400.

Ubertooth uses NXP's LPC1756 [16] as the main MCU, which also acts as a bridge between the USB and CC2400. The MCU is down-clocked to 88MHz via the built-in PLL. We use the LPC1756's

SSP module to transmit the DSSS bits into CC2400's mixers. The IO pins of LPC1756 run at 3.3V whereas the core voltage of CC2400 is 1.8V. We thus use two resistors to convert the voltage. We tie the inputs to I and Q mixers together and with voltage conversion, 3.3V corresponds to 1 and 0V corresponds to -1 to the mixers in the constellation plane.

LPC1756 has an ARM Cortex-M3 core and we implement the FSM using C codes. Our custom firmware also handles the USB traffic and controlling CC2400 via SPI commands and hardware pins. Outgoing packets arrive at LPC1756 in the form of PSK bits, encapsulated in USB packets. The SSP module either sends 10110111000 or 01001000111, depending on each PSK bit. For reception, LPC1756 performs descrambling and FCS check in real time using the Cortex-M3 core. If the check passes, the transmission of ACK is directly initiated by LPC1756. Complete WiFi packets are sent to the host via USB packets.

3.2 WiFi Driver

We write a custom driver (a Linux kernel module) to interface FSK chips to mac80211, which sits on top of WiFi drivers in modern Linux WiFi architecture. Our custom driver is a very thin layer (less than 1k lines of code) that handles various mac80211 function calls, most notably ieee80211_tx and ieee80211_rx. The driver also manages a queue that buffers outgoing packets. Packets are popped off from the queue sequentially. The driver then converts the packet to WiFi PSK bits by adding the PHY header and FCS, scrambling the entire packets and applying differential codings. All these steps are simple bit operations and do not require any floating-point or DSP computation. For received packets, the driver polls USB packets and checks the FCS of the WiFi packet. If the check passes, the driver passes the packet to mac80211.

Even with such a small footprint, our driver supports monitor mode *and* packet injection. Furthermore, we ensure that the driver does not drop packets even when they are injected at the maximum speed, unlike certain other (most notably Realtek and Ralink) drivers.

It is possible to reuse existing WiFi drivers instead of writing a custom one. Such a design is only implementation variations and is not our focus.

3.3 MLME, WiFi Security and Upper Layers

We write the driver and the firmware in such a way that they can directly work with unmodified mac80211 and upper layers. WiFi's MLME (MAC subLayer Management Entity) operations, such as scanning via Probe Requests, authenticating and associating with an AP, are already implemented in mac80211. IP/ICMP and TCP/UDP have already been implemented in the Linux kernel. Many Linux (and Android) distributions come with wpa_supplicant, the *de facto* open-source WiFi security implementation. All these components work without modification. Therefore, the Internet works out-of-the-box once the driver is implemented.

4 EVALUATION

The evaluation of FLEW can be divided into *physical-layer* and *system-level* evaluations. The former measures the physical-layer performance, such as PER (Packet Error Rate), in either WiFi-to-FSK or FSK-to-WiFi direction. Since we need to monitor the PHY traffic for physical-layer evaluation, these experiments are conducted with several WiFi NICs operating in monitor mode. Although these PHY metrics provide insights into synchronization, modulation and demodulation performance, such constant unilateral traffic never exists in real WiFi operations. Specifically, WiFi data packets must be acknowledged using ACK packets in the opposite direction. Therefore, the end-to-end performance depends on the PHY performance in both directions. Furthermore, the MAC layer also affects the end-to-end performance since wireless devices use a shared medium. Therefore, for system-level evaluation, we test FLEW with real WiFi APs and measure performance at the transport layer, which represents a more realistic performance characterization in the real world. The transmit power of FLEW is set to 20dBm for both evaluations.

4.1 Experimental Setup

Table 2: WiFi chips and APs used in experiments.

Chip Maker	PHY Evaluation	System Evaluation
Atheros	AR9462	GL.iNet GL-AR150 (AR9331)
Broadcom	BCM4313	ASUS RT-AC66U (BCM4331)
Ralink/ Mediatek	RT3072	TP-Link TL-WR841N (MT7628NN)
Marvell	-	Linksys EA3500 (88W8366)
Intel	AC 7260	TP-Link Archer AX3000 (WAV654A0)
Realtek	RTL8811AU	D-Link DIR-619L (RTL8192ER)

To test the performance and compatibility, we comprehensively evaluate FLEW with numerous commodity WiFi APs and devices with chips from all major WiFi chip-makers. The tested devices and chips are listed in Table 2.

