MethodsX 8 (2021) 101544

Contents lists available at ScienceDirect

MethodsX

journal homepage: www.elsevier.com/locate/mex

Method Article

A novel method for utilizing RF information from IEEE 802.11 frames in Software Defined Networks

Paul Zanna*, Dinesh Kumar, P.J. Radcliffe

RMIT University, Australia

ABSTRACT

Wireless security research using Radio Frequency (RF) data is complex and costly, often requiring expensive equipment and extensive offline processing. To make wireless research more accessible, we have integrated RF signal features with the existing IEEE 802.11 frame layer 2/3 data using the Software Defined Networking (SDN) paradigm. Combining low-cost RF hardware, a novel SDN processing method, and unique RF processing architecture has resulted in a framework that enables advanced wireless security and control research at a significantly lower cost. The method for enabling such functionality consist of the following stages:

- During the demodulation process, extract Carrier Frequency Offset (CFO) and Pilot Sub-Carrier Offset from the Frequency Offset Correction section, in addition to the Vector Magnitude from the equalization section.
- Append the new RF information to the wireless frame buffer and pass it to the SDR kernel driver module to further process before transferring them to a P4 application via a single API instantiation.
- Compile a custom P4 application that combines the new RF features alongside higher OSI layer features to perform advanced networking, security, or control plane operations.

ARTICLE INFO
Method name: Extraction of IEEE 802.11 RF features for use in Software Defined Networking applications
Keywords: IEEE 802.11, Software Defined Networking, Software defined Radio, FPGA, P4
Article history: Received 22 June 2021; Accepted 8 October 2021; Available online 11 October 2021

* Corresponding author.
E-mail address: mail@paulzanna.com (P. Zanna).

https://doi.org/10.1016/j.mex.2021.101544

Specifications Table

Subject Area:	Engineering
More specific subject area:	IEEE 802.11 Wireless Networking
Method name:	Extraction of IEEE 802.11 RF features for use in Software Defined Networking applications
Name and reference of original method	Openwifi: A free and open-source IEEE802. 11 SDR implementation on SoC [1]. OpenOFDM [2]. WP4: A P4 programmable IEEE 802.11 Data Plane [3].
Resource availability	Original Openwifi Source Code [4]. Modified Openwifi Source Code [5]. P4 Compiler Source Code [6]. WP4 Extension Source Code [7]. Analog Devices ADRV9361-Z7035 Evaluation Board [8].

Method details

The native SDN model has limited applicability in wireless networking due to its pipeline's fixed packet header format [9]. The OpenFlow protocol [10], which is the most common SDN approach, enforces a strict group of header fields based on the IEEE 803.2 Ethernet standard. Thus, any usage within a wireless networking environment has been limited to the underlying wired network and not at the IEEE 802.11 frame level without data plane customizations [11]. Consequently, the SDN paradigm could not provide the same benefits to wireless networks as it has in data centers. This new method has addressed these shortcomings in a novel way, facilitating several new SDN research areas associated with wireless network security, useability, and management.

This original approach allows a P4 [6] program to use IEEE 802.11 RF information to create more advanced, targeted networking algorithms. P4 is a domain-specific programming language designed for creating custom SDN packet processing pipelines. P4's strength lies in its protocol independence, enabling header fields to be defined by the end-user, unlike OpenFlow. In developing this new method, the goal was to extract, consolidate, and transfer specific RF data from an IEEE 802.11a/g frame for further processing using a P4 program explicitly written to process these new data types. This approach enables researchers to combine RF data with existing MAC layer information to develop new device identification, denial-of-service (DOS) attack detection and service optimization algorithms [12].

The selection of a test platform that replicated the performance and functionality equivalent to a Wireless Access Point (WAP) was essential to ensure an accurate evaluation of capabilities suitable for domestic applications. The Analog Devices ADRV9361-Z7035 [8] was chosen because it delivers the required functionality, supports the Openwifi framework [1] and is relatively low-cost. The device utilizes a Xilinx Zynq 7035-2L System-on-a-Chip (SoC) containing a Dual-core ARM Cortex-A9 processor operating at 800 MHz and a 275K logic cell Kintex-7 FPGA. Combined with 1 GB DDR3 RAM, 256 MB of flash memory, Gigabit Ethernet, USB2 and UART interfaces, the device is equivalent to a mid-range domestic wireless router.

