SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ingegneria

Tesi di Laurea Specialistica in:

INGEGNERIA DELLE TELECOMUNICAZIONI

COGNITIVE NETWORKING:

Network Sensing with application to IEEE 802.11 communication systems

Candidato:

Jesus Roldan Diaz

Relatore:

Prof.ssa Maria-Gabriella Di Benedetto

Anno Accademico 2009/2010

Acknowledgements

First, I would like to thank my parents for their support throughout all these years of studying. Without them, this experience would not have been possible. Special thanks go to Cristina Roldán, always helping me to improve my English and my grammar, no matter the time it took to achieve this: you are so special.

I wouldn't be done without acknowledging Professor Maria-Gabriella Di Benedetto's contribution to my development as a person and an Engineer: You're the best boss and teacher anyone could hope to have. Thank you for believing in us since the very beginning, and for giving us the opportunity to work with you these eight months. The things I've learned over this period are simply priceless.

During my stay at the ACTS Laboratory, I met two incredible people who also happen to be magnificent researchers. They made every day a funnier and easier one. Thank you, Luca, for the indoor soccer matches invitations; and Dome, for your assertive advice in all categories of life.

Thank to my fellows Stefano and Sergio for helping me with the language and guiding me through Italian manners. You made our workplace even better.

Finally, I would like to thank my partner, Carmen. You alone made these the best years of my life. It has been a struggle, but I wouldn't have made it without your endless love and support. I am sure times to come will just get better. I am so grateful to have you in my life!

I consider myself very fortunate to have met each and every one of you. I appreciate your availability and dedication to me. Total thanks to you all.

Table of Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 Cognitive Radio/Networking: State of the art & History

Lately, the research community has been devoting much attention to the Cognitive Radio and Cognitive networks paradigms. Characterized by the addition of cognition capabilities -reasoning and learning- to wireless devices and networks, Cognitive Radios and networks are being developed in an effort to provide enhanced adaptability and re-configurability to overcome the common challenges of radio communications.

Officially, the idea of cognitive radio was presented for the first time in an article by Joseph Mitola III and Gerald Maguire Jr., as a new way to approach wireless communication inserting in network's life a device that will be able to detect user communications needs as a function of use context and provide radio resources and wireless services most appropriate to those needs.

Mitola III described cognitive radio in [25] as "the point in which wireless PDAs and related networks become, in computational terms, intelligent enough in regards to radio resources and communications between computers to be able to detect the eventual communication needs of the user as function of the use context and to respond to it by assigning the most adequate wireless services and radio resources right away".

Mitola's definition is one of wide scope: the term radio here would identify any generic mobile terminal used for communication – smartphones, laptops or PDAs fall onto this category. Here, the terminal is assumed to perform as an intelligent agent: devices will need to observe the environment, evaluate possible strategies and make optimal decisions to satisfy user's needs; while learning from every experience.

Today, the most popular conception of Cognitive Radio denotes spectrum-agile devices capable of performing Dynamic Spectrum Access [29]&[29]. The use of the term "radio" has been largely interpreted as referring to the lower-layer characteristics of wireless communications, particularly the physical (PHY) layer. From a Game theory approach, Dynamic Spectrum sensing falls in this category of human behavior –such as cooperation and competition: analyzing a problem by comparing possible strategies to be adopted by the Cognitive Radio. In much the same approach, a Cognitive Network, as formulated by Thomas et al. [31], is view as a society: all networking devices interact among themselves on the pursuit of the own network's best performance.

Yet, in other cases, the term cognitive has been interpreted more in the sense of acting rationally, when Cognitive Radios are viewed as Intelligent Agents performing its actions to provide satisfactory communication services. Here, Artificial Intelligence techniques are used to solve the evaluation, optimization, decision and learning problems that arise.

Amongst the application proposed by Mitola III in [25] we find spectrum pooling, offering mobile users the opportunity to negotiate spectrum access tailored to their communication needs, to overcome licensing issues.

This application makes a great deal in today's reality: now, the brunt of available spectrum is already allocated to specific services, and -with the ever increasing demand for wireless connectivity- spectrum availability could be one of the biggest challenges.

Nowadays, unlicensed bands, such as the 2.4 GHz ISM band, are the hottest usage spots, hosting several protocols contemporaneously (802.11, Bluetooth, Zigbee and WiMAX, to name a few). As a result, Dynamic Spectrum Access techniques have raised ever-growing interest; making devices performing this task the most common interpretation of the term Cognitive Radio (as stated on its formal definition by the Federal Communications Commission [29]).

Cognitive radio was therefore conceived as the objective of the evolution of software defined radio platforms. It should become completely reconfigurable wireless system –as a "black box"- that was able to adjust its communication parameters automatically depending on the demands of the network and its users.

These are some of fundamental functions of Cognitive Radio are:

- Spectrum Sensing: A relevant requisite for appropriate functioning of cognitive radios is to be able to detect underutilized spectrum and utilize it without provoking negative interference to other users. The best way to find such "holes" in the spectrum is detecting legitimate users. Techniques to detect changes in the spectrum can be classified in three groups:

- Transmission Detection: Cognitive radios need to be in capacity to determine the presence of signal of any user utilizing a concrete part of the spectrum.

- Spectrum Administration: Assign and utilize the bandwidth that fits best the QoS required by the user amongst all available bandwidths. Spectrum administration follows two distinguishable steps: spectrum analysis and spectrum decision.

- Spectral mobility: The process by which a cognitive radio changes its frequency of transmission or reception. Cognitive radios are designed to constantly switch bands in search of the best fitting in a way that is imperceptible to primary users.

- Spectrum Sharing: Find a schematic method of spectrum distribution that is just and equitable to all cognitive radio users without interfering primary users' transmissions. This represents one of the greatest challenges of developing cognitive radios; the problem is similar to the generic issues of medium access (MAC) that plague today's systems.

This thesis study focuses on the ISM band, a populated and multi-channel band. Medium Access control poses significant challenges in a Dynamic Spectrum Access system, due to the difficulties in handling multiple channels. To cope with this issue, several solutions have been proposed in the recent literature.

In [32] the Dynamic Channel Access (DCA) scheme is proposed as a modification of the 802.11 MAC tailored to multi-channel ad-hoc networks. The DCA scheme introduces modifications to the RTS/CTS mechanism to include channel availability and preference information. The solution requires terminals to have two separate wireless interfaces -where one must transmit and receive on a fixed control channel, while the other is used for data transmission on dynamically selected channels.

Another modified version of 802.11 targeting unlicensed reuse of licensed spectrum is KNOWS [33]. Its main characteristics are cooperative sensing amongst nodes to identify unused spectrum bands, and resource advertising and reservation performed by means of the newly defined RTS/CTS/DTS handshake in place of the traditional 802.11 RTS/CTS.

Another multi-channel MAC protocol is proposed in [34]. This proposal purviews the use of a dedicated control channel. In [35], a distributed scheme with the aim to dynamically allocate frequencies to access points in infrastructured 802.11 WLANs is described.

Moving forward from the term Cognitive Radio, Cognitive Networks have been the “talk of the town”. The concept of a Cognitive Network, already foreseen by Mitola, when in [25] he suggested that his cognitive radios could interact within the system-level scope of a Cognitive Network. However, the contribution of Mitola to Cognitive Networking does not go much beyond Cognitive Radio.

In 2005 [36] Thomas et al. define a Cognitive Network as possessing a “cognitive process that can perceive current network conditions, and then plan, decide and act on those conditions", and which can "learn from these adaptations and use them to make future decisions, all while taking into account end-to-end goals". The same team analyzes similarities and differences between Cognitive Radio, Cross-layer Optimization and Cognitive Networks; one relevant similarity is that Cognitive Networks are foreseen to be based on the so-called Software Adaptable Network, just as Cognitive Radio is based on Software Defined Radio. Lake [37] proposed a similar approach: the Software Programmable Intelligent Network.

Cognitive Radio and Networking has quickly emerged as a promising wireless paradigm, researched in laboratories around the world. Eventually, the paradigm is expected to integrate benefits of software-defined radio with a complete aware communication behavior. To achieve this, a need for powerful algorithms for sensing the external environment needs to be fulfilled. In this dissertation, an algorithm that provides automatic detection of IEEE 802.11 networks present in the ISM band is proposed, exploiting the band information provided by a spectrum sensing generic device, i.e. an energy detector.

Previous work, as for example [22], has addressed a similar problem, by classifying Wi-Fi vs. Bluetooth, using a spectrum sensing procedure based on distributed detection theory. The present thesis extends beyond previous investigations by considering Wi-Fi real traffic captures, and by focusing feature extraction and classification on MAC sub-layer characteristics, leading to simplicity and computational efficiency.

The *AIR-AWARE project* that serves as umbrella for this work (described to extent in section 1.3) aspires to implement a cognitive radio node for the ISM band, particularly the spectrum sensing and classification module, i.e. a generic device that provides classification in terms of technologies present in the air interface. In this way, the road is paved for a cognitive engine [28] that, utilizing the obtained information, can make instantaneous decisions over multiple transmission parameters of nodes belonging to different networks (operating in the ISM 2.4GHz band) inside a crowded heterogeneous environment.

1.2 Coexistence of Heterogeneous wireless networks on ISM band

The most relevant example of co-existence of heterogeneous wireless networks in unlicensed bands is WLAN IEEE 802.11b/g and Bluetooth networks, both using and sharing the ISM band around 2.45 GHz. Two preemptive approaches can be taken to prevent interference in multi-network environments: the collaborative approach, where radios of different technologies exchange information regarding frequency of spectrum usage; and the non-collaborative approach, where a radio device sensed the frequency of spectrum occupancy and determines channel definition without communicating with other possible users.

The first, or collaborative, case will require both a common signaling protocol to negotiate the local frequency/time allocation and the optional establishment of orthogonal channel to prevent mutual interference. Here, either dual radio systems –with one acting as the common radio- or a single radio processing both its own protocol and the common one are required.

In the second, non-collaborative case, typically used by short-range, ad-hoc radio systems, a radio device considers the local spectrum conditions and reacts accordingly. In this approach there will not be a central Cognitive Radio controlling device instructing the other devices about radio parameters choices. When a common control channel is not predefined, the challenge that emerges is how to inform all peer devices operating on the same band.

Cognitive Radio devices define their radio parameters in order to produce minimum impact over other spectrum users. This makes the non-collaborative techniques more fitting for unlicensed bands where users expect and tolerate moderate amounts of interference. However, in incumbent QoS radio systems, operating in licensed bands, a controlled and predefined level of interference becomes necessary. This poses the challenge of either restructuring such systems in order to introduce Cognitive Radios or incorporate additional devices that monitor interference locally and communicate any possible violations to all devices.

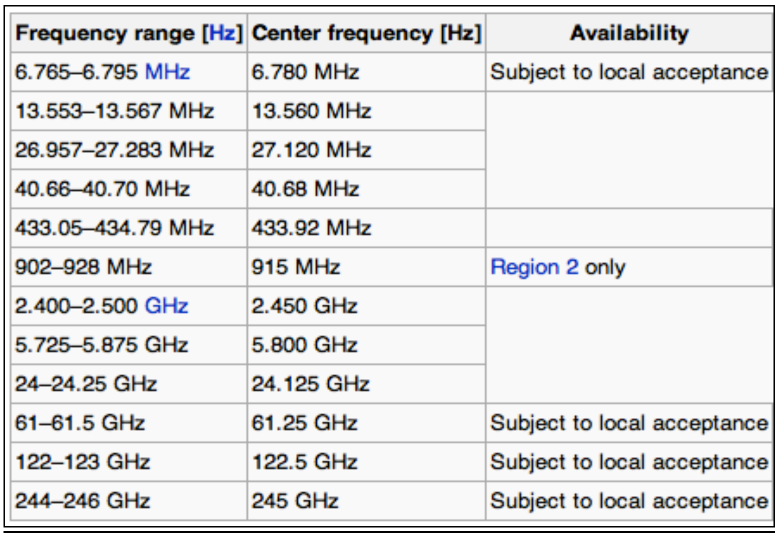
Particularly in medium to high traffic environments, such as packet-based wireless systems –typically used for Internet access- it is not clear how often and for how long the CR shall carry out a detection process. Repeated detection may present high overhead signaling, which may affect the CR link performance and the power consumption.

In the AIR-AWARE project, we propose the design and deployment of module that automatically recognizes what kind of network technologies are operating on the ISM band by feature extraction. This module will be on-line in such way that it can detect variations in spectrum's status in real time. The device will extract information regarding the ISM band through an energy detector that obtains information in all 802.11 channels; detecting presence of Bluetooth, Wi-Fi, Zigbee and other ISM band networks, by implementing passive feature extraction with low complexity.

The low complexity classification module will enable a Cognitive Radio node to make immediate decisions –based on real time band status information- to cover users –from all the different ISM networks- needs in the most efficient manner possible at any given moment.

1.3 DESCRIPTION OF THE AIR-AWARE PROJECT

This work is developed under the umbrella of AIR-AWARE Project, created with the objective of achieving differentiation –in any scenario- between present technologies and interference entities on the ISM Band.



Frequency range [Hz]	Center frequency [Hz]	Availability
6.765–6.795 MHz	6.780 MHz	Subject to local acceptance
13.553–13.567 MHz	13.560 MHz	
26.957–27.283 MHz	27.120 MHz	
40.66–40.70 MHz	40.68 MHz	
433.05–434.79 MHz	433.92 MHz	
902–928 MHz	915 MHz	Region 2 only
2.400–2.500 GHz	2.450 GHz	
5.725–5.875 GHz	5.800 GHz	
24–24.25 GHz	24.125 GHz	
61–61.5 GHz	61.25 GHz	Subject to local acceptance
122–123 GHz	122.5 GHz	Subject to local acceptance
244–246 GHz	245 GHz	Subject to local acceptance

Figure 1.1 - ISM Operational Frequency Band Chart

With the increasing diffusion of technologies operating on the ISM Band (illustrated on Figure 1.1), we noticed the relevance of creating a device (as a black box) capable of detecting and classifying different technologies networks present in a determined environment, as well as types of interferences in play. The information such a device will provide will support adjustment of radio resources and wireless services in the most appropriate way to fulfill the needs of each specific network. In order to do this, feedback between the device and different networks nodes of the distinct wireless technologies operating in that band, in a *cognitive radio*[1] context will be necessary.

[1] According to Mitola and Maguire (1999) cognitive radio "is a radio frequency transmitter/receiver that is designed to intelligently detect whether a particular segment of the radio spectrum is currently in use, and to jump into (and out of, if necessary) the temporarily-unused spectrum very rapidly, without interfering with the transmission of other authorized users."[25]

This above proposed classification is of vital relevance, considering that many commercial technologies are operating in this range of the spectrum (2.4*GHz* ISM band). The following list shows some examples of ISM operating technologies:

- IEEE 802.11 networks: operating both over the 2.4*GHz* and 5.8*GHz* bands.
- Bluetooth: operates over the 2.4*GHz* ISM band, using Frequency Hopping and chooses any of the 79 available channels on its operating band (80 MHz).
- HIPERLAN [High Performance Radio LAN]: An European alternative to IEEE 802.11, operating on the 5.8*GHz* bands in order to avoid interference entities present on the 2.4*GHz* range.
- Closed Circuit TV: Many security cameras operate over the band 2.4*GHz* ISM band.
- ZigBee IEEE 802.15.4: Wireless Data Networks operating in the 2.4 GHz ISM band.
- Wireless Mouse and Keyboard: Operating in the 2.4*GHz* band.

Regarding the interference entities in this band, probably the most common one is the microwave oven, which emits a very high power signal at a 2.45 GHz frequency[2], compromising temporarily the quality of an IEEE 802.11 network or a WPAN. Baby monitors operate also in the 2.4*GHz* band creating possible interference for 802.11 and Bluetooth

[2] This can cause considerable difficulties to Wi-Fi and Video senders, resulting in reduced range or complete blocking of the signal.

networks. Our last usual offender is the traditional DECT standard cordless phone, operating in the 1.9 GHz band, and able to interfere with the quality of connection in the first channels of a Wi-Fi network when in use.

The final goal of the AIR-AWARE project is to implement a classification strategy based on information regarding protocol layers above the physical one (PHY). In particular, the objective is to identify MAC sub-layer [1] specific features for each of the ISM 2.4*GHz* band operating technologies. Previous work, as for example [22], has addressed a similar problem, by classifying Wi-Fi vs. Bluetooth, using a spectrum sensing procedure based on distributed detection theory. The present project will try to extend beyond previous investigations by considering real traffic captures, and by focusing feature extraction and classification on MAC sub-layer communication procedures for different technologies networks, leading to simplicity and computational efficiency.

To achieve the mentioned goal, a device that is able to sense the spectrum with a good time resolution on the ISM 2.4*GHz* band must be built: the AIR-AWARE module. This module was conceived with the idea of a generic hardware, including an energy detector or radiometer as in [2], that provides information about the energy level in the whole ISM Band overtime. Evidently, with this piece of hardware, we will not be in a position to demodulate and decode the distinct signals in the air; we however, will be able to statistically study temporization of presence or absence of energy (i.e. packets) in different ISM sub-bands in real time. A software module (provided with classifiers) that analyzes pattern provided by the above-mentioned energy detector must be incorporated. This analysis should recognize similarity between presence/absence of energy periods and a Packet interchange sequence in an IEEE 802.11 network, a Bluetooth network or any other ISM band operating technology network.

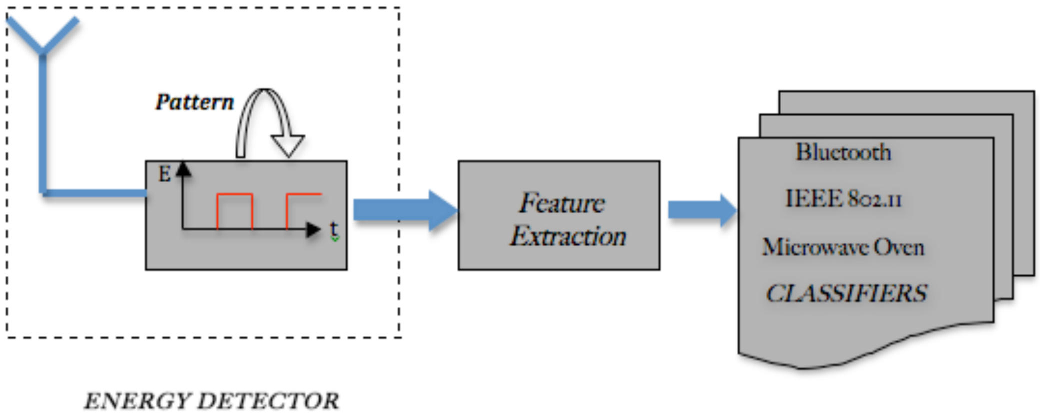


Figure 1.2- AIR-AWARE Device Schema

Figure 1.2 illustrates how also one classifiers block thought for each technology automatic recognition is embedded into the AIR-AWARE module. This device will great scalability, as can be easily noted, it will not be necessary to embed another adapted receiver every time a new available technology is desirable to classify; the update will be performed through programming a new software module with information of the PHY packet interchange behavior of this new technology network.

This software block (algorithms) will take a pattern (of energy variation over time) as an input. The above pattern will then undergo a feature extraction and, by way of classification algorithms[3], it will be recognized each pattern's observation interval as a specific kind of known network packet interchange, by feature matching. To that end, it will be necessary to study in depth the MAC sub-layer communications procedures of each and every commercial technology operating in this band –in terms of temporization of PHY packet interchange-. In addition, as allowed by our resources, it would be desirable a similar study for common interference entities –in terms of presence/absence of energy temporization while functioning–.

[3] This classifier would make a decision in function of its training set [3], provided in this project by making clean captures for each network technology operating in 2.4 GHz band.

In order to correctly select each technology features, it would be necessary achieve network sensing –packet capturing– for each one. Each workgroup is working independently during this preliminary phase of the project. Once packet capturing for each technology is achieved, could be verified the validity of the chosen features. Finally, these entire packet captures will be used as training set for our classifiers.

The final step of this phase is to test every software classification block and evaluate its performance in different scenarios. Later on, all modules will be integrated and a tested run will evaluate how they work together classifying a pattern that includes traffic from networks of different technologies and interference entities.

1.4 MOTIVATION AND PURPOSE OF THIS WORK

Network sensing together with an exhaustive research of typical MAC communication procedures for IEEE 802.11 communication systems, was the starting pre-requisite to kick off AIR-AWARE project (as described on 1.3). To this end, in this work, was developed an IEEE 802.11 packet capturing application (described in detail in Chapter 2), which provides real and clean IEEE 802.11 network's PPDUs[4] sequences.

The above-mentioned network sensing is necessary to generate an accurate analysis of the IEEE 802.11 PPDUs interchange temporization in real world, enabling us to verify if the selected features based on MAC sub-layer communications procedures are strong enough in real conditions. Feature identification here proposed, lays its foundation on the observation that packet interchange patterns are technology-specific. As such, by identifying patterns clues, network recognition could be achieved. The above features must be selected in a way that could be extracted from the pattern delivered by the energy detector –information independent–.

