Unify: Turning BLE/FSK SoC into WiFi SoC

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

ABSTRACT

With the potential for connecting billions of WiFi devices to low-power Bluetooth/FSK devices, WiFi–Bluetooth cross-technology communication (CTC) has drawn significant interests from academia and industry. However, state-of-the-art CTC solutions either are limited to *one-way communication*, or require *multiple chips with hardware modifications*.

We present a novel solution, `Unify`, that eliminates both of these undesirable features. By just updating firmware, `Unify` transforms BLE/FSK SoCs (Systems-on-a-Chip) into WiFi SoCs. It enables bidirectional communication between BLE/FSK devices and *unmodified* WiFi APs/devices, and is compatible with products of all major WiFi chip vendors. `Unify` is a single-chip solution and requires no hardware modifications. Thanks to their high integration, `Unify` devices are smaller, cheaper and consume significantly less power than both off-the-shelf WiFi chips and prior WiFi–FSK CTC solutions.

CCS CONCEPTS

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

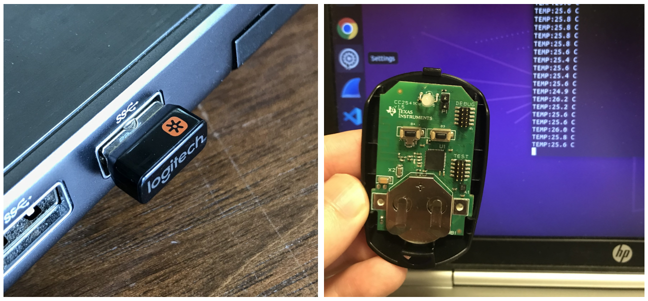
KEYWORDS

Cross-Technology Communication, WiFi, Bluetooth, BLE

ACM Reference Format:
Hsun-Wei Cho and Kang G. Shin. 2023. Unify: Turning BLE/FSK SoC into WiFi SoC. In *The 29th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '23), October 2–6, 2023, Madrid, Spain.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3570361.3592512

1 INTRODUCTION

WiFi and Bluetooth are the two leading wireless technologies that connect tens of billions of devices. Conventionally, these two technologies are deemed incompatible and treat signals from/to mismatched devices as interference. Recently, however, researchers in the field of cross-technology communication (CTC) have shown that this "interference" can actually become valid communication signals by leveraging digital signal processing to pre-process signals and/or by reusing various radio and digital blocks.



(a) Unifying dongles **(b) CC2541 BLE modules**

Figure 1: (a) `Unify` turns a Logitech Unifying dongle into a fully operational WiFi dongle without any hardware modification. The dongle directly connects to a standard WiFi AP. (b) `Unify` turns a CC2541 BLE keyfob into a WiFi thermostat. CC2541 uploads sensor readings directly to the Internet via a WiFi AP.

It is both important and practical to enable bi-directional communication between WiFi and Bluetooth devices. It allows a device to communicate in environments without any other device of the same wireless technology, which was previously impossible. For example, Bluetooth devices can directly connect to WiFi APs and access the Internet, eliminating the need to install any Bluetooth gateways. Furthermore, such a communication can combine the strengths of both WiFi and Bluetooth. For example, Bluetooth has lower cost, lower device complexity, and better power efficiency than WiFi, while WiFi infrastructure is omnipresent and provides direct Internet access. BLE chips are typically cheaper than WiFi chips, and the much simpler hardware of many BLE system-on-chips (SoCs) enables their operation for several years with a single coin cell battery. Bi-directional communication between WiFi and Bluetooth opens a new opportunity of connecting low-cost, energy-efficient Bluetooth devices to omnipresent WiFi APs and provides direct Internet connectivity to Bluetooth devices without requiring gateways.

This envisioned WiFi–Bluetooth bi-directional communication is particularly suitable for Internet of Things (IoTs)

ACM MobiCom '23, October 2–6, 2023, Madrid, Spain

ACM ISBN 978-1-4503-9990-6/23/10…$15.00
https://doi.org/10.1145/3570361.3592512

and wireless sensor networks (WSNs) in which nodes upload sensor data to, and receive actuation (control) messages from, the cloud. In these use-cases, the sensor data and the actuation messages are usually small but should be sent/received in a timely manner in order to accurately reflect and control real-world events/behaviors. Considering the large number of IoT/WSN nodes that have already been, or are expected to be, deployed in the real world, each of them should be cheap, small and highly power-efficient so that they can "operate" as intended without frequent battery changes/charging. BLE and FSK chips are ideal for meeting these requirements as they are very cheap, small, and suitable for "small but frequent" (as opposed to "large and bursty") communications with ultra low power consumption. However, IoT and WSN nodes require connection to the Internet, but BLE or FSK chips by themselves do not support such connectivity and hence require gateways. Bi-directional communication between WiFi and Bluetooth can fill this gap by equipping BLE/FSK chips with WiFi connectivity, thus allowing IoT or sensor nodes to use cheap and energy-efficient BLE/FSK chips while also enabling their direct use in conventional WiFi environments.

On the other hand, state-of-the-art (SOTA) CTC solutions have a number of limitations and have not yet fully realized this vision. Specifically, some CTC solutions only enable *one-way communication* whereas others require use of *multiple chips and hardware modifications*.

Numerous SOTA CTC solutions [1–5] enable one-way communication by modifying WiFi transmitters to send "magic packets" that can be received by Bluetooth devices. Specifically, these solutions carefully select the bits within a WiFi packet so that, after modulating these bits in WiFi modulation, the physical waveform resembles a legitimate Bluetooth waveform. However, since this approach can only provide one-way communication from WiFi Tx to Bluetooth Rx, it cannot be used when IoT/WSN nodes (using Bluetooth chips) need to upload sensor data to the cloud via WiFi infrastructure. Also, device discovery and encryption cannot be implemented with one-way communication without side channel information, which significantly complicates system deployment. As a result, the use of such solutions is limited to Bluetooth beacon broadcasting or one-way "downlink" (from WiFi to Bluetooth) communication.

FLEW [6] introduced a fundamentally different design that addresses the above shortcomings and is shown to work with actual WiFi APs. Other CTCs enable WiFi-to-Bluetooth one-way communication by modifying WiFi transmitters to emulate Bluetooth transmitters. Unlike these other CTCs, FLEW enables bi-directional communication between WiFi and Bluetooth by directly enabling conventional WiFi operations on FSK devices (i.e., the hardware of Bluetooth devices). It makes an FSK device appear as a standard WiFi device and the FSK device adheres to the conventional WiFi protocol. The modified FSK device can then connect to unmodified WiFi APs and use the standard WiFi encryption. Because the FSK device behaves like a standard WiFi device, users do not need to modify the WiFi AP or infrastructure, and no IoT gateway is needed either.

Despite its many salient features, FLEW also has a few shortcomings. In order to transmit WiFi waveforms using an FSK device, FLEW requires hardware modifications to inject the WiFi waveform into the mixers of the FSK chip. Furthermore, this waveform injection (along with all other packet processing) requires an external microprocessor. So, FLEW needs at least 2 chips. (In fact, all evaluations in FLEW use 3 chips.)

In this paper, we present a novel CTC solution, called `Unify`, which transforms modern BLE/FSK SoCs into fully operational WiFi SoCs **without any hardware modification**. `Unify` transmits and receives conventional WiFi packets. Because `Unify` behaves just like an off-the-shelf WiFi device, it can directly connect to unmodified, conventional WiFi APs. Unlike FLEW that relies on multiple chips, `Unify` is a **single-chip solution**, and hence is smaller, cheaper and more power-efficient.

`Unify` can directly run on very popular and widely-deployed BLE/FSK SoCs (CC254x) made by Texas Instruments (TI). For example, the CC2544 SoC is widely used in Logitech's Unifying dongles. Fig. 1a shows that `Unify` turns a Unifying mouse dongle into a proper WiFi dongle, which provides direct WiFi connection for the entire laptop. In another example, the CC2541 SoC is widely used in numerous highly popular BLE modules, such as the HM-10, HM-11, JDY-06 and JDY-08 modules. In Fig. 1b, `Unify` is shown to transform a CC2541 keyfob into a WiFi thermostat, which periodically uploads temperature readings directly to the Internet/cloud.

Enabling bi-directional Bluetooth–WiFi communication without any hardware modification is a significant advantage of `Unify`. Specifically, hardware modifications require either manufacturing brand-new devices, or retrieving and modifying the circuit board of the device already deployed in the field. In contrast, `Unify` only needs a firmware update and the circuit board need not be modified, which greatly simplifies and reduces the deployment difficulty. More importantly, some CC254x SoCs are capable of over-the-air (OTA) firmware upgrade [7]. That is, this firmware update process could be done wirelessly and even remotely, without ever physically accessing the device.

