INSTITUTO POLITÉCNICO DE BRAGANÇA Escola Superior de Tecnologia e Gestão

Analysis of Asymmetry of Traffic in Full-duplex Wireless Local Area Network

Felipe Soares

Dissertation submitted to School of Technology and Management to obtain the degree of Master of Science in Information Systems at the Polytechnic Institute of Bragança and the degree of Bachelor in Computer Science at the Federal University of Technology – Paraná in a double degree program

Work done under the guidance of:

PhD. Luisa Maria Garcia Jorge

PhD. Manuel Paulo de Albuquerque Melo

Msc. Saulo Jorge Beltrão de Queiroz

Bragança

May 2019

Dedication

To God, to my parents, Glaiton Soares and Maria Aparecida Soares and to my love, Rafaela Lunelli.

Abstract

The standard commodity wireless hardware is half-duplex because there are challenges in full-duplex wireless that need attention and improvement. The self-interference in radios is one of the big challenges, but, even though there is no standard yet, there are several proposals that cancel enough self-interference that it is possible for communication to be successfully made. The standard half-duplex rules of the media access control (MAC) protocol contained on wireless cards do not accept simultaneous transmissions, because simultaneous transmissions are likely to collide with each other. Therefore, full-duplex wireless networks need a new MAC protocol to be able to handle the different full-duplex transmissions, namely, symmetric and asymmetric.

Symmetric full-duplex transmissions ocurr between just two stations, which can be managed trivially by a suitable MAC protocol. On the other hand, asymmetric transmissions occur in communications involving three stations, and those transmissions are likely to produce collisions if one station receives simultaneously signals from the two others. From the different difficulties of each transmission type, emerges the doubt about how many opportunities are there for a full-duplex wireless network to make each type of transmission. With the focus on this question, this research proposes a method to collect traffic data from a real half-duplex wireless local area network (WLAN) to measure the amount of full-duplex symmetric and asymmetric transmission opportunities.

The proposed method relies on: the brcmfmac driver, to collect the traffic data in kernel space; the Ftrace tracing utility framework, to send the data from kernel to user space; a Raspberry Pi 3 B+, in which is installed the modified driver and tracing utility; and an estimate of the travel time of frames between the kernel and firmware.

The results of this research include a method to collect traffic data with the goal of measuring the amount of full-duplex transmissions opportunities and their types in a real half-duplex WLAN. It is also presented the analysis of a small amount of data collected during four days as an example of the proposed method, which shows that 4.096% of the frames presented the proper conditions to symmetric transmissions, while only 0.025% in the case of asymmetric transmissions.

Keywords: wireless networks, full-duplex, traffic symmetry

Resumo

Os dispositivos sem fio padrão são half-duplex, pois o full-duplex sem fio apresenta desafios que precisam receber atenção e melhorias. A auto-interferência presente é um dos desafios, mas, ainda que não haja padrão, existem algumas propostas que cancelam a auto-interferência a ponto de comunicações serem realizadas com sucesso. As regras padrão do protocolo de controle de acesso ao meio (MAC) half-duplex contido nas placas sem fio não permitem transmissões simultâneas, já que são propensas a causar colisões. Portanto, redes full-duplex sem fio precisam de um novo protocolo MAC para que os diferentes tipos de transmissão full-duplex (simétrico e assimétrico) sejam utilizados.

As transmissões simétricas ocorrem em comunicações entre apenas duas estações, o que pode ser gerido de forma trivial por um protocolo MAC apropriado. Por outro lado, as transmissões assimétricas envolvem comunicações entre três estações, e estas transmissões são propensas a gerar colisões no caso de uma das estações receber sinal das outras duas, simultaneamente. Devido às diferentes dificuldades de cada tipo de transmissão, surge a dúvida sobre quantas oportunidades existem para comunicação full-duplex de cada tipo de transmissão. Com foco nessa questão, esta pesquisa propõe um método para coleta de dados de tráfego de uma rede de área local sem fio (WLAN) half-duplex com o objetivo de calcular a quantidade de oportunidades de transmissões full-duplex simétricas e assimétricas.

O método proposto conta com: o driver brcmfmac, para coleta de dados de tráfego em ambiente de kernel; o Ftrace, ferramenta utilitária de rastreamento, usado para enviar os dados do kernel para o ambiente do usuário; um Raspberry Pi 3 B+, no qual é instalado o driver modificado e o utilitário de rastreamento; e, um cálculo para estimar o tempo de viagem de pacotes entre o kernel e o firmware.

Os resultados desta pesquisa incluem um método de coleta de dados de tráfego com o objetivo de quantificar as oportunidades de transmissões full-duplex e seus tipos em uma WLAN real. Também é apresentado uma coleta feita por quatro dias como um exemplo do mesmo. A análise mostra que 4.096% dos pacotes apresentam condições adequadas para transmissões simétricas, e apenas 0.025% para transmissões assimétricas.

Palavras-chave: redes sem fio, full-duplex, simetria de tráfego

General Index

List of Acronyms

ADC	Analog-to-Digital Converter
ATT	Average Time Travel
CSMA	Carrier Sense Multiple Access
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DAC	Digital-to-Analogic Converter
FD	Full-duplex
FIFO	First In First Out
IBFD	In-Band Full-Duplex
MAC	Media Access Control
MIMO	Multiple-Input Multiple-Output
MLME	MAC (sub)Layer Management Entity
OFDMA	Orthogonal Frequency Division Multiple Access
RTT	Round-Trip Time
SDIO	Secure Digital Input Output
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
WLAN	Wireless Local Area Network

Figures Index

Table Index

Listing Index

Chapter 1 Introduction

New possibilities are often created by the development and advancement of technologies. Those possibilities should be investigated to understand their growth possibility. It is clear the impact wireless systems have on people’s lives, since they are used for all sorts of things, for everyday use such as in smartphones and laptops, for the internet of things devices, for mesh networks, etc. However, despite its popularity, wireless connections also present some disadvantages when compared with wired connections.

Stations connected to a single wireless network share the same medium. As a result, they contend to transmit units of informations (frames) through the medium. If more than one station simultaneously sends a frame, the frames likely collide, which means they cannot be properly decoded at the receiver. The carrier sense multiple access (CSMA) is a method created with the goal to prevent collisions, but it is not able to eliminate them. Stations using CSMA checks whether the medium is idle (no other station is transmitting at the moment) before sending each frame. A collision can still occur when the medium is idle and two stations transmit their frames at the same time. In wired networks, an improved CSMA method is applied, called carrier sense multiple access with collision detection (CSMA/CD). CSMA/CD was developed because it is possible to send and receive information simultaneously in wired networks. Hence, if station A is sending a frame and starts reading signals different from the ones it sent in the media, it means there is a collision, and the transmission is aborted. Another improvement for CSMA is the carrier sense multiple access with collision avoidance (CSMA/CA) that says that even with an idle medium, a station has to wait a random amount of time before starting transmitting its frame, in order to reduce the chance of collisions. In wireless local area networks, the CSMA/CA is needed to avoid simultaneous transmissions.

In contrast with the wireless connections, the wired connections are able to detect collisions, transfer and receive data at the same time. They also present lower latency in comparison to WLANs and do not suffer from problems like the hidden station or exposed station [1]. On the other hand, advantages of wireless connections are the convenience of being connected everywhere, its ease of use and allowing for the creation of tiny devices that are able to be connected wirelessly. Therefore, all the limitations of wireless connections can be used as an incentive for research to be made to mitigate those limitations.

There are publications in wireless communications suggesting improvements by applying mixed radio techniques. E.g.: the multiple-input multiple-output (MIMO) system is defined by a station with a set of multiple antennas to transmit and receive frames from/to another station [2]; beamforming is a technique that takes advantages of channel information to improve the signal-to-noise ratio on the receiver [3]; and, the orthogonal frequency division multiple access (OFDMA) uses multiple orthogonal frequencies (do not interfere with each other) by dividing the bandwidth, which allows simultaneous transmissions to several users and improves robustness and scalability of connections [4], [5].