The IEEE 802.11 traffic analysis made on Chapter 3, was based on captures developed in a "clean scenario"[5], which allowed to characterize precisely the IEEE 802.11 PPDUs interchange based on its temporization.

Evidently, thinking at the AIR-AWARE module, the extracted pattern –energy detector's output– will be contaminated by the traffic of different technologies and interference entities. As a consequence of this, we propose

[4] PLCP Protocol Data Unit (wireless LANs), Wi-Fi PDU as sent over the air interface.

[5] We define "clean scenario" as one in which only the chosen technology is present. This is almost impossible in reality, but because the developed Sniffer receives information from the network wireless adapter of the computer –which is matched for IEEE 802.11 signals- it can take as interference any other traffic type present within the band, and not display it on the capture file.

extracting multiple features in different time[6] intervals to this pattern, and comparing them to the each technology's proposed ones through linear classifiers; this way, will be possible to recognize what kind of network or interference are present in the air.

In conclusion, particularly this work provides AIR-AWARE project with feature selection and statistical characterization for IEEE 802.11 PPDUs interchange in the 2.4*GHz* band (version b/g). This data will be utilized, in a later stage of the project, by linear classifiers that will compare these reliable features against pattern's ones, building the software block described on Figure 1.2. In addition, real IEEE 802.11 PPDUs sequences obtained by network sensing procedures will be utilized for above classifiers as training set for this technology.

Preliminary results of this work are reported in "*Automatic network recognition by feature extraction: a case study in the ISM band*" (see Appendix), by Maria-Gabriella Di Benedetto, *Senior Member, IEEE*, Stefano Boldrini, Camen Juana Martin Martin, and Jesus Roldan Diaz. Paper accepted for publication in *Proceedings of the 5th International Conference on Cognitive Radio Oriented Wireless Networks and Communications*, Special Session on Cognitive Radio and Networking for Cooperative Coexistence of Heterogeneous Wireless Networks, June 9-11 2010, Cannes, France.

The rest of this thesis is organized as follows. In Chapter 2 we will provide detailed description of network sensing applied to this work. The purpose of this chapter will be to make the reader familiar with packet capturing and sniffing concepts, also to describe how experimental-data from 802.11 PPDUs interchange was achieved. In the subsequent two chapters, we will present the major contribution of this thesis: A robust

[6] Study time will depend on the quantity interference entities and technologies operating in the medium. We predict that some time intervals –on the pattern extracted by the energy detector- can be recognized respectively as traffic types and technologies in the air.

feature for automatic recognition of Wi-Fi networks inside a multi-network environment. In particular, Chapter 3 will propose different features and extraction algorithms. Chapter 4 will test the strongest Wi-Fi proposed feature inside a Bluetooth network pattern. In Chapter 5 the conclusions will be drawn. Finally, in Appendix section all developed codes for this research –Sniffer Code and Data processing Codes- will be reported, and also the Crowncom 2010 accepted paper "*Automatic network recognition by feature extraction: a case study in the ISM band*" with first results of this research.

Chapter 2

NETWORK SENSING APPLIED TO IEEE 802.11 COMMUNICATION SYSTEMS

2.1 INTRODUCTION TO PACKET CAPTURING: SNIFFERS

Sniffers are applications installed in a computer that makes part of a network to capture information not specifically directed to them. The idea is to sense the network and monitor transmissions, while capturing data flowing through the system. Commonly, sniffers are programmed with two main functions: the first type is designed to obtain the content of transmissions –telephone conversation, e-mail message -; whereas the second type analyzes network traffic. Because of their capabilities, user discretion determines if these programs are used with administrative purposes or to illegally capture information.

In this particular case, the sniffer was designed to extract real-traffic from an 802.11 scenario; temporization and other properties of the PPDUs (PLCP Protocol Data Unit) interchange between nodes in a 802.11 network with BSS or IBSS infrastructure that would be statistically analyzed later on.

Some sniffers for 802.11 and Bluetooth networks are currently available in the market as free or commercial software. Here are some examples of sniffers working on a Linux platform:

- Kismet, is a detector of 802.11 networks that can identify intrusions. This application analyses traffic 802.11a, 802.11b, 802.11g and 802.11n. The software can be executed from BSD, MacOS (KIsMAC) and Linux-ARM (available at [20]).
- AirTraf: a 802.11 network analyzer, developed using JAVA, C and PHP (available at [19]).
- Wireshark: is a Unix based analyzer with ability to work under MacOS, OpenBSD, BeOS, Solaris and Iris. This application is oriented towards wired networks (Ethernets) and 802.11. Wireshark supports hcidump files (available at [18]).

- Hcidump is a Bluetooth packet analyzer that works on the HCI interface and can be executed from Linux console (available at [16]).

Some sniffers can work under Windows, if packet-capturing hardware is installed on the computer. These are some examples of what's available on the market:

- Air Magnet: a WLAN monitoring Windows based solution that analyzes threats and audits regulatory compliance of Wi-Fi users; the application provides complete visibility of traffic the wireless airspace. Can work with some commercial network adapters, but only if using special drivers (available at [17]).
- NetStumbler, is a tool that can monitor the network, as well as analyze packets in 802.11 networks (available at [4]).
- AirPcap: a hardware-software solution that captures full 802.11 data and offers management and control frames that can be also viewed using Wireshark (available at: [14]).
- OmniPeek: this hardware-software solution addresses wireless network monitoring and analysis from the network edge to the data center (available at [13]).
- FTE4BT, first protocol analyzer and packet sniffer for Bluetooth v1.2, developed for commercial use. Analyzes data in real time enabling the user to capture, display, decode, filter and detect errors at the same time (available at [12]).

Currently, most sniffers working under 802.11 protocol have common capabilities: packet traffic analysis, ad-hoc network detection (IBSS, BSS and ESS), monitoring of client stations and access points, network mapping, troubleshooting networks and decoding WEP (Wired Equivalent Privacy) encryption. These applications work with network adapters that possess Broadcom, Intel, Prism2, Orinoco, Atheros or Cisco chipsets, using Open Source drivers: Linux based solutions; or depending on specialized hardware that allows to them perform this operations: Windows based solutions. Sniffers working under Bluetooth protocol require specialized hardware to capture air traffic without establishing prior communications (i.e. FTE4BT, Merlin I and II).

The WLAN sniffers we know today have either been created by communities of independent programmers and distributed online for free access, or by companies who develop commercial software. This applications are targeted to network administrators, software developers and companies that integrate wireless technology in their products in order to test them before launching them in the market. Most Windows based sniffers are expensive and require specialized hardware, this seems to be the reason why many developers chose to work with free applications under Linux based platforms.

The sniffer developed for this project allows creation of data packet capture sessions in real time, as well as displaying packet content in hexadecimal format and report detailed information of captured MAC PDUs:

- Packet Length [bytes] (Radiotap Header Length + MAC Layer PDU Length).
- Captured Length [bytes].
- Radiotap Header Length [bytes].
- Frame Length (MAC Layer PDU Length [bytes]).
- TSFT (Time Synchronization Function timer [usec]).
- Band (2.4 GHz or 5 GHz)
- Modulation Type.
- Bit rate.
- PLCP Preamble Type.
- Contention-Free Period or Contention Period

- PPDU (PLCP Protocol Data Unit) Duration [usec] (Calculated by the application)
- Frame type.
- Frame SubType.
- Receiver MAC Address.
- Transmitter MAC Address.
- Fragment Number.
- Sequence Number.

The advantage of developing this sniffer using JAVA is that it can be executed under any operative system provided with Java Virtual Machine (JVM) [23], as long as the network's wireless adapter can be set to monitor mode[7]. This packet-capturing application is executed by console under any Java Software Development Environment, most likely embedded with an option to store capture sessions as files (.txt is a common example). Capture files can later be loaded and displayed anytime, making possible to import this data into a higher level programming language[8] when desirable. Once the capture data is imported into MATLAB, a reproduction of the network PPDUs sequences in a Time Diagram will be implanted. This time diagram shows presence or absence of PPDUs in the air (by way of determining use or not use of the physical resource, based on the duration of each PPDU transmitted over the air) during a preset period of study –capture time-.

Having the capture information imported into a higher level programming language with the capabilities to develop statistical studies of the data set, will enable optimal feature selection and friendly

[7] We define monitor mode [24] as intercepting packets with no exceptions within an IEEE 802.11 BSS or IBSS domain, with no active participation in sending frames.

[8] MATLAB was chosen in this work.

classification algorithm design and testing. Those ways we will be paving the way for future work in this Project (as described in section 1.3); remembering that the idea is to accomplish accurate classification of the PPDUs sequences as IEEE 802.11 traffic independently of sent data and physical parameters of transmissions[9].

[9] Such as modulation, bit rate amongst others.

2.2 Getting Raw-Traffic from an IEEE 802.11 scenario

This WLAN sniffer was developed under Linux in order to attain optimal control over the network dispositive in use. Chosen distribution was Linux Ubuntu Karmic Koala 9.10 with the real-time kernel version *2.6.31-9-rt*, choice parameters were stability and support to manage wireless networks. The wireless adapter in use for the scenario was an AirForce 54g® 802.11a/b/g PCI Express® Transceiver with a BCM 4311 chipset.

To support and facilitate configuration of the wireless adapter we used Wireless tools for Linux, a package of Linux with simple text-based commands. The **iwconfig** tool allowed us to display and change the parameters of the network adapter, particularly those specific to the wireless operation, such as frequency of the WLAN operation channel and monitor mode settings for the adapter[10]. Management and configuration of the network adapter were handled with the **b43** driver, which provides reasonably good performance in terms of packet loss and a Monitor Mode option (available at [9]). Additionally, **b43** works with **Radiotap Header**, which has become a de facto standard for 802.11 frame injection and reception.

"The radiotap header format is a mechanism to supply additional information about frames, from the driver to userspace applications such as libpcap, and from a userspace application to the driver for transmission. Designed initially for NetBSD systems by David Young, the radiotap header format provides more flexibility than the Prism or AVS header formats and allows the driver developer to specify an arbitrary number of fields based on a bitmask presence field in the radiotap header"[7].

[10] Only when the driver supports this configuration.

This Radiotap Header will provide us physical information of the IEEE 802.11 frame, such as modulation, bitrate, PLCP preamble type, TSF[11], and frame failure reception from a FCS check, amongst other utilities. These capabilities are fundamental to reconstruct presence or absence of data in the air during observation time, and allow us to recreate PPDU duration from the captured MAC PDU. The recreation will follow this formula:

PPDU = PLCP HEADER + PLCP PREAMBLE + MPDU

The 802.11 PHY Layer Convergence Procedure (PLCP) transforms each 802.11 frame (MPDU) that a station wishes to send into a PLCP protocol data unit (PPDU) that will be transmitted over the air. This transformation adds different length sub-fields (header and preamble) depending on the standard's version (a/b/g/n).

Our WLAN sniffer captures information inside every single MPDU transmitted over the air into an IEEE 802.11 network, while registering length (in bytes) and transmission bitrate. In this fashion, we are able to calculate the duration in time units of the MPDU. For future developments, we need to be able to determine temporal duration of PPDU, which correspond with the duration of the packet at the air interface. Our application calculates this PHY duration by adding together the MPDU time duration, the PLCP HEADER[12] duration and the PLCP Preamble duration[13] according to PHY characteristics of this PPDU transmission.

As an example, for the 802.11a version, we find the following structure in the PPDU:

---

[11] Time Synchronization Function Timer, when the first bit of the MPDU arrived at the MAC, related to the wireless card clock at that instant.

[12] Fixed length field.

[13] Information provided by the radiotap header.

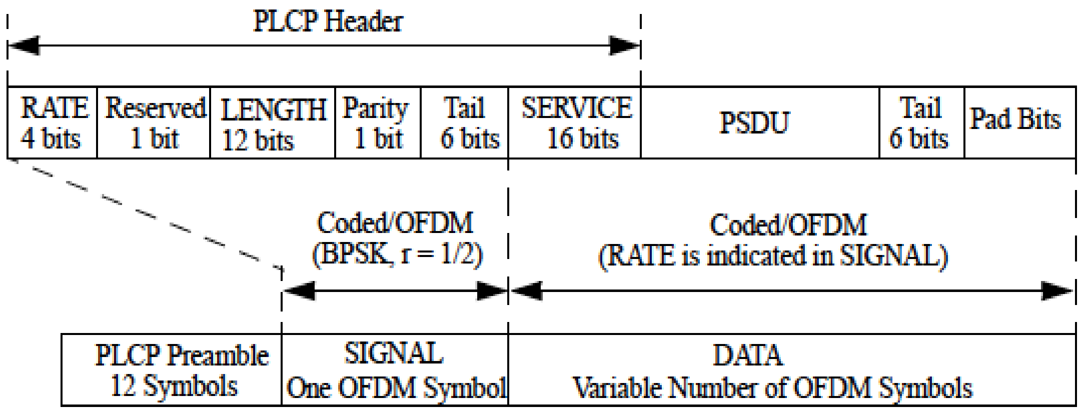


Figure 2.1 - IEEE 802.11a PPDU format

Calculations for airtime of the PLCP Header and PLCP Preamble are reported at the IEEE Standard[1]. These are utilized by our application to report temporal duration of the PPDU transmission over the air.

For the 802.11b version, we have the following structure, where both PCLP Header and Preamble are transmitted over the air at a bitrate of 1 Mbps in the long version or at 2 Mbps for the short one as described on[1], following figures illustrate both long and short preamble cases:

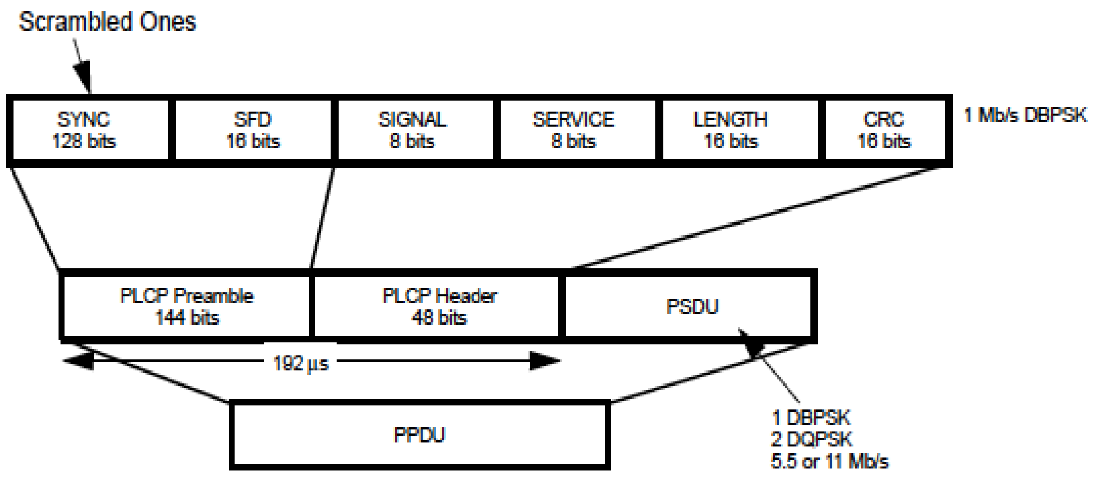


Figure 2.2 - IEEE 802.11b Long-Preamble PPDU fortmat

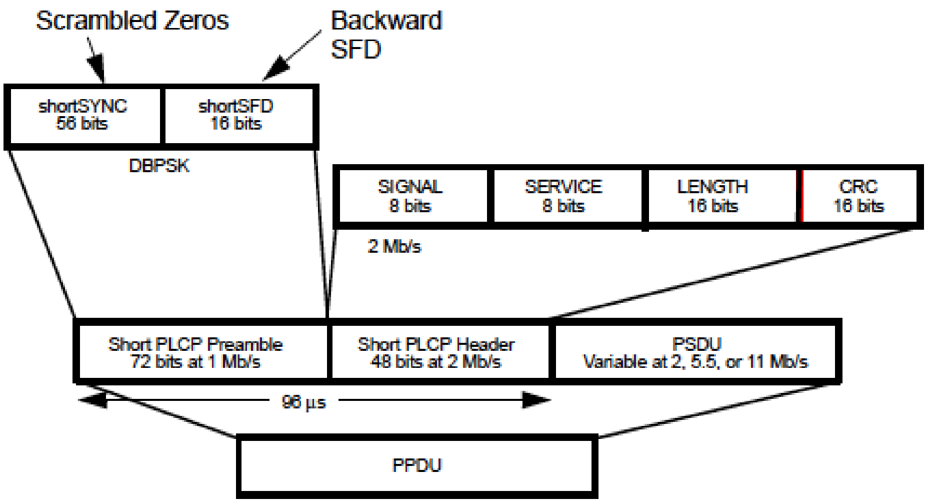


Figure 2.3 - IEEE 802.11b Short-Preamble PPDU format

For 802.11g versions, PPDU structure supports three distinct combinations of PLCP Preamble and Header as described on [1]:

- The first is the long preamble and header based on 802.11b with redefinition of reserved bits defined therein. This PPDU provides interoperability with 802.11b STAs when using the 1, 2, 5.5, and 11 Mb/s data rates; the optional DSSS-OFDM modulation at all OFDM rates; and the optional ERP-PBCC modulation at all ERP-PBCC rates.

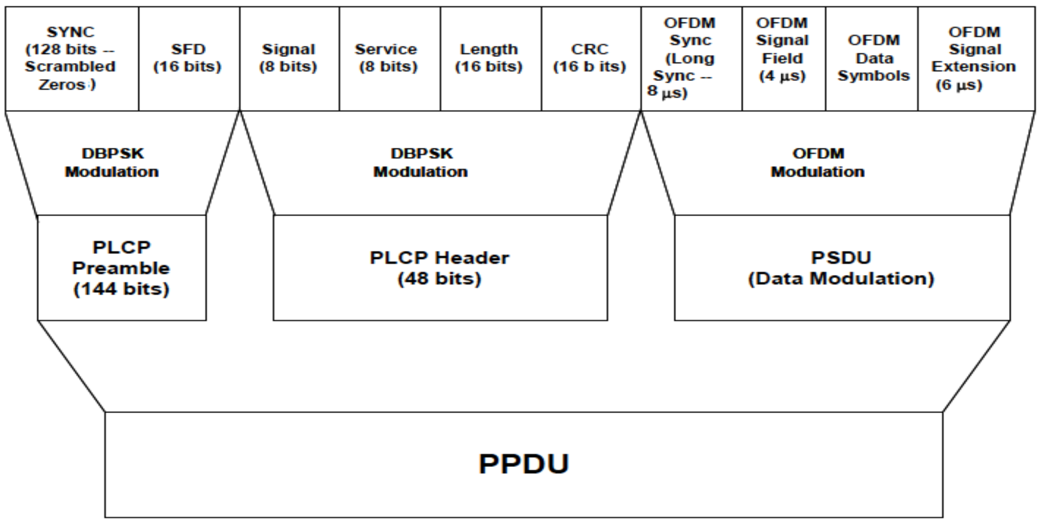


Figure 2.4 - IEEE 802.11g Long-Preamble PPDU format (DSSS-OFDM)

- The second is the short preamble and header based on 802.11b (where it is optional). The short preamble supports the rates 2, 5.5, and 11 Mb/s as well as DSSS-OFDM and ERP-PBCC.

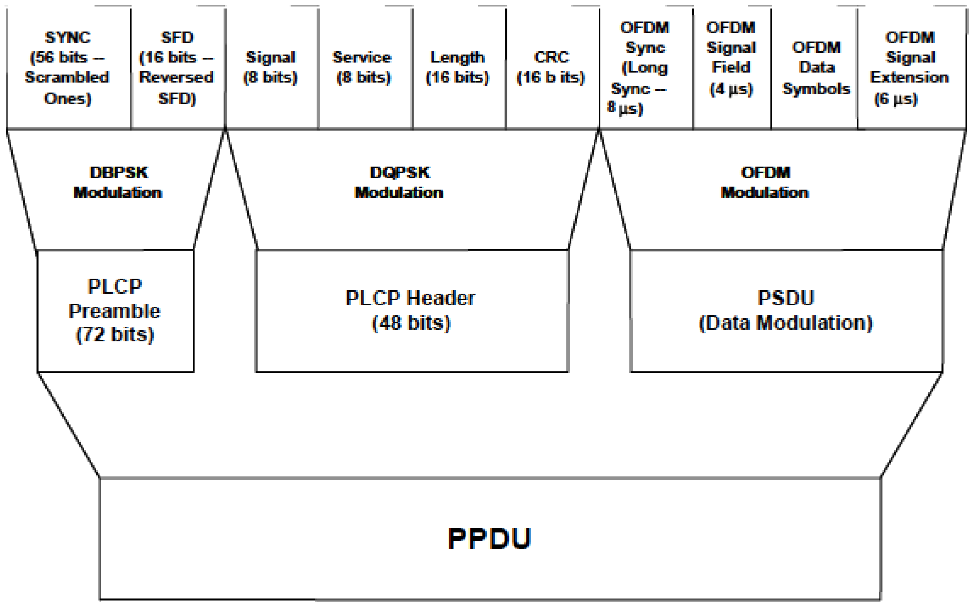


Figure 2.5 - IEEE 802.11g Short-Preamble PPDU format DSSS-OFDM

- The third is the ERP-OFDM preamble and header based on 802.11a (Figure 2.1). For the ERP-OFDM modes, the DATA field that contains the SERVICE field, the PSDU, the TAIL bits, and the PAD bits shall follow.