The single-chip operation of `Unify` allows for significantly smaller, cheaper, and more power-efficient devices, even when compared to FLEW. For example, CC2544 uses the QFN32 [8] package with a dimension of 5mm×5mm, which is already smaller than the FSK transceiver chip (QFN48 [9]) used in FLEW. FLEW also requires an additional microprocessor (LQFP80 [10], 12mm×12mm) for a complete operation. Also, CC2544 is as cheap as $1.2 USD [11] and the HM-11 module is available for less than $2 USD. Both are considerably cheaper than the FSK chip used in FLEW. Finally, as we will show in Sec. 4.6, Unify consumes much less power than not only standard WiFi chips but also FLEW.

Unify is groundbreaking in that *WiFi connectivity can be achieved with the same device complexity/cost and with the same power budget as BLE connectivity*. We achieve this vision with three key technical concepts: a) streaming DAC IQ samples with an innovative use of DMA, b) capturing FSK signals with SPI, and c) satisfying WiFi timings with power overrides. These concepts are directly applicable to the CC254x SoC. In addition, outputting FSK signals and using power overrides are common on BLE SoCs for the purpose of BLE certification. Therefore, these techniques can be easily applied to other BLE SoCs if the register map of the SoC is known. Some vendors consider the register maps proprietary information and thus the configuration procedure is not publicly available. However, the vendors themselves or trusted developers should have access and can thus apply the techniques. Unify transmits WiFi waveforms with an innovative DMA scheduling. This concept can be generalized as using the DMA to control the phase of the carrier on BLE SoCs. For example, we can apply our concept to other BLE SoCs by using and scheduling the DMA to efficiently control the radio configuration, such as the PLL or the AFE (analog front end) settings.

We develop Unify entirely from publicly available information and we do not use any proprietary information.

2 SYSTEM DESIGN

2.1 SoC: Challenges and Opportunities

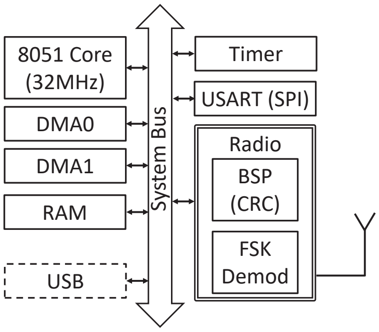


Figure 2: Simplified block diagram of CC254x SoC.

There are numerous technical challenges in transforming BLE/FSK SoCs into WiFi SoCs. Unlike the highly configurable FSK transceiver chip used in FLEW, BLE/FSK SoCs abstract away low-level configurations and have less flexibility in customizing radio protocols. For example, CC254x does not have the infinite receive mode used in FLEW. Moreover, since SoCs integrate everything on a single chip, the connections between a processor and the radio circuits are internal, thus preventing use of certain techniques that are applicable only when the processor and the radio circuits are two separate chips. For example, FLEW implements WiFi transmission by using an external processor to inject digital waveforms into the analog pins on the FSK transceiver. Such a transmission design is not feasible on CC254x.

On the other hand, the SoC includes a processor core, a memory bus, and a number of peripherals. Fig. 2 shows the components (utilized by Unify) on a CC254x chip. By leveraging the different functions provided by the peripherals and because of the much faster access to the radio or peripheral registers, these resources create a unique opportunity for Unify to overcome numerous hardware challenges and perform standard WiFi operations on SoCs without any hardware modification.

2.2 Design Goal

The ultimate goal of Unify is to make BLE/FSK SoCs behave like standard WiFi chips. To achieve this goal, the SoC must **transmit** and **receive** standard, un-coded WiFi packets. We take a bottom-up approach to designing both the WiFi transmission (Sec. 2.3) and the WiFi reception (Sec. 2.4). We first design transmission/reception of the physical waveform of a single WiFi bit. We then design robust mechanisms for transmitting and receiving complete packets. Sec. 2.5 presents an innovative way of drastically reducing the turnaround times between transmission and reception, which is critical for meeting the WiFi timing requirements. Finally, Sec. 2.6 puts all components together and schedules the Tx and the Rx to transform BLE/FSK SoCs into WiFi SoCs.

2.3 Transmitting WiFi Packets

To transmit a WiFi packet, Unify divides its physical waveform into segments, each of which is copied to the DAC registers for radio transmission. The segment copying is done by precise, robust scheduling of two DMA controllers.

2.3.1 DAC Registers. Similar to earlier TI transceivers, CC254x chips have registers for controlling transmission DACs. When set appropriately, these registers allow the transmitting IQ samples to be overridden by software. Although these registers are not documented for CC254x chips, we were able to pinpoint them among the undocumented memory locations and verify that the IQ overrides function correctly. Specifically, the IQ overrides are activated by loading a value of 41 to location 0x61AA (in the 8051 XDATA region), and the actual I and Q overrides are located at 0x61A7 and 0x61A8. We also find that the DACs do not seem to be clock-gated. That is, any I/Q overrides are directly reflected in the transmitted waveforms after memory stores, and no synchronization between the DAC and the memory bus is required. Conceptually, DACs are always synchronized with memory bus.

For CC254x, these DAC registers, unlike those in older TI transceivers, provide a unique opportunity to transmit WiFi waveforms. Specifically, since CC254x are SoCs, DAC registers can be updated at a high frequency. In contrast, older TI transceivers are not SoCs and register writes require using much slower inter-chip communications, such as an SPI bus, and thus the register updates on older transceivers are too slow to be useful for WiFi waveforms.

To transmit proper WiFi waveforms, the DAC registers must be updated fast enough. Moreover, to reliably transmit WiFi packets from CC254x to WiFi devices every time, these register updates should have precise, predictable timings and any glitch should be eliminated. Transmitting one WiFi packet involves thousands of consecutive writes to the DAC register. Missing or duplicating one memory write results in the shifting of all subsequent memory writes (and therefore, the transmitted WiFi waveforms), which will cause bit misalignment between CC254x and WiFi devices, resulting in packet errors.

A straightforward way to update the DAC registers is to use the 8051 on the SoC. The 8051 CPU on CC254x is a TI-enhanced variant with a typical 8051 memory layout (RAM, SFR, XDATA, etc.) and reduced clock cycles for the instructions (as opposed to 12 cycles of the original 8051 [12]). The CPU runs at 32MHz. We have extensively explored use of the 8051 CPU for writing DAC registers that transmit WiFi waveforms. Although we find it possible to transmit a WiFi packet by our highly-optimized assembly code and with the CPU set to the real-time mode, glitches are observed and a less precise WiFi waveform needs to be used because of longer cycles of accessing the XDATA memory. So, we conclude that using the CPU does not meet the speed and reliable timing requirements for transmitting WiFi packets.

2.3.2 DMA Controllers. Fortunately, CC254x is equipped with DMA controllers. Conceptually, a DMA controller is hardware that implements `memcpy()`. Therefore, a WiFi IQ waveform can first be stored in memory and then a DMA controller copies (and therefore transmits) the IQ waveform to the DAC registers. With this method, no CPU intervention is needed throughout the transmission of WiFi packets. Also, since the memory reads and writes are implemented in hardware, using the DMA controller enables much faster DAC register updates, and thus the transmission of precise WiFi waveforms. Moreover, the memory access priority of the DMA controllers on CC254x is configurable. With the priority set to high, the DMA memory access always gets priority over the CPU access, thus guaranteeing precise, glitch-free timing. We find that the DMA controllers actually have a hidden level ("absolute") with an even higher priority and use this level for copying IQ samples to DAC registers.

Similar to FLEW, `Unify` transmits DSSS BPSK WiFi waveforms, which is the most robust WiFi modulation format. The DSSS BPSK modulation can be viewed as typical BPSK modulations at a higher speed, which can be implemented by updating I samples only and keeping the Q samples to midpoint (i.e., Q-branch is always 0 on the constellation plane). While it is possible to use a single DMA controller to update both I and Q samples, we choose to implement BPSK by only updating the I samples. This is because the I and Q samples are in separate registers and updating both samples reduces the update rate by half.

By only modulating the I-branch, the spectrum of the transmitted waveform is symmetric. The standard DSSS WiFi waveform can be generated by modulating the I-branch at 11MHz. This corresponds to the 11-bit Barker sequence the WiFi DSSS uses. That is, after scrambling and differential coding, a WiFi packet is encoded in an 1Mbps bitstream. For any 1's in the bitstream, the Barker sequence [1,-1,1,1,-1,1,1,1,-1,-1,-1] is sent to the I-branch at 11Mbps. For any 0's, its complement ([-1,1,-1,-1,1,-1,-1,-1,1,1,1]) is used.