(Qualcomm) **Atheros** and **Broadcom**[3] have long been considered industry leaders and each has a full WiFi portfolio from low-cost 1T1R to high-end, enterprise-grade chips. The overwhelming majority of Apple's products uses Broadcom's WiFi chips.
Ralink/Mediatek[4] chips are commonly found in low-cost APs. Note that Qualcomm and Mediatek SoC are widely used in high-end and low-cost Android smartphones/tablets, respectively. Therefore, although smartphones are not the main focus of FLEW, testing these chips ensures the compatibility of FLEW with the majority of smartphones. For example, we have verified that FLEW works with an unmodified iPhone acting as a WiFi hotspot.

Marvell[5] chips are mainly used in niche markets (high-end or enterprise APs) and are rarely used as general NICs nowadays. They lack Linux drivers or the drivers do not support monitor mode. So, we evaluate their performance at the system level. Marvell chips are commonly found in Cisco's 802.11ac APs, although Cisco's newer APs use Qualcomm chips. We use a Linksys EA3500 (which was launched during the Cisco era and internally uses a Cisco PCB) to

[3]Broadcom sold part of its WiFi portfolio to Cypress
[4]Mediatek acquired Ralink in 2011
[5]Marvell sold its WiFi portfolio to NXP

evaluate the performance. On the other hand, **Intel** and **Realtek** chips are mostly used in client NICs and have much less presence in APs. Even so, we include their results for completeness.

In what follows, we refer to each system with its WiFi chip-maker. We do not modify the WiFi devices. In fact, we use the original firmware supplied with each equipment and we do not flash any new firmware. For WiFi APs, we use their browser interface to configure the WiFi channels and WiFi passwords. We use WiFi channel 4 and all APs use WPA2-PSK, unless specified otherwise.

4.2 PHY Layer and PER

Table 3: PER Evaluation (%)

Direction	WiFi-to-FSK			FSK-to-WiFi		
Distance	5m	10m	20m	5m	10m	20m
Atheros	1.93	1.46	2.29	0.00	0.02	0.07
Broadcom	3.10	3.15	2.59	0.27	0.78	0.17
Ralink	3.56	4.88	4.22	0.63	1.32	5.83
Intel	5.59	3.03	2.49	2.39	2.17	1.90
Realtek	3.32	4.47	4.69	6.52	8.54	8.86

We evaluate the PHY performance in both FSK-to-WiFi and WiFi-to-FSK directions. The transmitter (either FSK or WiFi) continuously sends a 1508-byte (1512-byte including FCS) packet to the receiver. For all (including FSK) receivers, all 4 bytes of FCS are received and compared; FCS-error frames, if any, are discarded. Since 4 bytes of FCS are used, it is practically impossible ($\frac{1}{2^{32}} = 2.33 \cdot 10^{-10}$) for an error frame to have the correct FCS.

To measure the PER, the transmitter marks these packets with unique sequence numbers. We iterate all possible sequence numbers and thus 4096 packets are sent for each test. We collect the sequence numbers of packets received at the other end. Because the sequence numbers sent are unique, any number missing within [0, 4095] indicates packet losses/errors.

Table 3 shows the PER in the WiFi-to-FSK direction. For signal transmission, the Atheros chip shows superior performance. We use exactly the same set of typical WiFi antennas for Atheros, Broadcom and Intel chips, and their performance is largely the same. Even at 20m, the PER is around 2.5% or lower. For Broadcom and Intel chips, the PER actually increases slightly with shorter distances. This may be because the signal is slightly stronger than the optimal ranges, and they potentially have a slightly less accurate waveform. It is possible to further tweak the AGC (Automatic Gain Control) in the FSK chip, instead of using the default values, to extend the optimal range. For Ralink and Realtek chips, the NICs are in the USB form (since the mini PCI-E versions are hard to find). Because of the form factor limitation, USB NICs typically do not have the best performance, as evident in the table. `FLEW` still achieves a PER of less than 5% at 20m. Although Realtek chips do contain the SYNC length bug, to the FSK chip, this bug only affects the packet format and should be mostly unrelated to the PER.

Table 3 also shows the PER in the FSK-to-WiFi direction. The Atheros chip again shows phenomenal performance. At a distance of 5m, it is even possible to achieve 0% PER. Even at 20m, the PER is still much less than 1%. The Broadcom chip also has great performance with less than 1% PER under all conditions. For the Ralink chip, we believe the performance is mostly limited by the USB form factor, and thus the PER steadily increases with distance. The Realtek chip has the worst receiving performance among all chips (including receiving using FSK chips!), which may be partly contributed by its tiny USB form factor. It is also possible that bugs in Realtek chips affect the performance (e.g., expecting 126 bits of SYNC instead of 128 bits).