2.3 SNIFFER DESIGN AND IMPLEMENTATION

The sniffer developed in this work, allowed implementing network sensing, i.e. capturing packets in real time. By choosing JAVA as its programming language, the application inherited the ability to be executed in any operative system, as well as a solid troubleshooting performance.

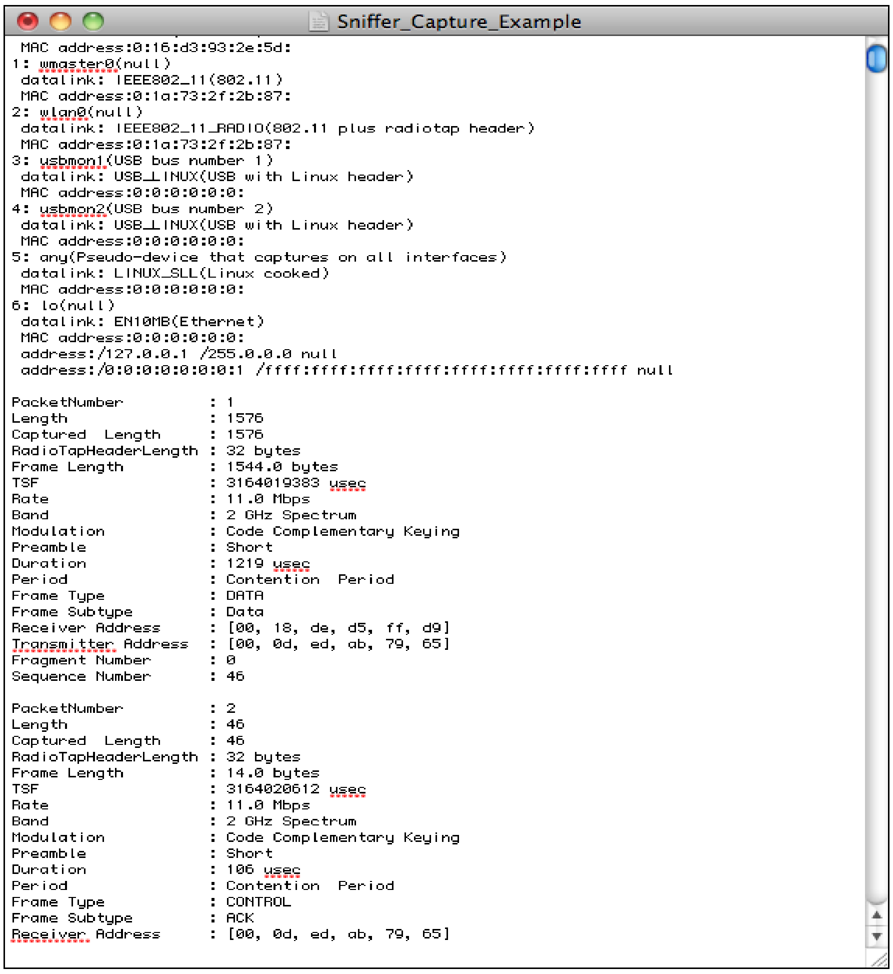
To achieve support for 802.11 packet capturing, the ***jpcap*** library was utilized[14] (available at [6]). This library contains classes written in JNI (Java Native Interface) to wrap capture functionalities of the common use *libpcap* library (available at [16]). For this reason, *libpcap* must be installed previously in the chosen operative system for ***jpcap*** to run. Most Linux distributions pre-install this library. On the other hand, the windows based version (winpcap) enables the sniffer to run also under this platform, but with the limitation of the monitor mode implementation.

Our sniffer serves three main functions: real time packet capturing, enable file capture saving, and packet content visualization. A detailed description of each function follows:

- Capture sessions can be created at any time and are executed in real time. Each session will display a list of all captured packets (refer to figure 2.6) that can be stored as a file.
- Any capture session saved as .txt files can be loaded and its data imported into MATLAB, by way of a .m file, explained in section 2.4.
- Packet content can be visualized in hexadecimal format.

[14] "A Java library for capturing and sending network packets" designed by Keita Fujii as an open source library and licensed under GNU LGPL, this resource is available at http://netresearch.ics.uci.edu/kfujii/jpcap/doc/index.html.

Following figure shows an example of a real capture made by developed sniffer:



```
 MAC address:0:16:d3:93:2e:5d:
1: wmaster0(null)
 datalink: IEEE802_11(802.11)
 MAC address:0:1a:73:2f:2b:87:
2: wlan0(null)
 datalink: IEEE802_11_RADIO(802.11 plus radiotap header)
 MAC address:0:1a:73:2f:2b:87:
3: usbmon1(USB bus number 1)
 datalink: USB_LINUX(USB with Linux header)
 MAC address:0:0:0:0:0:0:
4: usbmon2(USB bus number 2)
 datalink: USB_LINUX(USB with Linux header)
 MAC address:0:0:0:0:0:0:
5: any(Pseudo-device that captures on all interfaces)
 datalink: LINUX_SLL(Linux cooked)
 MAC address:0:0:0:0:0:0:
6: lo(null)
 datalink: EN10MB(Ethernet)
 MAC address:0:0:0:0:0:0:
 address:/127.0.0.1 /255.0.0.0 null
 address:/0:0:0:0:0:0:0:1 /ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff null

PacketNumber          : 1
Length                : 1576
Captured  Length      : 1576
RadioTapHeaderLength : 32 bytes
Frame Length          : 1544.0 bytes
TSF                   : 3164019383 usec
Rate                  : 11.0 Mbps
Band                  : 2 GHz Spectrum
Modulation            : Code Complementary Keying
Preamble              : Short
Duration              : 1219 usec
Period                : Contention  Period
Frame Type            : DATA
Frame Subtype         : Data
Receiver Address      : [00, 18, de, d5, ff, d9]
Transmitter Address   : [00, 0d, ed, ab, 79, 65]
Fragment Number       : 0
Sequence Number       : 46

PacketNumber          : 2
Length                : 46
Captured  Length      : 46
RadioTapHeaderLength : 32 bytes
Frame Length          : 14.0 bytes
TSF                   : 3164020612 usec
Rate                  : 11.0 Mbps
Band                  : 2 GHz Spectrum
Modulation            : Code Complementary Keying
Preamble              : Short
Duration              : 106 usec
Period                : Contention  Period
Frame Type            : CONTROL
Frame Subtype         : ACK
Receiver Address      : [00, 0d, ed, ab, 79, 65]
```

Figure 2.6 - Sniffer capture example

2.4 Importing the sniffer extracted data into a higher-level programming language: MATLAB

The resulting capture files in .txt format, will be used as an input for a MATLAB script[15] to import data collected by the sniffer in its capture session and generate a Time Diagram (using as input PPDU duration and PPDU time of arrival) that recreates the behavior of the medium in terms of presence/absence of PPDU over the air.

The Time Diagram was designed to be transparent to the transmitted data, displaying only PPDU activity without reporting any PHY parameters or data over the air. On the ordinate axis, a "1" value is shown at any microseconds of the capture session when a PPDU was detected in the air, and a "0" value whenever no PPDU was detected.

Regarding the temporal axis, we have a microsecond resolution in a domain that extends from the moment the first PPDU enters the wireless medium, through the moment when the last PPDU exits it.

This Time Diagram recreates the output provided by the energy detector of our AIR-AWARE module. The following two figures represent two distinct capture sessions: the first reports a wireless medium with no traffic and the second depicts a PPDU interchange between a station and an Access Point (Frame types are Data and Acknowledgement).

The information provided by the sniffer's capture afforded to verify that periodic packets in figure 2.7 are Beacon Frames, while in figure 2.8 we found a DATA-ACK communication procedure:

[15] MTD.m see code on section 7.2

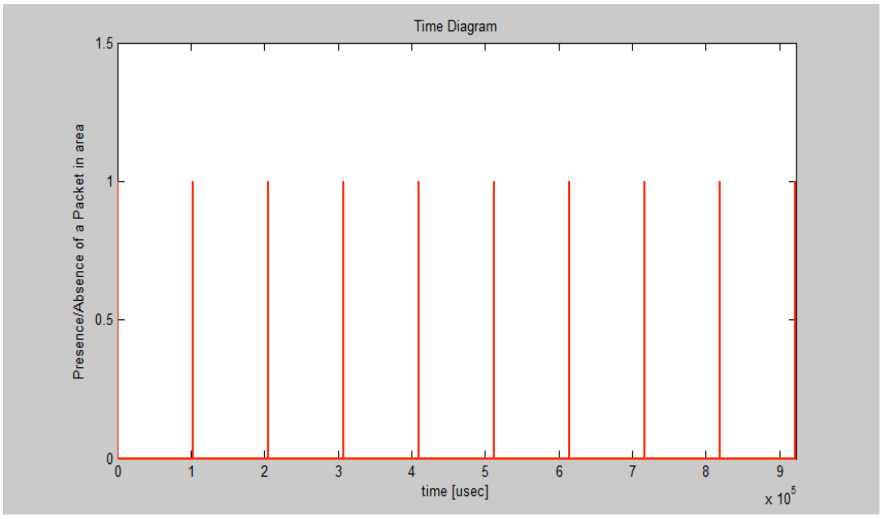


Figure 2.7 - Time Diagram with Beacon Frames

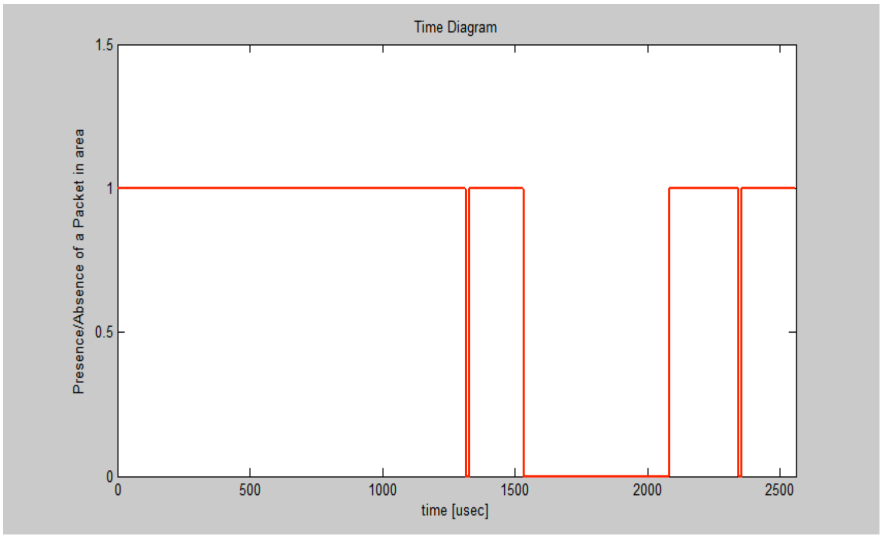


Figure 2.8 - Time Diagram with Data-Ack Procedure

The Time Diagram was designed in a way that there is no further knowledge as to what kind of information the PPDU is carrying over the wireless medium, later on an optimal feature selection to this pattern is rendered and a classification algorithm must concludes that the PPDUs in fact represent IEEE 802.11 traffic[16].

[16] As described in extent on section 1.4.

Chapter 3

IEEE 802.11 REAL TRAFFIC ANALYSIS

3.1 STRATEGY

Once IEEE 802.11 real-traffic is effectively acquired, it's time to analyze these networks PPDUs interchange. When we set off with this project to recognize each technology operating on the 2.4*GHz* ISM band based on temporization of different networks packet interchange, we took as granted the presence of characteristic behavior for each one of them: an accurate differentiation parameter.

The goal of the analysis we develop on this section is to determine different parameters based on MAC sublayer communication procedures, which identify unequivocally a packet interchange sequence as part of the IEEE 802.11 traffic when true. The analysis will utilize different captures of the implemented sniffer (Chapter 2), to test how solid are the chosen parameters[17] in realistic situations. Such parameters will then become our IEEE 802.11 networks features.

These features selection would be made using clean patterns (only with IEEE 802.11 traffic over the air), being careful that packets are not lost to interference while studying the "Sequence Number" frame subfield for each intercepted PPDU during capture sessions. Once the features are selected, will be then evaluated if the presence of different technologies could affect theirs trends [26].

Later on (in Chapter 4), a Bluetooth network packet sequence will be simulated based on specifications reported at [27], and the algorithms used to extract features from 802.11 PPDUs sequences, will be then applied to this Bluetooth packet sequence; this way, will be, at least, proved the differentiation Wi-Fi vs. Bluetooth based on selected features [26].

[17] The idea is to encounter the same behavior for each parameter at multiple measurements in the clean scenario described in detail on section 1.5.

3.2 FEATURES SELECTION

In order to select solid features from captures in different scenarios, we need to determine standard fixed parameters that identify Wi-Fi interchange type packets unequivocally. In other words, we need to find a set of rules –related to temporization of presence/absence of PPDUs over the air- that repeatedly appears in an IEEE 802.11 network.

The first relevant parameter that could help discriminate among other technologies is the interval of possible PPDU duration for each version of IEEE 802.11 traffic. Minimum and maximum values of these parameters for versions 802.11b y 802.11g, as well as a description of physical parameters for each value are reported on Figure 3.1.

Table 3-1 - 802.11 PPDUs Duration (critical values)

	Min PPDU Duration µs	**Max PPDU Duration µs**
802.11b	**106 µs** @ 11 Mbps with Short Preamble Frame Subtype: ACK, RTS, CTS	**18624 µs** @ 1 Mbps with Long Preamble Frame Subtype: DATA *MTU*[18] *= 2304 bytes and*
802.11g	**29 µs** @ 54 Mbps using ERP-OFDM PPDU Frame Subtype: ACK, RTS, CTS	**18624 µs** @ 1 Mbps with Long Preamble Frame Subtype: DATA *MTU*[2] *= 2304 bytes and Fragmentation Threshold disabled.*

[18] *Maximum Transfer Unit* is the maximum payload the link can handle, for IEEE 802.11 is 2304 bytes long (frame payload size before encryption). However, most WLAN drivers use the Ethernet standard of 1500 bytes.

The next proposed parameter, is focused on PPDU interchange temporization: the time interval between frames, known as Interframe Space (IFS). Five different IFS spaces are defined to provide priority levels for access to the wireless media:

a) **SIFS:** Short Interframe Space.

b) **PIFS:** PCF Interframe Space.

c) **DIFS:** DCF Interframe Space

d) **AIFS:** Arbitration Interframe Space (Used by the QoS facility)

e) **EIFS:** Extended Interframe Space

Different IFSs shall be independent of STA bit rate. The IFS timings are defined as time gaps on the medium, and all of them except AIFS are fixed (even in multirate-capable PHYs). The IFS values are determined from attributes specified by the physical layer in each of the versions (a/b/g/n).

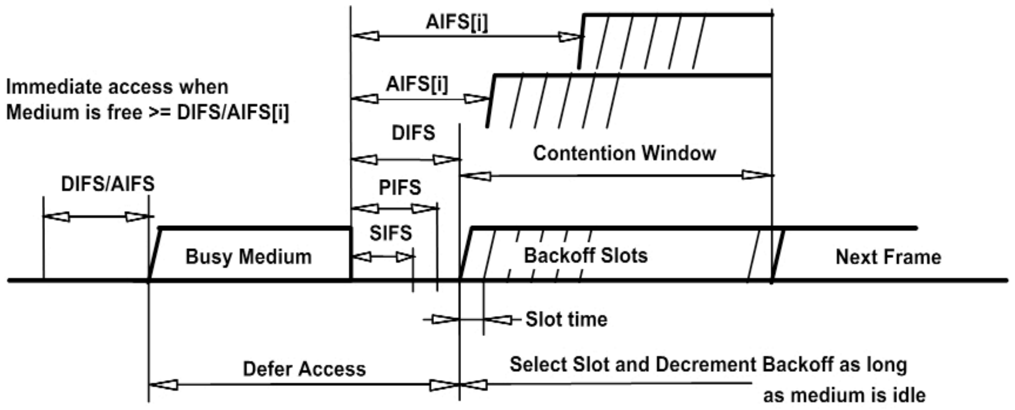

AIFS[i]
AIFS[i]
Immediate access when Medium is free >= DIFS/AIFS[i]
DIFS
Contention Window
DIFS/AIFS
PIFS
SIFS
Busy Medium
Backoff Slots
Next Frame
Slot time
Defer Access
Select Slot and Decrement Backoff as long as medium is idle

Figure 3.1 - Illustration of Multiple IFSs

Of all existing IFS types, SIFS was chosen as the reference. This interval is both shortest and most likely to occur in a scenario with medium to high traffic, because it shall be used prior to transmission of an ACK frame, a CTS frame, the second or subsequent MAC PDU of a fragment burst, and by a STA responding to any polling by the PCF. The SIFS may also be used by a Point Coordinator (PC) for any types of frames during the Contention Free Period (CFP) [1].

The **SIFS** is the time from the end of the last symbol of the previous frame to the beginning of the first symbol of the preamble of the subsequent frame as seen at the air interface. An IEEE 802.11 network shall not allow the space between frames that are defined to be separated by a SIFS time, to vary from the nominal SIFS value by more than ±10% of *aSlotTime* [19] for the version in use.

SIFS has a nominal value of **10 µs** for ISM Sub Band *2.4GHz* operating versions (b/g/n). The following figure illustrates the transmission of a multiple-fragment MSDU (MAC Service Data Unit) from one station to the Access Point using the SIFS:

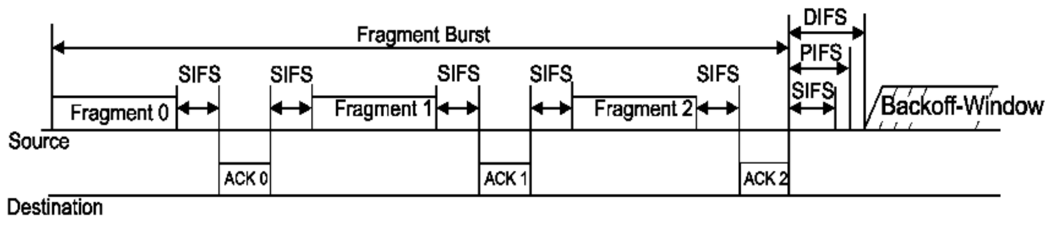


Figure 3.2 - TX of Multiple MSDU's fragments using SIFS

Every time a station needs the channel to transmit to or receive from the access point a quantity of data superior to the Maximum Transfer Unit *MTU* or the Fragmentation Threshold, the illustrated procedure will occur. In addition, whenever a STA or AP send a DATA frame will receive back an ACK frame with a SIFS in the middle. This communication procedures will enable us to identify IEEE 802.11 traffic, finding absence of energy in

[19] Has a 20 µs value for 802.11b version and 9 µs value for OFDM Preamble 802.11g version

consecutive gaps of 10 μs. Experiments to observe this parameter's behavior across multiple captures are available on section 3.3.2.

For this feature extraction, we must individuate from IEEE 802.11 acquired traffic gaps those corresponding with SIFS interval and discard others. To extract the gap on *DATA-ACK* procedures, the most common, the propose criteria is an algorithm (See Appendix, FeatureExtraction.m) that automatically extracts the SIFS gap. Estimating its statistical behavior, SIFS could be well differentiated from a non-SIFS when the extraction is only rendered in the case that two consecutive PPDUs, durations were such that: $0.65*PPDU_{ith} > PPDU_{ith+1}$.

The factor has been justified thinking at the difference between the length of an ACK and DATA frames; this difference in bytes could go from 16 to 2300. Aditionally, we'll only choose values shorter than 625 μs (Bluetooth time-slot) to build the statistic of this parameter, excluding bigger gaps that would neither correspond to the Data-Ack 802.11 procedure nor a Bluetooth[20] silence gap between a PHY packet and its ACK. This way we are also discarding uncorrelated PHY packets (long silence gap between them) that fit the above-mentioned duration condition. This feature's will be characterized analyzed on section 3.3.2.

The last parameter was selected thinking about very low-traffic scenarios (where the *DATA-ACK* procedure is uncommon) or even no traffic (having just an Access Point sending Beacon Frames). In these cases, we could take advantage of the periodicity of Beacon Frames, since every IEEE 802.11 network has an Access Point (Basic Service Set case) or multiple stations (Independent Basic Service Set case) sending this kind of frames regularly with network's information and PHY parameters. The beginning of two Beacon Frames is separated by a *Beacon Interval*, defined

[20] Thinking always in a posteriori Wi-Fi vs. Bluetooth classification.

in the standard as a multiple positive integer of the *Time Unit* (TU = 1024 μs), which by default is fixed at every Access Point to 100:

Common Beacon Interval = 100*1024 μs = 102400 μs

Because we have no certainty that this value will always be equal to the *Beacon Interval* assumption and our energy detector is unable to read its related data inside Beacon Frames, an algorithm is proposed (See Appendix, FeatureExtraction.m code) to detect this type of frames.