In `Unify`, the DMA updates the I samples at 8MSps. Therefore, a sequence, resampled in the phase domain from the Barker sequence, is used. Specifically, for 1's in the 1Mbps bitstream, [1,-1,1,-1,1,1,-1,-1] is sequentially sent to the I registers using DMA. For 0's, [-1,1,-1,1,-1,-1,1,1] is used. Let the maximum of the autocorrelation of the Barker sequence be 100%, then the maximum of the cross-correlation between the Barker sequence and the resampled sequence (when both are upsampled to 88MHz) is 81%, indicating that they are highly similar. Technically, the resampled sequence has a smaller bandwidth (modulating at 8MHz instead of at 11MHz). However, since conventional WiFi systems are designed to withstand interference from nearby channels and since the high frequency components near channel boundaries are already attenuated either by the circuits or the channel filter in practice, this resampled sequence works properly and reliably with chips from all major WiFi vendors.

In theory, WiFi packets can be transmitted by updating the DAC registers using a single DMA transfer. However, this is infeasible in practice due to memory and processing constraints. CC254x only has 2~8kB of SRAM. In `Unify`, each WiFi bit is represented by 8 bytes of I data. For sending a 200 byte WiFi packet, a contiguous memory of $200 \cdot 8 \cdot 8 = 12.8$kB is required, which far exceeds the available memory on these low-cost chips. Furthermore, even if an unlimited amount of SRAM is available, generating and putting all the I data in the memory in the first place would take too much time for the 8051 core, resulting in low throughputs.

2.3.3 Multiple DMA Scheduling. To overcome these limitations, we must reuse the memory regions, meaning that we must configure and trigger multiple DMA transfers during the transmission of a single WiFi packet. Conceptually, each DMA transfer sends a small segment of the WiFi packet and the whole WiFi packet is transmitted by concatenating all DMA-transferred segments.

Using multiple DMA transfers and concatenating them introduce new challenges. Instead of configuring and then triggering one DMA transfer, we need to configure multiple DMA transfers on the fly, and each transfer should be triggered precisely on time. If any DMA transfer is not triggered precisely at the scheduled instant, the waveform segment sent during the DMA transfer is shifted and corrupted. If a DMA transfer is not correctly configured before its triggering, an incorrect waveform segment will be sent.

We address these challenges by devising a robust mechanism that schedules the DMA transfers without any CPU involvement during transmission. The scheduling is very tricky to design but works elegantly and reliably in `Unify` as described below.

Our solution uses a timer and two DMA controllers on CC254x. One DMA controller (DMA0) periodically copies a waveform segment from a memory region to the DAC register. Since waveform segments must be copied to the DAC register starting at precise time instants, each transfer issued by DMA0 must start at a precise time instant. This is guaranteed by triggering DMA0 in hardware using a timer on CC254x. Note that a waveform segment does not have to contain the waveform of just one WiFi bit. In fact, `Unify` triggers DMA0 every 4μs, corresponding to the duration of 4 WiFi bits. The basic idea is that, depending on the WiFi bits to transmit, DMA0 will copy a waveform segment from different memory locations (by modifying the DMA0 configuration). The memory region is pre-loaded with all possible waveform segments corresponding to all bit permutations within the duration of a waveform segment.

After a segment (corresponding to a group of WiFi bits) is transferred via DMA0, the configuration of DMA0 must be updated to transmit the next segment. Specifically, the source memory address of DMA0 must be updated to point to the starting location of the next waveform segment. On CC254x, DMA is configured by specifying a DMA descriptor. The DMA descriptor itself is actually 8 bytes of memory in the XDATA space. Another key idea is that, since the descriptor of DMA0 is also a memory region, we can use a second DMA (DMA1) to keep updating the source memory addresses of DMA0. Additionally, because DMA1 can be hardware-triggered upon completion of each DMA0 transfer, the address update happens exactly once after each DMA0 transfer with guaranteed ordering. While the 8051 core, instead of DMA1, could theoretically be used for updating addresses, we find the timing is not reliable using the 8051. Also, the DMA0 triggering is too fast to allow interrupt-based designs for updating the descriptor.

Since DMA0 transfers are the most time-critical, the priority of DMA0 is set to“absolute” while the priority of DMA1 is “high”. DMA0 is also set to repeated block mode where DMA0 transfers a block of data once triggered, and after each transfer, DMA0 automatically reads the new descriptor and waits for the next trigger.

A small turnaround time is required between DMA0 transfers. (This turnaround time is still present if, alternatively, DMA0 and DMA1 are interleaved to update the DAC register.) Therefore, although 4μs corresponds to 32 memory accesses, the block size of DMA0 is set to 28. During the remaining 4 cycles, the DAC register is not updated and the waveform stays unchanged. To account for this, DMA0 transfers are not aligned with the bit boundaries of WiFi bitstreams. Each transfer spans 5 WiFi bits where the first cycle in each transfer copies the last I sample of the first WiFi bit, and the last 3 cycles copy the first 3 I samples of the last WiFi bit. That is, the DMA0 transfer is repeated every 4 WiFi bits and the waveform for one of these 4 bits is transmitted by the last 3 cycles of a transfer and the first 1 cycle of the next transfer. The 4 cycles of idle time in between are used for turnaround.

2.3.4 Waveform Segments. DMA0 copies I samples from a memory region in which every possible waveform segment within 4μs can be found. A naive approach to storing all these segments is to store each segment separately in a non-overlapping manner. Since every possible 5 WiFi bit permutation needs an entry, the overall required memory is $2^5 \cdot 28 = 896$ bytes. While this memory size is within the limits of CC254x chips, it is still relatively large. More importantly, since a memory region of 896 bytes cannot be indexed using a single byte, DMA1 needs to update the source address (in the DMA0 descriptor) in two bytes after each DMA0 transfer and each address waiting to be copied to DMA0 descriptor by DMA1 needs to be stored in two bytes of the main memory.

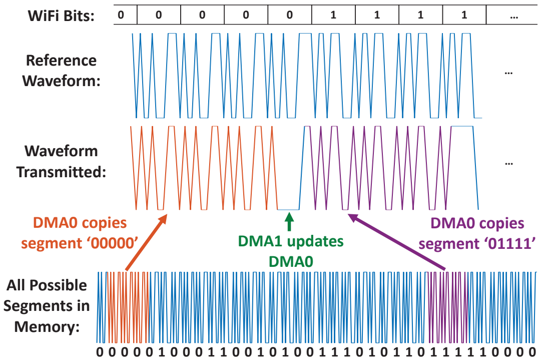


Figure 3: Transmit waveforms using 2 DMAs.

A better approach is to store all segments in the memory with maximum overlaps between segments, resulting in the minimum required space. This approach makes use of the de Bruijn sequence [13]. Specifically, every 5-bit permutation is a subsequence of "000001000110010100111010110111110000" and we can store all possible segments in this manner. Each '0' or '1' in this sequence represents one WiFi bit, which actually occupies 8 bytes (representing 8 I samples) in the memory. (That is, [1,-1,1,-1,1,1,-1,-1] for '1' and [-1,1,-1,1,-1,-1,1,1] for '0'.) With these stored segments, we can transmit a WiFi bitstream by setting, using DMA1, a series of appropriate offsets as the source address of DMA0. Let us consider Fig. 3 as an example and suppose the last WiFi bit in the last iteration is '0' and we want to transmit WiFi bits '0000', then we need to specify the offset for '00000', which is 0. However, since we only transmit the last I sample (i.e., skipping the first 7 samples) of the first WiFi bit, the offset is 7. Suppose the WiFi bits after '0000' is '1111', we need to specify the offset for '01111', which is $7 + 26 \cdot 8 = 215$. Since the maximum offset is $7 + 31 \cdot 8 = 255$, each offset can be represented with just one byte. Therefore, DMA1 only needs to update a single byte of the DMA0 descriptor each time and each offset waiting to be copied by DMA1 occupies only 1 byte in the main memory.

The ability to use a single byte to store offsets is also the reason why DMA0 is triggered every $4\mu s$, since $4\mu s$ is the longest interval that all possible waveform segments for every bit permutation can still be addressed by a single byte.