4.3 TCP/UDP Throughput

For system-level evaluation, we measure the transport-layer throughputs with unmodified WiFi APs. We use iperf3 [17], the standard tool for measuring network performance. An iperf3 server is either run on the AP itself (Atheros), or on a host connected (via LAN) to the AP (Broadcom, Ralink, Marvell, Intel, Realtek). At the other end, a Ubuntu laptop with `FLEW` driver installed is running an iperf3 client. The laptop connects to these APs via the FSK chip. We measure both TCP and UDP throughputs in both directions with the following commands:

```
TCP Uplink: iperf3 -c <server's IP>
UDP Uplink: iperf3 -c <server's IP> -u
TCP Downlink: iperf3 -c <server's IP> -R
UDP Downlink: iperf3 -c <server's IP> -R -u
```

We record the throughput reported by the receiving end.

Table 4 shows the uplink throughputs at different distances. All AP chips of four popular chip-makers (Atheros, Broadcom, Ralink, Marvell) have similarly good performances. The Atheros chip is still slightly better than others. UDP throughputs are higher than TCP throughputs as TCP requires additional TCP ACKs sent at the transport layer. To the physical layer, these TCP ACKs are simply packet payloads traveling in the opposite direction. Although it is redundant to have ACKs at both the physical layer and the transport layer, it is the way the Internet works. Because WiFi is half-duplex, these TCP ACKs consume some bandwidth, and thus decrease the "real" throughput. In addition, these TCP ACKs create more contention as the AP now also tries to access the channel to transmit TCP ACKs. Nevertheless, `FLEW`'s CSMA/CA works well and the throughput is only 5~10% lower. The Intel chip has equally good performance but is less stable at longer distances.

The only outlier in the uplink experiments is Realtek. This may be attributed to its inferior receiving performance, potentially due to the SYNC or other bugs. We would like to stress that Realtek chips are much less commonly used in commodity APs, and are especially rare in enterprise APs. They can still sustain a transport-layer throughput of at least 350kbps. We leave specific optimizations for Realtek chips as future work.

Table 4 also shows the downlink throughputs. Downlink throughputs are, in general, higher than uplink throughputs because, in the FSM (Fig. 7), a small amount of time is spent on copying transmit data between each transmission. That is, even if the channel is clear/available, the FSK chip will not transition to Tx mode if the data is not ready. This design simplifies the re-transmit and USB logic, but the throughputs are slightly lower due to unused airtime. The difference is smaller for TCP because the unused airtime can be reclaimed by the AP to transmit TCP ACKs.

Table 4: Throughput Evaluation (kbps)

Direction	Uplink						Downlink					
Transport	TCP			UDP			TCP			UDP		
Distance	5m	10m	20m	5m	10m	20m	5m	10m	20m	5m	10m	20m
Atheros	646	661	671	721	700	697	766	771	768	839	840	857
Broadcom	616	602	654	697	698	697	703	686	707	813	818	800
Ralink	654	639	615	695	692	708	767	749	734	846	842	834
Marvell	645	636	656	693	701	698	679	678	694	792	803	791
Intel	651	636	471	699	578	673	743	721	586	811	814	816
Realtek	350	357	449	375	346	386	588	525	544	700	658	667

Downlink has a similar trend to uplink. The first four chip-makers have most consistent results across different distance ranges. The Intel chip can be equally good but is less consistent. The Realtek chip performs much better, since downlink involves mostly transmission, but is still inferior to other chips.

Unlike uplink, downlink throughputs show higher variations among different chip-makers. For example, Atheros and Ralink chips consistently outperform Broadcom and Marvell chips. We believe the rate adaptation algorithms (RAAs) used largely contribute to this since Atheros and Ralink (and even Intel) have their own proprietary RAAs [18–20]. When data packets are sent by the AP, the AP may briefly try a different rate before falling back to 1Mbps. These RAAs seem to better balance the throughput and exploration of other rates. As discussed in Sec. 1, using 1Mbps allows RAAs to converge and not diverge or cause disconnection. Even at distance of 20m, FLEW stays connected throughout all the experiments.

4.4 RTT

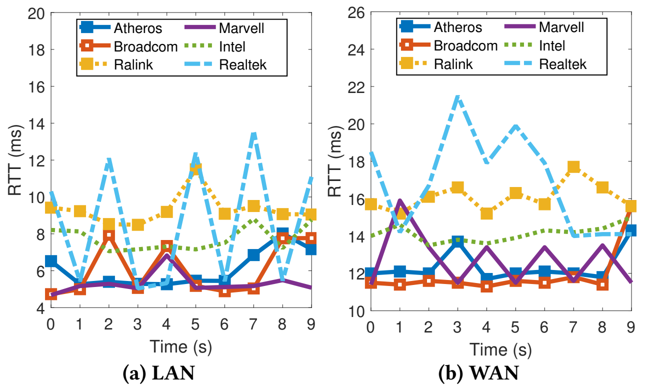


Figure 8: RTT

We also measure the round-trip time (RTT) of each setting. Furthermore, we measure the RTT over both LAN and WAN. For LAN, we `ping` the AP to which the FSK chip is connected. For WAN, we connect the AP to the Internet and FLEW tries to `ping` 8.8.4.4, Google's public DNS server.