Designing the algorithm: For each 802.11 network, the length –in bytes- for this type of frames is fixed and the transmission rate is the lowest possible of the *Basic Rate Set (BRS)*. PPDUs with the same Temporal Duration are grouped for captures approximately two (2) seconds[21] long. The proposed algorithm generates a vector [x] for each group of PPDUs with the same Duration, and each vector's cell stores inter-arrival time between PPDU_{ith} and the $\text{PPDU}_{\text{ith+1.}}$ Then, the median for the vector is calculated and the percentage of consecutive PPDUs for each group with an inter-arrival time of 100 μs over and under the median calculated is determined[22].

To calculate percentage PPDUs under this condition, we use the median instead of the average to build a solid algorithm that resists the phenomenon of packet loss. In the uncommon cases of packet loss or no detection, the inter-arrival time between two PPDUs gets duplicated; these values will go to the extreme of vector [x] and will be discarded when setting the threshold of expected inter-arrival time:

[21] With the default beacon interval, we will be able to capture around 20.

[22] We called this parameter Expected Interarrival Time, keeping in mind that 100 μs is three orders of magnitude under the Beacon Interval default value.

$\mu_{1/2}(x)\cdot$ -100 μs < Expected Intearrival Time < $\mu_{1/2}(x)\cdot$ +100 μs

If the majority of PPDUs in a group meet the condition, we can safely conclude that they correspond to Beacon Frames. In section 3.3.3 we report the results of the algorithm tests.

3.3 FEATURES CHARACTERISTICS FOR MULTIPLE CAPTURES

In this section, we report the behaviors of the selected features at different network scenarios. First, a snapshot of our experimental setup: The tests were run in the ACTS laboratory, situated on the 2nd floor of the Information and Communication department (InfoCom department), in the School of Engineering of the University of Rome "La Sapienza". To run the tests we used two (2) Access Points (APs), three (3) Stations (STAs) and one (1) Sniffer Station (SS) described on Chapter 2. Following is a description of each component:

- **AP1:** Cisco Aironet 1200 Series (802.11 b) set at channel 8 for all tests.
- **AP2:** U.S. Robotics USR808054 (802.11b/g) set at channel 1 for all tests.
- **STA1:** Sony VAIO VGN SZ-450 N/C with INTEL PRO WIRELESS 3945ABG adapter.
- **STA2**: HP Pavilion dv6000 Series with Broadcom BCM 4311 802.11 a/b/g wireless adapter.
- **STA3:** Asus *eeepc* 1201 with Atheros AR9285 wireless adapter.
- **SS:** HP pavilion dv2000 Series with AirForce 54g 802.11 a/b/g PCI Express, with a Broadcom 4311 chipset.

For every test, AP1 was situated at 5 meters from the table where all STAs were located, while AP2 was on STAs table at 50 centimeters from them. STAs and SS where close to each other: 10 centimeters between each of them. During test runs, AP1 was permanently set to channel eight (8) without presence of interference entities in that channel and AP2 was set to channel 1.

3.3.1 PPDUs Duration Feature

To study the behavior of this feature, we run tests using AP1 (Cisco Aironet 1200 Series) in two scenarios, either with medium to high traffic or with no traffic. A thousand (1000) packets were captured in each run. Please refer to these histograms to illustrate our results:

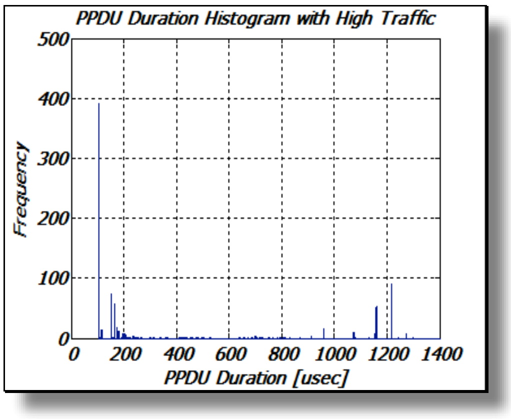


Figure 3.3

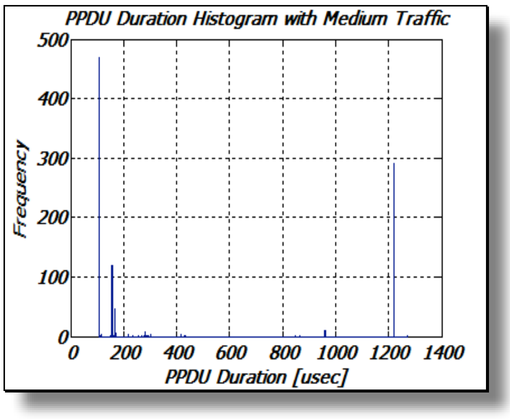


Figure 3.4

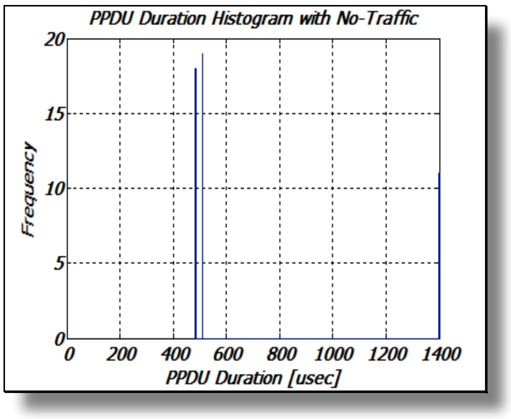


Figure 3.5

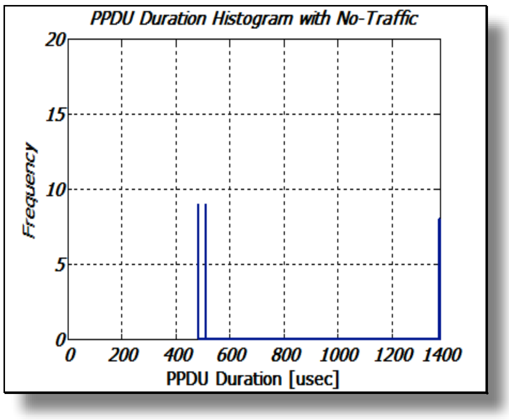


Figure 3.6

Figure 3.3 displays results of a test in which 1000 packets were captured in 1.8 seconds in a high traffic scenario; with STA3 processing a video call and STA1 downloading two (2) files through AP1.

Another 1000 packets were capture in 2 seconds at a medium traffic scenario with STA2 downloading two files through AP1, as displayed on figure 3.4. Only participating STAs were associated to the network during these trials.

Figures 3.5 and 3.6 reflect the results inside a no-traffic scenario. The first capture rendered 26 packets in 0.9 seconds, while the second resulted in 47 packets in 1.8 seconds. Here, we had three (3) APs sending beacon frames with no STAs associated to them. Both AP2 and the two interfering APs are set to channel 1, with the Sniffer Station (SS) amongst their coverage range.

The capture file allowed us to verify that all APs had set the same *Beacon Interval*; therefore, in a situation with no packet loss we should expect pretty much the same number of PPDUs to be captured from each AP. However, a look at the histogram shows us differently; while same AP's beacon frames have exactly the same duration, the peak in the middle represents captures of AP2's frames (the maximum value in both graphics). We were able to reach this conclusion because AP2 was situated at 50 centimeters and we had high signal power level at the SS receiver. Oppositely, for the interfering APs, we verified lower signal power levels at the SS receiver, particularly on the one that sends longer beacon frames. Here we see the representation of the packet loss phenomenon.

Regarding the duration value in the four (4) captures, we are able to confirm that every sample falls within the range defined at figure 3.1 for the version 802.11b [106 µs, 18624 µs]. Lastly, figures 3.4 and 3.5 display concentration of PPDU Duration values around 200 µs (corresponding to ACK frames) and at 1200 µs (corresponding Data frames), from this we infer that the *Data-ACK* procedure is very common in medium to high traffic scenarios.

3.3.2 SIFS Feature

This feature was selected to fit medium to high traffic scenarios where the *Data-ACK* procedure is very common, as showed in precedent section. To test this feature, we run six (6) captures of 1000 packets, in three (3) different scenarios. Each scenario was enacted twice, in different days. The following graphics display our results regarding this feature behavior over the six captures; each row represents the behavior for exactly the same scenario:

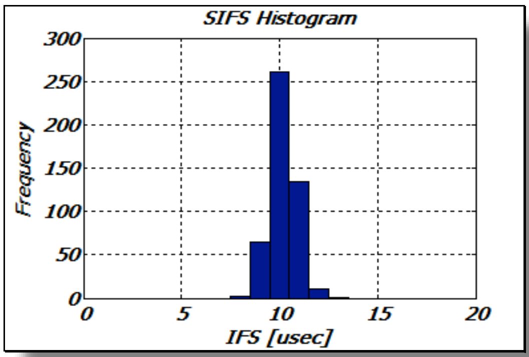


Figure 3.7

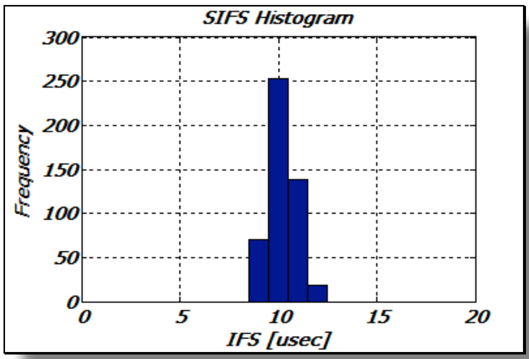


Figure 3.8

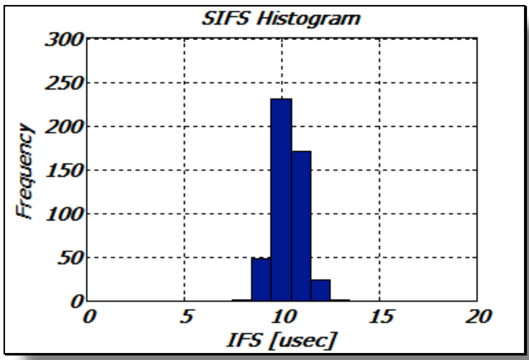


Figure 3.9

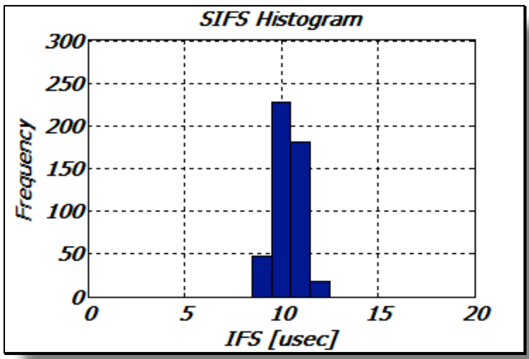


Figure 3.10

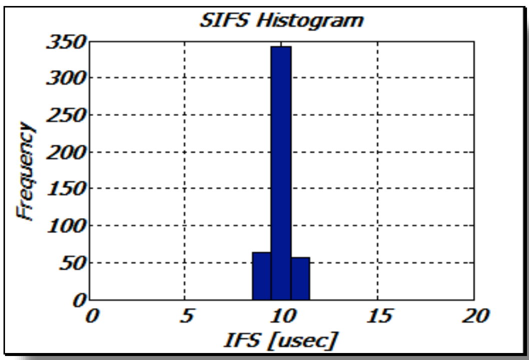


Figure 3.11

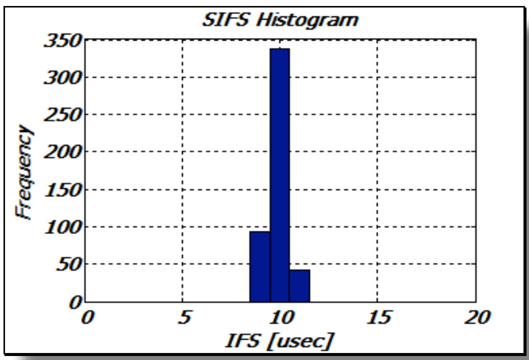


Figure 3.12

The AP1 (Cisco Aironet 1200 Series) was used in all tests. Figures 3.7 and 3.8 correspond to different dates in which AP1 was associated to STA1 and STA3. STA1 was processing a video call while downloading one file, and STA3 was surfing the web.

The results on figures 3.9 and 3.10 represent a scenario in which STA1, STA2 and STA3 were associated to AP1. STA1 was downloading two (2) files, while STA2 and STA3 were on a video call together. STA3 was also downloading one (1) file at the time.

Finally, results reported on graphics 3.11 and 3.12 reflect STA2 associated to AP1 while downloading one (1) file.

We noted that test runs performed in analogue scenarios reported very similar results. Also, all trials were run over a 1000 packet capture and, for each case, 40% of the PPDUs sequence's gaps were reported as SIFS. Therefore, made assumption that this parameter will be very common at medium or high traffic scenarios proved to be right. Following Table 3-2 illustrates this fact:

Table 3-2 – Results of SIFS Feature extraction

Capture	Silence Gaps	Gaps reported as SIFS by the proposed algorithm	% Of total gaps reported as SIFS	Non-SIFS gaps reported as SIFS by the algorithm
First	999	475	47.55%	2
Second	999	453	45.34%	2
Third	999	481	48.14%	5
Fourth	999	471	47.14%	3
Fifth	999	451	45.14%	4
Sixth	999	452	45.23%	6

Additionally, the fact that in all tests ran this parameter takes mostly the value of 10 μs (as defined at [1]) was confirmed, and so was the deviation of this value, always within the range of tolerance ±10% of *aSlotTime* (2 μs for 802.11b version), plus 1 μs of MAC Processing that could be induced by the Sniffer Station (SS). We also observe more deviation of this nominal value when a specific STA (STA 3) is associated to the AP, but still within the expected behavior.

It could also be noted in Table 3-2 that some gaps reported as SIFS by the algorithm are not really SIFS. Looking theirs measured value, is still shorter than 625 μs Bluetooth time-slot so its reported, but very far from 10 μs expected, and consequently are not visible in histograms -scale issue-. As reported on [26], this will not be a big problem at Wi-Fi vs. Bluetooth classification because theirs low frequency of occurrence (inferior to 1% most cases).

As a feature, the SIFS value is evidently solid respect interference of other technologies operating in the ISM band. It's almost impossible to find regularly a similar distribution of a parameter, with this same frequency within the capture and with this small deviation from its expected value in other technologies or interference entities. Anyhow, at chapter 4, a Bluetooth communication will be simulated, and this same algorithm will be applied to that packet sequence; that way we will really know if this feature helps to differentiate at least Bluetooth vs. Wi-Fi networks.

3.3.3 Beacon Periodicity Feature

This feature was selected with low or no traffic scenarios in mind, where SIFS are uncommon and not definitive to conclude whether or not traffic over the air corresponds to IEEE 802.11. During the 4 trials –with captures of approximately 2 seconds- the Sniffer Station (SS) was sensing on Wi-Fi channel 1, in which AP1 and two interfering APs were operating with no stations associated to them –sending just beacon frames-.

Along the four (4) tests (A, B, C, D), our SS receiver was at AP2 signal power level of -64 dBm, and interfering APs with ID “WiFiArea” and “NETGEAR”, with a signal power lever at SS receiver of -84 dBm and -79 dBm, respectively. In addition, was calculated the percentage of same length PPDUs (all of them beacon frames, in this case) with a “fixed”[23] inter-arrival time, as described by the end of section 3.2.

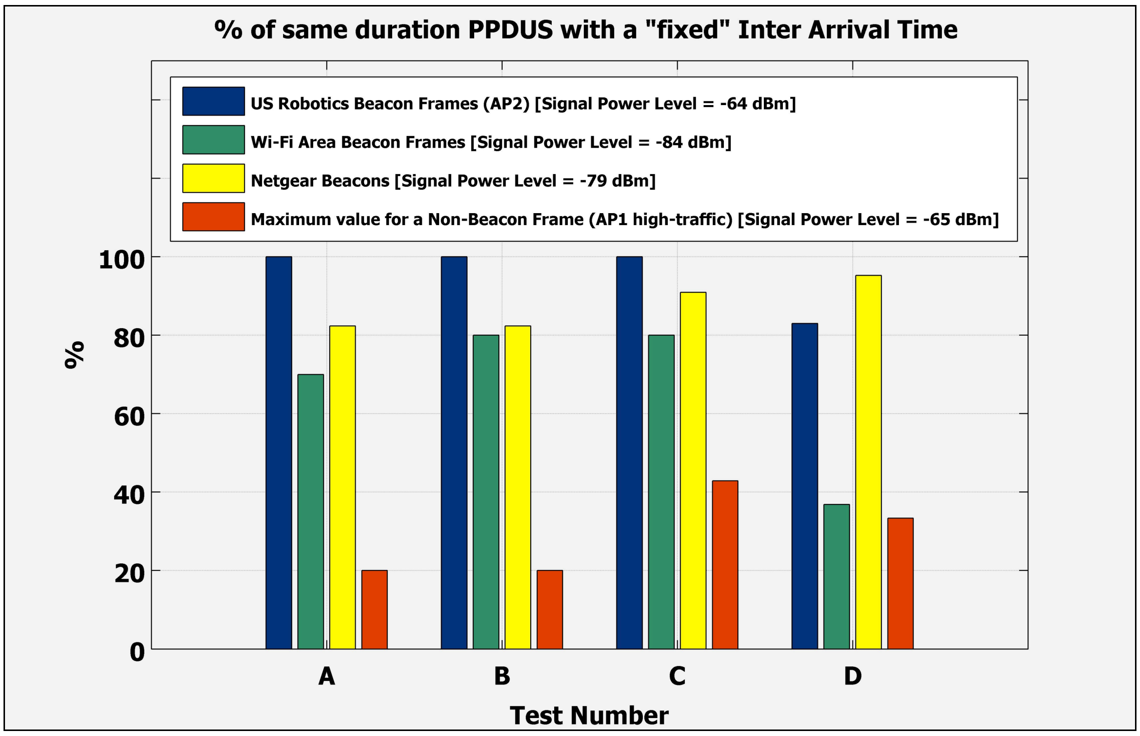


Figure 3.13 - Beacon Periodicity Feature

[23] Within an Expected Inter-Arrival time range. It has a tolerance ±100 μs respect to the median of inter-arrival times of all grouped same length PPDUs

Figure 3.13 shows how AP2 with the higher signal power level at the (SS) receiver presented 100% of PPDUs inter-arrival times within range for 3 out of 4 tests. During the last test, two (2) packet losses were registered. We are also able to observe that a decline in signal power level at the SS receiver makes more frequent the reception of PPDUs (beacon frames for this case) with inter-arrival times outside of expected range (packet loss will explain this behavior).

Finally, to evaluate the solidity of this feature, the same algorithm was also applied to four (4) captures (see red bar on figure 3.13) of 1000 packets used in the test of SIFS' feature behavior (through AP1). The idea was to study its behavior when applied to non-periodical PPDU's sequences without pre-established inter-arrival time. To accomplish this, we grouped all same length PPDUs in four (4) captures (A,B,C,D also) on a high traffic scenario –from the preceding section- and calculated the Expected Inter-Arrival Time range for each group. The last step was to calculate the percentage of packets within the tolerance range for each group –except beacon frames group of AP1- and report on figure 3.13 the "maximum percentage value" at any group for each of the four (4) tests (A, B, C, D red bar).

We were able to conclude that this feature is in capacity to reveal and discriminate a certain periodicity in presence of packets of the same duration with good precision[24] and, therefore, is adequate to detect IEEE 802.11 beacon frames. We can responsibly assume that this feature is not sufficient per se, because it doesn't enable differentiation between this traffic and other technologies with fixed length packets and fixed inter-arrival times or an interfering entity with certain periodicity. This feature will then be useful in combination with others like PPDU Duration to

[24] As seen in figure 3.13, it will be necessary to detect (receive) the majority of packets in a measurement; otherwise, the performance of this feature could be compromised.

achieve accuracy, or even with other technologies features that will allow us to make a more accurate conclusion.

3.4 Fitting SIFS feature to a Probability Density Function using experimental data

The objective of this section is to fit to a common probability density function (p.d.f.) the most solid and characteristic feature founded on experiments. Test here are Gaussian based because its simplicity; and particularly for the SIFS feature, measurements seems to fit very well to this model.