2.4 Receiving WiFi Packets

To receive standard, un-coded WiFi packets, Unify uses the radio circuit and the FSK demodulator to receive raw WiFi bits. Unify also reuses the SFD detector on CC254x, which is intended to detect FSK/BLE packets, to detect standard WiFi packets. Upon detection of a WiFi preamble, Unify starts collecting raw WiFi bits and descrambling the incoming bit stream to recover each byte in the WiFi packet. CRC checks are performed on every WiFi packet, which is critical for standard WiFi operations. All this processing needs to be done in real time on CC254x, which is only equipped with a low-power, 8-bit 8051. Although receiving standard WiFi packets is usually assumed to be computationally-intensive, Unify achieves this goal by leveraging recent signal processing discoveries, by reusing certain hardware accelerators on CC254x and by optimizing the codes run on 8051.

2.4.1 Receive a raw WiFi bit. To receive a raw WiFi bit, Unify leverages a simple and known technique used in the latest WiFi–FSK CTC works [4–6]: FSK demodulators, operated at a relative frequency offset, can receive WiFi DSSS bits. Unify operates the FSK receivers at -3MHz (relative to the center frequency of the selected WiFi channel) with an intermediate frequency of -1MHz. The frequency offset is chosen based on the receive performance. Although the latest WiFi–FSK CTC results show that a fractional frequency offset should ideally be used, the frequency can only be set in 1MHz increments/decrements on CC254x. To address this, the frequency offset compensation feature on CC254x is used, which provides additional fractional frequency offsets based on the received waveforms. A similar technique can be found in [4]. Unify configures CC254x to continuously track and compensate for any frequency offset before an SFD is detected. Once an SFD is received, the frequency offset is frozen and this estimate is used to demodulate the rest of a packet. This setting yields the best performance among all different frequency compensation configurations.

Although the above techniques are useful in demodulating raw WiFi bits, they do not enable CC254x to directly receive WiFi packets. In particular, since the bit decoding and packet logic on CC254x are designed for BLE/FSK packets, they cannot be directly used for conventional WiFi packets. The prior work of [4], which enables one-way WiFi-to-BLE communication, circumvents these limitations by pre-processing and pre-coding WiFi packets in the form of BLE packets. However, this method would require modifications on every WiFi device from which the BLE chip receives packets.

We design Unify to directly detect and decode standard WiFi packets. This eliminates the need for pre-coding WiFi packets so that Unify can work with conventional WiFi devices. To achieve this goal, we must address other challenges imposed by the hardware.

2.4.2 Packet Detection. To detect WiFi packets using FSK/BLE packet logics, CC254x is configured to detect the standard WiFi preamble from the received raw WiFi bitstream. Although detecting WiFi preambles can, in theory, be implemented in software using sliding windows, it is computationally prohibitive, especially on an 8-bit 32MHz CPU.

Unify exploits the BLE/FSK SFD matching hardware on CC254x for the detection of WiFi packets. The SFD to be matched is set to a bit sequence within the scrambled WiFi SYNC field. This technique can be found in FLEW. However, Unify uses different SFD and SFD length because of the frequency compensation requirements. The SFD used in FLEW is 0x05AE4701. In standard WiFi packets, the 8-bit pattern before 0x05AE4701 is '11010101'. On the other hand, CC254x uses the SFD, along with the 8 bits received before SFD, to estimate frequency offsets. For CC254x, the 8-bit pattern should be either '01010101' or '10101010', and receiving '11010101' lowers the accuracy of the frequency estimates.

We meet this requirement by using 0x0B5C8E03 as the SFD, since the WiFi bit pattern is '10101010' before 0x0B5C8E03. CC254x is configured to expect this bit pattern before SFD. 0x05AE4701 is 0x0B5C8E03 shifted right by 1 bit. Therefore, the length of SFD is set to 31 so as to align the subsequent bytes to the appropriate byte boundaries.

With appropriate bit descrambling, the WiFi PLCP starts 5 bytes (40 bits) after the SFD (0x05AE4701). This holds for packets conforming to the 802.11 standard [14]. As described in [6], Realtek’s chips are buggy in that their PLCP (and subsequent bytes) are 2 bits early (38 bits after bit descrambling). For connecting to Realtek devices, `Unify` sets the length of SFD to 29 in order to automatically align the bytes without any software processing.

2.4.3 Packet Length. The length of a WiFi packet can be determined after parsing the PLCP header. `Unify` also performs sanity checks on the PLCP to prevent erroneous or unsupported PLCP affecting the packet reception.

The hardware limitations of CC254x make it very difficult to decode standard WiFi packets, especially due to the packet-length limitation. Since we are not pre-coding WiFi packets with BLE packet format, we cannot use CC254x’s packet-length parsing, which determines the packet length and adjusts the reception duration accordingly. If the length parsing is not used, we need to use the fixed length packet format, which requires specifying the length before starting each packet reception. This is infeasible because the WiFi packet lengths are variable and we do not know the length of each packet until its PLCP is received.

In addition, the packet logic (with or without length parsing) only supports packets up to 255 bytes and CC254x immediately terminates the reception afterwards. Conventional WiFi packets easily exceed this size.

We use a special design that solves the size limitation and dynamically adjusts the reception duration. The key idea is to prevent the packet logic from terminating the reception before a complete WiFi packet is received. From extensive experiments, we find this achievable by forcing the packet logic “stuck” at its initial state. Specifically, we find that after an SFD is received, we can arbitrarily prolong the reception by periodically overwriting one register in an undocumented RAM region the packet logic uses. The register is mapped to 0x607E in the XDATA space when the RF memory page is set to 5 (RFRAMCFG = 5). By constantly overwriting 0 to 0x607E, it prevents the packet logic from terminating the reception. Furthermore, by controlling the total duration of overwriting 0, the reception duration of a WiFi packet can be adjusted dynamically after its PLCP is received.

One side effect of overwriting the packet logic’s register is that the packet buffer goes haywire. Consequently, the raw WiFi bits cannot be read out from the packet buffer. To address this problem, we utilize CC254x’s RF observation signals, which are various signals from the FSK demodulator. By selecting “demodulated bits”, “clock” and “SFD matched”, these signals emulate an SPI sender. We can then use an SPI receiver (USART hardware on CC254x) to collect the demodulated bits (the raw WiFi bit stream), as shown in Fig. 4.

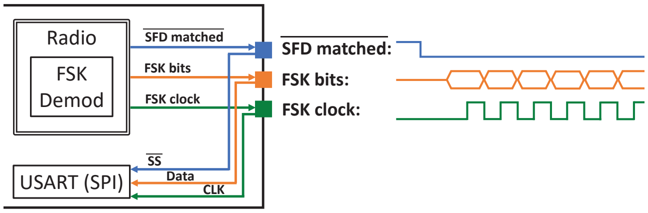


Figure 4: Receiving FSK bits with SPI.

While it might seem that external wires are needed to connect the SPI sender to the SPI receiver, `Unify` does not require such external connections. We find that the peripheral inputs are (implicitly) always connected to the pins, even when those pins are configured as outputs. Therefore, by using the IO matrix to route the RF observation and SPI receiver signals to the same pins, they are connected without any external connections.

2.4.4 Packet Decoding & CRC. After raw WiFi bits are received from SPI, `Unify` descrambles the bit stream. `Unify` performs descrambling immediately upon receiving each byte. The WiFi descrambling can be simplified as performing XOR on 3 shifted versions of the bit stream. We optimize the descrambling as 8-bit computations (as opposed to shifting and XOR’ing three 16-bit integers). We devise the following C code to produce each WiFi byte (`output`):

```
sb = <one byte received from SPI, LSB first>;
output = lb ^ (lb>>7) ^ (lb>>3) ^ (sb<<1) ^ (sb<<5);
lb = sb.
```

Further optimization is possible. Unlike ARM’s shift instructions, rotate left/right on 8051 only shifts 1 bit at a time and extra cycles are required to mask bit-rotations to get logical shifts. Consequently, the second line in the above snippet takes approximately 31 cycles (depending on the register allocation) using the IAR compiler. This line can be replaced by an optimized assembly routine. The key idea is to construct two lookup tables in the 8051 CODE space to store the results for `lb ^ (lb>>7) ^ (lb>>3)` and `(sb<<1) ^ (sb<<5)`. During descrambling, we can load `lb` (or `sb`) into `A` and use the `MOVC` instruction to directly fetch the results. The number of cycles is reduced to 17 using this method. We process the WiFi PSDU (excluding the CRC) with this optimization.