In-Band Full-Duplex (IBFD) is another radio technique that can leverage the performance of a wireless link. With a IBFD radio, a station can receive and transmit simultaneously within the same frequency band [6]. Some works show how to design IBFD-capable radios e.g., [7], [8]. The design presented in [8] achieved an improvement of 87% over the half-duplex mode. However, this (and other similar physical layer IBFD results) consider a single link. To translate the gains of IBFD radios into higher performance for actual WLANs, the medium access layer need to be able to identify and exploit the IBFD opportunities. These opportunies are classified as either symmetric, in which the transmissions happen between two stations; and, asymmetric, in which the transmissions happen between three stations. With the goal to implement in-band FD in WLANs transmissions, it is needed to understand its nature and to design processes to be applied to WLAN transmissions. Collecting data relating to the behavior of a working WLAN can help researchers understand the opportunities of in-band FD transmissions, and it can also encourage them to focus their work according to the behavior details of a real WLAN flow.

1.1. Organization of the work

This research is organized as follows. In Chapter 2, the state of the art is discussed, including concepts fundamental to this research, as well as existing problems on improving today's wireless technology and the attempts to work around them. In Chapter 3, the methodology is presented, divided into discussions of fundamental strategies, goals, tools, the of collection method of and other choices taken. In Chapter 4, the data collection environment is explained and the results are discussed. Finally, in Chapter 5, the impact of this research in its field is reasoned about, and some possible future works are suggested.

Chapter 2 State of the art

This chapter is dedicated to present the technologies that motivated this research and its basic concepts and issues, including in-band full-duplex and half-duplex radios, self-interference and medium access problem.

2.1. Introduction

In this chapter are discussed some ideas related to wireless characteristics, such as the strategies usually applied to wireless networks. The discussion also includes attempts and challenges to make wireless more efficient, through the use of technologies already in use in the wired networks, such as full-duplex transmissions.

2.2. Full-duplex

A transmission is called full-duplex (FD) if it occurs between two stations simultaneously. If these simultaneous transmissions use the same frequency band, the transmission is classified as In-Band FD. Otherwise it is classified as Out-of-Band FD. When the transmission happens between two stations, but it cannot be at the same time, the transmission is called half-duplex. Finally, if only one station is able to transmit and the other station is only able to receive, the transmission is called simplex [1].

In WLANs following the IEEE 802.11 standard, transmissions are half-duplex. FD transmissions provide the theoretical benefit of doubling the network throughput in contrast to half-duplex transmissions, but they are only possible by using the concept of the out-of-band FD.

This work focus on in-band FD, therefore in-band FD is going to be simply referred to as FD. In the following sections, two challenges of FD WLANs are discussed, namely, self-interference and the medium access problem.

2.3. Self-interference

FD is attractive, but it includes serious challenges. When a station starts transmitting, its own receptor gets the transmitting signal. This is called self-interference and it makes the radio unable to properly demodulate signals from other stations. The self-interfering signal acts at the radio's signal reception path with practically no loss, whereas the signal coming from a third party radio arrives at the same receiver much weaker, because of the path propagation loss. Therefore, self-interference prevents reception from other nodes.

Some works describe self-interference cancellation strategies, as in [6], [8], [9], [10], [11], [12] and [13]. The self-interference cancellation techniques can be classified into digital circuit domain, analogue circuit domain or propagation domain [14]. Figure 1 presents a scheme with the three types of self-interference cancellation:

- The digital domain cancellation is applied in the digital signal processing, after the frame is received by the antenna and converted to digital;
- The analog cancellation takes place after the transmitting signal is converted to analog. A copy of the analog signal is sent to the canceller circuit, which processes it and sends it to the receptor to cancel the self-interference; and,
- The propagation cancellation acts in the transmitter/receptor and it takes advantages of electromagnetic properties to suppress the self-interference.

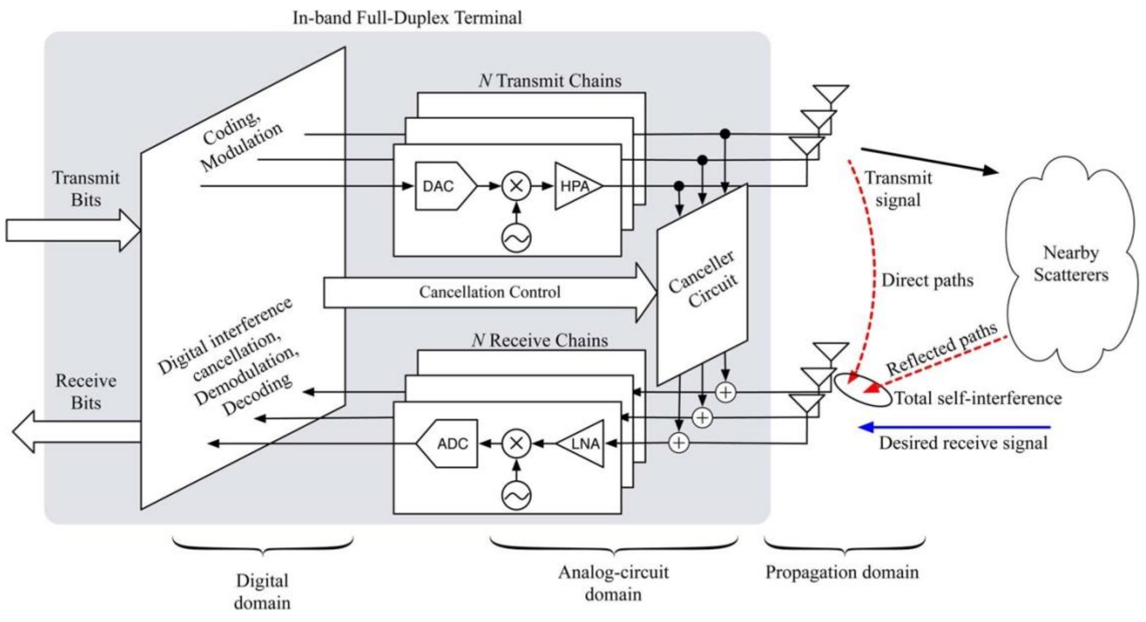

Figure 1: Representation of self-interference domains (reproduced from [15])

2.4. Medium Access Problem in Full-Duplex WLANs

The opportunities to FD transmissions depends on traffic behavior and can be classified into two transmission types.

A symmetric FD transmission opportunity can happen when two stations have packets to each other. The symmetric transmissions can be considered trivial since the two stations are communicating exclusively with each other and its assumed that no other node can transmit any frame at the same time.

In an asymmetric FD type, the transmission can happen between three stations, as illustrated in Figure 2. Station A starts transmitting a frame after gaining the CSMA/CA contention. After demodulating A's frame header, station B verifies that there is no queued frame to A but there is a frame to station C and starts this new transmission. However, if station C is in the range of the signal from station A (dotted arrow), there will be a collision between the frame from A and the frame from B at C. In this situation, it is not trivial for station B to detect whether it is safe to make the asymmetric transmission or not.

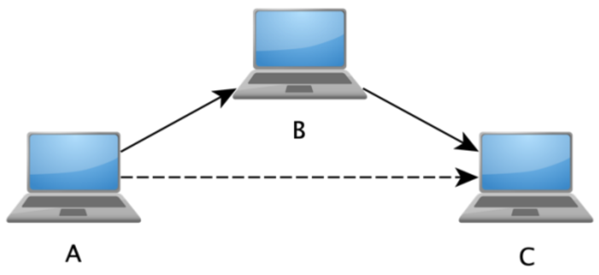

Figure 2: Asymmetric transmission (adapted from [10])

The CSMA/CA based media access control (MAC) used in half-duplex does not exploit FD opportunities. To fill this gap, novel MAC protocols haven been proposed in the literature [6], [16] and [17].

In spite of the fact that novel MAC protocols are mandatory for future IBFD-based WLANs, they do not suffice to ensure the expected throughput gain. In fact, the IBFD gain also depends on the existence of symmetric or asymmetric frames during a transmission opportunity. In turn, benefiting from these opportunities is not only a matter of protocol stack design but also of the traffic pattern in the network. Therefore, the central question of this work is: how often the IBFD opportunities arises in a real-world WLAN? Aiming to answer this question, Oliveira [18] presents results assuming different probability distributions in a simulated environment. Under that assumption, the author verifies that the IBFD opportunities happens in 36.93% of times for symmetric transmissions and 42.21% of times for asymmetric transmissions. Although the relevance of this contribution is remarkable, the assumed traffic patterns were not verified in an actual WLAN.