For each system, we perform 10 pings. We measure RTTs at 20m, since the RTTs of shorter distances are largely the same as those of 20m. Even at 20m, no ping was lost for every configuration.

Figs. 8a and 8b plot the measurement results. Atheros, Broadcom and Marvell generally have the lowest RTTs. Intel and Ralink lag behind, but their RTTs are still stable. Realtek has the worst RTTs that jump all over the place. This can be partly attributed to Realtek chips' non-ideal wireless performance. WAN RTTs are mostly the same as LAN RTTs. The only difference is the ~6ms delay for traveling across the Internet.

4.5 Coexistence

Table 5: Coexistence with multiple WiFi devices (bps)

# of Devices		TCP UL	UDP UL	TCP DL	UDP DL
1	FLEW	663 k	708 k	730 k	841 k
2	FLEW	412 k	698 k	293 k	603 k
	AC7260	13.1 M	1.05 M	25.0 M	1.05 M
3	FLEW	123 k	676 k	117 k	517 k
	AC7260	10.5 M	1.05 M	19.8 M	1.05 M
	AR9271	19.3 M	1.05 M	11.5 M	1.00 M
4	FLEW	92.7 k	616 k	178 k	504 k
	AC7260	8.27 M	1.05 M	12.5 M	1.06 M
	AR9271	7.73 M	1.05 M	6.2 M	1.04 M
	RTL8811AU	10.9 M	1.05 M	12.5 M	1.05 M
5	FLEW	90.9 k	621 k	166 k	580 k
	AC7260	6.98 M	1.05 M	10.3 M	1.05 M
	AR9271	1.95 M	1.05 M	3.09 M	1.03 M
	RTL8811AU	2.80 M	1.05 M	10.4 M	1.05 M
	BCM43602	15.2 M	1.05 M	7.12 M	1.05 M

4.5.1 Coexistence with WiFi Devices. To evaluate FLEW's performance when it coexists with other WiFi devices, we also use iperf3 to measure the performance when up to 5 WiFi clients are simultaneously sending or receiving data. To accommodate multiple iperf3 connections, multiple iperf3 servers simultaneously run on the Atheros AP and listen on different ports.

Table 5 shows the results. By default, iperf3 injects UDP data to and from each client at around 1.05Mbps. In these cases, the channel is not saturated and all clients access the channel efficiently. For UDP uplink, the throughput of FLEW does not decrease much, indicating that the MAC layer of FLEW functions properly and allows different WiFi clients to access the spectrum efficiently. Throughputs decrease more for downlink, which may be due to slight delays for a single AP to prepare and send multiple streams.

For TCP, iperf3 injects the data at the maximum speed, which saturates the channel. In these situations, any throughput gain at one client comes at the expense of the throughput decrease at another client. For FLEW, throughput decreases are more pronounced as 2 or 3 devices coexisting and plateau out as more devices are added. We believe that from FLEW 's perspective, interference increases drastically from 1 to 2 or 3 devices. Although interference does continue to increase for more devices, the increase in ratio is comparatively lower. We observe that even though WiFi devices follow the WiFi MAC standard, throughput imbalance still occurs, even among COTS WiFi cards from different vendors, when the WiFi channel is saturated to the max. Therefore, we conclude that the WiFi MAC alone cannot guarantee a perfect throughput distribution among WiFi terminals in practical wireless environments. However, the key takeaway is that even under saturation, throughput of FLEW does not starve to 0 and is still sufficient for many IoT applications. Furthermore, a WiFi channel may only truly saturate occasionally during bursty data transmission. In such cases, FLEW can still provide decent overall throughput by utilizing the channel idle time.

A fairer throughput distribution can be achieved by rate limiting or a better load balancing at the AP. Alternatively, PCF can be used to allow the AP to arbitrate the traffic and eliminate spectrum contentions. It is also possible to fine-tune the MAC layer to access the spectrum more aggressively. We leave these as future work.

Table 6: Throughputs with multiple FLEW devices (kbps)

# of Devices		TCP UL	UDP UL	TCP DL	UDP DL
1	FLEW #1	663	714	753	833
2	FLEW #1	357	385	390	295
	FLEW #2	358	386	307	396

Table 7: Coexistence with background BT traffic (kbps)

# of BT Devices	TCP UL	UDP UL	TCP DL	UDP DL
0	655	715	761	819
1	667	716	758	820
2	650	714	748	821

4.5.2 Coexistence with FSK Devices. FLEW coexists with other FSK devices well. Table 6 shows the results of multiple FLEW nodes sending/receiving data simultaneously. For uplink, throughputs are almost perfectly divided by multiple nodes, indicating that the MAC layer design allows each node to access the spectrum equally and efficiently. Furthermore, the aggregate uplink throughputs of multiple nodes are slightly higher, which is a result of the utilization, by the second node, of the small unused uplink airtime (described in Sec. 4.3) of the first node. For downlink, the throughputs are split in about 57:43 between nodes, which may be due to the different channel conditions and packet processing in the AP. The aggregate downlink throughputs of multiple nodes are slightly lower because of more spectrum contention.