CRC must be performed on WiFi packets, since the CRC result is used to determine whether an ACK should be transmitted after 10 μs. However, CRC32 is simply too computationally intensive to run on the 8-bit 8051 at WiFi speed. Fortunately, CC254x has a configurable hardware CRC module (known as the BSP co-processor). We devise an appropriate initialization sequence for the CRC module. Specifically, the CRC32 polynomial is configured once at startup. After the SFD (0x0B5C8E03) is received *and just before the start of WiFi PLCP*, `Unify` initializes the CRC32 shift register (to 0xFFFFFFFF). Then, for each byte (excluding the last 4 bytes) in the WiFi PSDU, `Unify` sends the descrambled byte to the BSP. The 4 bytes received last are matched with the CRC calculated by the BSP. An ACK should be transmitted if the CRC matches.

The CC254x has a register called `FREQTUNE`. Contrary to its name, we find that it has little effect on the RF frequency. Instead, it fine-tunes the crystal oscillator. We find this register particularly useful in fine-tuning CC254x to receive long WiFi packets, since it compensates for the small timing offsets between CC254x and a WiFi device. This fine-tuning is needed only when large packets are expected and it only needs to run once for each WiFi device. We iterate 5 possible values and choose the best `FREQTUNE`.

2.5 Transitions between Tx and Rx

2.5.1 Rx to Tx Turnaround. Normal WiFi operation requires a device to respond (i.e., with ACK or CTS packets) almost instantly (10μs) after receiving packets. Meeting this WiFi turnaround time is highly challenging for `Unify`, since CC254x is not designed with such a short turnaround time. In particular, we find that the shortest turnaround time of CC254x, using the standard Rx to Tx transition, is around 130μs. This is far too long for any WiFi devices.

After extensive experiments, we fundamentally resolved this challenge. Our solution starts from a key insight that, since `Unify` transmits waveforms by directly controlling the DAC registers, we can initiate a transmission with minimum delay by directly turning on the PA, mixer and DAC. The power signals of these components can be overridden by modifying the "power down" registers. Although the locations of these registers on CC254x are completely undocumented, we are able to pinpoint the two registers after extensive experiments. Specifically, the transmit chain is turned on if 0x61AC (in the XDATA space) is set to 8.

By directly controlling the power signals, the Rx to Tx turnaround is reduced to approximately 10μs, thus satisfying the timing requirement of transmitting an ACK (or CTS) after receiving the corresponding packet. Unlike FLEW, `Unify` meets this timing requirement without truncating packet reception, and thus full 4 bytes of CRC32 are received, just like any conventional WiFi chip. We observe that terminating packet reception affects the RF carrier signal, and hence do not terminate the reception until an ACK is transmitted. That is, the reception is prolonged to receive the full WiFi packet, plus 10μs, plus the duration of an ACK.

2.5.2 Tx to Rx Turnaround. In the opposite direction, after `Unify` sends a packet to an AP, the AP will send an ACK after 10μs if the AP properly receives the packet. While it is technically possible that a WiFi system is functional without ACK detection, such a design would have very poor performance because a reliable re-transmission mechanism cannot be implemented without detecting ACKs. However, 10μs is also a far too short turnaround time for CC254x.

To address this challenge, we use a similar idea, except we change the order of Tx and Rx. That is, to send a packet, `Unify` initializes reception, sets the Tx frequency, and then immediately overwrites the power registers (which block the signal from going into the receive chain). In addition to loading 8 to 0x61AC, 32 is loaded to 0x61AB to reduce power consumption. Once the transmission is completed, DMA1 becomes inactive. After detecting this, the 8051 cancels DMA0 transfers and resumes the highest access priority. Afterwards, it sets the Rx frequency, sets both 0x61AB and 0x61AC to 0, and waits for a packet (which should be an ACK). The key idea is that by the time `Unify` finishes transmission, the receive circuit will be primed and ready for the next packet. All we need to do is "unblock" the receive signal by removing the power overrides, and a fast Tx to Rx transition is achieved. The timing of this control logic is not very tight since 10μs leaves enough room for the 32MHz 8051. (The main source of delay was the receive circuit, not the 8051.)

The Tx to Rx turnaround can be followed by an Rx to Tx turnaround. For example, if a unicast packet (instead of an ACK) is received after sending a packet, `Unify` will receive the unicast packet and transmit an ACK (after ∼10μs).

2.6 MAC Layer

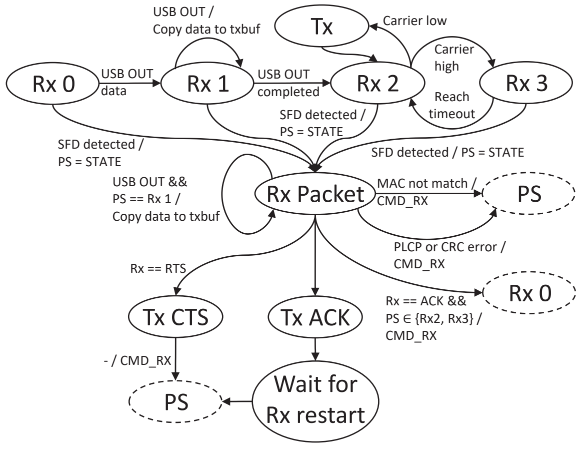


Figure 5: The FSM of `Unify`.

The finite state machine (FSM) of `Unify` consolidates all important components, schedules transmission and reception in a half-duplex manner, and implements the MAC-layer functions of WiFi.

Fig. 5 shows the high-level FSM of `Unify`. After initialization, a `CMD_RX` is issued and `Unify` enters the `Rx 0` state. In

Rx 0~3, the hardware is in receiving mode and waiting for a valid SFD. These states differ only in software. From Rx 0 to Rx 2, Unify copies and prepares the WiFi data until a complete WiFi packet (in the form of offsets) is ready to be transmitted in the memory. State Rx 3 represents that the packet is ready but the channel is busy.

Unify follows CSMA/CA, which is the fundamental medium access mechanism in WiFi. Before each packet transmission, the Rx 2 state senses the channel by monitoring the RSSI value. If an idle channel is observed, Unify transmits a packet. After Tx, Unify returns to Rx 2. If the connecting AP sends an ACK, Rx 2 can detect the SFD and transition to Rx Packet. Otherwise, Unify initiates re-transmissions.

Rx Packet is the main state that collects and processes (descrambling and CRC) the bitstream and extends the reception to a complete packet. Rx Packet provides information (the frame type, address and CRC results) so that necessary actions are performed thereafter. For packets with a wrong MAC address or other errors, Unify simply returns to the last state ("PS") before Rx Packet. If an ACK is received after packet transmission, Unify goes to Rx 0 to prepare the next packet for transmission. CTS is transmitted if RTS is received. If a unicast packet with a matched MAC address is received, Unify transmit an ACK. Then, Unify does *not* issue a new CMD_RX because each CMD_RX is configured as repeated with fast warm-up. Since Rx Packet to Tx ACK is the normal reception flow, the receiver will become ready faster by simply waiting for the repeated reception to start.

We also include an optimization in Rx Packet. When Unify is processing the incoming WiFi bit stream from the radio, the 8051 also copies the outgoing WiFi bytes from the USB buffer to the memory. This allows Unify to go to Rx 2 much faster after reception.

3 IMPLEMENTATION

3.1 Hardware

To directly compare Unify with SOTA CTC, we implement Unify on the CC2544 since it has built-in USB. We use Logitech Unifying dongles as the hardware. We do not modify the hardware of Logitech Unifying dongles. We simply load the Unifying dongle with Unify firmware and WiFi connectivity is achieved.

3.2 Firmware

The firmware of Unify is developed from scratch. This includes in-house USB firmware codes that are completely developed from the ground up. The firmware is compiled with IAR EW8051 [15].

After USB initialization and exchanging USB standard requests, the firmware initializes the radio hardware. The 8-bit Timer 3 is initialized to provide a constant 4μs trigger. The radio is set to the fixed length mode (with a length of 120) and repeated reception with the "synthesizer on" option. The actual reception duration will be dynamically adjusted at run time. The AGC is off and the radio is manually set to the maximum gain. The firmware initializes 1 DMA0 and 3 DMA1 descriptors. Three descriptors are for transmitting a normal packet, an ACK and an CTS, respectively. This allows Unify to switch between different transmissions by simply specifying a different descriptor for DMA1.

The firmware also initializes an XDATA region that stores all possible waveform segments to be sent by DMA0. Using the IAR compiler, this XDATA region always starts at 0x0001. A small optimization is used here. In Sec. 2.3, the lowest possible offset (index) is 7. Therefore, we generate all possible waveform segments, truncate the very first 7 bytes and store them in XDATA starting at 0x0001. All offsets are adjusted (i.e., subtracting 6) accordingly.

After initializations, the firmware enters the FSM loop. If any USB error (e.g., USB reset) is detected, the firmware jumps to 0x0000 and the USB initialization follows.