The current work, presents a methodology identify FD opportunities in a real-world half-duplex WLAN. The methodology comprises collecting and analyzing real-world data, based on which different IBFD MAC protocols can be compared.

2.5. Radio Data Path

The identification of IBFD opportunities in a half-duplex radio is not a trivial task. The reason is that such opportunities have to be identified just after the header of the incoming frame is processed at the primary receiver, which constitutes a hard real-time deadline to meet. To support this study, in this section we explain the basic transmission and reception paths followed by a frame within a half-duplex radio.

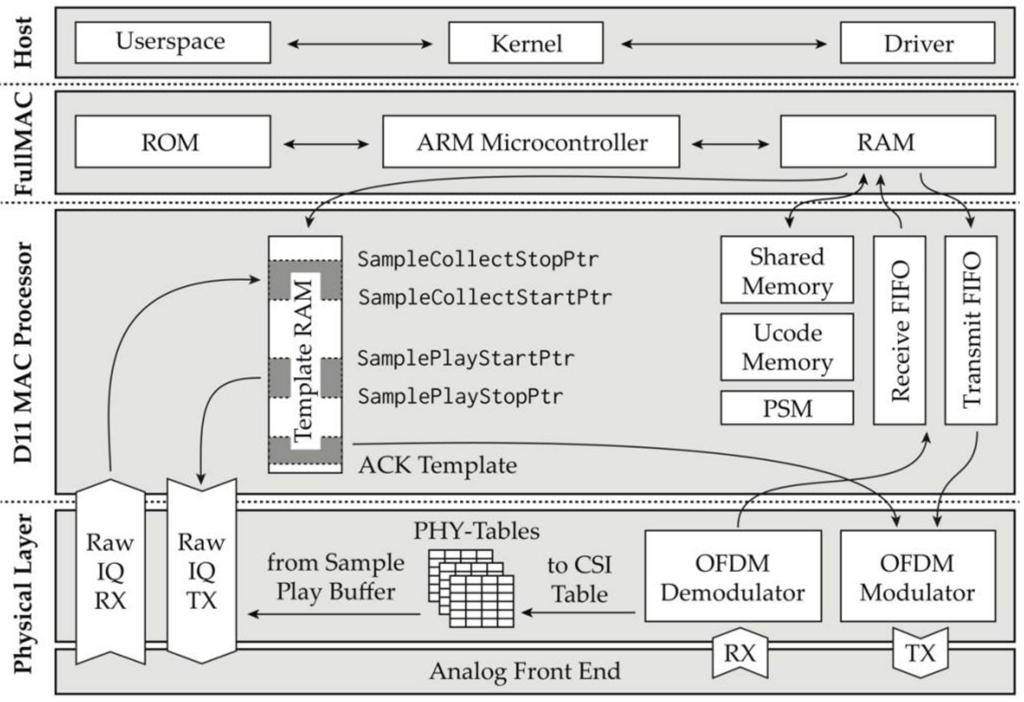

Figure 3: Broadcom/Cypress FullMAC internal (reproduced from [19])

As Figure 3 shows, when a device receives a wireless frame, the antenna demodulates the signal and sends the frame to the analog-to-digital converter (ADC). The digital frame is then sent to the firmware of the network chip on that device (D11 MAC Processor), in which the frame is added to the receive FIFO and sent to MLME handling. The next step is to send up the frame to the device driver running in the host, then to the kernel and finally, to the operating system.

On the other hand, on transmission, the data follows opposite steps: the frame is sent from the operating system kernel and then to the driver, where the frame is included in the transmission queue. It is then sent to the firmware, where it is added to the transmit FIFO, it is sent to the digital-to-analog converter (DAC), then finally to the antenna circuit, going through the OFDM modulator, then the transmission is made.

Any device communication goes through the MAC (sub)Layer Management Entity (MLME), which performs the physical layer MAC. Depending on the chip, the MLME is handled in different steps of communication. If the MLME is managed in hardware, the chip is called a FullMac wireless card, and if the MLME is managed in software, the chip is called a SoftMAC wireless card [20].

2.6. Conclusion

It is straightforward to understand why the wireless routers people have at home are not yet able to execute FD transmissions: FD in the WLAN environment is still in development, and it still has challenges that needs attention and research.

In this context, some solutions to the self-interference problem (which was the main reason FD was never considered to be used in WLANs) were presented in this section. Publications as [8] and [6] show results that cancel the self-interference sufficiently, which makes FD technology one step closer to be implemented. But there is still a lot to be studied, as new ways for MAC process to fulfill FD needs. This work contributes to this environment by providing the investigation of a methodology to identify the FD connection opportunities and its types (symmetric and asymmetric).

Chapter 3 Methodology

This chapter discusses the publications that fundament the goals of this work, the concepts that motivate them and how to achieve them. In section 3.1 and 3.2, it is reasoned about existing concepts that support the core of this research, as an FD MAC protocol, tools, and equipment. Finally, sections 3.3 and 3.4 presents the development of the resulting methodology to collect and handle the data.

3.1. Introduction

In order to properly study the behavior of a real WLAN, a suitable environment must be set up and a strategy to collect the data defined.

In [6] the design and implementation of a real-time FD MAC process is described. This MAC process is based on the strategy presented in [10], which is limited to symmetric communications only. Figure 4 shows the way an FD transmission is handled in this process. Station A sends a frame to station B. B first receives the frame header, then checks if there is a frame in its transmission queue with A as its destination. If B finds such a frame, it immediately starts the FD transmission to A, as presented in Figure 4 (a). In this case, both transmissions finish at the same time, meaning that the body of the frame from A is the same size as the whole frame (header and body) from B. Those frames can be defined as compatible. However, if the frames are incompatible, as in Figure 4 (b), one station will finish the transmission before the reception is finished, so a *busy tone* is sent until the communication fully ends (on both sides). Finally, Figure 4 (c) shows that, if B finds no frame destined to A, it immediately starts sending a *busy tone*. Those actions prevent the hidden terminal problem, because any other station in the range of A or B will notice that the media is busy and, therefore will not send any frame.

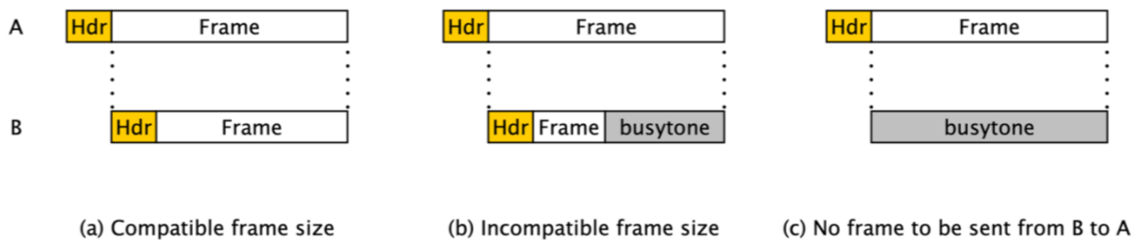
A
B
Hdr
Frame
Hdr
Frame
(a) Compatible frame size
Hdr
Frame
Hdr
Frame
busytone
(b) Incompatible frame size
Hdr
Frame
busytone
(c) No frame to be sent from B to A

Figure 4: Station A and station B make FD transmission (adapted from [6])

A data collection process is proposed assuming the stations will follow the MAC process just presented (originally defined in [6]). In order to check if there are opportunities for symmetric FD communication, the following goals for this data collection were set:

- Identify the moment the header of a frame is received by the station, then
- Get access to the transmission queue of the station, and
- Store offline data from the header of the receiving frame and the headers of the entire transmission queue.

Once the data is collected, it is possible to check the network behavior and the nature of the possible transmissions, that is, to check if it would be possible to make use of the full-duplex concepts and send a frame, at the same time, back to the origin station.

In order to have an accurate data collection, the ideal scenario would be to identify the exact moment the station receives a header of a frame. That happens in the MLME and, as explained in Chapter 2, it depends on the wireless card. A SoftMAC card seems more direct and accessible way to achieve the goals since there are free and open source drivers as the ath9k [21]. On the other hand, a set of FullMAC cards by Broadcom/Cypress are supported by the Nexmon framework, which simplifies the process of collecting information by enabling firmware patches.