FLEW achieves excellent performance in the presence of coexisting Bluetooth devices. We evaluate the performance of FLEW when there are multiple Bluetooth connections in the background streaming music. Table 7 shows that the throughputs of FLEW are virtually unaffected by the number of Bluetooth audio streams in the environment. We attribute this result to the frequency-hopping design and the adaptive frequency-hopping mechanism in Bluetooth protocols. Specifically, Bluetooth transmitters are constantly switching channels over the span of 79MHz, thus making a Bluetooth transmitter unlikely to hop to the same frequency as the FSK chip. Furthermore, Bluetooth systems use the adaptive frequency-hopping mechanism, which automatically avoids Bluetooth channels within active WiFi channels. Additionally, Bluetooth systems typically operate at a lower transmit power, which, combined with robustness of DSSS, mitigates the impact on WiFi performance even if a collision does occur. Since FLEW uses the standard WiFi waveform and WiFi MAC design (i.e., timing), the existing results of WiFi–Bluetooth coexistence and interference-reduction mechanisms are directly applicable to FLEW.

4.6 Mobile and Outdoor Environments

Table 8: Throughputs in mobile environments (kbps)

Condition	TCP UL	UDP UL	TCP DL	UDP DL
Stationary	641	700	682	809
Walking	651	642	673	747
Running	647	677	670	733

4.6.1 Mobile Environments. To evaluate the performance under mobile conditions, we measure the throughputs when a person with the FLEW terminal is walking or running back and forth toward/away from the AP at a distance of 10∼15 meters. Interestingly, Table 9 shows that mobile environments have more impacts on UDP than on TCP throughputs, although UDP still has higher throughputs. From the results, we conclude that the extra robustness and throttling provided by TCP may mitigate the throughput variations in certain conditions.

Further optimizations for mobile environments are possible. For example, we use the default packet size, which has a relatively long time duration. If shorter packets are used (e.g., by fragmentation), the channel response may be more similar within the duration of each packet, making it closer to the (piecewise) stationary condition.

Table 9: Throughputs in outdoor environments (kbps)

Distance (m)	TCP UL	UDP UL	TCP DL	UDP DL
25	654	661	734	821
50	646	702	731	844
75	652	709	761	862
100	674	716	739	828
125	652	710	719	763
150	677	710	610	689

4.6.2 Outdoor Environments. We also evaluate FLEW in practical outdoor environments. Table 9 shows the performance in a typical university campus environment with very few interference sources around the Atheros AP. Uplink throughputs maintain consistently good performance for at least 150m, which validates that FLEW is ideal for IoT applications where sensor data travels in the uplink direction. Downlink throughputs are consistently high within 100m and are reduced gradually for longer ranges, which is likely a result of the higher PER. Even so, at 150m, throughputs of FLEW in real-world outdoor environments are at least 2x the maximum throughput of Zigbee. (In addition, Zigbee at 2.4GHz typically only has a range of 10~100m [21].)

Outdoor and industrial WiFi APs are allowed to transmit at a higher power than indoor APs. For example, a typical Cisco outdoor AP [22] has a transmit power of 30 dBm, which is at least 10~12 dB (corresponding to 3~4x in range) higher than the AP we use. This additional transmission power can significantly increase downlink range. The Cisco outdoor AP also has a better sensitivity at -103 dBm. Therefore, we expect an even better performance when FLEW is paired with APs designed for outdoor and industrial use.

4.7 Secured vs. Open Network

Table 10: Throughputs w/wo WPA2-PSK (kbps)

WPA2-PSK	TCP UL	UDP UL	TCP DL	UDP DL
Enabled	661	700	755	850
Disabled	676	697	780	858

All system evaluations presented so far are done with WiFi security (WPA2-PSK) enabled, since it is the recommended setting. Enabling WiFi security incurs only a very small amount of overhead. For example, WPA2-PSK incurs an overhead of 16 bytes per packet, which is about 1% of a typical WiFi data packet (~1500 bytes). Throughput may increase very slightly using open networks. Using the Atheros AP as an example, we measure the throughputs with and without WPA2-PSK and leave other parameters, including device placement, intact. Table 10 shows slightly higher throughputs using open networks. In practice, other factors, such as background traffic and interference, can easily outweigh the effect of WiFi security settings.