3.3 Firmware Update

The Unify firmware can always be loaded to CC254x using a CC254x programmer. The official CC Debugger [16] from TI can be used. Alternatively, an Arduino board can be used as the debugger [17]. We also design the Unify firmware so that Unifying dongles with a compatible bootloader can be directly updated via USB. Specifically, the bootloader on the Unifying dongles in our possession occupies 0x0000~0x03FF and 0x7400~0x7EBF (in the CODE space). A compatible Unify firmware is first generated by placing the Unify codes starting at 0x052C. Then, all instructions between 0x0400~0x052B are replaced by jumps to 0x056D, which is the entry point to the Unify codes. CRC16 is calculated over 0x0400~0x6BF9 and is placed at 0x6BFA. Finally, CRC16 is followed by a magic string (0xFE,0xC0,0xAD,0xDE). With this code layout, Unify firmware can be flashed to compatible dongles using USB flashing tools for Unifying dongles. Finally, our firmware is also designed to be able to revert back to the original Unifying firmware.

The Unify firmware focuses on WiFi communication. TI developed a "Boot Image Manager", which allows running multiple firmware [18]. Alternatively, BLE functions can later be added to our firmware. Therefore, BLE and WiFi communication can both be supported with a single SoC.

3.4 WiFi Driver

The WiFi driver of Unify is similar to FLEW and interfaces the firmware with the mac80211 module in the Linux kernel. Similar to FLEW, the transmission path converts the WiFi packets to WiFi (phase) bit streams. Then, every 4 bits (and the one bit that precedes them) are converted, using a simple lookup table, to an offset that points to the corresponding waveform segment. All offsets are sent to the firmware for transmission. In the receiving path, the firmware sends descrambled WiFi packets, along with two status bytes for each packet, to the driver. The status bytes include a CRC flag indicating if there is a packet error. The driver simply checks this flag and either relays a packet to `mac80211` or ignores an error packet. In the driver, we set the MTU to avoid overflowing CC2544's memory and to limit the sampling offsets. For these purposes, we can use an MTU of 256, which is the minimum MTU setting on the modern Linux. However, we use a higher MTU (552) because it increases throughputs and 552 was the minimum MTU of Linux for a long time [19–22].

4 EVALUATION

4.1 Experimental Setup

Table 1 shows the WiFi devices and APs used in the evaluation of `Unify`. The experimental setup is similar to that of FLEW. Various NICs from major WiFi chip makers are used for measuring physical-layer performance. A difference between our setup and that of FLEW is that all NICs use exactly the same antennas (on an HP 2570p laptop), making their performance directly comparable. Marvell does not seem to manufacture any standard half-length mini PCIe card and its chip does not support monitor mode well. Therefore, the evaluation is done at the system level. Various commodity WiFi APs are used for evaluating system-level performance. These system evaluations represent real-world use-cases where `Unify`, just like any typical WiFi device, directly connects to conventional WiFi APs. They characterize end-to-end performance where all aspects of WiFi protocol (e.g., re-transmissions, CSMA/CA, etc.) are taken into account.

All WiFi devices and APs are unmodified and all NICs use their default driver supplied with Ubuntu 20.04 LTS. We use WiFi channel 9 and the system evaluations (Sec. 4.3∼4.5) use the WPA2-PSK encryption. This WiFi channel is chosen because it has the lowest activity in the test environment.

Table 1: 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)
Intel	Advanced-N 6205	TP-Link Archer AX3000 (WAV654A0)
Marvell	-	Linksys EA3500 (88W8366)
Ralink/ Mediatek	RT3290	TP-Link TL-WR841N (MT7628NN)
Realtek	RTL8188CE	D-Link DIR-619L (RTL8192ER)

4.2 PHY Layer and PER

Table 2: PER Evaluation (%)

Direction	WiFi to `Unify`			`Unify` to WiFi		
Distance	5m	10m	20m	5m	10m	20m
Atheros	0.44	0.73	2.64	2.47	3.83	4.00
Broadcom	5.25	6.30	6.40	2.88	3.10	3.81
Intel	5.52	8.35	9.74	1.78	3.37	3.74
Ralink	5.27	7.52	10.91	5.71	7.89	9.01
Realtek	1.25	3.08	5.40	12.82	13.21	13.26

We set `Unify` and NICs to the monitor mode and continuously send/receive standard WiFi packets with a PSDU of 564 bytes (including 4 bytes of CRC). In each setting, 4096 packets are sent so that each packet has a unique sequence number. On the receiver side, CRC checks are performed. We calculate the number of received packets with a correct CRC and the PER is defined as $1 - \frac{\text{\# of correct packets}}{4096}$.

Using `Unify` to receive standard WiFi packets transmitted by standard WiFi chips, Table 2 shows that transmission by (Qualcomm) Atheros has the best performance with less than 0.5% PER at 5m. The performance with Broadcom at 5m is similar to that of Intel and Ralink but the PER increases less with longer distances. Intel and Ralink have very similar performance across different distances with Intel being slightly better at 20m. Interestingly, Realtek has the second best PER performance. This is in part due to the fact that `Unify` uses a shorter SFD length (to compensate for Realtek's packet format bug), which distinguishes Realtek's packets from background interferences and yields good performance.

In the opposite direction, `Unify` transmits conventional WiFi packets and standard WiFi NICs receive them. Atheros and Broadcom have similar performance and the PER is ≤4% even at 20m. Intel actually shows the best receive performance in our testing. Ralink and Realtek chips have noticeably worse receive performance. This is consistent with the observations, reported in the prior work, of the inferior receive performance of Ralink and Realtek NICs. We believe these are in part due to the comparatively worse circuit or DSP designs. It is also possible that Realtek uses a non-standard WiFi SYNC field in the Rx chain (as it does in the Tx) and leads to the worst performance. `Unify` uses a shorter SFD length for Realtek tests in both directions, and this increases the possibility of packets transmitted by `Unify` collide with background interference since the shorter SFD length is for receiving Realtek's packets.

These physical-layer evaluations measure one-way, fixed-data-rate and no-retransmission performance. For packet transmission (i.e., WiFi to Unify), WiFi cards may have IQ imbalance, constellation imperfections, or timing or frequency drifts. For packet reception (i.e., Unify to WiFi), different vendors may use different preamble detection, bit demodulation, and RF circuit implementations, which can lead to noticeable performance differences. On the other hand, higher layer issues such as the rate adaptation algorithm and MAC layer implementations will not contribute to the performance differences at the physical layer.

4.3 TCP/UDP Throughput

Table 3: Throughput Evaluation (kbps)

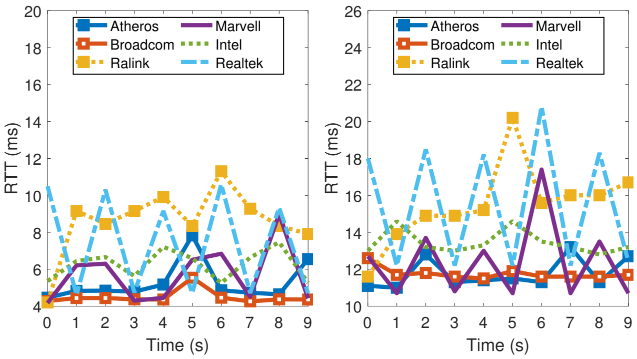
Uplink						
Transport	TCP			UDP		
Distance	5m	10m	20m	5m	10m	20m
Atheros	421	420	412	471	479	459
Broadcom	397	405	391	461	455	444
Intel	390	388	369	446	438	415
Marvell	395	385	310	435	427	419
Ralink	401	395	387	438	441	437
Realtek	405	399	360	458	454	410
Downlink						
Transport	TCP			UDP		
Distance	5m	10m	20m	5m	10m	20m
Atheros	552	530	428	655	652	609
Broadcom	507	491	473	621	597	535
Intel	501	494	474	586	576	544
Marvell	425	424	395	512	520	479
Ralink	469	448	422	571	555	553
Realtek	475	468	451	570	560	527

In the system-level evaluation, Unify directly connects to unmodified WiFi APs. Using the standard iperf3 [23] tool, we evaluate the transport-layer throughputs in both the uplink (Unify to AP) and downlink (AP to Unify) directions. We run an iperf3 server on a Ubuntu laptop, which is connected (via Ethernet) to the first LAN port of the testing AP. Unify joins the AP's WiFi network and runs an iperf3 client to measure the TCP and UDP throughputs in both directions. The measurements are always taken from the receiving end. Besides the MAC layer and packet retransmissions, physical-layer performances in both directions affect the throughputs since data packets are ACKed in the opposite direction. Different rate adaptation algorithms also affect data packets sent by an AP.