The normal distribution or Gaussian distribution is a continuous probability distribution that describes data that cluster around the mean. The simplest case of a normal distribution is known as the standard normal distribution, described by the probability density function:

Equation 1 - Standard Normal Distribution PDF

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$$

The constant 1/(√(2 π)) in this expression ensures that the total area under the curve Ø(x) is equal to one, and 1/2 in the exponent makes the "width" of the curve (measured as half of the distance between the inflection points of the curve) also equal to one. It is far more common to describe a normal distribution by its mean μ and its standard deviation σ . Changing to these new parameters allows us to rewrite the probability density function in a convenient standard form:

Equation 2 - Gaussian PDF in a convenient standard form

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Notice that for a standard normal distribution, mean μ=0 and variance σ^2=1. The last part of the equation above shows that any other normal distribution can be regarded as a version of the standard normal distribution that has been stretched horizontally by a factor σ and then translated rightward by a distance μ. Thus, μ specifies the position of the bell curve's central peak, and σ specifies the "width" of the bell curve.

To do the fitting, we will refer to the six (6) experiments detailed on section 3.3.2. The data sets used to generate all of the histograms reported on figures 3.7 through 3.12, will be now analyzed statistically and its mean and standard deviation would be calculated. This both parameters would be used to define a Gaussian pdf, which would be superimposed to a normalized version of the previously mentioned histograms (as showed on figures 3.14 through 3.19).

Regarding the six (6) data sets, we define as x_i the i-th measurement of the SIFS value (i-th cell of the data set vector) and as N the number of measurements of this parameter made on the six experiments (length of the data set vector). This way we estimate the Gaussian model mean as:

Equation 3 - Calculation of the mean from the histogram

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i,$$

and finally we estimate the model variance as:

Equation 4 - Calculation of the variance from the histogram

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N}\left(x_i - \hat{\mu}\right)^2$$

Following, we present the normalized histograms from section 3.3.2, with the superposition the proposed Gaussian model for each case:

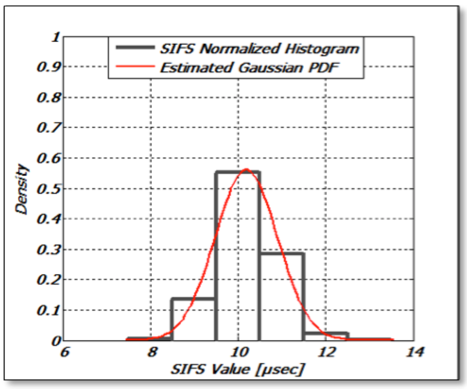


Figure 3.14

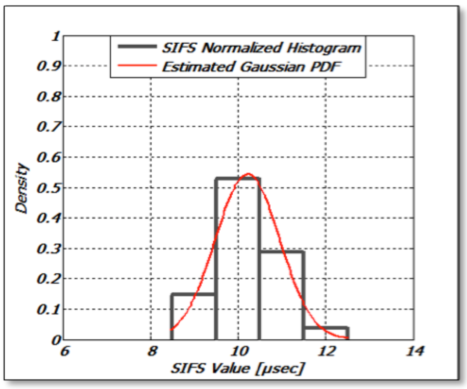


Figure 3.15

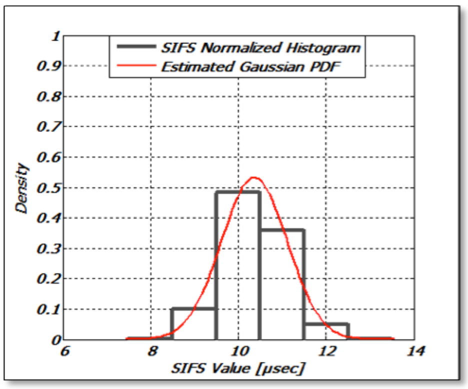


Figure 3.16

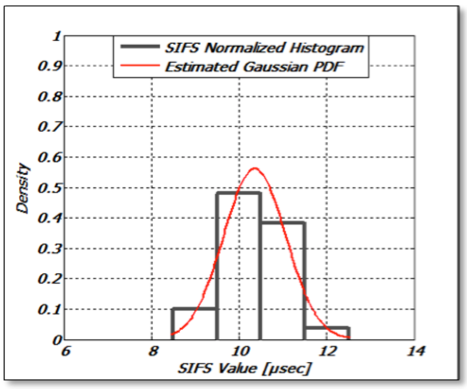


Figure 3.17

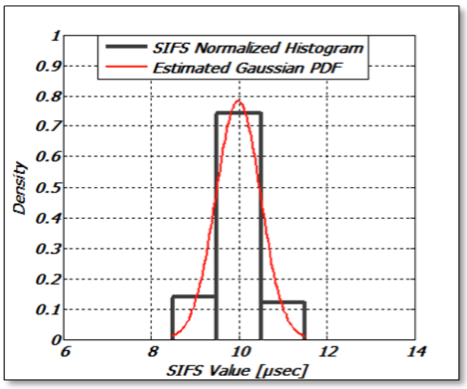


Figure 3.18

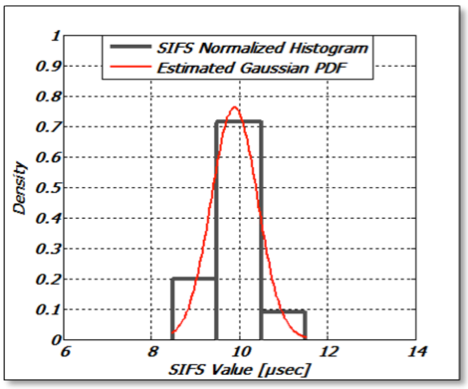


Figure 3.19

In Table 3-3, obtained results after applying Equation 3 and Equation 4 to the data sets generated by the six captures are:

Table 3-3 – Estimated Mean and Variance for Gaussian model

Parameter	Fig 3.14	Fig 3.15	Fig 3.16	Fig 3.17	Fig 3.18	Fig 3.19
Mean [μs]	10.1886	10.2171	10.3621	10.3586	9.9827	9.8919
Variance [μs²]	0.5058	0.5385	0.5606	0.5053	0.2595	0.2750

Analyzing results, in a general way, we propose to assume a Gaussian Probability Density Function with *Mean* 10 μs and *Variance* 0.5 μs^2 to model an extraction of the SIFS Feature if needed. The following figures display the above proposed p.d.f., superimposed to the resulting histograms for each of the six (6) experiments. This way, we can visually confirm that the feature follows this feature's trend adequately:

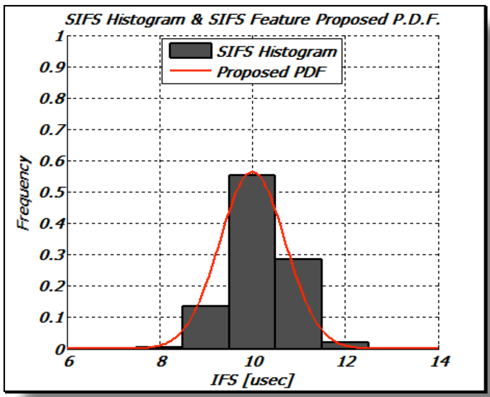


Figure 3.20

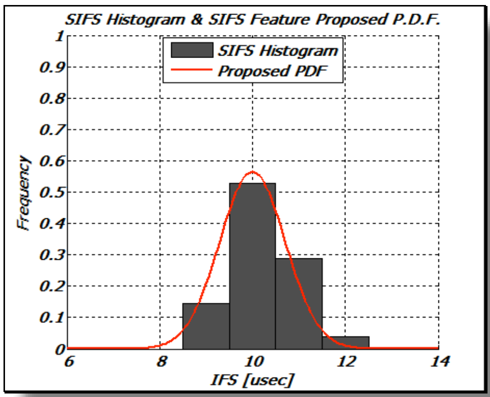


Figure 3.21

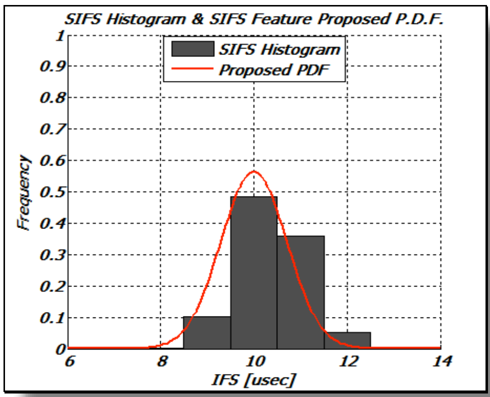


Figure 3.22

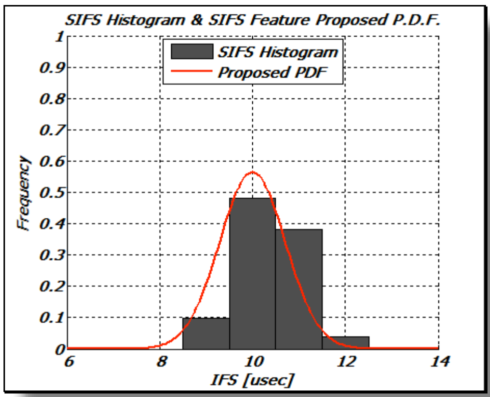


Figure 3.23

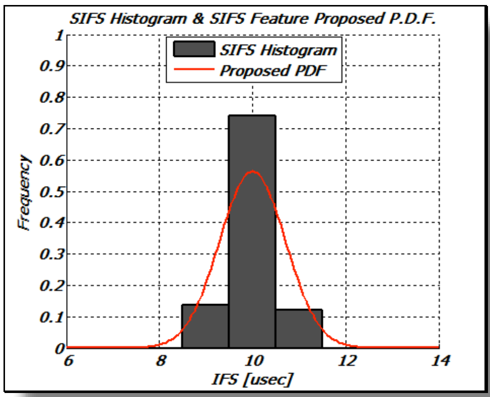


Figure 3.24

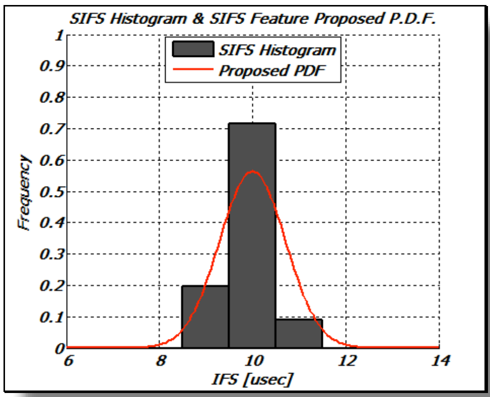


Figure 3.25

Chapter 4

TESTING SELECTED FEATURES IN A MULTI-NETWORK ENVIRONMENT

4.1 Analyzing SIFS feature extraction inside a Bluetooth Network communication pattern

4.2 Future work

4.1 Analyzing SIFS feature extraction inside a Bluetooth Network communication pattern

The experiments in Chapter 3 proved that the SIFS feature was particularly useful for Wi-Fi networks identification. Data from across all PPDUs captures analyzed render results practically identical for this feature's measurement. Nonetheless, no clues have been collected about this feature's behavior when extracted from a pattern generated by others technologies networks communication.

Evidently, it is necessary to at least prove that extraction of the feature in the above-mentioned patterns will render clearly different results, in respect to those reported after extraction in a Wi-Fi network pattern. By doing this, we could securely state that this feature could allow differentiating this set of technologies.

As a first case study, it was decided to extract this feature from a Bluetooth network packet interchange; because of the wide diffusion of this network type. To do such extraction, the same algorithm utilized for measurements in section 3.3.2 will be applied –where extraction will only be rendered in case that two consecutive PPDUs duration were such that: *0.65*PPDUith > PPDUith+1*-.

To generate the Bluetooth packet sequence as extracted by the energy detector, simulated Bluetooth PHY packets were recreated using MATLAB. The reference for this simulation is the IEEE Standard 802.15.1 – 2002 [27], i.e. bitrate of 1 *Mbit/s*. Piconets of two devices, one master and one slave, in connection state (one master and one slave) were considered. The two devices send their packets alternately: one device (the master, for example) sends its data packets, and for every received packet, the other device (the slave) sends back an acknowledgement –as reported on [27]-. Data packets sent by the master can occupy 1, 3 or 5 time slots (where the time slot is 625 *μs*), according to their length, whereas acknowledgement packets (NULL packets, with a fixed length of 126 *bits*) occupy 1 time slot.

Regarding to the communication Scenario, the length of the simulated packets was modeled as follows:

- 80% Data Packets sent by the master will occupy 1 Time Slot
- 15% Data Packets sent by the master will occupy 3 Time Slots
- 5% Data Packets occupy sent by the master will occupy 5 Time Slots

Additionally, 70 % of the data packets have the maximum possible duration that is fixed by the protocol to 366 μs, as showed in Figure 4.1.

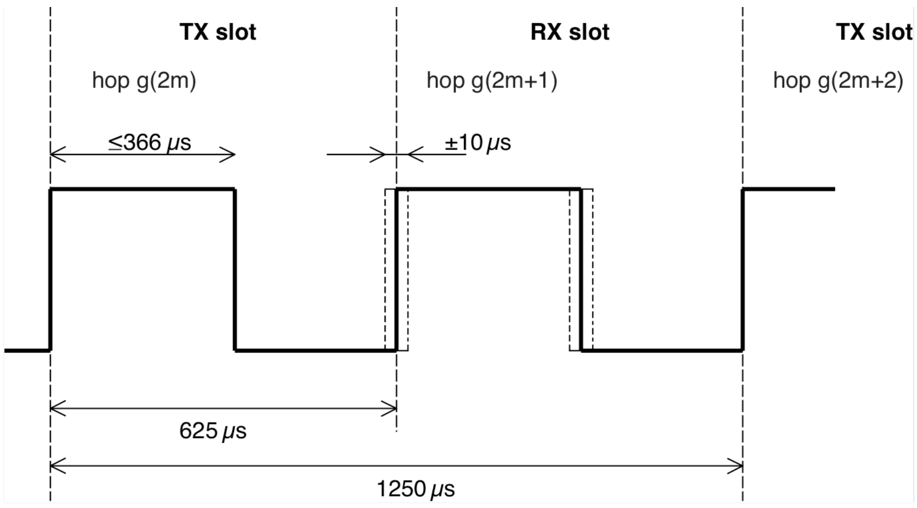


Figure 4.1 -Rx-Tx cycle of Bluetooth master transceiver in connection state

The duration of the remaining 30% is uniformly distributed between minimum and maximum values (see Table 4-1). According to the standard, for every packet arrival time a jitter of ± 10 *μs* has been set, to consider imperfect synchronization between the two devices. The jitter was modeled by a Gaussian distribution with zero mean and standard deviation "=10/3 *μs*; given the model, 99% of jitter values fell within a ± 10*μs* interval, while the remaining 1% exceeded this interval and were readjusted to edge values in order to meet the standard specifications.

Table 4-1 - Bluetooth simulated packet parameters

	Fixed duration	Min. Duration	Max. Duration
Time slot	625 μs		
1-time-slot-packet		126 μs	366 μs
3-time-slot-packet		1250 μs	1622 μs
5-time-slot-packet		2500 μs	2870 μs
NULL packet	126 μs		

Figure 4.2 reports the extraction results of Wi-Fi proposed SIFS feature in a 1000 Bluetooth packets simulated capture. The characteristic of this feature in a Bluetooth communication pattern is displayed in blue; whereas characteristic in one of the Wi-Fi 1000-packets capture as reported in section 3.3.2, are shown in red:

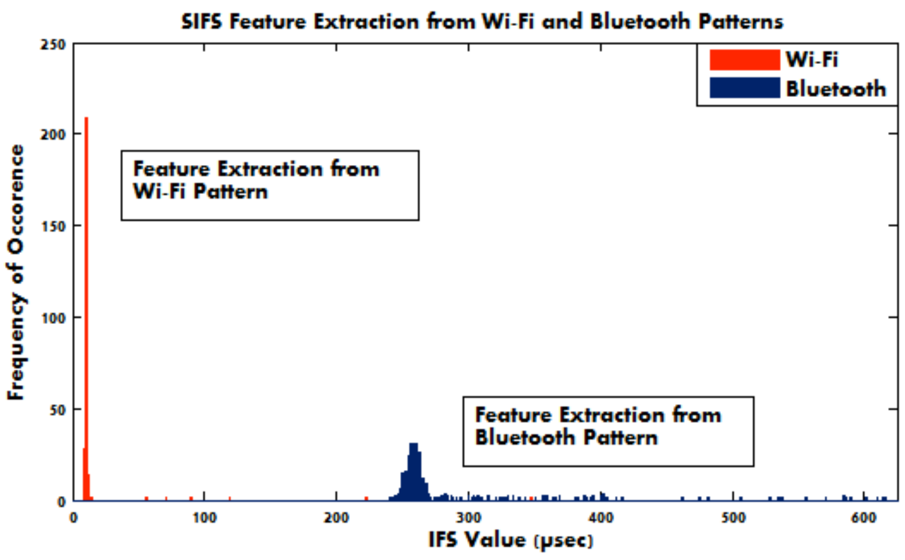


Figure 4.2 - SIFS Feature Extraction from Wi-Fi and Bluetooth patterns

As desirable, the extractions made from different packet sequence patterns (Wi-Fi vs. Bluetooth) showed very different results. This is a good sign for classification of these two technologies. Such difference is a consequence of what is reported in Bluetooth standard, where Bluetooth time slot a maximum duration for PHY packet transmitted over the air is fixed:

“In the connection mode, the Bluetooth transceiver transmits and receives alternately (see Figure 4.1). In the figure, depending on the type and the payload length, the packet size can be up to 366 µs”[27].

Even in the worst of cases, for Bluetooth maximum length packets, we will find a silence gap between PHY packets communications procedures of at least 249 µs, far away from the 10 µs expected in a Wi-Fi PPDUs sequence.

As reported in the paper, “Automatic network recognition by feature extraction: A case study in the ISM band” [available on appendix section 7.3], this feature, as proposed here, allowed a great performance for the purposes of Wi-Fi vs. Bluetooth network differentiation.

4.2 FUTURE WORK

After proposing a feature that afforded precise differentiation between Wi-Fi and Bluetooth technologies in the tests rendered [26], the next natural step will be identify communication procedures with an unique behaviors for other ISM band operating technologies, as Bluetooth, ZigBee, CCTV, etc. This way, additional features that identify these technologies could be added eventually to a classifier, improving its performance.

In addition, it came to knowledge that for ZigBee networks, a Short Interframe Space is also defined in the specifications, with duration of 12 symbols that equate to 192 μs on the version that operates over the ISM band. This information leads us to believe that capturing packet sequences from a ZigBee network and applying the proposed SIFS feature extraction utilized for Wi-Fi, ZigBee networks will appear as a new class in the middle of Wi-Fi and Bluetooth pattern's extracted values.

Finally, implementation of a classification block, as foreseen in the AIR-AWARE module, that displays all proposed features from a multi-network environment pattern delivered by the energy detector will become necessary. The module, using specific algorithms, will be able to find clues in the pattern that will enable it to identify each one of the networks of different technologies in play.

In the future, when analyzing and subsequent extractions are performed also for interference entities, this device could also help manage interference, i.e. microwave oven.

Chapter 5

CONCLUSIONS

Utilizing the experimental packet captures from Wi-Fi networks and simulated captures for Bluetooth networks towards feature extraction statistical analysis, we were able to conclude that the SIFS feature[25] proposed in this work was adequate to recognize what type of technology network is operating -in a medium or high traffic scenario- over the air, and, as reported on [26], enabled us to discriminate between Wi-Fi and Bluetooth networks, which are the two most commonly found technologies over the ISM band.

The implication of this fact is clear: a "black box" spectrum sensing –the AIR-AWARE module- device that discriminates between technologies operating in the ISM band, in a heterogeneous environment-through linear classifiers- can be created and implemented in a cognitive radio context.

We set out on the assumption that packet interchange patterns were technology specific; and, as such, by identifying pattern clues, network recognition could be achieved. For this study on the ISM 2.4GHz band, this assumption was verified to be correct, by granting network recognition through the extracted SIFS feature –as showed results on elaborated paper regarding differentiation Wi-Fi vs. Bluetooth-.

The present work extends beyond previous investigations by considering Wi-Fi real traffic captures, and by focusing feature extraction

[25] Defined for 802.11 communication systmes in [1], is the time interval between PPDUs, defined in as Short Inter Frame Space corresponding to silence gaps on the medium when DATA-ACK procedures are in play, and also in some other less frequent procedures.

and classification on MAC sublayer characteristics, leading to simplicity and computational efficiency.

Features such as the proposed SIFS can be extracted using a generic device that detects only temporization of presence/absence of energy in the medium. Using temporization presence/absence of packets can be mapped overtime in that pattern, allowing identification of the technology, regardless of the information being transmitted. Extracted features are based on MAC sublayer communication procedures.

The device will improve spectrum administration by detecting types of technologies and the obtained to information could serve as an aid to assign the most adequate fit of PHY parameters for transmission at each node -amongst all available- matching user needs.

On the other hand, PPDUs duration feature help us understand better the typical duration of PPDUs in a real world Wi-Fi network. In the future, this information could become useful to differentiate Wi-Fi networks from slotted technologies with shorter packet duration.

In the next stage of the project, the Wi-Fi packet captures achieved during experimentation will serve as a training set for the linear classifiers for this class; while the simulated Bluetooth captures will become the training set for their class.