4.8 Application Examples

To the network stack and the AP, FLEW behaves just like a conventional WiFi chip. Network applications can use FLEW for Internet access without even recognizing the use of an FSK chip, instead of a WiFi chip. General web browsing works normally as well. In addition, FLEW can support streaming high-quality audio in real time using Spotify. FLEW is also shown to be able to support streaming 480p Youtube videos in real time. When little/no background interference is present, it can even stream 720p Youtube videos in real time. Higher resolution videos can be supported with enough buffering (or potentially in real time with better video compression).

4.9 Power Consumption

Table 11: Comparison of power consumption

	Idle (A)	Tx (A)	Rx (A)	Rx-Idle (A)
AR9271	0.02	0.49	0.07	0.05
RTL8811AU	0.01	0.32	0.07	0.06
RT3072	0.04	0.28	0.13	0.09
FLEW	0.08	0.16	0.11	0.03

FSK chips are very power-efficient. CC2400 only draws 19mA in transmit mode (at 0dBm) and 24mA in receive mode. For a fair comparison (and since the transmit current of conventional WiFi chips is typically measured at near 20dBm), we consider that an external PA [23] is used, which is also the case in Ubertooth. The external PA boosts the signal to 20dBm but draws 100mA (at 3.3V). Even with the PA, the overall power consumption is considerably lower than the WF200 in Table 1.

Table 11 shows the overall power consumption of different USB WiFi cards, including Ubertooth with FLEW. We measure their USB (5V) supply current when in idle, in continuous transmission or reception of 1Mbps WiFi waveforms. FLEW has the lowest power consumption in Tx mode. Ubertooth has a higher idle current, even when CC2400 and the PA are turned off, than other WiFi cards. If we adjust the Rx current with the idle current, FLEW also has the lowest power consumption in Rx mode.

5 DISCUSSION

5.1 OFDM

The 802.11 standard has always stressed backward compatibility, which enables FLEW to communicate with newer systems, even though they might primarily use OFDM modulation. Specifically, 802.11g is a superset of 802.11b and 802.11a and the standard explicitly specifies that any 802.11g device should fully support 802.11b operations. Similarly, the standard states that every 802.11n device should also be an 802.11g device; every 802.11ax device should also be an 802.11n device, etc. Therefore, even if new devices use OFDM, they should still support FLEW operations. We have verified that FLEW works with 802.11b, 802.11g, 802.11n, 802.11ac and 802.11ax APs and devices.

Compared to OFDM waveforms, DSSS waveforms are more suitable for IoT applications, where device complexity, power consumption and cost are more important than throughput. In addition, DSSS waveforms are much more robust than OFDM (a 9dB [7] higher link budget). OFDM waveforms are also known to have high PAPR (Peak-to-Average Power Ratio), which prevents the use of simple, power-efficient PA and LNA. In contrast, PSK waveforms are generally considered as a form of constant-envelope modulation since information is encoded entirely in the phase of the signal [24]. Therefore, DSSS is much more suitable for the PA and LNA inside FSK chips, since FSK is also a constant-envelope modulation.

The data rate of 802.11b overlaps with common FSK data rates (1Mbps), which allows FLEW to directly reuse demodulation hardware, including the packet handling circuits, on common FSK chips, thus enabling low-power operations. The data rate of WiFi OFDM does not overlap with common FSK data rates. Consequently, the demodulator and packet handling circuits on FSK chips are unlikely to work with OFDM waveforms. Furthermore, processing OFDM waveforms requires significantly more complex and power-hungry circuits, including much wider radio front-end, faster A/D conversion and FFT circuits. These essential OFDM building blocks are simply not present on FSK chips.

Finally, we found that 802.11g/802.11n/802.11ac/802.11ax APs, by default, use 802.11b waveforms to transmit beacons and management frames. Therefore, even if there exists a solution that enables FSK chips to receive/transmit OFDM waveforms, the support of 802.11b is still essential, especially if the APs cannot be modified. In this regard, FLEW will still be integral in such solutions to handle management functions, such as association with APs, and to ensure maximum compatibility with unmodified WiFi devices.

From the practicality perspective, if existing devices already use FSK, Bluetooth or BLE protocols, then the throughput of FLEW is sufficient since FLEW matches their mandatory data rate. Thus, for these devices, the benefits of OFDM waveforms are not critical, especially considering DSSS waveforms are more robust and FSK chips lack the OFDM building blocks.

The limitation of emulating OFDM mainly comes from the fundamental hardware limits of FSK chips. It might be possible to use the radio circuits on FSK chips and emulate OFDM building blocks in firmware using a high-speed micro-controller. However, such a design would require a significant amount of computing power and would likely require floating point units (FPUs). These requirements will likely prevent the adoption of such a solution to simple, low-power and low-cost IoT devices. In contrast, FLEW does not require any floating point calculations. In fact, the Cortex-M3 micro-controller on Ubertooth is not equipped with any FPU.