3.2. Tools

The data collection is not a trivial task and needs to handle low-level data in a very specific moment. The tools selected to make data collection possible are presented in this section.

3.2.1. Nexmon

Nexmon, presented in [19], is a firmware patching framework for Broadcom cards. It aims to provide to the community means to edit the proprietary Wi-Fi firmware. With this method, the researchers do not need to use custom hardware, which is expensive, more energy consuming and hard for cross-layer evaluation. The reasons Nexmon do not focus on SoftMAC cards are that those cards can be patched by updating the driver code (commonly available, as the ath9k and br43) and many common devices, e.g. smartphones and Raspberry Pi, use Broadcom FullMAC cards, according to [19].

To learn how the firmware works, it was necessary for Nexmon developers to reverse engineer essential parts of the firmware. Then, they were able to design a patching tool that uses C code as an input, compiles it and patches the binary into the firmware. This is possible because Nexmon compresses the D11 core (the component that handles MAC-layer events), and this compression frees up a little space that can be used to store patch symbols.

Nexmon provides, by default, a patch to enable monitor mode and frame injection for most of the supported chips. All patches need to be called from inside the firmware, hence, if there is research to do requiring experiments with firmware features not present on the default patches, the effort to write a completely new feature is considerably larger, since there is the need to know the firmware address to insert the call to the new patch.

The attempts made to actually install Nexmon resulted in several internal issues, even though the installation instructions were followed. Tests were made with the goals to activate the monitor mode (not available by default in the device without Nexmon) and to collect frames. Another test was made using the device in AP mode, which would be the way to collect data to this research. But unfortunately, after some effort applied to this matter with no success, an issue was opened in Nexmon project page [22], and the answer was that it was needed to reverse engineer the firmware in order to write a new patch that would satisfy this project needs. After some time studying the firmware code, it was estimated that there would not be enough time to use Nexmon in this research. Nevertheless, the Nexmon work is relevant for providing an explanation of the workflow of the firmware, as well as presenting experiments and results in the low-level that are used in this research.

3.2.2. Raspberry Pi

Raspberry Pi is a small size single board computer with ARM architecture used to learn programming, embedded projects or regular computer use. Its recommended operating system is called Raspbian, but many other ARM distros work fine, as Ubuntu Mate, or even Windows 10 IoT Core [23], [24].

The model used for experiments in this work is the Raspberry Pi 3B+, whose wireless card is the Broadcom bcm43455 (also called Cypress cyw43455), a FullMAC chip supported by Nexmon. The driver used to handle this card is the brcmfmac [25]. The Raspberry Pi 3B+ was used as an access point to collect spontaneous traffic data.

3.2.3. Ftrace

Ftrace, described in [26], is a framework with tracing utilities, included on Linux Kernel since version 2.6.27, which can be used as a debugger and also to analyze latencies and performance in kernel space. Ftrace has the capabilities to trace functions calls in the Kernel in real time and send it to userspace using a set of files. The possible trace utilities (tracers) include listing function calls, function entry and exit and hardware latency.

Table 1: Ftrace files (Based on information available in [26])

File name	Type	Description
current_trace	Input	Holds the current tracer name. The value *nop* is used to disable all tracers and it is the default value.
tracing_on	Input/output	Holds/sets the current tracing state. 0 means Ftrace is disabled and 1 means Ftrace is enabled. The kernel functions tracing_off() and tracing_on() can be used directly in kernel code to the same purpose.
trace	Output	Holds the current information in a human-readable form, sent from Ftrace to user space. Its contents are consumed as the buffer gets full.
trace_pipe	Output	The output of reading this file is the same as reading from *trace*, however reading from this file causes its contents to be consumed, that is, the same content will not be available to be read again.
buffer_size_kb	Input	Holds the number of kilobytes used as buffer to each CPU.

Ftrace files are used to configure Ftrace and to store its functions output data. Its files are located in the folder */sys/kernel/debugging/tracing* by default. Table 1 describes the files used in this project. By default, Ftrace is disabled.

In order to write in the Ftrace buffer, one of the functions that can be used is *trace_printk()*. It works exactly as the standard C language *printf()* function, except that its output is directed to the Ftrace buffer only.

3.3. Collecting network data

The goals of this research will be pursued via an estimation, using values collected from the kernel.

To do so, timing data from previous research will be used. In [19] is described an experiment (using Nexmon) to manage the ping application directly from the firmware in the same way the kernel manages it. It used two Android smartphones (with Linux Kernel) and it sent exactly one frame per ping request and reply. The authors report the round-trip time (RTT) of frames. The data was collected using a third station, a laptop running in monitor mode. The result shows that, for a complete round-trip of pings handled in the firmware, it takes 230µs, while for pings handled in the kernel it takes around 2ms for more than 28 frames per second.

Based on RTT results, it is possible to infer the average time of frame travel between kernel and firmware (ATT). The idea is illustrated in Figure 5. First, subtract the firmware RTT (dotted arrow) from kernel RTT, resulting in the remaining sum of all travels between kernel and firmware (continuous arrows). Then devide the result by 4, resulting in the ATT (442.5µs). This reasoning is summarized in Equation (1).

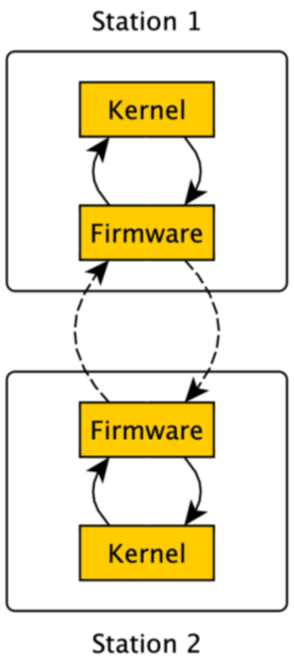

Figure 5: Round-trip scheme

$$\left(RTT_{kernel} - RTT_{firmware}\right) / 4 = ATT \quad (1)$$

With the ATT computed it is now trivial to estimate when a frame collected in the kernel was/will be in the firmware. For incoming frames, subtracting the ATT from the frame timestamp results in the timestamp the frame was sent from the firmware to the kernel. And for outgoing frames, summing up the ATT to the frame timestamp results in the timestamp the frame arrives in the firmware, coming from the kernel.

As explained in Chapter 2, when a frame to be transmitted reaches the firmware, it is inserted into a FIFO. It is only removed from the FIFO when the antenna is ready to transmit it. As explained in [6], according to the FD MAC process presented, to identify the opportunities for symmetric FD transmissions means to search in the transmission FIFO for a frame destined to A every time the station receives a frame from A.

This calculation that uses ATT, results in one timestamp per frame. With this, the calculation does not provide the amount of time that a frame will spend on the firmware FIFO before transmitted. However, using the firmware RTT presented in [6], it is possible to estimate this value.

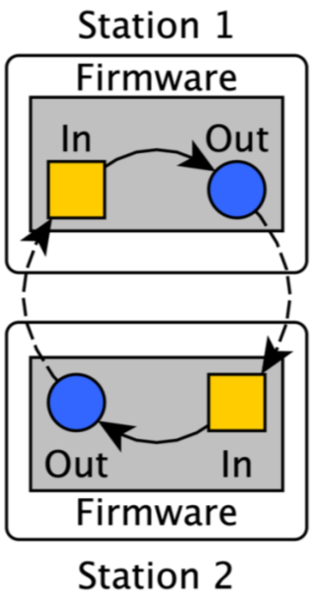

Figure 6: Firmware round-trip route

Figure 6 illustrates the round trip path, which, as discussed, takes 230µs. If the firmware RTT path is divided into four parts, it results in the average time to process each part, namely, 57,5µs per part. The path from In to Out in the same firmware is an RTT step that may take less time than the path from Station 1 Out to Station 2 In. For this reason, the average value (57,5µs) is a lower bound to identify FD opportunities.

Let us say that, according to Figure 6, Station 1 has a frame to be sent to Station 2, and Station 2 has a frame to be sent to Station 1. When the frame from Station 1 leaves the firmware output, the timestamp value is Xµs. As discussed, the input process of the frame on Station 2 will take 57,5µs. If the frame Station 2 wants to send has arrived on its output queue at timestamp X-57,5µs, then it is possible to infer that there it is an opportunity for an FD transmission.