Bibliography

REFERENCES

[1] LAN MAN Standards committee of the IEEE Computer Society. Part 11 Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. ANSI/IEEE Std. 802.11, 2007 Edition.

[2] Danijela Cabric, Artem Tkachenko, Robert W. Brodersen, "Experimental Study of Spectrum Sensing based on Energy Detection and Network Cooperation", in TAPAS 06.

[3] Theodiridis Sergios, Konstantinos Koutroumbas, "Pattern Recognition", Fourth Edition, Elsevier Inc, 2009.

[4] http://www.netstumbler.com

[5] http://java.sun.com

[6] http://netresearch.ics.uci.edu/kfujii/jpcap/doc/

[7] http://www.radiotap.org

[8] http://www.ubuntu.com

[9] http://wireless.kernel.org/en/users/Drivers/b43

[10] http://www.bluej.org

[11] http://www.mathworks.com/products/matlab/

[12] http://www.fte.com/products/FTS4BT-01.asp

[13] http://www.wildpackets.com/solutions/wireless_analysis

[14] http://www.cacetech.com/products/airpcap.html

[15] http://www.netstumbler.com/

[16] http://www.linuxcommand.org/man_pages/hcidump8.html

[17] http://www.airmagnet.com/

[18] http://www.wireshark.org/

[19] http://airtraf.sourceforge.net

[20] http://www.kismetwireless.net/

[21] http://www.wi-fiplanet.com

[22] Gandetto M. and Regazzoni C., "Spectrum Sensing: A Dsitributed Approach for Cognitive Terminals," *IEEE Journal on selected areas in communications*, Vol.25 (3), 2007.

[23] http://en.wikipedia.org/wiki/Java_Virtual_Machine

[24] http://en.wikipedia.org/wiki/Monitor_mode

[25] J. Mitola III and G. Q. Maguire, Jr., "Cognitive radio: making software radios more personal," IEEE Personal Communications Magazine, vol. 6, nr. 4, pp. 13-18, Aug. 1999.

[26] Di Benedetto MG., Boldrini S., Martin C., Roldan J., "Automatic Network Recognition by Feature Extraction: A case study in the ISM band", Crowncom 2010. 9-11, June, 2010.

[27] LAN MAN Standards committee of the IEEE Computer Society. Part 15 Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. ANSI/IEEE Std. 802.15, 2002 Edition.

[28] Timothy Newman (Virginia Tech, USA); Joseph Evans (University of Kansas, USA),*"Parameter Sensitivity in Cognitive Radio Adaptation Engines", DySPAN 2008.*

[29] S. Haykin, “Cognitive Radio: Brain-Empowered Wireless Communications,” IEEE Journal on Selected Areas In Communications, vol. 23, no. 2, p. 201, 2005.

[30] Federal Communications Commission, “Facilitating opportunities for flexible, efficient, and reliable spectrum use employing cognitive radio technologies,” 2005.

[31] R. W. Thomas, D. H. Friend, L. A. DaSilva, and A. B. MacKen- zie, “Cognitive Networks: Adaptation and Learning to Achieve End- to End Performance Objectives,” IEEE Communications Magazine, vol. 44, no. 12, dec 2006.

[32] S. Wu, C. Lin, Y. Tseng, and J. Sheu, “A New Multi-Channel MAC Protocol with On-Demand Channel Assignment for Multi-Hop Mobile Ad Hoc Networks,” International Symposium on Parallel Architec- tures, Algorithms, and Networks, I-SPAN, pp. 232–237, 2000.

[33] Y. Yuan and P. Bahl, “Knows: Cognitive radio network over white spaces,” in IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), apr 2007.

[34] A. Motamedi and A. Bahai, “Mac protocol design for spectrum-agile wireless networks: Stochastic control approach,” in IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), apr 2007.

[35] D. Leith and P. Clifford, “A Self-Managed Distributed Channel Selec- tion Algorithm for WLANs,” Proc IEEE RAWNET 2006, 2006.

[36] R. Thomas, L. DaSilva, and A. MacKenzie, “Cognitive networks” New Frontiers in Dynamic Spectrum Access Networks, 2005. DyS-PAN 2005. 2005 First IEEE International Symposium on, pp. 352–360, 2005.

[37] S. S. Lake, "Cognitive networking with software programmable intel- ligent networks for wireless and wireline critical communications," in Military Communications Conference (MILCOM), 2005.