5.2 Security Implementation

WiFi security algorithms can be implemented in software. However, modern WiFi security frameworks (e.g., WPA2 and WPA3) use AES and most CPUs (Intel since Sandy Bridge, AMD since Bulldozer, modern ARM processors) support AES instructions. Many micro-controllers, especially ARM TrustZone MCUs and even low-end MCUs designed for IoT, also have hardware AES accelerators. These AES hardware help efficiently encrypt and decrypt WiFi payloads.

6 RELATED WORK

Multiple systems [25–28] have demonstrated the communication from modified WiFi devices to Zigbee chips. However, they aim to turn WiFi devices into a Zigbee transmitter, and in those systems, Zigbee chips do *not* work as a conventional WiFi device.

Several prior studies explore communication from modified WiFi devices to Bluetooth/BLE devices. BlueFi [29] modifies WiFi devices to transmit Bluetooth signals, such as BLE beacons or audio packets. WiBeacon [30] modifies WiFi APs to transmit BLE beacons. These systems only allow one-way broadcast (beacons) or one-way unicast (audio) communication *from* WiFi *to* Bluetooth devices. In addition, modifications must be made to the WiFi device. NBee [31] shows the possibility of bit-level communication from 802.11b (with QPSK formulation) to BLE receivers, NBee uses this to construct the magic packets that a modified 802.11b transmitter should send so that the BLE receivers can decode the packet as a normal BLE packet. In contrast, FLEW analyzes BPSK waveforms and constructs a fully functional 802.11b receiver for packets with arbitrary payloads, transmitted by unmodified WiFi transmitters. In addition, FLEW presents a general model of FSK demodulation and we show, with theoretical insights and simulations, how such a communication will work on the general, architectural level. The core analysis in NBee is under the condition of BLE receivers being tuned to the same frequency as the WiFi channel. FLEW uses a frequency offset and we explain, with theoretical reasoning, why such an offset plays a critical role in converting DSSS to PSK waveforms and in demodulating DBPSK waveforms with FSK circuits. With the general FSK receiver model, the WiFi-to-FSK result only holds when the frequency offset is introduced. FLEW also demonstrates how the packet handling hardware on FSK chips can be used to detect and efficiently decode conventional WiFi packets, not just certain packets that are precoded in the BLE form. FLEW also addresses the subtle issues of waveforms transmitted by different WiFi chips, such as the bug of Realtek chips. NBee only evaluates the performance using USRP whereas FLEW extensively evaluates the performance with real, unmodified WiFi chips from all major WiFi chip makers. Therefore, FLEW targets a very different use-case and presents a practical design and results under real-world conditions.

Although another prior work [32] does enable bi-directional communication between SDR-emulated WiFi devices and modified Bluetooth devices, not only does it require modifications to both Bluetooth and WiFi devices, it also does not validate the running of such a method on real commercial WiFi chips (instead of SDR devices). Moreover, a custom encoding is used on top of Bluetooth modulation, thus significantly reducing its throughput. All these systems aim to use WiFi to transmit Bluetooth waveforms and their Bluetooth devices do not operate as a conventional WiFi client. Their Bluetooth devices can only receive WiFi packets with specifically-crafted payloads. Inter-scatter [33] enables bi-directional communication between a backscatter, which uses a modified Bluetooth transmitter as the RF source, and modified WiFi devices. It also requires the RF source to be placed very close (<1m) to the backscatter.

BlueBee [34] enables communication from modified Bluetooth transmitters to Zigbee receivers and XBee [35] enables communication from modified (for generating access codes) Zigbee transmitters to modified (for cross-coding) Bluetooth receivers. Zigbee uses MSK [36], which is a special form of FSK. In contrast, WiFi uses PSK with DSSS, which is much more different than FSK. In addition, WiFi is 4x faster than Zigbee and it is theoretically impossible (250kbps<1Mbps) for a standard Zigbee chip to receive WiFi packets with arbitrary payloads. Therefore, their methods are not directly applicable to our system. Furthermore, even at the transport layer, FLEW is still much faster than Zigbee CTCs (250 kbps). XBee configures the access address of BLE receivers to detect Zigbee packets. The contribution of FLEW here is that we methodologically find a specific SFD and show via extensive experiments that the SFD works well for WiFi-to-FSK communication. In addition, the process of selecting SFD in FLEW can be generalized for other WiFi applications, such as reducing the interferences between WiFi and FSK devices, or waking up WiFi receivers with BLE chips. These contributions are very different from XBee, which is designed for Zigbee-to-BLE communication.

7 CONCLUSION

We have presented FLEW that enables low-power FSK chips to directly communicate with unmodified WiFi APs. We have leveraged several insights on FSK and WiFi modulation. We have evaluated FLEW extensively to demonstrate its compatibility with chips from all major WiFi chip-makers with good performance.