When investigating which RF features to include, it was found that those requiring compensation due to device manufacturing variations or environmental factors tended to be the best candidates. For example, Carrier Frequency Offset (CFO), caused by differences in local oscillator frequencies, has been shown to be a reliable identifier [13]. Other features such as Carrier Phase Offset (CPO) and Carrier Frequency Amplitude (CFA) also show strong differential values and therefore facilitate the ability to identify individual devices or behaviors [14]. While only modulation features may be available, there is still the entire PHY layer pipeline to consider, from sample decimation to modulation decoding. The AD9361 utilizes the OFDM module within the FPGA with a 32-bit serial I/Q data stream composed of 16-bit in-phase (I) and 16-bit quadrature (Q) samples.

Although OpenOFDM [2] provides some fundamental values in a raw format, such as the Received Signal Strength Indicator (RSSI) and timestamps, other features require new extraction and computation methods. Several new functions have been added as Verilog code in the FPGA and the remainder in the SDR Linux kernel module. The capabilities of each component were used to determine where the new code should reside. For example, functions requiring complex mathematics are more suited to C code in the driver, while timing specific functions may only be

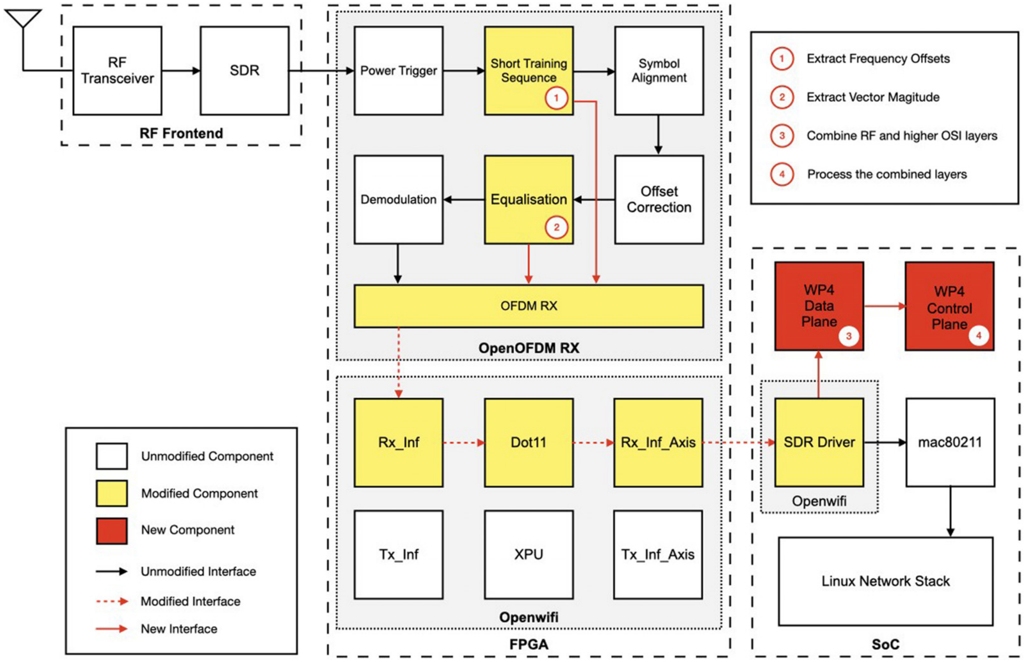

Fig. 1. Method overview.

possible in the FPGA. However, the occasional compromise will be explained, such as the inability to calculate floating-point numbers within a kernel module resulting in scaled values. The overall system architecture shown in Fig. 1, with the computation and interface techniques of each RF feature extracted in this method, will be discussed in detail in the following sections.

An IEEE 802.11a/g frame header includes two RF training fields, a Short Training Field (STF) and a Long Training Field (LTF). These training fields are used for various synchronization and frequency offset calculations to ensure the frame's reliable demodulation. The STF consists of 10 identical repetitions, each containing 16 complex values used for frame synchronization, Automatic Gain Control (AGC), and coarse Carrier Frequency Offset (CFO) estimation. Simultaneously, the LTF comprises two repetitions of 64 complex-valued samples used for channel estimation and residual CFO correction. While the OpenWifi platform currently supports 802.11n, the methods for decoding High Throughput (HT) frames vary from 802.11a/g frames and, therefore, will be addressed in future versions of this method.