With the discussed values, it is possible now to collect frame information from the kernel and estimate the behavior on the firmware. Hence, it is needed to discuss the kernel changes in order to access and collect the proper information.

3.3.1. Driver changes

As discussed, the Raspberry Pi 3B+ wireless card is handled by the brcmfmac driver, which is included in Raspbian. The driver communicates to the wireless card according to its interface, which in the case of the BCM43455 is the SDIO [27].

The main data type to handle reception and transmission of frames is a complex socket buffer called *struct sk_buff* [28]. Its basic layout is presented in Figure 7.

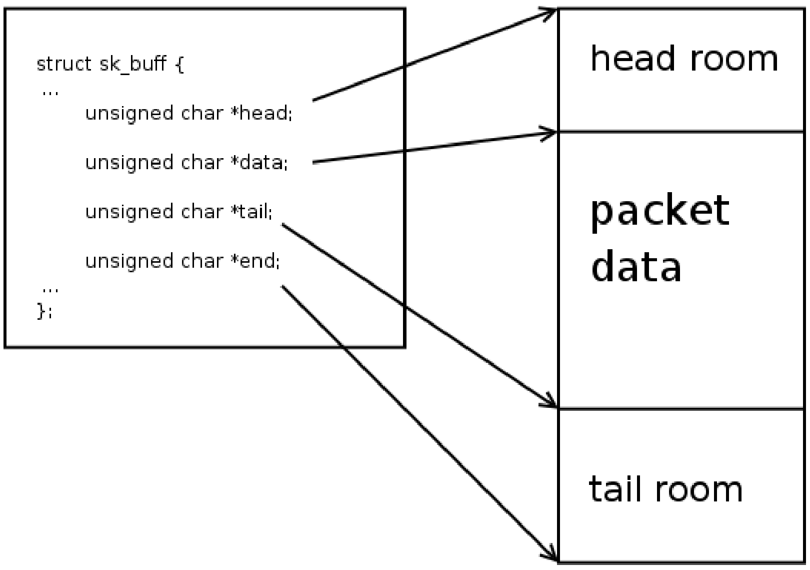

Figure 7: Socket buffer layout (reproduced from [29])

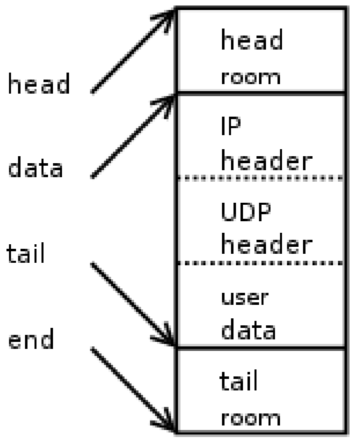

Figure 8: Populated socket buffer (reproduced from [29])

Figure 8 shows an example of a frame with header and user data included. The poiters in the buffer are constantly changing to handle the inclusion and exclusion of any needed data as the frame is being processed or created. Usecases includes handling fragmented and linear data, control frames, UDP and TCP frames, etc. Because of this dynamic behavior, the structrure needs a set of functions of access the correct data, independent of the frame status.

The file skbuff.h introduces a set of functions and macro definitions to support the use of the structure. It includes the definition of the *struct sk_buff* itself, functions to deliver frame information (as those explained in Table 2), and macros to handle the *sk_buff* queue, e.g. *skb_queue_walk()*, *skb_dequeue()* and *skb_queue_empty()*.

In the file bcmsdh.c of the driver, there is a function called *brcmf_sdiod_send_pkt()*, which is responsible to send the frames from the kernel queue to the firmware. Before the function sends each frame, it was added a call the function *trace_printk_frame()* (available in Appendix A).

The function *brcmf_rx_frame()*, located in the file core.c of the driver, receives frames from the interfaces (e.g. SDIO or USB), process them and then send them to another function that will be part of the path for the frame to reach the operating system. After the frame is process in *brcmf_rx_frame()*, a call to *trace_printk_frame()* was added.

Table 2: Collected information.

Frame field (function called/accessed variable)	**Definition**
Length (*skb->len*)	Total number of bytes
Data length (*pkt->data_len*)	Number of bytes of fragmented data [30]
Head length (*skb_headlen()*)	Number of bytes of linear data [30]
MAC header length (*skb_mac_header_len()*)	Number of bytes of the MAC header
Network header length (*skb_network_header_len()*)	Number of bytes of the network header
MAC header source address (*skb_mac_header()->h_source*)	6 bytes source address
MAC header destination address (*skb_mac_header()->h_dest*)	6 bytes destination address

The utility of the function *trace_printk_frame()* is to collect the information and print it into a temporary file using the tracing utility Ftrace. Before the core of *trace_printk_frame()*, it is added a call to the function *tracing_on()* in order to activate Ftrace, then the function *trace_printk()* is used to print the required information and, at the end of *trace_printk_frame()*, it is added a call to the function *tracing_off()* in order to stop the writing on the debug temporary file. It is important to turn off the tracing utility to prevent loss of data, because, once the temporary file fills up, old data is removed from it and new data is written.

The function *trace_printk_frame()* collects the timestamp of the moment the frame is being sent to the firmware, and it also collects from the frame the pieces of information presented in Table 2.

Once the changes are made in the kernel, the current Raspbian installed on the station needs to have its kernel rebuilt using the data collector driver code.

3.3.2. Collecting data from Ftrace

As discussed, Ftrace print functions store the values on consumable debugging trace files. To automate the data collection from Ftrace files, the *cron* program runs the *collector.sh* script every minute. The script checks if the *trace* file is empty and it registers in a log file if any data was recorded or not. The verification process is applied every 3 seconds.

The *auto_email.sh* script is responsible for compacting all the files generated in the current day, including the result of the *df* command, which gives it information about the station storage. Those files are sent to an email at 23:59h every day.

Both scripts and the *cron* configuration are available in Appendix A.

3.4. Handling raw data

After the data is collected by Ftrace and it is organized in files by the *collector.sh* script, the data looks like the example in Listing 1 and it needs to be parsed to format to be imported into Excel.

In Listing 1, the relevant data is located after the *trace_printk_frame* function name (which indicated where the Ftrace call came from). All the data before (including) the function name can be discarded. Each frame starts with the symbol ">>", and it is followed by the frame type (out/in) in the next line. The following lines follow a pattern, which is a title followed by its value in the next line. Finally, the frame ends with the symbol "<<".

The raw data example shows two frames. The first frame (lines 1 to 19) is outgoing from the address 101682148359480 to the address 614646115652650. And the second frame (lines 20 to 38) is incoming from the address 101682148359480 to the address 614646115652650.