8 ACKNOWLEDGEMENTS

This work was supported in part by the NSF under Grant No. CNS-1646130 and the ARO under Grant No. W911NF-21-1-0057.

REFERENCES

[1] Wi-Fi Alliance. Wi-fi® in 2019. https://www.wi-fi.org/news-events/newsroom/wi-fi-in-2019, Feb 2019.

[2] Silicon Labs. Wf200 data sheet: Wi-fi network co-processor. https://www.silabs.com/documents/public/data-sheets/wf200-datasheet.pdf, Sep 2020.

[3] Texas Instruments. Cc2500 low-cost low-power 2.4 ghz rf transceiver. https://www.ti.com/lit/ds/swrs040c/swrs040c.pdf, May 2008.

[4] Texas Instruments. Cc2650 simplelink™ multistandard wireless mcu. https://www.ti.com/lit/ds/symlink/cc2650.pdf, Feb 2015.

[5] 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.

[6] Cisco. Cisco wireless mesh access points, design and deployment guide, release 7.4. https://www.cisco.com/c/en/us/td/docs/wireless/technology/mesh/7-4/design/guide/mesh74/mesh74_chapter_011.html, Jun 2020.

[7] 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.

[8] Texas Instruments. Cc2620 simplelink zigbee rf4ce wireless mcu. https://www.ti.com/lit/ds/symlink/cc2620.pdf, Dec 2020.

[9] Atmel. At86rf233. http://ww1.microchip.com/downloads/en/devicedoc/atmel-8351-mcu_wireless-at86rf233_datasheet.pdf, Jul 2014.

[10] Texas Instruments. Cc2652r simplelink multiprotocol 2.4 ghz wireless mcu. https://www.ti.com/lit/ds/symlink/cc2652r.pdf, Mar 2021.

[11] Qualcomm. Csr8811. https://www.qualcomm.com/products/csr8811, 2021.

[12] Cypress. Cyw43012. https://www.cypress.com/file/497511/download, Jun 2020.

[13] 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.

[14] Michael Ossmann. Ubertooth one. https://greatscottgadgets.com/ubertoothone/, 2021.

[15] Texas Instruments. Cc2400 2.4 ghz low-power rf transceiver. https://www.ti.com/lit/ds/symlink/cc2400.pdf, Mar 2006.

[16] NXP. Lpc1756fbd80: Scalable mainstream 32-bit microcontroller (mcu) based on arm® cortex®-m3 core. https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc1700-cortex-m3/scalable-mainstream-32-bit-microcontroller-mcu-based-on-arm-cortex-m3-core:LPC1756FBD80, 2021.

[17] iPerf. iperf - the ultimate speed test tool for tcp, udp and sctp. https://iperf.fr/, 2020.

[18] Duy Nguyen, J. J. Garcia-Luna-Aceves, and Cedric Westphal. Throughput enabled rate adaptation in wireless networks. In *2013 International Conference on Computing, Networking and Communications (ICNC)*, pages 1173–1178, 2013.

[19] Electronic Products. Ralink premiers industry's first 450 mbps 802.11n router solution with beamforming technology at ces 2009. https://www.electronicproducts.com/ralink-premiers-industrys-first-450-mbps-802-11n-router-solution-with-\beamforming-technology-at-ces-2009/#, Jan 2009.

[20] Rémy Grünblatt, Isabelle Guérin-Lassous, and Olivier Simonin. Simulation and performance evaluation of the intel rate adaptation algorithm. In *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '19, page 27–34, New York, NY, USA, 2019. Association for Computing Machinery.

[21] Connectivity Standards Alliance. Zigbee faq. https://zigbeealliance.org/zigbee-faq/, 2021.

[22] Cisco. Cisco aironet 1570 series outdoor access point data sheet. https://www.cisco.com/c/en/us/products/collateral/wireless/aironet-1570-series/datasheet-c78-732348.html, Mar 2021.

[23] Texas Instruments. Cc2591 2.4-ghz rf front end. https://www.ti.com/lit/ds/swrs070b/swrs070b.pdf, Sep 2014.

[24] James E. Gilley. Digital phase modulation. https://www.efjohnson.com/resources/dyn/files/75832z342fce97/_fn/Digital_Phase_Modulation.pdf, Aug 2003.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] Vikram Iyer, Vamsi Talla, Bryce Kellogg, Shyamnath Gollakota, and Joshua Smith. Inter-technology backscatter: Towards internet connectivity for implanted devices. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 356–369, New York, NY, USA, 2016. Association for Computing Machinery.

[34] 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.

[35] 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.

[36] Mohammed Abdullah Zubair, Ajay Kumar Nain, Jagadish Bandaru, P. Rajalakshmi, and U.B. Desai. Reconfigurable dual mode ieee 802.15.4 digital baseband receiver for diverse iot applications. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 389–394, 2016.