Extraction of CFO from STF

As depicted in Step 1 of Fig. 1, the CFO is the first calculation in a series of procedures used to compute the signal's frequency offset based on the STF. Oscillator variations in the carrier frequency are often due to manufacturing tolerances, temperature changes and the Doppler effect resulting from device movement. The frame received at frequency f_c is down-converted to baseband and sampled at S_R, producing 160 samples. The last five repetitions are used for frequency offset estimation, denoted as $S_m$where $m = 0$–79, thereby the coarse CFO estimate of α_{ST} is produced by:

$$\alpha_{ST} = \angle\left(\sum_{m=0}^{64} S_m^* S_{m+16}\right), \tag{1}$$

A division by 16 is typically performed to calculate the offset of a single sample within the FPGA by shifting four bits to the left before being output to the SDR kernel module. However, doing so produces a significant loss of accuracy due to rounding. The required division in this method is no longer performed in the FPGA, and therefore the output is the phase shift across the entire STF, ensuring no loss of information. Alternatively, performing the division within the kernel module will not produce any further accuracy as it is not possible to perform floating-point calculations within the Linux kernel.

Previously within the OpenOFDM module, the CFO ε was calculated using:

$$\varepsilon = \frac{S_R \cdot \left(\frac{\alpha_{ST}}{16}\right)}{2\pi}, \tag{2}$$

S_R equals a sample rate of 20 Msp/s, and the CFO of the entire STF is divided by 16 to equal a single symbol CFO.

To capture the CFO across the entire STF before division, a new 32-bit register, *phase_offset_full_out,* was added to the *sync_short.v* FPGA module so the *phase_out_neg* value can be captured in full. If the traditional CFO is required later, it is possible to calculate it within the WP4 Control Plane by dividing the value by 16 to produce a floating-point value of higher precision without integer rounding.

Extraction of vector magnitude from equalizer stage

Due to several environmental effects, each sub-carrier will exhibit different magnitudes, which require normalization before decoding. To compensate for these variations for 802.1a/g, the mean value H_i of the two LTFs is calculated using:

$$H_i = \frac{1}{2}\left(LTF_i + LTF_i' \ . \ L_i\right), \ i \in (-26, 26), \tag{4}$$

Where L_i is the polarity of the LTF sequence at sub-carrier i as per the IEEE 802.11a/g standard.

Each sub-carrier Y_i is then normalized by:

$$Y_i = \frac{X_i}{H_i}, \ i \ \in (-26, 26), \tag{5}$$

Where X_i is the FFT of sub-carrier i.

The mean value H_i is sent to the SDR driver as Vector Magnitude (VM), another value that was previously unavailable outside the FPGA. To facilitate the extraction of the VM, a *mag_sq_out* wire was added to the *equalizer.v* module to capture the value during the calculation, as shown in Step 2 of Fig. 1.

Extraction of pilot carrier offset

In addition to coarse-grained CFO adjustment, residual CFO correction uses the pilot sub-carriers $\in (-21, -7, 7, 21)$ also shown in Step 2 of Fig. 1, which is estimated by:

$$\theta_n = \angle\left(\sum\nolimits_{i\in(-21,-7,7,21)} X_i^n . P_i^n . H_i\right) \tag{3}$$

Where P^n is the polarity of the pilot sub-carrier at symbol n.

θ_n of the last symbol, which was not previously available outside of the FPGA, is copied to a new wire, *pilot_phase_out* and sent to the SDR driver.

Additional wires and interconnects were created and added to the Verilog source files to consolidate and route the new measurements out of the OpenOFDM into the core Openwifi modules. Three new 32-bit input wires, *phase_offset_full, mag_sq_out* and *pilot_phase_out,* were added to the Openwifi receiver interface module *rx_intf.v*.

WP4 data plane

The Openwifi implementation uses the Xilinx Advanced Extensible Interface (AXI) to transfer data between the FPGA and the Linux SoC. There are two AXIs interconnects currently in use, one for

Table 1
Header format.

Data	Type	Size (bits)	Offset (bytes)
Timestamp (low bits)	Unsigned	32	0
Timestamp (high bits)	Unsigned	32	4
RSSI	Unsigned	16	8
IEEE 802.11 frame length	Unsigned	16	12
Data Rate Index	Unsigned	16	14
Carrier Frequency Offset	Integer	32	16
Pilot Frequency Offset	Integer	32	20
Magnitude Vector	Unsigned	32	24
Unused	Unsigned	32	28
Raw IEEE 802.11 frame data	Unsigned	-	32

frame data and the other for side-channel information such as CSI. Attempting to use data from both interfaces simultaneously would create undue complexity. Therefore, this method uses the novel approach of adding the RF information to the existing IEEE 802.11 frame interface. A total of 16 bytes were appended onto the frame after the existing timestamp and RSSI values, between the metadata header and the IEEE 802.11 frame, using the DMA header function in the *rx_intf_pl_to_m_axis.v* module.