For uplink, Table 4.3 shows Atheros has the best performance, which is due to the superior overall (Tx and Rx) physical-layer performance. Broadcom also has good performance and similar to Atheros, the throughputs do not decrease much even at 20m. Intel and Marvell have similar performance but their throughputs tend to decrease more at 20m. The newer Mediatek chip inside the Ralink AP seems to have a better receive performance with little throughput decrease at 20m. Realtek is similar to Ralink at shorter distances but the throughput drops more at 20m. These decreases are caused by the inferior physical layer performance that requires more re-transmissions.

For downlink, Atheros shows the best performance in Table 4.3. Broadcom and Intel have almost identical TCP throughputs but have different UDP throughputs. This could have been caused by different rate adaptation algorithms where the Intel AP tries to use a different data rate more proactively after multiple successful transmissions. Ralink and Realtek have similar throughputs and some throughput drops can be observed at 20m. Marvell has the lowest downlink throughputs, which could be caused by the radio performance and the rate adaptation algorithms. We observe that during rate exploration, the Marvell's AP simply iterates all data rates from high to low, which takes more air time.

4.4 Round-Trip Time



(a) LAN **(b) WAN**

Figure 6: RTT

We connect each AP to the Internet and measure the RTT of Unify over LAN (pinging the AP) and over WAN (pinging 8.8.4.4). The distance between the AP and Unify is 20m.

The wireline RTT between our location and 8.8.4.4 is approximately 6.80ms. This explains that the WAN RTT (Fig. 6b) is similar to the LAN RTT (Fig. 6a) except for a roughly 7ms offset. Broadcom and Atheros show the best performance, followed by Marvell and Intel. Ralink and Realtek have inferior RTT performances. We observe that Realtek RTT tends to oscillate between high and low. This might be due to its rate adaptation design or a bug. Overall, the RTT is 4~10ms for LAN and 11~21ms for WAN, making Unify highly suitable for latency-sensitive applications.

4.5 Coexistence with Multiple Devices

We conducted WiFi coexistence tests, demonstrating Unify's ability to coexist with conventional WiFi devices without "starvation" even when the WiFi channel is fully saturated. We use the Atheros AP and concurrently run multiple iperf3 servers. Unify and different conventional WiFi chips will simultaneously upload or download via TCP or UDP. By default, each UDP link is rate-limited by iperf3 to 1Mbps whereas each TCP link is not.

Table 4: Coexistence with multiple WiFi devices (bps)

# of Devices		TCP UL	UDP UL	TCP DL	UDP DL
1	Unify	432 k	484 k	554 k	654 k
2	Unify	218 k	417 k	193 k	634 k
	Intel 6205	8.07 M	1.05 M	26.3 M	1.07 M
3	Unify	170 k	442 k	118 k	588 k
	Intel 6205	18.2 M	1.05 M	13.3 M	1.05 M
	AR9462	1.88 M	1.05 M	17.6 M	1.00 M
4	Unify	92.3 k	411 k	92.4 k	553 k
	Intel 6205	12.6 M	1.05 M	14.4 M	1.05 M
	AR9462	5.18 M	1.05 M	14.5 M	1.05 M
	BCM4313	2.03 M	1.05 M	5.36 M	1.05 M
5	Unify	67.0 k	423 k	115 k	523 k
	Intel 6205	14.0 M	1.05 M	11.9 M	1.05 M
	AR9462	1.84 M	1.05 M	7.79 M	1.05 M
	BCM4313	1.13 M	1.05 M	6.30 M	1.05 M
	RTL8811AU	13.2 M	1.05 M	3.91 M	1.05 M
6	Unify	55.8 k	330 k	101 k	471 k
	Intel 6205	5.85 M	1.05 M	6.68 M	1.05 M
	AR9462	5.68 M	1.04 M	7.84 M	1.05 M
	BCM4313	554 k	1.04 M	5.95 M	1.05 M
	RTL8811AU	4.61 M	1.05 M	5.69 M	1.05 M
	MT7612U	6.06 M	978 k	3.38 M	1.04 M

For UDP, the channel is less saturated than for TCP and Table 4 shows that the throughput decrease of iperf3 is relatively moderate as the number of active transmissions increases. For TCP, the channel is maximally saturated and each device must contend for the channel and share the overall bandwidth. For TCP uplink, the Unify throughput is roughly halved for 2 devices and is 39% (of the one device throughput) for 3 devices, etc. This validates that the MAC layer is working well and accesses the channel properly. For TCP downlink, the Unify throughput has an initial drop but stays relatively similar thereafter. We believe this is because the downlink arbitration is mostly done by the AP whereas the uplink arbitration is mostly achieved by multiple devices contending for the channel.

Table 5: Throughputs of multiple Unify devices (kbps)

# of Devices		TCP UL	UDP UL	TCP DL	UDP DL
1	Unify #1	432	483	567	684
2	Unify #1	233	297	280	346
	Unify #2	229	296	292	347
3	Unify #1	107	86.7	94.7	242
	Unify #2	123	112	224	171
	Unify #3	90.1	150	118	276

Table 5 shows the results of evaluating the coexistence of multiple Unify nodes. For two Unify nodes with active traffic, the throughputs are roughly halved for TCP or UDP downlink. For uplink, the throughputs are actually more than 50%. This is because when one Unify is copying the uplink packet, the other Unify can use the time to transmit packets. For three nodes, the throughputs are affected more, particularly in the uplink direction, because of the increased possibility of collision. However, this occurs only when all Unify nodes saturate the channel, and even so, no node is starved. Note that throughput variations can be observed even among standard WiFi vendors in Table 4. The bandwidth control, a common feature on WiFi APs, can be used to guarantee minimum throughputs for all devices. This approach should be better than relying purely on MAC layer implementations, which come with lower guarantees and can be slightly different among vendors.

Table 6: Coexistence with background BT traffic (kbps)

	TCP UL	UDP UL	TCP DL	UDP DL
0 BT Devices	431	485	551	672
1 BT Devices	432	484	551	672
2 BT Devices	432	485	547	671

We also tested the coexistence of Unify with Bluetooth devices. Table 6 shows Unify's throughputs when 0, 1 or 2 Bluetooth headphones are simultaneously streaming audio. The results show that Bluetooth traffic has virtually no effect on Unify, because Bluetooth is designed to adaptively avoid active WiFi traffic and Unify is designed to behave exactly like a WiFi device.

4.6 Power Consumption

Table 7: Comparison of power consumption

	Tx (A) (Peak Power)	Rx (A)
AR9271	0.49 (19dBm)	0.07
RTL8811AU	0.32 (≤20dBm)	0.07
RT3072	0.28 (20dBm)	0.13
FLEW	0.16 (~20dBm)	0.11
Unify	0.04 (4dBm)	0.04
Unify (Med. Rx gain)	0.04 (4dBm)	0.03

We measure the power consumption of Unify and compare it with off-the-shelf WiFi cards and prior work. For every device, currents are measured from the USB 5V power line when the device is constantly sending or receiving DSSS WiFi packets. Table 7 shows that Unify has significantly lower power consumption than standard WiFi cards and even FLEW. For a comparison with the same transmit power, we can first compare FLEW and Unify. Specifically, FLEW uses an FSK transceiver (0dBm) and an additional PA (+20dB, 100mA at 3V [24]). Just like FLEW, Unify could use the same PA (+20dB) for higher power. In such a case, the transmit power of Unify would be greater than 20dBm, and the overall power consumption ($0.04 + 0.1 = 0.14$A) is still lower than all other devices after considering the power consumption of the additional PA. (In practice, the PA would draw less than 0.1A at 5V.) Unify is set to the maximum Rx gain in all evaluations. Using a medium Rx gain consumes even lower power. Medium gain has a practical range of about 5∼10m.

4.7 Applications

Unify enables CC254x to behave just like a WiFi chip and its low cost, tiny footprint and low power consumption make Unify ideal for IoT applications, such as the thermostat shown in Fig. 1b. Other applications that normally use WiFi can directly use Unify as well. For example, Unify is capable of streaming 360p Youtube videos in real time. Alternatively, Unify can stream high-quality Spotify audio in real time. Unify can also directly access web pages, such as weather websites. The novel communication paradigm of Unify can pave the way for innovative IoT applications in the future.

5 DISCUSSION

We select the DSSS waveform since it is the most robust WiFi modulation and offers as high as 10dB higher sensitivity than OFDM. The benefit is that Unify maintains the same distance with a much lower transmit power (or allows a longer distance with the same transmit power).