Listing 1: Ftrace raw data

```
1  <...>-508  [003] ...  1903.517622: trace_printk_frame: >>
2  <...>-508  [003] ...  1903.517628: trace_printk_frame: out
3  <...>-508  [003] ...  1903.517630: trace_printk_frame: current_tstamp:
4  <...>-508  [003] ...  1903.517637: trace_printk_frame: 1557255063046762
5  <...>-508  [003] ...  1903.517639: trace_printk_frame: pkt->len:
6  <...>-508  [003] ...  1903.517641: trace_printk_frame: 90
7  <...>-508  [003] ...  1903.517643: trace_printk_frame: pkt->data_len:
8  <...>-508  [003] ...  1903.517644: trace_printk_frame: 0
9  <...>-508  [003] ...  1903.517646: trace_printk_frame: pkt_headlen:
10 <...>-508  [003] ...  1903.517647: trace_printk_frame: 90
11 <...>-508  [003] ...  1903.517649: trace_printk_frame: pkt_mac_header_len:
12 <...>-508  [003] ...  1903.517650: trace_printk_frame: 14
13 <...>-508  [003] ...  1903.517652: trace_printk_frame: pkt_network_header_len:
14 <...>-508  [003] ...  1903.517668: trace_printk_frame: 20
15 <...>-508  [003] ...  1903.517670: trace_printk_frame: pkt_mac_header->h_source
16 <...>-508  [003] ...  1903.517672: trace_printk_frame: 101682148359480
17 <...>-508  [003] ...  1903.517673: trace_printk_frame: pkt_mac_header->h_dest
18 <...>-508  [003] ...  1903.517675: trace_printk_frame: 614646115652650
19 <...>-508  [003] ...  1903.517677: trace_printk_frame: <<
20 <...>-508  [003] ...  1906.195180: trace_printk_frame: >>
21 <...>-508  [003] ...  1906.195185: trace_printk_frame: in
22 <...>-508  [003] ...  1906.195186: trace_printk_frame: current_tstamp:
23 <...>-508  [003] ...  1906.195192: trace_printk_frame: 1557255063292862
24 <...>-508  [003] ...  1906.195194: trace_printk_frame: pkt->len:
25 <...>-508  [003] ...  1906.195212: trace_printk_frame: 52
26 <...>-508  [003] ...  1906.195213: trace_printk_frame: pkt->data_len:
27 <...>-508  [003] ...  1906.195213: trace_printk_frame: 0
28 <...>-508  [003] ...  1906.195214: trace_printk_frame: pkt_headlen:
29 <...>-508  [003] ...  1906.195215: trace_printk_frame: 52
30 <...>-508  [003] ...  1906.195216: trace_printk_frame: pkt_mac_header_len:
31 <...>-508  [003] ...  1906.195216: trace_printk_frame: 4294967214
32 <...>-508  [003] ...  1906.195217: trace_printk_frame: pkt_network_header_len:
33 <...>-508  [003] ...  1906.195218: trace_printk_frame: 65535
34 <...>-508  [003] ...  1906.195219: trace_printk_frame: pkt_mac_header->h_source
35 <...>-508  [003] ...  1906.195220: trace_printk_frame: 614646115652650
36 <...>-508  [003] ...  1906.195221: trace_printk_frame: pkt_mac_header->h_dest
37 <...>-508  [003] ...  1906.195221: trace_printk_frame: 101682148359480
38 <...>-508  [003] ...  1906.195222: trace_printk_frame: <<
```

A C++ program called *prepare_string.cpp* (available in Appendix A) was written with the goal to parse the raw data file into a comma separated text file.

Listing 2: Output of prepare_string.cpp

```
id,type,begin,end,alterated,pkt->len,pkt->data_len,pkt_headlen,pkt_mac_header_len,pkt_network_header_len,pkt_mac_header->h_source,pkt_mac_header->h_dest
out,1557255063046762,90,0,90,14,20,101682148359480,614646115652650
in,1557255063292862,52,0,52,4294967214,65535,614646115652650,101682148359480
```

The Listing 1, after processed by the *prepare_string.cpp* program will be converted to the data presented in Listing 2. After this conversion, the data is now ready to be imported in Excel and analyzed.

3.5. Conclusion

In this chapter, it was the goals of this research and the tools used to achieve them were discussed. The decision to do not use the Nexmon framework was explained, as well as a way to try to circumvent its consequences through the use of previous Nexmon results in the firmware environment. The equipment used to collect data, the software modifications required and the logic of the data collection methodology were also presented.

Chapter 4 Results and discussion

In this chapter, it is discussed the environment of collection of data, its characteristics, and the way the equipment was installed. It is shown the results of the data, its meaning and tendency.

4.1. Environment of data collection

To demonstrate the method proposed in Chapter 3, it was made a collection of data in a classroom on ESTiG of the Polytechnic Institute of Bragança during four weekdays. The Raspberry Pi 3B+ was used as an access point, and it was placed inside the classroom.

This particular classroom was chosen because it does not have a good wifi reception from the regular access points available in ESTiG. This fact would contribute as an incentive for people to connect to the new access point.

4.2. Analysis of the collected data

Once the data is collected and imported to Excel, a macro is used to calculate the ATT value for each frame, honoring ATT definition: for incoming frames, ATT is subtracted from the frame timestamp, and for outgoing frames, ATT is added to the frame timestamp. Then, the macro is now able to identify the existing symmetric and asymmetric FD opportunities.

For symmetric transmissions, the macro searches for pairs of incoming and outgoing frames with the same source and destination addresses, respectively. In each pair, the incoming frame timestamp must be at most 57,5µs bigger than the outgoing frame timestamp, according to the discussion on Chapter 3.

Although the MAC protocol presented in [6] does not handle asymmetric transmissions, the asymmetric opportunities were calculated similarly to the symmetric opportunities. But for asymmetric opportunities, each pair of the incoming and outgoing frames must have different source and destination addresses, respectively.

The number of collected frames per day and its respective amount of symmetric and asymmetric FD opportunities are shown in Figure 9. The total amount of collected frames is 1.961.233, the symmetric opportunity amount is 80.337, and the asymmetric opportunity amount is 494. There were 1.498.568 incoming frames and 462.665 outgoing frames.

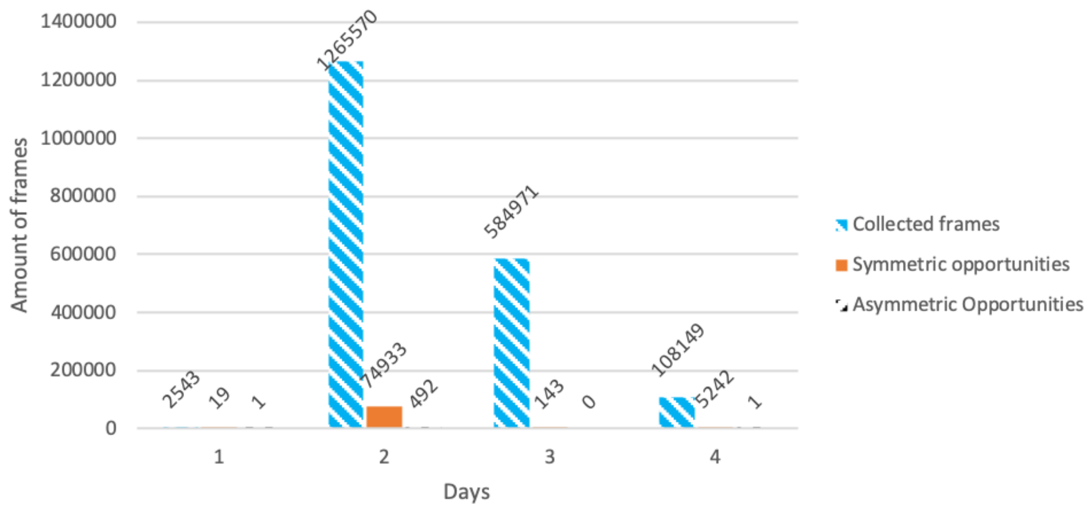

Figure 9: Number of frames and FD opportunities per day

The symmetric opportunities represent 4.096% of the total amount, while the asymmetric opportunities represent 0.025% only. Even thought the collected data amount is not enough to a proper comparision, the result discussed in [18] (36.93% of symmetric opportunities and 42.21% of asymmetric opportunities) is significantly higher than the result of this reseach.

During the four days, 22 users were connected to the access point as follow: 3 users on day one, 9 users on day two, 5 users on day three, and 5 users on day four. The small percentage of symmetric and asymmetric opportunities is directly related to the number of frames and users connected, as explained in [18], which shows that there is a strong correlation between the amount of traffic and the percentage of FD opportunities, this is, the more intense the network traffic is, the more FD opportunities there are.

The asymmetric opportunity, by its definition, depends mainly on the number of connected stations. As this scenario shows, a number of users as small as 22 and not simultaneously connected are not even enough to generate more asymmetric opportunities than symmetric ones.

On the other hand, a small number of users connected simultaneously results in more opportunities for symmetric transmissions, this is, with a high amount of traffic, the fewer the users connected simultaneously, the higher the chance for the access point to have frames to any stations communicating with the access point.

As discussed, an FD transmission might need to transmit the *busy tone* in the MAC proposed in [6]. The decision to transmit a *busy tone* is made according to the compatibility of the size of the frames. With the goal to verify the compatibility of frame sizes on FD opportunities, Table 3 and Table 4 show and classify the number of incoming/outgoing frames used on FD opportunities. Both tables, in the collumns, show the number of outgoing frames that are larger than the incoming frames, and show the opposite as well. The lines classify the frames by size relation between frames, e.g., on Table 3, line 1, column 1, it shows that 50 outgoing frames are between 0% and 25% larger than its correspondent incoming frame.