Appendix

DEVELOPED CODES

7.1 Sniffer Code

```
import jpcap.*;
import jpcap.packet.*;
import java.util.*;
import java.io.*;
import java.lang.*;
import java.lang.Object;

public class PacketPHY implements PacketReceiver
{
int PacketCount = 1;
public PacketPHY(){
}
public void receivePacket(Packet packet){
//Prints out Packet and Packet Length:
System.out.println();
System.out.println("PacketNumber        : " + PacketCount );
System.out.println("Length              : " + packet.len );
System.out.println("Captured  Length    : " + packet.caplen );
Vector<String> vecdata = new Vector<String>();
Vector<String> vecheader = new Vector<String>();

//Prints out Packet Data:
System.out.print("DATA: ");
for (byte d : packet.data)
System.out.print(Integer.toHexString(d&0xff) + ":");
System.out.println();

//Packet Data Vector Creator
for(byte item2: packet.data)
{
String aux = Integer.toHexString(item2&0xff);
if(aux.length() < 2)
aux = "0" + aux;
vecdata.addElement(aux);
}
//System.out.println("Data Vector: " + vecdata);

classifier(vecdata,packet);
PacketCount++;
}
public void classifier (Vector<String> vecdata,Packet packet)
{
Object RadioTapLength1 = vecdata.elementAt(2);
Object RadioTapLength2 = vecdata.elementAt(3);
String RTL1 = RadioTapLength1.toString();
String RTL2 = RadioTapLength2.toString();
String FinalRTL = RTL2+RTL1;
int RadioTapFinish = Integer.parseInt(FinalRTL,16);
System.out.println("RadioTapHeaderLength : " + RadioTapFinish + " bytes");
float Framelen=packet.len-RadioTapFinish ;// The frame include FCS and Shared Key
System.out.println("Frame Length        : " + Framelen + " bytes" ); // Stampa la durata dell trama;
```

```
//Getting Rate from RadioTapHeader
Object flag= vecdata.elementAt(4);
String flaghex= flag.toString();
int flagint= Integer.parseInt(flaghex,16);
String flagbinary=Integer.toBinaryString(flagint);
while (flagbinary.length()<8){
flagbinary="0"+ flagbinary;
}
char presentrateC= flagbinary.charAt(5);
int presentrateI=Integer.parseInt(Character.toString(presentrateC));
char [] cell= {flagbinary.charAt(6),flagbinary.charAt(7)};
String twobits= new String (cell);
int twobitsint=Integer.parseInt(twobits,2);

//Others Parameters
char channelC = flagbinary.charAt(4);
int channelI= Integer.parseInt(Character.toString(channelC));
int flagintTSF=Integer.parseInt(Character.toString(flagbinary.charAt(7))) ;
Vector <String> VecdataTSF = new Vector<String>();
if    (flagintTSF == 1)
{    int ivec;
for (ivec=8; ivec<16 ;ivec++){
VecdataTSF.addElement(vecdata.elementAt(ivec));
}

String TSF01=
(VecdataTSF.elementAt(7)+VecdataTSF.elementAt(6)+VecdataTSF.elementAt(5)+VecdataTSF.elementAt(
4)+VecdataTSF.elementAt(3)+VecdataTSF.elementAt(2)+VecdataTSF.elementAt(1)+VecdataTSF.element
At(0));
long TSFI01=Long.parseLong(TSF01,16);
//System.out.println("TSFHEXA: " + TSF01);
System.out.println("TSF                : " + TSFI01 + " usec");

}
if(channelI == 0) {

if (presentrateI==1)
{

switch (twobitsint)
{
case 00:
Object rate00= vecdata.elementAt(8);
String rateS00=rate00.toString();
float rateI00=Integer.parseInt(rateS00,16);
float Rate00 = rateI00/2;
Math duration00=null;

System.out.println("Rate               : " + Rate00 + " Mbps");
System.out.println("Duration Frame      : "  + duration00.round(Framelen*8/Rate00)+ " usec");

break;

case 01:
Object rate01= vecdata.elementAt(16);
String rateS01=rate01.toString();
```

```
float rateI01=Integer.parseInt(rateS01,16);
float Rate01 = rateI01/2;
Math duration01=null;

System.out.println("Rate              : " + Rate01 + " Mbps");
System.out.println("Duration Frame      : "  + duration01.round(Framelen*8/Rate01)+ " usec");


break;

case 02:
Object rate02= vecdata.elementAt(9);
String rateS02=rate02.toString();
float rateI02=Integer.parseInt(rateS02,16);
float Rate02 = rateI02/2;
Math duration02=null;
System.out.println("Rate              : " + Rate02 + " Mbps");
System.out.println("Duration Frame      : "  +duration02.round(Framelen*8/Rate02)+ " usec");
//Preamble's Type
Object flag02=vecdata.elementAt(8);
String flag02S=flag02.toString();
int flag02I=Integer.parseInt(flag02S,16);
String flag02B=Integer.toBinaryString(flag02I);
System.out.println(flag02B);
while (flag02B.length()<8){
flag02B="0"+ flag02B;
}

char preamble02C = flag02B.charAt(6);
int preamble02I=Integer.parseInt(Character.toString(preamble02C));
if (preamble02I==1)
{

System.out.println("Preamble          : Short");
}
else
{
System.out.println("Preamble          : Long");
}
//Indication CFP
char CFP02C = flag02B.charAt(7);
int CFP02I= Integer.parseInt(Character.toString(CFP02C));
if (CFP02I==0)
{

System.out.println("Period            : Contention Period");
}
else
{
System.out.println("Period            : Contention Free Period");
}

break;

case 03:
Object rate03= vecdata.elementAt(17);
```

```
String rateS03=rate03.toString();
float rateI03=Integer.parseInt(rateS03,16);
float Rate03 = rateI03/2;
Math duration03=null;
System.out.println("Rate                : " + Rate03 + " Mbps");
System.out.println("Duration Frame      :"  +duration03.round(Framelen*8/Rate03)+ " usec");
//Preamble's Type
Object flag03=vecdata.elementAt(16);
String flag03S=flag03.toString();
int flag03I=Integer.parseInt(flag03S,16);
String flag03B=Integer.toBinaryString(flag03I);
while (flag03B.length()<8){
flag03B="0"+ flag03B;
}
char preamble03C = flag03B.charAt(6);
int preamble03I=Integer.parseInt(Character.toString(preamble03C));
if (preamble03I==1)
{
System.out.println("Preamble            : Short");
}
else
{
System.out.println("Preamble            : Long");
}
//Indication CFP
char CFP03C = flag03B.charAt(7);
int CFP03I= Integer.parseInt(Character.toString(CFP03C));
if (CFP03I==0)
{

System.out.println("Period              : Contention Period");
}
else
{
System.out.println("Period              : Contention Free Period");
}
break;
}
}

else {

switch (twobitsint)
{

case 02:
//Preamble's Type
Object flag02=vecdata.elementAt(8);
String flag02S=flag02.toString();
int flag02I=Integer.parseInt(flag02S,16);
String flag02B=Integer.toBinaryString(flag02I);

while (flag02B.length()<8){
flag02B="0"+ flag02B;
}
```

```
char preamble02C = flag02B.charAt(6);
int preamble02I=Integer.parseInt(Character.toString(preamble02C));
if (preamble02I==1)
{

System.out.println("Preamble           : Short");
}
else
{
System.out.println("Preamble           : Long");
}
//Indication CFP
char CFP02C = flag02B.charAt(7);
int CFP02I= Integer.parseInt(Character.toString(CFP02C));
if (CFP02I==0)
{

System.out.println("Period             : Contention Period");
}
else
{
System.out.println("Period             : Contention Free Period");
}
break;

case 03:
//Preamble's Type
Object flag03=vecdata.elementAt(16);
String flag03S=flag03.toString();
int flag03I=Integer.parseInt(flag03S,16);
String flag03B=Integer.toBinaryString(flag03I)
while (flag03B.length()<8){
flag03B="0"+ flag03B;
}

char preamble03C = flag03B.charAt(6);
int preamble03I=Integer.parseInt(Character.toString(preamble03C));
if (preamble03I==1)
{

System.out.println("Preamble           : Short");
}
else
{
System.out.println("Preamble           : Long");
}
//Indication CFP
char CFP03C = flag03B.charAt(7);
int CFP03I= Integer.parseInt(Character.toString(CFP03C));
if (CFP03I==0)
{

System.out.println("Period             : Contention Period");
}
else
{
```

```
System.out.println("Period              : Contention Free Period");
}

break;

}
}

}
//Getting Channel Information

else {
if (presentrateI==1)
{

switch (twobitsint)
{
case 00:
Object rate00= vecdata.elementAt(8);
String rateS00=rate00.toString();
float rateI00=Integer.parseInt(rateS00,16);
float  Rate00 = rateI00/2;
Math duration00=null;


System.out.println("Rate                : " + Rate00 + " Mbps");
System.out.println("Duration Frame      : "  +duration00.round(Framelen*8/Rate00) +" usec");

Object chV12= vecdata.elementAt(12);
Object chV11 = vecdata.elementAt(11);
String chS = (chV12.toString()+chV11.toString());
int chI=Integer.parseInt(chS,16);
String chB=Integer.toBinaryString(chI);
while (chB.length()<16){
chB="0"+ chB;
}

char GFSK=chB.charAt(4);
int GFSKI=Integer.parseInt(Character.toString(GFSK));
if(GFSKI==1)
{
System.out.println("Modulation          :Gaussian Frequency Shift Keying Modulation");
}
char Dynamic=chB.charAt(5);
int DynamicI=Integer.parseInt(Character.toString(Dynamic));
if(DynamicI==1)
{
System.out.println("Modulation          : Dynamic CCK-OFDM ");
}
char Spec5=chB.charAt(7);
int Spec5I=Integer.parseInt(Character.toString(Spec5));
if(Spec5I==1)
{
System.out.println("Band                : 5 GHz Spectrum");
}
char Spec2=chB.charAt(8);
```

```
int Spec2I=Integer.parseInt(Character.toString(Spec2));
if(Spec2I==1)
{
System.out.println("Band               : 2 GHz Spectrum");
}
char OFDM=chB.charAt(9);
int OFDMI=Integer.parseInt(Character.toString(OFDM));
if(OFDMI==1)
{
System.out.println("Modulation         : Orthogonal Frequency Division Multiplexing");
}

char CCK=chB.charAt(10);
int CCKI=Integer.parseInt(Character.toString(CCK));
if(CCKI==1)
{
System.out.println("Modulation         : Code Complementary Keying");
}

break;

case 01:
Object rate01= vecdata.elementAt(16);
String rateS01=rate01.toString();
float rateI01=Integer.parseInt(rateS01,16);
float Rate01 = rateI01/2;
Math duration01=null;

System.out.println("Rate               : " + Rate01 + " Mbps");
System.out.println("Duration Frame      : "  +duration01.round(Framelen*8/Rate01)+" usec");
Object chV2001 = vecdata.elementAt(20);
Object chV1901 = vecdata.elementAt(19);
String chS01= (chV2001.toString()+chV1901.toString());
int chI01=Integer.parseInt(chS01,16);
String chB01=Integer.toBinaryString(chI01);
while (chB01.length()<16){
chB01="0"+ chB01;
}

char GFSK01=chB01.charAt(4);
int GFSKI01=Integer.parseInt(Character.toString(GFSK01));
if(GFSKI01==1)
{
System.out.println("Modulation         : Gaussian Frequency Shift Keying Modulation");
}
char Dynamic01=chB01.charAt(5);
int DynamicI01=Integer.parseInt(Character.toString(Dynamic01));
if(DynamicI01==1)
{
System.out.println("Modulation         : Dynamic CCK-OFDM ");
}
char Spec501=chB01.charAt(7);
int Spec5I01=Integer.parseInt(Character.toString(Spec501));
if(Spec5I01==1)
{
System.out.println("Band               : 5 GHz Spectrum");
}
char Spec201=chB01.charAt(8);
int Spec2I01=Integer.parseInt(Character.toString(Spec201));
if(Spec2I01==1)
{
System.out.println("Band               : 2 GHz Spectrum");
}
char OFDM01=chB01.charAt(9);
int OFDMI01=Integer.parseInt(Character.toString(OFDM01));
if(OFDMI01==1)
{
System.out.println("Modulation          : Orthogonal Frequency Division Multiplexing");
}

char CCK01=chB01.charAt(10);
int CCKI01=Integer.parseInt(Character.toString(CCK01));
if(CCKI01==1)
{
System.out.println("Modulation          : Code Complementary Keying");
}

break;

case 02:
Object rate02= vecdata.elementAt(9);
String rateS02=rate02.toString();
float rateI02=Integer.parseInt(rateS02,16);
float Rate02 = rateI02/2;
Math duration02=null;

System.out.println("Rate                : " + Rate02 + " Mbps");

//Preamble's Type
Object flag02=vecdata.elementAt(8);
String flag02S=flag02.toString();
int flag02I=Integer.parseInt(flag02S,16);
String flag02B=Integer.toBinaryString(flag02I);
while (flag02B.length()<8){
flag02B="0"+ flag02B;
}
char preamble02C = flag02B.charAt(6);
int preamble02I=Integer.parseInt(Character.toString(preamble02C));
Object chV1302 = vecdata.elementAt(13);
Object chV1202 = vecdata.elementAt(12);
String chS02 = (chV1302.toString()+chV1202.toString());
int chI02=Integer.parseInt(chS02,16);
String chB02=Integer.toBinaryString(chI02);
while (chB02.length()<16){
chB02="0"+ chB02;
}
char GFSK02=chB02.charAt(4);
int GFSKI02=Integer.parseInt(Character.toString(GFSK02));
if(GFSKI02==1)
{
System.out.println("Modulation          : Gaussian Frequency Shift Keying Modulation");
}
```

```
char Dynamic02=chB02.charAt(5);
int DynamicI02=Integer.parseInt(Character.toString(Dynamic02));
if(DynamicI02==1)
{
System.out.println(" Modulation          : Dynamic CCK-OFDM ");

if (preamble02I==1)
{

System.out.println("Preamble           : Short");
System.out.println("Duration           : "  +(duration02.round(Framelen*8/Rate02+96))+ " usec");

}
else
{
System.out.println("Preamble           : Long");
System.out.println("Duration           : "  +(duration02.round(Framelen*8/Rate02+192))+ " usec");
}
}
char Spec502=chB02.charAt(7);
int Spec5I02=Integer.parseInt(Character.toString(Spec502));
if(Spec5I02==1)
{
System.out.println("Band               : 5 GHz Spectrum");
}
char Spec202=chB02.charAt(8);
int Spec2I02=Integer.parseInt(Character.toString(Spec202));
if(Spec2I02==1)
{
System.out.println("Band               : 2 GHz Spectrum");
}
char OFDM02=chB02.charAt(9);
int OFDMI02=Integer.parseInt(Character.toString(OFDM02));
if(OFDMI02==1)
{
System.out.println("Modulation          : Orthogonal Frequency Division Multiplexing");

if (preamble02I==1)
{

System.out.println("Preamble           : Short");
System.out.println("Duration           : "
+(16+4+6+4*(duration02.round((16+6+Framelen*8)/(Rate02*4))))+ " usec");
}
else
{
System.out.println("Preamble           : Long");
System.out.println("Duration           : "
+(16+4+6+4*(duration02.round((16+6+Framelen*8)/(Rate02*4))))+ " usec");

}
}

char CCK02=chB02.charAt(10);
int CCKI02=Integer.parseInt(Character.toString(CCK02));
if(CCKI02==1)
```

```
{
System.out.println("Modulation          : Code Complementary Keying");

if (preamble02I==1)
{

System.out.println("Preamble            : Short");
System.out.println("Duration            : "  +(duration02.round(Framelen*8/Rate02+96))+ " usec");

}
else
{
System.out.println("Preamble            : Long");
System.out.println("Duration            : "  +(duration02.round(Framelen*8/Rate02+192))+ " usec");
}
}
//Indication CFP
char CFP02C = flag02B.charAt(7);
int CFP02I= Integer.parseInt(Character.toString(CFP02C));
if (CFP02I==0)
{

System.out.println("Period              : Contention Period");
}
else
{
System.out.println("Period              : Contention Free Period");
}

break;

case 03:

Object rate03= vecdata.elementAt(17);
String rateS03=rate03.toString();
float rateI03=Integer.parseInt(rateS03,16);
float Rate03 = rateI03/2;
Math duration03=null;

System.out.println("Rate                : " + Rate03 + " Mbps");
//Preamble's Type
Object flag03=vecdata.elementAt(16);
String flag03S=flag03.toString();
int flag03I=Integer.parseInt(flag03S,16);
String flag03B=Integer.toBinaryString(flag03I);
while (flag03B.length()<8){
flag03B="0"+ flag03B;
}
char preamble03C = flag03B.charAt(6);
int preamble03I=Integer.parseInt(Character.toString(preamble03C));

Object chV2103 = vecdata.elementAt(21);
Object chV2003 = vecdata.elementAt(20);
String chS03 = (chV2103.toString()+chV2003.toString());
int chI03=Integer.parseInt(chS03,16);
String chB03=Integer.toBinaryString(chI03);
```

```
while (chB03.length()<16){
chB03="0"+ chB03;
}
char GFSK03=chB03.charAt(4);
int GFSKI03=Integer.parseInt(Character.toString(GFSK03));
if(GFSKI03==1)
{
System.out.println("Modulation          : Gaussian Frequency Shift Keying Modulation");
}
char Dynamic03=chB03.charAt(5);
int DynamicI03=Integer.parseInt(Character.toString(Dynamic03));
if(DynamicI03==1)
{
System.out.println("Modulation          : Dynamic CCK-OFDM ");
if (preamble03I==1)
{

System.out.println("Preamble            : Short");
System.out.println("Duration            : "  +(duration03.round(Framelen*8/Rate03+96))+ " usec");

}
else
{
System.out.println("Preamble            : Long");
System.out.println("Duration            : "  +(duration03.round(Framelen*8/Rate03+192))+ " usec");
}

}
char Spec503=chB03.charAt(7);
int Spec5I03=Integer.parseInt(Character.toString(Spec503));
if(Spec5I03==1)
{
System.out.println("Band                : 5 GHz Spectrum");
}
char Spec203=chB03.charAt(8);
int Spec2I03=Integer.parseInt(Character.toString(Spec203));
if(Spec2I03==1)
{
System.out.println("Band                : 2 GHz Spectrum");
}
char OFDM03=chB03.charAt(9);
int OFDMI03=Integer.parseInt(Character.toString(OFDM03));
if(OFDMI03==1)
{
System.out.println("Modulation          : Orthogonal Frequency Division Multiplexing");

if (preamble03I==1)
{

System.out.println("Preamble            : Short");
System.out.println("Duration            : "
+(16+4+6+4*(duration03.round((16+6+Framelen*8)/(Rate03*4))))+ " usec");

}
else
{
```

```
System.out.println("Preamble           : Long");
System.out.println("Duration           : "
+(16+4+6+4*(duration03.round((16+6+Framelen*8)/(Rate03*4))))+ " usec");

}
}
char CCK03=chB03.charAt(10);
int CCKI03=Integer.parseInt(Character.toString(CCK03));
if(CCKI03==1)
{
System.out.println("Modulation         : Code Complementary Keying");

if (preamble03I==1)
{

System.out.println("Preamble           : Short");
System.out.println("Duration           : "  +(duration03.round(Framelen*8/Rate03+96))+ " usec");

}
else
{
System.out.println("Preamble           : Long");
System.out.println("Duration           : "  +(duration03.round(Framelen*8/Rate03+192))+ " usec");
}

}
//Indication CFP
char CFP03C = flag03B.charAt(7);
int CFP03I= Integer.parseInt(Character.toString(CFP03C));
if (CFP03I==1)
{

System.out.println("Period             : Contention Free Period");
}
else
{
System.out.println("Period             : Contention  Period");
}

break;

}
}
else {

switch (twobitsint)
{

case 00:
Object chV11 = vecdata.elementAt(11);
Object chV10 = vecdata.elementAt(10);
String chS = (chV11.toString()+chV10.toString());
int chI=Integer.parseInt(chS,16);
String chB=Integer.toBinaryString(chI);
while (chB.length()<16){
chB="0"+ chB;
```

```
}

char GFSK=chB.charAt(4);
int GFSKI=Integer.parseInt(Character.toString(GFSK));
if(GFSKI==1)
{
System.out.println("Modulation          : Gaussian Frequency Shift Keying Modulation");
}
char Dynamic=chB.charAt(5);
int DynamicI=Integer.parseInt(Character.toString(Dynamic));
if(DynamicI==1)
{
System.out.println("Modulation          : Dynamic CCK-OFDM ");
}
char Spec5=chB.charAt(7);
int Spec5I=Integer.parseInt(Character.toString(Spec5));
if(Spec5I==1)
{
System.out.println("Band                : 5 GHz Spectrum");
}
char Spec2=chB.charAt(8);
int Spec2I=Integer.parseInt(Character.toString(Spec2));
if(Spec2I==1)
{
System.out.println("Band                : 2 GHz Spectrum");
}
char OFDM=chB.charAt(9);
int OFDMI=Integer.parseInt(Character.toString(OFDM));
if(OFDMI==1)
{
System.out.println("Modulation          : Orthogonal Frequency Division Multiplexing");
}

char CCK=chB.charAt(10);
int CCKI=Integer.parseInt(Character.toString(CCK));
if(CCKI==1)
{
System.out.println("Modulation          : Code Complementary Keying");
}
break;

case 01:

Object chV1201 = vecdata.elementAt(19);
Object chV1101 = vecdata.elementAt(18);
String chS01= (chV1201.toString()+chV1101.toString());
int chI01=Integer.parseInt(chS01,16);
String chB01=Integer.toBinaryString(chI01);
while (chB01.length()<16){
chB01="0"+ chB01;
}

char GFSK01=chB01.charAt(4);
int GFSKI01=Integer.parseInt(Character.toString(GFSK01));
if(GFSKI01==1)
{
```

```
System.out.println("Modulation          : Gaussian Frequency Shift Keying Modulation");
}
char Dynamic01=chB01.charAt(5);
int DynamicI01=Integer.parseInt(Character.toString(Dynamic01));
if(DynamicI01==1)
{
System.out.println("Modulation          : Dynamic CCK-OFDM ");
}
char Spec501=chB01.charAt(7);
int Spec5I01=Integer.parseInt(Character.toString(Spec501));
if(Spec5I01==1)
{
System.out.println("Band                : 5 GHz Spectrum");
}
char Spec201=chB01.charAt(8);
int Spec2I01=Integer.parseInt(Character.toString(Spec201));
if(Spec2I01==1)
{
System.out.println("Band                : 2 GHz Spectrum");
}
char OFDM01=chB01.charAt(9);
int OFDMI01=Integer.parseInt(Character.toString(OFDM01));
if(OFDMI01==1)
{
System.out.println("Modulation          : Orthogonal Frequency Division Multiplexing");
}

char CCK01=chB01.charAt(10);
int CCKI01=Integer.parseInt(Character.toString(CCK01));
if(CCKI01==1)
{
System.out.println("Modulation          : Code Complementary Keying");
}
break;

case 02:
Object chV1202 = vecdata.elementAt(12);
Object chV1102 = vecdata.elementAt(11);
String chS02 = (chV1202.toString()+chV1102.toString());
int chI02=Integer.parseInt(chS02,16);
String chB02=Integer.toBinaryString(chI02);
while (chB02.length()<16){
chB02="0"+ chB02;
}
char GFSK02=chB02.charAt(4);
int GFSKI02=Integer.parseInt(Character.toString(GFSK02));
if(GFSKI02==1)
{
System.out.println("Modulation          : Gaussian Frequency Shift Keying Modulation");
}
char Dynamic02=chB02.charAt(5);
int DynamicI02=Integer.parseInt(Character.toString(Dynamic02));
if(DynamicI02==1)
{
System.out.println("Modulation          : Dynamic CCK-OFDM ");
}
```

```
char Spec502=chB02.charAt(7);
int Spec5I02=Integer.parseInt(Character.toString(Spec502));
if(Spec5I02==1)
{
System.out.println("Band                : 5 GHz Spectrum");
}
char Spec202=chB02.charAt(8);
int Spec2I02=Integer.parseInt(Character.toString(Spec202));
if(Spec2I02==1)
{
System.out.println("Band                : 2 GHz Spectrum");
}
char OFDM02=chB02.charAt(9);
int OFDMI02=Integer.parseInt(Character.toString(OFDM02));
if(OFDMI02==1)
{
System.out.println("Modulation          : Orthogonal Frequency Division Multiplexing");
}

char CCK02=chB02.charAt(10);
int CCKI02=Integer.parseInt(Character.toString(CCK02));
if(CCKI02==1)
{
System.out.println("Modulation          : Code Complementary Keying");
}

//Preamble's Type
Object flag02=vecdata.elementAt(8);
String flag02S=flag02.toString();
int flag02I=Integer.parseInt(flag02S,16);
String flag02B=Integer.toBinaryString(flag02I);
while (flag02B.length()<8){
flag02B="0"+ flag02B;
}

char preamble02C = flag02B.charAt(6);
int preamble02I=Integer.parseInt(Character.toString(preamble02C));
if (preamble02I==1)
{

System.out.println("Preamble            : Short");
}
else
{
System.out.println("Preamble            : Long");
}
//Indication CFP
char CFP02C = flag02B.charAt(7);
int CFP02I= Integer.parseInt(Character.toString(CFP02C));
if (CFP02I==0)
{

System.out.println("Period              : Contention Period");
}
else
{
```

```
System.out.println("Period              : Contention Free Period");
}

break;

case 03:

Object chV1303 = vecdata.elementAt(20);
Object chV1203 = vecdata.elementAt(19);
String chS03 = (chV1303.toString()+chV1203.toString());
int chI03=Integer.parseInt(chS03,16);
String chB03=Integer.toBinaryString(chI03);
while (chB03.length()<16){
chB03="0"+ chB03;
}
char GFSK03=chB03.charAt(4);
int GFSKI03=Integer.parseInt(Character.toString(GFSK03));
if(GFSKI03==1)
{
System.out.println("Modulation          : Gaussian Frequency Shift Keying Modulation");
}
char Dynamic03=chB03.charAt(5);
int DynamicI03=Integer.parseInt(Character.toString(Dynamic03));
if(DynamicI03==1)
{
System.out.println("MOdulation          :  Dynamic CCK-OFDM ");
}
char Spec503=chB03.charAt(7);
int Spec5I03=Integer.parseInt(Character.toString(Spec503));
if(Spec5I03==1)
{
System.out.println("Band                : 5 GHz Spectrum");
}
char Spec203=chB03.charAt(8);
int Spec2I03=Integer.parseInt(Character.toString(Spec203));
if(Spec2I03==1)
{
System.out.println("Band                : 2 GHz Spectrum");
}
char OFDM03=chB03.charAt(9);
int OFDMI03=Integer.parseInt(Character.toString(OFDM03));
if(OFDMI03==1)
{
System.out.println("Modulation          : Orthogonal Frequency Division Multiplexing");
}

char CCK03=chB03.charAt(10);
int CCKI03=Integer.parseInt(Character.toString(CCK03));
if(CCKI03==1)
{
System.out.println("Modulation          : Code Complementary Keying");
}
//Preamble's Type
Object flag03=vecdata.elementAt(16);
String flag03S=flag03.toString();
int flag03I=Integer.parseInt(flag03S,16);
```

```
String flag03B=Integer.toBinaryString(flag03I);
while (flag03B.length()<8){
flag03B="0"+ flag03B;
}

char preamble03C = flag03B.charAt(6);
int preamble03I=Integer.parseInt(Character.toString(preamble03C));
if (preamble03I==1)
{

System.out.println("Preamble          : Short");
}
else
{
System.out.println("Preamble          : Long");
}
//Indication CFP
char CFP03C = flag03B.charAt(7);
int CFP03I= Integer.parseInt(Character.toString(CFP03C));
if (CFP03I==0)
{

System.out.println("Period          : Contention Period");
}
else
{
System.out.println("Period          : Contention Free Period");
}
break;

}
}
}
Object auxob = vecdata.elementAt(RadioTapFinish);
String auxstring = auxob.toString();
int auxint = Integer.parseInt(auxstring,16);

Vector<String> DestAddress = new Vector<String>();
Vector<String> SorAddress = new Vector<String>();
int fragmentnumberI=0;
int sequencenumberfinalI=0;

if (Framelen > RadioTapFinish+14)

{
Object sequencecontrol1=vecdata.elementAt(RadioTapFinish+22);
String sequencecontrol1S=sequencecontrol1.toString();
int sequencecontrol1I=Integer.parseInt(sequencecontrol1S,16);
String sequencecontrol1B=Integer.toBinaryString(sequencecontrol1I);
while (sequencecontrol1B.length()<8){
sequencecontrol1B="0"+ sequencecontrol1B;
}

Object sequencecontrol2=vecdata.elementAt(RadioTapFinish+23);
String sequencecontrol2S=sequencecontrol2.toString();
int sequencecontrol2I=Integer.parseInt(sequencecontrol2S,16);
```

```
String sequencecontrol2B=Integer.toBinaryString(sequencecontrol2I);
while (sequencecontrol2B.length()<8){
sequencecontrol2B="0"+ sequencecontrol2B;
}

String fragmentnumberS=sequencecontrol1B.substring (4);
fragmentnumberI=Integer.parseInt(fragmentnumberS,2);
String sequencenumberS=sequencecontrol1B.substring (0,4);
String sequencenumberfinalS=sequencecontrol2B + sequencenumberS;

while (sequencenumberfinalS.length()<12){
sequencenumberfinalS="0"+ sequencenumberfinalS;
}

sequencenumberfinalI=Integer.parseInt(sequencenumberfinalS,2);
}

int mac;

switch(auxint)
{
case 00:
System.out.println("Frame Type          : MANAGEMENT");
System.out.println("Frame Subtype       : Association Request");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{  DestAddress.addElement(vecdata.elementAt(mac) );
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Destination Address  : " + DestAddress);
System.out.println("Source Address      : " + SorAddress);
System.out.println("Fragment Number     : " + fragmentnumberI);
System.out.println("Sequence Number     : " + sequencenumberfinalI);
break;

case 16:
System.out.println("Frame Type          : MANAGEMENT");
System.out.println("Frame Subtype       : Association Response");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Destination Address  : " + DestAddress);
System.out.println("Source Address      : " + SorAddress);
System.out.println("Fragment Number     : " + fragmentnumberI);
System.out.println("Sequence Number     : " + sequencenumberfinalI);
break;

case 32:
System.out.println("Frame Type          : MANAGEMENT");
System.out.println("Frame Subtype       : Reassociation Request");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Destination Address  : " + DestAddress);
System.out.println("Source Address      : " + SorAddress);
System.out.println("Fragment Number     : " + fragmentnumberI);
System.out.println("Sequence Number     : " + sequencenumberfinalI);
break;
```

```
case 48:
System.out.println("Frame Type          : MANAGEMENT");
System.out.println("Frame Subtype        : Reassociation Response");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Destination Address  : " + DestAddress);
System.out.println("Source Address       : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 64:
System.out.println("Frame Type          : MANAGEMENT");
System.out.println("Frame Subtype        : Probe Request");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{  DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Destination Address  : " + DestAddress);
System.out.println("Source Address       : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 80:
System.out.println("Frame Type          : MANAGEMENT");
System.out.println("Frame Subtype        : Probe Response");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Destination Address  : " + DestAddress);
System.out.println("Source Address       : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 96:
System.out.println("Frame Type          : MANAGEMENT");
System.out.println("Frame Subtype        : Reserved");

break;

case 112:
System.out.println("Frame Type          : MANAGEMENT");
System.out.println("Frame Subtype        : Reserved");
break;

case 128:
System.out.println("Frame Type          : MANAGEMENT");
System.out.println("Frame Subtype        : Beacon Frame");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Destination Address  : " + DestAddress);
System.out.println("Source Address       : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);

//Beacon Timestamp
vecdata.elementAt(RadioTapFinish+25+6)+vecdata.elementAt(RadioTapFinish+25+5)+vecdata.elementA
t(RadioTapFinish+25+4)+vecdata.elementAt(RadioTapFinish+25+3)+vecdata.elementAt(RadioTapFinish+
25+2)+vecdata.elementAt(RadioTapFinish+25+1)+vecdata.elementAt(RadioTapFinish+25)+vecdata.elem
entAt(RadioTapFinish+24);
break;

case 144:
System.out.println("Frame Type          : MANAGEMENT");
System.out.println("Frame Subtype       : ATIM");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Destination Address  : " + DestAddress);
System.out.println("Source Address      : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 160:
System.out.println("Frame Type          : MANAGEMENT");
System.out.println("Frame Subtype       : Disassociation");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Destination Address  : " + DestAddress);
System.out.println("Source Address      : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 176:
System.out.println("Frame Type          : MANAGEMENT");
System.out.println("Frame Subtype       : Authentication");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Destination Address  : " + DestAddress);
System.out.println("Source Address      : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 192:
System.out.println("Frame Type          : MANAGEMENT");
System.out.println("Frame Subtype       : Deauthentication");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac) );
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Destination Address  : " + DestAddress);
System.out.println("Source Address      : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
```

```
break;

case 208:
System.out.println("Frame Type         : MANAGEMENT");
System.out.println("Frame Subtype       : Action");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{  DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Destination Address  : " + DestAddress);
System.out.println("Source Address      : " + SorAddress);
System.out.println("Fragment Number     : " + fragmentnumberI);
System.out.println("Sequence Number     : " + sequencenumberfinalI);
break;

case 224:
System.out.println("Frame Type         : MANAGEMENT");
System.out.println("Frame Subtype       : Reserved");

break;

case 240:
System.out.println("Frame Type         : MANAGEMENT");
System.out.println("Frame Subtype       : Reserved");
break;

case 04:
System.out.println("Frame Type         : CONTROL");
System.out.println("Frame Subtype       : Reserved");
break;

case 116:
System.out.println("Frame Type         : CONTROL");
System.out.println("Frame Subtype       : Reserved");
break;

case 132:
System.out.println("Frame Type         : CONTROL");
System.out.println("Frame Subtype       : Block Ack Request");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address    : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
break;

case 148:
System.out.println("Frame Type         : CONTROL");
System.out.println("Frame Subtype       : Block Ack");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address    : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
break;

case 164:
```

```
System.out.println("Frame Type          : CONTROL");
System.out.println("Frame Subtype       : PS-Poll");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address    : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
break;

case 180:
System.out.println("Frame Type          : CONTROL");
System.out.println("Frame Subtype       : RTS");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address    : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
break;

case 196:
System.out.println("Frame Type          : CONTROL");
System.out.println("Frame Subtype       : CTS");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));}

System.out.println("Receiver Address    : " + DestAddress);

break;

case 212:
System.out.println("Frame Type          : CONTROL");
System.out.println("Frame Subtype       : ACK");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));}

System.out.println("Receiver Address    : " + DestAddress);
break;

case 228:
System.out.println("Frame Type          : CONTROL");
System.out.println("Frame Subtype       : CF-End");

for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address    : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
break;

case 244:
System.out.println("Frame Type          : CONTROL");
System.out.println("Frame Subtype       : CF-End+CF-ACK");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address    : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
break;

case 0x08:
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       : Data");

for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address     : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 24:
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       : Data+CF-Ack");

for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address     : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 40:
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       : Data+CF-Poll");

for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address     : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 56:
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       : Data+CF-Ack+CF-Poll");

for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address     : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 72:
```

```
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       : Null(No Data)");

for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address    : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
System.out.println("Fragment Number     : " + fragmentnumberI);
System.out.println("Sequence Number     : " + sequencenumberfinalI);
break;

case 88:
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       : CF-Ack (No Data)");

for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address    : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
System.out.println("Fragment Number     : " + fragmentnumberI);
System.out.println("Sequence Number     : " + sequencenumberfinalI);
break;

case 104:
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       : CF-POll (No Data)");

for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address    : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
System.out.println("Fragment Number     : " + fragmentnumberI);
System.out.println("Sequence Number     : " + sequencenumberfinalI);
break;

case 120:
System.out.println("Frame Type          : DATA");
System.out.println("Frame Sutype        : CF-Ack+CF-Poll (No Data)");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address    : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
System.out.println("Fragment Number     : " + fragmentnumberI);
System.out.println("Sequence Number     : " + sequencenumberfinalI);

break;

case 136:
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       : QoS Data");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address     : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 152:
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       : QoS Data + CF-Ack");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address     : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 168:
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       : Qos Data + CF-Poll");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address     : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 184:
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       : QoS Data+CF-Ack+CF-Poll");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address     : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 200:
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       : QoS Null");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address     : " + DestAddress);
System.out.println("Transmitter Address  : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 216:
```

```
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       : Reserved");
break;

case 232:
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       : QoS CF-Poll");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address     : " + DestAddress);
System.out.println("Transmittert Address : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

case 248:
System.out.println("Frame Type          : DATA");
System.out.println("Frame Subtype       :QoS + CF-Ack(No Data)+CF-Poll(No Data)");
for ( mac = (RadioTapFinish+4); mac<(RadioTapFinish+10); mac++)
{ DestAddress.addElement(vecdata.elementAt(mac));
SorAddress.addElement(vecdata.elementAt(mac+6));}
System.out.println("Receiver Address     : " + DestAddress);
System.out.println("Transmittert Address : " + SorAddress);
System.out.println("Fragment Number      : " + fragmentnumberI);
System.out.println("Sequence Number      : " + sequencenumberfinalI);
break;

default:
System.out.println("CHECK CODE");
break;
}
}

public static void main(String[] args)
{
NetworkInterface[] devices = JpcapCaptor.getDeviceList();
//for each network interface
for (int i = 0; i < devices.length; i++)
{
//print out its name and description
System.out.println(i+": "+devices[i].name + "(" + devices [i].description+")");

//print out its datalink name and description
System.out.println(" datalink: "+devices[i].datalink_name +  "(" + devices[i].datalink_description+")");

//print out its MAC address
System.out.print(" MAC address:");
for (byte b : devices[i].mac_address)
System.out.print(Integer.toHexString(b&0xff) + ":");
System.out.println();

//print out its IP address, subnet mask and broadcast address
for (NetworkInterfaceAddress a : devices[i].addresses)
System.out.println(" address:"+a.address + " " + a.subnet +  " "+ a.broadcast);
}
```

```
int index=2;  // set index of the interface that you want to open.

//Open an interface with openDevice(NetworkInterface intrface,  int snaplen, boolean promics, int
to_ms)
try
{
JpcapCaptor captor=JpcapCaptor.openDevice(devices[index],  1600, true, 1);
captor.loopPacket(1020,new PacketPHY());
captor.close();
}
catch (Exception e){
}
}
```