Unify works with old and new WiFi devices, because of WiFi's backward compatibility. Another benefit is that DSSS does not have the problem of OFDM's high PAPR where the linearity requirement precludes low-power implementations. DSSS also allows for reusing existing FSK hardware, which is more difficult for OFDM. Unify thus achieves very low power consumption, which is particularly useful when IoT nodes are listening for an extended period but need to take immediate actions once a packet arrives.

Chatty DSSS connections may affect coexisting OFDM traffic. However, the advantage of Unify here is that since every device follows the WiFi signal and protocol, it provides good coexistence between them. This is supported by our coexistence test where no device is starved even under network saturation. In contrast, strong OFDM signals may drown out concurrent BLE connections under network saturation. Also, for narrow band signals like BLE, it has been shown that OFDM is susceptible to narrow band interference [25] and narrow band interference may greatly affect OFDM timing synchronization and detection [26]. Unify avoids starving and other coexistence issues by directly leveraging the WiFi design that already exists in access points.

6 RELATED WORK

Earlier CTC works [27–34] modulate packets' transmit power and a receiver measures the signal strength to decode information. These designs have considerably lower throughputs than the SOTA and require modifications on both ends. Numerous recent works demonstrate communication from modified WiFi transmitters to non-WiFi receivers, by carefully constructing magic WiFi packets. WiBeacon [1] broadcasts Bluetooth beacons by modifying WiFi APs to generate and send 802.11b waveforms that also resemble Bluetooth waveforms. BlueFi [2] transmits Bluetooth beacon or audio packets by sending special 802.11n packets. TransFi [3] works similarly but in a MIMO setting. NBee [4] and WiBle [5] modify WiFi devices to send DSSS packets that encode BLE packets. OfdmFi [35–37] enables communication between WiFi and LTE-U. Interscatter [38] uses WiFi chips to send a custom AM signal. WiFi-to-Zigbee [39–44] and WiFi-to-LoRa [45, 46] are also possible by selecting WiFi packets.

Two prior studies enable communication between Bluetooth and WiFi, but none of them can work without modifying the WiFi receivers. Due to the extent of modifications required for WiFi receivers, they are only shown to work in simulation or with software-defined radios. The approach in [47] receives BLE packets using the FFT signals within the WiFi demodulation process. It is unclear whether such a feature is available across different WiFi vendors, or if collecting the FFT signals from standard WiFi chips is fast enough to provide continuous and seamless baseband samples. The WiFi implementation uses software-defined radios. Another study [48] tries to detect BLE packets using the preamble detector on WiFi receivers but concludes that the default WiFi preamble detection cannot be used. It thus proposes an extended preamble detection and demodulates BLE by collecting WiFi payloads (once a packet is detected). Its evaluation was done in simulation only.

FLEW [6] enables two-way communication between unmodified WiFi devices and modified FSK devices. However, it requires hardware modifications and using multiple chips, including a configurable FSK chip. In contrast, Unify is a single-chip solution for popular BLE/FSK SoCs.

7 CONCLUSION

We have designed and evaluated Unify, which transforms popular BLE/FSK SoCs into WiFi SoCs. Unify overcomes numerous significant challenges imposed by the hardware so that standard, bi-directional WiFi operations are achieved in SoCs without hardware modification. Without modifying WiFi devices, Unify is compatible with all major WiFi vendors and has low latency, good throughput and coexistence performance, and low power consumption.

8 ACKNOWLEDGEMENTS

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

REFERENCES

[1] 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, pages 83–96, New York, NY, USA, 2021. Association for Computing Machinery.

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

[3] 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.

[4] 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.

[5] 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.

[6] Hsun-Wei Cho and Kang G. Shin. FLEW: Fully Emulated WiFi. In *Proceedings of the 28st Annual International Conference on Mobile Computing and Networking*, MobiCom '22, New York, NY, USA, 2022. Association for Computing Machinery.

[7] Microwave Journal. TI delivers industry's only OTA download capabilities for seamless software updates of Bluetooth Smart products. microwavejournal.com/articles/19352-ti-delivers-industrys-only-ota-download-capabilities-for-seamless-software-updates-of-bluetooth-smart-products, Mar 2013.

[8] Texas Instruments. CC2544: System-on-Chip for 2.4-GHz USB Applications. ti.com/lit/ds/symlink/cc2544.pdf, 2012.

[9] Texas Instruments. CC2400 2.4 GHz Low-Power RF Transceiver. ti.com/lit/ds/symlink/cc2400.pdf, Mar 2006.

[10] NXP. LPC1756FBD80: Scalable Mainstream 32-bit Microcontroller (MCU) based on Arm® Cortex®-M3 Core. 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.

[11] Texas Instruments. CC2544 2.4 GHz RF Value Line SoC with 32kB flash, USB, SPI and UART. ti.com/product/CC2544, 2022.

[12] Intel Corporation. MCS51 Microcontroller Family User's Manual, Feb 1994.

[13] T. Aardenne-Ehrenfest, van and N.G. Bruijn, de. Circuits and trees in oriented linear graphs. *Simon Stevin : Wis- en Natuurkundig Tijdschrift*, 28:203–217, 1951.

[14] 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.

[15] IAR Systems. IAR Embedded Workbench for 8051. iar.com/products/architectures/iar-embedded-workbench-for-8051/, 2021.

[16] Texas Instruments. CC-DEBUGGER. ti.com/tool/CC-DEBUGGER, 2014.

[17] Ioannis Charalampidis. CCLib: An arduino library that implements the CC.Debugger protocol of TI. github.com/wavesoft/CCLib, 2017.

[18] Texas Instruments. New TI Bluetooth low energy software stack delivers over-the-air downloads and support for multiple stacks on one SoC. https://www.prnewswire.com/news-releases/new-ti-bluetooth-low-energy-software-stack-delivers-over-the-air-downloads-and-support-for-multiple-stacks-on-one-soc-184270451.html, Dec 2012.

[19] kernel.org. ip-sysctl.txt. kernel.org/doc/Documentation/networking/ip-sysctl.txt, 2022.

[20] Ludovic Jacquin, Vincent Roca, and Jean-Louis Roch. Too big or too small? the ptb-pts icmp-based attack against ipsec gateways. In *2014 IEEE Global Communications Conference*, pages 530–536, 2014.

[21] Nokia. Nokia SR Linux: Network-instances. infocenter.nokia.com/public/SRLINUX200R6A/index.jsp?topic=%2Fcom.srlinux.configbasics%2Fhtml%2Fconfigb-network_instances.html, 2020.

[22] Matthew Luckie and Ben Stasiewicz. Measuring path mtu discovery behaviour. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 102–108, New York, NY, USA, 2010. Association for Computing Machinery.

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

[24] Texas Instruments. CC2591 2.4-GHz RF Front End. ti.com/lit/ds/swrs070b/swrs070b.pdf, Sep 2014.

[25] Arun Batra and James R. Zeidler. Narrowband interference mitigation in ofdm systems. In *MILCOM 2008 - 2008 IEEE Military Communications Conference*, pages 1–7, 2008.

[26] Mohamed Marey and Heidi Steendam. Analysis of the narrowband interference effect on ofdm timing synchronization. *IEEE Transactions on Signal Processing*, 55(9):4558–4566, 2007.

[27] 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.

[28] 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, pages 317–330, New York, NY, USA, 2015. Association for Computing Machinery.

[29] 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.

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

[31] Xiuzhen Guo, Xiaolong Zheng, and Yuan He. WiZig: Cross-technology energy communication over a noisy channel. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017.

[32] Zicheng Chi, Yan Li, Zhichuan Huang, Hongyu Sun, and Ting Zhu. Simultaneous Bi-directional Communications and Data Forwarding using a Single ZigBee Data Stream. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 577–585, 2019.

[33] Zicheng Chi, Zhichuan Huang, Yao Yao, Tiantian Xie, Hongyu Sun, and Ting Zhu. EMF: Embedding multiple flows of information in existing traffic for concurrent communication among heterogeneous IoT devices. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017.

[34] Wenchao Jiang, Zhimeng Yin, Song Min Kim, and Tian He. Side Channel Communication over Wireless Traffic: A CTC Design: Poster Abstract. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, SenSys '16, pages 346–347, New York, NY, USA, 2016. Association for Computing Machinery.

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

[36] 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.

[37] 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.

[38] 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, pages 356–369, New York, NY, USA, 2016. Association for Computing Machinery.

[39] 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, pages 2–14, New York, NY, USA, 2017. Association for Computing Machinery.

[40] 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.

[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] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.