In both tables, the outgoing frame size is larger than double of the incoming frame size for most FD opportunities. The same occurs (but less markedly so) when the incoming frame size is greater than the outgoing frame size.

As the Table 3 shows, in the symmetric FD opportunities found, the outgoing frame was bigger in 79.865 cases, the incoming frame was bigger in only 471 cases, and in only one opportunity the outgoing and incoming frames were the same size.

Table 3: Relation of frames size on symmetric FD opportunities

	Number of larger outgoing frames	**Number of larger incoming frames**	**Number of equal size frames**
0% - 25%	50	33	1
25% - 50%	45	25	
50% - 75%	507	23	
75% - 100%	228	19	
> 100%	79035	371	
Total	**79865**	**471**	

In asymmetric FD opportunities found, presented in Table 4, the outgoing frame was bigger in 447 cases, the incoming frame was bigger in 32 cases, and the outgoing and incoming frames were the same size in 15 cases.

Table 4: Relation of frames size on asymmetric FD opportunities

	Number of larger outgoing frames	**Number of larger incoming frames**	**Number of equal size frames**
0% - 25%	13	3	15
25% - 50%	5	2	
50% - 75%	22	1	
75% - 100%	35	4	
> 100%	372	22	
Total	**447**	**32**	

Although there are results of this data collection, the amount of data is insufficient to provide a proper answer regarding FD opportunities and must be seen as a demonstration of the proposed methodology.

4.3. Conclusion

The results presented in this chapter can be interpreted as a demonstration of the methodology discussed in Chapter 3. The data were analyzed according to the operation of the MAC protocol presented in [6].

The focus of the analysis was to count the number of FD opportunities of each type, symmetric and asymmetric. The size of the frames was also used to understand the number of FD transmissions that would need to send the *busy tone*, because of the incompatibility of incoming and outgoing frame sizes.

Chapter 5 Conclusion

In this reseach it was discussed that FD technologies for real WLANs are the next step to improve the WLANs efficiency. Even though FD technologies already exist, namely out-of-band full-duplex, the in-band full-duplex brings a set of advantages. It also brings challenges, but once those are solved, FD has the potential to become the next standard in commodity routers and general WLANs.

The self-interference used to be one of the main problems to the FD WLANs, but there are already techniques to sufficiently cancel the self-interference and allows successful FD transmissions. Another present problem on FD WLANs is the absence of a standard MAC protocol to take as much advantage of FD opportunities as possible (although there are some proposals, as in [10], [12], [16] and [17]).

The main result of this research is the methodology based on estimations to collect information from incoming and outgoing frames of a device with Linux kernel and a Broadcom chip that uses the brcmfmac driver. It was also presented the result of the analysis of collected data in a real WLAN with spontaneous traffic as a form of validation of the proposed methodology of data collection.

5.1. Future works

The results presented by this research can be enhanced with follow-up studies.

The referred Nexmon framework is a powerful tool for low-level investigations, and it could be used to identify the FD opportunities with better precision than the estimates made in this research.

The methodology presented in this research can be extended to collect information about the amount of users connected simultaneously, with the goal to verify the relation between simultaneous users and each type of FD opportunities. It can also be applied in a different scenario, for a longer period of time and with more spontaneous users at the same time, in order to verify the relation between symmetric and asymmetric FD opportunities. Statistics about the behavior of the real WLANs can be very useful as fundamental information for studies proposing new MACs for cards to be used in FD WLANs.

Bibliography

[1] B. A. Forouzan, "Data Communications," in *Data Communications and Networking*, 4th ed., McGraw-Hill, 2007, pp. 6-7.

[2] J. Gong, M. R. Soleymani and J. F. Hayes, "A Rigorous Proof of MIMO Channel Capacity's Increase with Antenna Number," *Wireless Personal Communications,* vol. 49, pp. 81-86, April 2009.

[3] Y. Jing and H. Jafarkhani, "Network Beamforming Using Relays With Perfect Channel Information," *IEEE Transactions on Information Theory,* vol. 55, pp. 2499-2517, May 2009.

[4] H. Yin and S. Alamouti, "OFDMA: A Broadband Wireless Access Technology," in *2006 IEEE Sarnoff Symposium*, Princeton, USA, 2006.

[5] G. Miao, N. Himayat, Y. Li and D. Bormann, "Energy-Efficient Design in Wireless OFDMA," in *2008 IEEE International Conference on Communications*, 2008.

[6] M. Jain, J. I. Choi, T. Kim, D. Bharadia, S. Seth, K. Srinivasan, P. Levis, S. Katti and P. Sinha, "Practical, real-time, full duplex wireless," in *Proceeding MobiCom '11 Proceedings of the 17th annual international conference on Mobile computing and networking,* Las Vegas, 2011.

[7] D. Korpi, L. Anttila, V. Syrjälä and M. Valkama, "Widely Linear Digital Self-Interference Cancellation in Direct-Conversion Full-Duplex Transceiver," *IEEE Journal on Selected Areas in Communications,* vol. 32, pp. 1674-1687, September 2014.

[8] D. Bharadia, E. McMilin and S. Katti, "Full Duplex," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM,* Hong Kong, 2013.

[9] D. Korpi, M. Turunen, L. Anttila and M. Valkama, "Modeling and Cancellation of Self-interference in Full-Duplex Radio Transceivers: Volterra Series–Based Approach," in *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, Kansas City, USA, 2018.

[10] B. Radunovic, D. Gunawardena, A. Proutiere, N. Singh, V. Balan and P. Key, "Efficiency and Fairness in Distributed Wireless Networks Through Self-interference Cancellation and Scheduling," Microsoft Research, Cambridge, 2009.

[11] S. Sen, N. Santhapuri, R. R. Choudhury and S. Nelakuditi, "Successive Interference Cancellation: A Back-of-the-Envelope Perspective," in *IX Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Monterey, 2010.

[12] J. I. Choi, M. Jain, K. Srinivasan, P. Levis and S. Katti, "Achieving Single Channel, Full Duplex Wireless Communication," *Proceedings of the 16th Annual International Conference on Mobile Computing and Networking*, September 2010.

[13] S. Gollakota and D. Katabi, "Zigzag decoding: combating hidden terminals in wireless networks," *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, pp. 159-170, 2008.

[14] C. D. Nwankwo, L. Zhang, A. Quddus, M. A. Imran and R. Tafazolli, "A Survey of Self-Interference Management Techniques for Single Frequency Full Duplex System," *IEEE Access*, vol. 6, pp. 30242-30268, 20 November 2017.

[15] A. Sabharwal, P. Schniter, D. Guo, D. W. Bliss, S. Rangarajan and R. Wichman, "In-Band Full-Duplex Wireless: Challenges and Opportunities," *IEEE Journal on Selected Areas in Communications*, vol. 32, pp. 1637-1652, 12 June 2014.

[16] Y. Liao, K. Bian, L. Song and Z. Han, "Full-Duplex MAC Protocol Design and Analysis," *IEEE Communications Letters*, vol. 19, pp. 1185-1188, July 2015.

[17] S. Goyal, P. Liu, O. Gurbuz, E. Erkip and S. Panwar, "A distributed MAC protocol for full duplex radio," in *2013 Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, USA, 2013.

[18] L. F. S. Oliveira, “Avaliação de redes sem fio com rádios full duplex,” Niterói, 2013.

[19] M. T. Schulz, "Teaching Your Wireless Card New Tricks: Smartphone Performance and Security Enhancements Through Wi-Fi Firmware Modifications," Darmstadt, 2018.

[20] "Glossary," Linux Wireless, [Online]. Available: https://wireless.wiki.kernel.org/en/developers/documentation/glossary.

[21] "About ath9k," Linux Wireless, [Online]. Available: https://wireless.wiki.kernel.org/en/users/Drivers/ath9k.

[22] "Get raw frames with Raspberry Pi 3B+ #282," Nexmon, [Online]. Available: https://github.com/seemoo-lab/nexmon/issues/282.

[23] "FAQs - Raspberry Pi Documentation," Raspberry Pi, [Online]. Available: https://www.raspberrypi.org/documentation/faqs.

[24] "Raspberry Pi Downloads - Software for the Raspberry Pi," Raspberry Pi, [Online]. Available: https://www.raspberrypi.org/downloads/.