Two options were viable to pass the new feature values from the SDR kernel module to the P4 interface, a function call or frame headers. As the RF feature values are not part of the IEEE 802.11a/g frame, it could be considered acceptable to maintain the separation of the frame's payload from any additional data. However, doing so would circumvent the inherent value of P4, that of the customizable header paradigm. Therefore, new RF values are added to the beginning of the IEEE 802.11 frame before being passed to the P4 interface.

The system architecture shown in Fig. 2 highlights the overall flexibility of such an approach which provides multiple customization opportunities. While the WP4 kernel module is created using the P4 programming language, the control plane interface facilitates advanced algorithms such as machine learning. For example, the rule or "flow" table function standard in the SDN model lends itself elegantly to a Learning Classifier System (LCS). Such a methodology is also well suited to a device fingerprinting solution, which parenthetically is the archetype's genesis.

Step 3 in Fig. 1 depicts the integration point of the SDR kernel module from the *openwifi_rx_interrupt* function within Openwifi's *sdr.c* driver file, which allows the new RF values to be extracted from the *rx_cyclix_buf* buffer. Calling the WP4 entry point directly from within this function allows access to all wireless frame types, including management frames. The *wp4_packet_in* function is called passing the frame buffer pointer (*pdata_tmp)*, the buffer size (*len*) and a null *port* value. This method could also facilitate the inclusion of other information if required by simply adding additional parameters. The P4 parser's flexibility allows data in the new variables to be appended to the beginning of the frame and parsed the same way as the frame's native header contents. The contents of the header fields, including size, pointer offset and data type, are shown in Table 1.

The WP4 kernel module is compiled using the generic P4 compiler [6] with the custom WP4 backend extension [7], producing the WP4 target source file from a P4 language [15] file. First, the P4 compiler source code is installed on a suitable development system, the WP4 backend source is added to the extensions directory, then the entire P4 compiler binary is built. The P4 compiler's output from a P4 file is a C language kernel module source code file called *wp4-p4.c* and associated header file *wp4-p4.h,* which are then added to the WP4 driver folder in the Openwifi repository. Modifications were made to the driver makefile to allow the WP4 kernel module to be compiled together with the other Openwifi driver modules and loaded as part of the startup process. Additionally, a common functions file called *wp4_runtime.c* is included in the wp4 compilation to provide several helper functions such as kernel module instantiation, flow table lookup functions, and the control plane application interface.

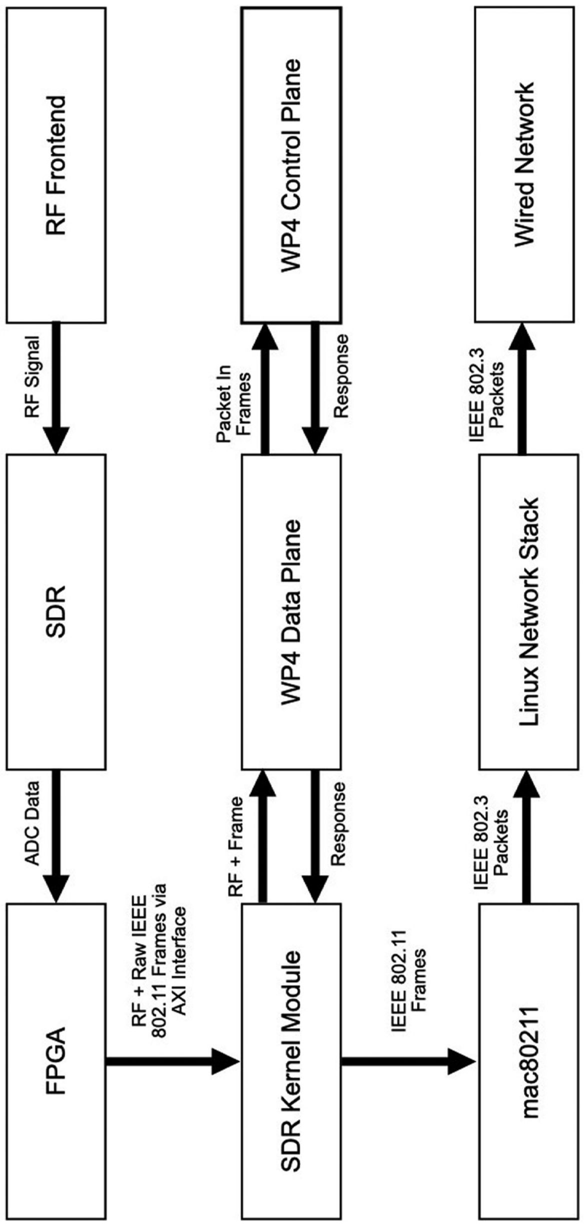

Fig. 2. Interactions between components.