[25] "Techdata: Raspberry Pi Foundation Raspberry Pi 3 B+," OpenWrt, [Online]. Available: https://openwrt.org/toh/hwdata/raspberry_pi_foundation/raspberry_pi_3_bplus.

[26] S. Rostedt, "ftrace - Function Tracer," [Online]. Available: https://www.kernel.org/doc/Documentation/trace/ftrace.txt.

[27] "Broadcom brcmsmac(PCIe) and brcmfmac(SDIO/USB) drivers," Linux Wireless, [Online]. Available: https://wireless.wiki.kernel.org/en/users/drivers/brcm80211#sdio_devices.

[28] S. Venkateswaran, Essential linux device drivers, Prentice Hall Press, 2008.

[29] D. S. Miller, "SKB data," [Online]. Available: http://vger.kernel.org/~davem/skb_data.html.

[30] D. S. Miller, "SKB," [Online]. Available: http://vger.kernel.org/~davem/skb.html.

Appendix A – Code files

Listing 3: Function trace_printk_frame

```
#define PKT_ETH_ALEN    6       /* Octets in one ethernet addr   */
#define PKT_IN  1
#define PKT_OUT 0

struct pkt_ethhdr {
    unsigned char   h_dest[PKT_ETH_ALEN];   /* destination eth addr */
    unsigned char   h_source[PKT_ETH_ALEN]; /* source ether addr    */
    __be16      h_proto;        /* packet type ID field */
};


void trace_printk_frame(struct sk_buff *pkt, int pkt_in)
{
    u16 *pkt_temp;
    struct pkt_ethhdr *pkt_mac_header;
    u16 pkt_headlen;
    u32 pkt_mac_header_len, pkt_network_header_len,
          pkt_inner_network_header_len;
    tracing_on();

    trace_printk(">>\n");
    if (pkt_in)
        trace_printk("in\n");
    else
        trace_printk("out\n");

    pkt_mac_header = (struct pkt_ethhdr *) skb_mac_header(pkt);
    pkt_headlen = skb_headlen(pkt);
    pkt_mac_header_len = skb_mac_header_len(pkt);
    pkt_network_header_len = skb_network_header_len(pkt);
    pkt_inner_network_header_len = skb_inner_network_header_len(pkt);

    trace_printk("current_tstamp:\n");
    trace_printk("%lld\n", ktime_to_us(ktime_get_real()));
    trace_printk("pkt->len:\n");
    trace_printk("%u\n", pkt->len);
    trace_printk("pkt->data_len:\n");
    trace_printk("%u\n", pkt->data_len);
    trace_printk("pkt_headlen:\n");
    trace_printk("%u\n", pkt_headlen);
    trace_printk("pkt_mac_header_len:\n");
    trace_printk("%u\n", pkt_mac_header_len);
    trace_printk("pkt_network_header_len:\n");
    trace_printk("%u\n", pkt_network_header_len);
    if(pkt_mac_header) {
        pkt_temp = (u16 *) pkt_mac_header->h_source;
        trace_printk("pkt_mac_header->h_source\n");
        trace_printk("%hu%hu%hu\n", pkt_temp[0], pkt_temp[1], pkt_temp[2]);

        pkt_temp = (u16 *) pkt_mac_header->h_dest;
        trace_printk("pkt_mac_header->h_dest\n");
        trace_printk("%hu%hu%hu\n", pkt_temp[0], pkt_temp[1], pkt_temp[2]);
    }
    trace_printk("<<\n");
    tracing_off();
}
```

Listing 4: Script collector.sh

```
#!/bin/bash

basis_folder=/home/pi/trace_frames
trace_folder=$basis_folder/trace_backup

for ((secs=0; secs<60; secs+=$1)); do
    if ! diff -q /sys/kernel/debug/tracing/trace $basis_folder/std_trace.txt ; then
        cat /sys/kernel/debug/tracing/trace_pipe >> $(echo -e $trace_folder/trace_$(date "+%Y-%m-%d_%H-%M-%S").txt);
        echo -e +$(date "+%Y-%m-%d_%H-%M-%S") >> $(echo -e $trace_folder/log_trace_$(date "+%Y-%m-%d").txt);
    else
        echo -e -$(date "+%Y-%m-%d_%H-%M-%S") >> $(echo -e $trace_folder/log_trace_$(date "+%Y-%m-%d").txt);

    fi
    /bin/sleep $1;
done
```

Listing 5: Script auto_email.sh

```
basis_folder=/home/pi/trace_frames
trace_folder=$basis_folder/trace_backup
report_folder=$basis_folder/reports
today_date=$(date "+%Y-%m-%d")

df > $(echo -e $report_folder/df_$today_date.txt)

tar -zcf $report_folder/report_$today_date.tar.gz --directory=$basis_folder $trace_folder/*$today_date* $report_folder/df_$today_date.txt

echo | mutt -a $report_folder/report_$today_date.tar.gz -s $(echo -e "Report-$today_date") -- email@address.com
```

Listing 6: crontab -e configuration

```
* * * * * /home/pi/trace_frames/collector.sh 3
59 23 * * * /home/pi/trace_frames/auto_email.sh
```

Listing 7: C++ program prepare_string.cpp

```
#include <iostream>
#include <fstream>
#include <string>
```

```
#include <bits/stdc++.h>
#include <filesystem>
namespace fs = std::filesystem;
using namespace std;

int main(int argc, char const *argv[])
{
  if (argc == 1 )
      return -1;
  ifstream coleta;
  ofstream table;
  string line, util_line, data_row;
  bool valid_block = false;
  int begin_idx, qtd_line_block = 0, count_block = 1, count_args = 0,
      count_all_blocks = 0, count_files = 0, count_invalid = 0, count_valid = 0;
  std::vector<std::string> vec;
  std::string path{argv[1]};
  table.open (argc == 3 ? argv[2] : "comma-separated-table.txt");
  table <<
"id,type,begin,end,alterated,pkt->len,pkt->data_len,pkt_headlen,pkt_mac_header_len,
pkt_network_header_len,pkt_mac_header->h_source,pkt_mac_header->h_dest\n";
  for (const auto & entry : fs::directory_iterator(path))
  {
    vec.push_back(entry.path());
  }
  sort(vec.begin(), vec.end());
  // It walks through all files in the folder
  for (const auto entry : vec)
  {
    if(entry.find(".DS_Store") != string::npos) continue;
    // It opens each file
    coleta.open (entry);
    if (coleta.is_open())
    {
      cout << "File #" << ++count_files << ":" << entry << endl;
      // It walks through each line of the file
      while ( std::getline (coleta,line) )
      {
        if(!line.size()) continue;
        // It searches the beggining of useful string
        begin_idx = 20+line.find("trace_printk_frame");
        if (begin_idx == string::npos)
          begin_idx = 0;
        util_line = line.substr(begin_idx, line.size()-begin_idx);
        // >> means it is a new frame
        if (!util_line.compare(">>"))
        {
          valid_block = true;
          count_all_blocks++;
          count_args = 0;
          data_row = to_string(count_block);
        // >> means the frame ended
        } else if (!util_line.compare("<<"))
        {
          if (valid_block && count_args == 18)
          {
            count_block++;
            count_valid++;
            table << data_row << endl;
          } else
            count_invalid++;
          valid_block = false;
        }
        if (valid_block)
        {
          // it checks if the current line is valid
           if(count_args > 18 || (count_args == 1 && util_line.compare("out") && util_line.compare("in"))
                  || (count_args%2 && (util_line.find("pkt") != string::npos || util_line.find("S") != string::npos)))
          {
              valid_block = false;
          }
          // Separate timestamp in two collumns
          else if (count_args == 3 && util_line.size() > 6)
          {
                    data_row+=","+util_line.substr(0, 5)+","+util_line.substr(5, util_line.size()-6)+",";
          }
          // it handles all other valid lines
          else if (count_args%2)
          {
            data_row+=","+util_line;
          }
          count_args++;
        }
      }
    }
    else cout << "Unable to open file " << entry << endl;
    coleta.close();
  }
  cout << "Invalid blocks: " << count_invalid << endl;
  cout << "Valid blocks: " << count_valid << endl;
  cout << "Blocks collected: " << count_all_blocks << endl;
  table.close();
  return 0;
}
```