WP4 control plane

The WP4 control plane application has been added as a Linux native user space application utilizing a memory map (mmap) interface between it and the WP4 data plane kernel module, shown as Step 4 in Fig. 1. Using mmap allows the two components to share a common memory space, allowing flow table updates from the control plane to be read by the data plane. A new folder has been created in the *user_space* directory titled *wp4_control,* which contains the control plane source files and required makefile for compilation on the device. The control plane application can be updated and re-compiled directly on the device independent of the data plane kernel module. The only caveat being, if the layout of the flow table data is changed, both components must be modified to ensure the synchronicity of the data layout in the flow table.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to acknowledge the support of the Australian Government through the Research Training Program Scholarship. The authors would like to express their gratitude to the developers and maintainers of Openwifi, OpenOFDM and the P4 programming language and compiler.

References

[1] X. Jiao, W. Liu, M. Mehari, M. Aslam, I. Moerman, Openwifi: a free and open-source IEEE802. 11 SDR implementation on SoC, in: Proceedings of the IEEE VTC2020, the 91st Vehicular Technology Conference, 2020, pp. 1–2.
[2] J. Shi, OpenOFDM: Synthesizable, Modular Verilog Implementation of 802.11 OFDM Decoder — OpenOFDM 1.0 documentation, 2017. [Online]. Available: https://openofdm.readthedocs.io/en/latest/index.html
[3] P. Zanna, P. Radcliffe, D. Kumar, WP4: A P4 Programmable IEEE 802.11 Data Plane, in: Proceedings of the 30th International Telecommunication Networks and Applications Conference (ITNAC), 2020, pp. 1–6, doi:10.1109/ITNAC50341.2020.9315141.
[4] X. Jiao, W. Liu, M. Mehari, Open-source IEEE802.11/Wi-Fi baseband chip/FPGA design, Github (2019). https://github.com/open-sdr/openwifi.
[5] P. Zanna, WP4 modified openwifi (tag:methodsx), Github (2020). https://github.com/pzanna/openwifi/releases/tag/MethodsX.
[6] P. Bosshart, et al., P4: Programming protocol-independent packet processors, ACM SIGCOMM Comput. Commun. Rev. 44 (3) (2014) 87–95, doi:10.1145/2656877.2656890.
[7] P. Zanna, WP4 Compiler extension, Github (2020). https://github.com/pzanna/p4c-wp4.
[8] Analog Devices, ADRV9361-Z7035 Evaluation Board, 2020. [Online]. Available: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/ADRV9361-Z7035.html
[9] K.K. Yap, et al., Blueprint for introducing innovation into wireless mobile networks, in: Proceedings of the 2nd ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures, VISA '10, Co-located with SIGCOMM, 2010, 2010, pp. 25–32, doi:10.1145/1851399.1851404.
[10] Nick McKeown, et al., OpenFlow : enabling innovation in campus networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74 [Online]. Available: http://www.openflow.org/ .
[11] H. Moura, G.V.C. Bessa, M.A.M. Vieira, D.F. Macedo, Ethanol: software defined networking for 802.11 wireless networks, in: Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015, 2015, pp. 388–396, doi:10.1109/INM.2015.7140315.
[12] Q. Xu, R. Zheng, W. Saad, Z. Han, Device fingerprinting in wireless networks: Challenges and opportunities, IEEE Commun. Surv. Tutor. 18 (1) (2016) 94–104, doi:10.1109/COMST.2015.2476338.
[13] V. Brik, S. Banerjee, M. Gruteser, S. Oh, Wireless device identification with radiometric signatures, in: Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM, 2008, pp. 116–127, doi:10.1145/1409944.1409959.
[14] B.W. Ramsey, T.D. Stubbs, B.E. Mullins, M.A. Temple, M.A. Buckner, Wireless infrastructure protection using low-cost radio frequency fingerprinting receivers, Int. J. Crit. Infrastruct. Prot. 8 (2015) 27–39, doi:10.1016/j.ijcip.2014.11.002.
[15] M. Budiu, C. Dodd, The P4 16 programming language, Oper. Syst. Rev. 51 (1) (2017) 5–14.