7.2 MATLAB DEVELOPED FUNCTIONS AND SCRIPTS

1. MTD.m: Generates from sniffer's capture file data a pattern as if it was provided by some energy detector.

```
[TSFVector,DurationVector]=import_WiFi(filename)

lTSF = length(TSFVector);
TimeDiagram = zeros(1,TSFVector(lTSF)+DurationVector(lTSF)+1);


 for i1 = 1:lTSF

    timebegin = TSFVector(i1);
    Durationflag = DurationVector(i1);


    for j1 = 1 : Durationflag+1
        position = timebegin +  j1;
        TimeDiagram(position) = TimeDiagram(position)+1;
    end
 end

 figure(1)
 x = linspace(0,length(TimeDiagram)-1,length(TimeDiagram));
 y = TimeDiagram;
 p = plot(x,y);
 axis([0 length(TimeDiagram) 0 1.5]);
 title('Time Diagram');
 xlabel('time [usec]');
 ylabel('Presence/Absence of a Packet over the air');
 set(p,'Color','red','LineWidth',2);
```

2. Import_WiFi.m: Imports from sniffer's capture file necessary data to recreate PPDU sequences.

```
function [TSFVector,DurationVector]=import_WiFi(filename)

%filename = '2503capture5.txt'
%PPDU duration extraction
fid = fopen (filename,'a+');
acqstring = fileread(filename);
index1 = regexp(acqstring,'Duration');
PacketsCaptured = length(index1);
index2 = regexp(acqstring,'Period               :');
indexlast = index2-1;
DurationVector = zeros(1,PacketsCaptured);

for i0 = 1:PacketsCaptured

 textdurationline = acqstring(index1(i0):indexlast(i0));
 textscannedduration = textscan(textdurationline,'%*s %*s %f %*s');
 DurationVector(i0) = cell2mat(textscannedduration);

end

%TSF extraction
fclose(fid);

fid = fopen (filename,'a+');
acqstringTSF = fileread(filename);
indexTSF1 = regexp(acqstringTSF,'TSF');
PacketsCaptured = length(indexTSF1);
indexTSF2 = regexp(acqstring,'Rate');
indexlastTSF = indexTSF2-1;
TSFVector = zeros(1,PacketsCaptured);

for j0 = 1:PacketsCaptured

 textTSFline = acqstringTSF(indexTSF1(j0):indexlastTSF(j0));
 textscannedTSF = textscan(textTSFline,'%*s %*s %f %*s');
 TSFVector(j0) = cell2mat(textscannedTSF);
end
```

```
TSFVector = TSFVector - TSFVector(1);
fclose(fid);

%Preamble extraction
fid = fopen (filename,'a+');
acqstringPre = fileread(filename);
indexPre1 = regexp(acqstringPre,'Preamble');
PacketsCaptured = length(indexPre1);
indexPre2 = regexp(acqstring,'Duration             :');
indexlastPre = indexPre2-1;
PreVector = zeros(1,PacketsCaptured);

for j0 = 1:PacketsCaptured

 textPreline = acqstringPre(indexPre1(j0):indexlastPre(j0));
 textscannedPre = textscan(textPreline, '%*s %*s %s');
 strtest = cell2str(textscannedPre);
 TF = strcmp('Long', strtest);
 if (TF == 1)
     TSFVector(j0) = TSFVector(j0)-192;

 else
     TSFVector(j0) = TSFVector(j0)-96;

 end
end

TSFVector = TSFVector - TSFVector(1);
fclose(fid);
```

3. FeatureExtraction.m: Algorithm that allow extraction of SIFS feature from any packet sequence pattern.

```
Function[IFSVector,MaxPPDUduration,Interference]=FeatureExtraction(Du
rationVector,TSFVector)


lDV = length(DurationVector)-1;

 cellIFS = 0;

 IFSVector=[];

 for DurationCounter = 1:lDV

     IFS =  TSFVector(DurationCounter+1)-TSFVector(DurationCounter)-
DurationVector(DurationCounter);

      if
(0.65*DurationVector(DurationCounter)>DurationVector(DurationCounter+
1)&(IFS<625)&(IFS>0))

          cellIFS = cellIFS+1;

          IFSVector(cellIFS)=IFS;

          CounterImportantValues(cellIFS)=DurationCounter;
      end
 end

 for fsc = 1:length(CounterImportantValues)

    if (fsc==1)

        MaxPPDUduration(1)= DurationVector(1);

        IFSPlotted(1)=IFSVector(1);
     else
        intmin = CounterImportantValues(fsc-1);

        intmax = CounterImportantValues(fsc);
```

```
            PPDUset = DurationVector(intmin:intmax);

            MaxPPDUduration(fsc) = max(PPDUset);

            IFSPlotted(fsc) = IFSVector(fsc);


    end
end

Interference=0;
```

4. MSBluetoothTrafficGeneration.m : Simulates a Bluetooth connection state Master-Slave communication packet sequence:

```
function
[DurationVectorBluetooth,TSFVectorBluetooth]=MSBluetoothTrafficGenera
tion

%IFs and Max PPDU Duration

    TS_duration = 625e-6;

    jitter = 10e-6;

    maxP_duration = 366e-6;

    NULL_duration = 126e-6;

    l = 15000;

    probability = 0.7;

    packet_duration = [];

    arrival_time = [];



% Scenario 3:
% 80% packets last 1 time slot
% 15% packets last 3 time slot
% 5% packets last 5 time slot

    arrival_time(1) = 0;

for i = 1:l

    if mod(i,2) == 1
        % odd packet -> data packet (master)

        chooser = rand(1,1);
        if chooser <= 0.8 % 1 time slot

            hmts = 1; % how many time slots

        elseif chooser > 0.95 % 5 time slot

            hmts = 5; % how many time slots

        else % 3 time slot

            hmts = 3; % how many time slots

        end

        switch hmts

            case 1  % 1 time slot

                if rand <= probability

                    packet_duration(i) = maxP_duration;

                else

                    packet_duration(i) = NULL_duration + 1e-
6*randint(1,1,[0,(maxP_duration-NULL_duration)*1e6]);

                end

                if i < l

                    arrival_time = [arrival_time
arrival_time(length(arrival_time))+TS_duration];

                end

            case 3  % 3 time slot

                if rand <= probability

                    packet_duration(i) = 1622e-6;
                else

                    packet_duration(i) = 2*TS_duration + 1e-
6*randint(1,1,[0,372]);

                end

                if i < l

                    arrival_time = [arrival_time
arrival_time(length(arrival_time))+3*TS_duration];

                end


            case 5   % 5 time slot

                if rand <= probability

                    packet_duration(i) = 2870e-6;

                else

                    packet_duration(i) = 4*TS_duration + 1e-
6*randint(1,1,[0,370]);

                end

                if i < l

                    arrival_time = [arrival_time
arrival_time(length(arrival_time))+5*TS_duration];

                end

        end

    else
        % even packet -> NULL packet as ACK (slave)
        packet_duration(i) = NULL_duration;

        if i < l

            arrival_time = [arrival_time
arrival_time(length(arrival_time))+TS_duration];

        end

    end


end


for i = 2:l

    j = randn*jitter/3; % 99% of values in +or- 3 sigma -> in +or-
jitter

    if j < - jitter

        j = - jitter;

    end

    if j > jitter

        j = jitter;

    end

    arrival_time(i) = arrival_time(i)+j;
end

DurationVectorBluetooth = 10^6.*packet_duration;

TSFVectorBluetooth = 10^6.*arrival_time;
```

7.3 Elaborated Paper: *"Automatic Network recognition by feature extraction: A case study in the ISM band"*

Accepted for publication in *Proceedings of the 5th International Conference on Cognitive Radio Oriented Wireless Networks and Communications*, Special Session on Cognitive Radio and Networking for Cooperative Coexistence of Heterogeneous Wireless Networks, June 9-11 2010, Cannes, France.

Automatic network recognition by feature extraction: a case study in the ISM band

Maria-Gabriella Di Benedetto, *Senior Member, IEEE*, Stefano Boldrini, Carmen Juana Martin Martin, and Jesus Roldan Diaz

***Abstract*— Automatic network recognition offers a promising framework for the integration of the cognitive concept at the network layer. This work addresses the problem of automatic classification of technologies operating in the ISM band, with particular focus on Wi-Fi vs. Bluetooth recognition. The proposed classifier is based on feature extraction related to time-varying patterns of packet sequences, i.e. MAC layer procedures, and adopts different linear classification algorithms. Results of classification confirmed the ability to reveal both technologies based on Mac layer feature identification.**

***Index Terms*—Cognitive networking, network discovery, automatic network classification**

I. Introduction

This work is framed under the umbrella of the AIR-AWARE Project, developed to achieve classification amongst technologies and interference entities operating over the ISM band. This project aims at creating a black box — the AIR-AWARE module — capable of classifying technologies, as well as different types of interference in play.

Such classification is important for cognitive mechanisms to be implemented in the network, given the numerous commercial technologies operate in this range of the spectrum, such as:

- IEEE 802.11 networks: 2.4 *GHz* and 5.8 *GHz* bands;
- Bluetooth: 2.4 *GHz* band, using Frequency Hopping and any of 79 available channels;
- HIPERLAN [High Performance Radio LAN]: European alternative to IEEE 802.11, operating in the 5.8 *GHz* band to avoid interference entities at 2.4 *GHz;*
- Closed-Circuit TV: Security cameras at 2.4 *GHz*;
- ZigBee IEEE 802.15.4: 2.4 *GHz* range band;
- Wireless Mouse and Keyboard: 2.4 *GHz* band.

Probably, the most common interference at 2.4 *GHz* comes from the microwave oven, followed by baby monitors and cordless Wi-Fi phones, and the interference produced by DECT standard cordless phones, operating in the 1.9 *GHz* band. When in use, these devices can compromise the quality of an IEEE 802.11 network.

The final goal is to propose a classification strategy based on information regarding protocol layers above the physical one (PHY). In particular, the objective is to identify MAC sublayer [1,2] specific features for each of the above technologies. Previous work, as for example [3], has addressed a similar problem, by classifying Wi-Fi vs. Bluetooth, using a spectrum sensing procedure based on distributed detection theory. The present work extends beyond previous investigations by considering Wi-Fi real traffic captures, and by focusing feature extraction and classification on MAC sublayer characteristics, leading to simplicity and computational efficiency.

In this work, feature identification lays its foundation on the observation that packet exchange patterns are technology-specific. As such, by identifying patterns clues, network recognition can be achieved. In particular, the study focuses on the ISM 2.4 *GHz* band. A first step consists in providing the AIR-AWARE module with a device capable of sensing the spectrum with a good time resolution. Albeit this piece of hardware will not be in a position to demodulate and decode the distinct signals in the air, it will enable AIR-AWARE to statistically study temporization of presence or absence of energy - against pre-defined thresholds – and therefore decode the packet sequence structure, in real time. Figure 1 illustrates the schematic of the cognitive energy detector, as well as software modules for the recognition of each technology embedded onto the device.

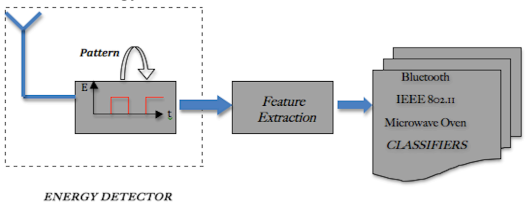


Figure 1 - The AIR-AWARE module

The study presented in this paper focuses on 802.11 (Wi-Fi) and 802.15.1 (Bluetooth) network recognition. To achieve recognition of both technologies, a set of features will be proposed. These features will serve as input for a linear classifier, that automatically performs classification among these two technologies.

The paper is organized as follows. Section II contains the description of the experimental set-up, i.e. the environment and tools by which data were collected and generated. The packet data-base is described in Section III, and particular focus on the Wi-Fi data-base (Section III.A) vs, the Bluetooth data-base (Section III.B). The classification algorithms are all linear classifiers as analyzed in Section IV; four different approaches are taken into consideration, i.e. Perceptron, Pocket, Least Mean Squares (LMS), and Sum of Error Squares Estimation (SOE). Experimental results on automatic classification of Wi-Fi vs. Bluetooth are reported in Section V. These results are discussed in Section VI, which also contains the conclusions of this study.

Manuscript submitted March 21, 2010. This work was supported in part by the European Commission in the framework of COST Action IC0902: Cognitive Radio and Networking for Cooperative Coexistence of Heterogeneous Wireless Networks.

The authors are with the Info-Com Department, School of Engineering, University of Rome La Sapienza, Via Eudossiana 18, 00184, Rome, Italy. E-mail address: dibenedetto@newyork.ing.uniroma1.it.

II. EXPERIMENTAL SET-UP

This Section describes the environment and tools used for collecting the reference data and build-up the data-base.

As for Wi-Fi, real data packets were detected within experimental measurements, using a packet capturing device ("Sniffer Station"). These measurements were carried out in the ACTS laboratory, located on the 2nd floor of the Information and Communication Department (InfoCom Dept.), in the Faculty of Engineering of the University of Rome "La Sapienza", Rome, Italy.

A long corridor, with offices and laboratories on both sides, composes the 2nd floor of the building, the ACTS lab being one of these laboratories. The (main) Access Point, to whose frequency channel the sniffer was tuned to, is located in this corridor, near the ceiling, at a height of about 2.5 - 3 *meters* from the ground, and at a distance of about 4 - 5 *meters* from the ACTS laboratory door.

The computers were placed on a fixed location on a table, at a height of about 1 - 1.5 *meters* from the ground, at a distance of about 1.5 *meters* from the nearest wall and of 2 *meters* from a door, that leads to the corridor mentioned above, i.e. the distance between the computers and the Access Point was about 5 - 6 *meters*. The distance between each computer was about 0.5 *meters*. On this same floor there are about 20 additional offices, containing about 2 - 3 computers each. Since we expected that any of these may be connected to the main Access Point during the measurements, and in the aim of capturing "clean data" for the training set, we performed measurements at night after verification that no other computer was active during captures.

There are also other Access Points in the same building, but tuned to other frequency channels. However, some packets of these other Access Points may have been captured. That occurs because of the frequency channels partial overlapping, that is expected in the IEEE 802.11 Standard. To this respect, collected data were carefully examined, in order to ensure again that only packets of the relevant network were captured.

Four computers were used to perform measurements. Three computers were used to generate traffic in heterogeneous traffic conditions. The fourth, used as the Sniffer Station, had the following technical specifications:

- Model: *HP Pavilion DV2320US*
- Processor: *AMD Turion 64 X2 TL 56*
- RAM Memory: 2*GB*
- Network Wireless Adapter: *AirForce 54g 802.11 a/b/g PCI Express*, with a *BCM4311* chipset

Operating System was *Linux Ubuntu 9.10*, with the real-time kernel *2.6.31-9-rt*. The driver used for the Broadcom 4311 chipset was the b43. The main Access Point was a *Cisco Aironet 12xx 802.11 b*.

As for Bluetooth packets, in this first phase we decided to design a complete simulator by which Bluetooth packets were obtained. By doing so, one of the two data packet stream was fully controllable by software.

III. PACKET DATA-BASE

A. Capturing Wi-Fi packets

In order to capture Wi-Fi Packets, a packet capturing application was developed using the Java library *jpcap* [4]. The Wi-Fi standard foresees a logic unit, the MAC PDU, while the unit sent over the air interface, called PPDU, includes beyond the MAC PDU, two additional fields (preamble and header). The driver used for the 802.11 network adapter enabled to intercept data of every PPDU [1] within the Sniffer range by means of its monitor mode [5]. This driver was also compatible with the radiotap header [6], which provides information such as preamble type, or time of arrival of first bit, for the captured MAC PDU.

Experiments were made in three different conditions: one, two, and three computers (nodes) associated to the Access Point. In all conditions, each computer was downloading at least two files or processing a video call; this way, we could ensure a high traffic scenario.

Two 1000-packet captures were run for each condition.

B. Generation of Bluetooth packets

Bluetooth simulated packets were generated using MATLAB. The reference standard for this simulation is the IEEE Standard 802.15.1 – 2005 [2], i.e. bitrate of 1 *Mbit/s.* Piconets of two devices in connection state (one master and one slave) were considered. The two devices send their packets alternately: one device (the master, for example) sends its data packets, and for every received packet, the other device (the slave) sends back an acknowledgement. Data packets sent by the master can occupy 1, 3 or 5 time slots (where the time slot is 625μs), according to their length, whereas acknowledgement packets (NULL packets, with a fixed length of 126 *bits*) occupy 1 time slot.

Two different scenarios were considered:

- Scenario 1: data packets occupy only 1 Time Slot
- Scenario 2:
  - 80% Data Packets occupy 1 Time Slot
  - 15% Data Packets occupy 3 Time Slots
  - 5% Data Packets occupy 5 Time Slots

In every scenario, 70 % of the data packets have a duration that is fixed by the 802.15.1 standard specification to the values shown in Table I. The duration of the remaining 30% is uniformly distributed between minimum and maximum values (see Table I).

According to the standard, for every packet arrival time a jitter of ± 10μs has been set, to consider imperfect synchronization between the two devices. The jitter was modeled by a Gaussian distribution with zero mean and standard deviation σ=10/3μs; given the model , 99% of jitter values fell within a ± 10μs interval, while the remaining 1% exceeded this interval and were readjusted in order to meet the standard specifications.

TABLE I
BLUETOOTH STANDARD SPECIFICATION

	Fixed duration	Min. Duration	Max. Duration
Time slot	625μs		
1-time-slot-packet		126μs	366μs
3-time-slot-packet		1250μs	1622μs
5-time-slot-packet		2500μs	2870μs
NULL packet	126μs		

IV. AUTOMATIC CLASSIFICATION

We started by designing a generic linear classifier that was able to distinguish amongst C classes, each class being characterized by M features. In general, a linear classifier divides the feature space into C regions; this division comes about through calculation of discrimination functions [7] that characterize the region of the space where each class is located:

$$g_j(x) = w_{0,j} + \sum_{i=1}^{M} w_{i,j} x_i, \qquad j = 1,2,...,C \quad , \tag{1}$$

where $w = [w_0, w_1, ..., w_M]$ is known as the weight vector, and $X = [x_1, x_2, ..., x_M]$ is a point on the decision hyperplane.

The classifier objective is to find each discrimination function, and generate a decision using the major score criterion. This score represents a measure of similarity between the object and each class. The classification module was implemented on MATLAB, based on four classification methods selected because of their simplicity and computational appeal:

- Perceptron. One of the oldest methods, its convergence depends on the separability of the classes. The idea is to calculate the weight vector through an iterative method, formulated in this way [8]:

$$w(t+1) = w(t) - \rho_t \frac{\partial J(w)}{\partial w} \Big|_{w=w(t)} \quad , \tag{2}$$

where, as displayed in the equation, there is a learning coefficient ρ_t and the minimization of a cost function is required, defined in this case as :

$$J(w) = \sum_{X \in Y} \delta_X w^T X \quad , \tag{3}$$

where Y is the subset of training vectors which are misclassified by the weight vector w, and δX is a coefficient that takes value equal to -1 if X∈Class1, and equal to 1 if X∈Class2 or vice-versa. In the multiclass case, δX equal to -1 for all classes.

- Pocket. A version of Perceptron, that provides a better behavior when separability of the classes is not totally guaranteed. The key [9] is to run perceptron learning, while keeping an extra set of weights in the pocket. By this way, if a new iteration provides a weight vector that classifies a greater number of training vectors than its predecessor, then that last is chosen and used in the next step; otherwise, the vector obtained in the preceding iteration is maintained as the weight vector.
- Least Mean Squares Method (LMS). Similar to Perceptron, but the cost function, that is minimized corresponds to the error. Here, the weight vector is computed so as to minimize the Mean Square Error (MSE) between true and desired output (y) [8]:

$$J(w) = E\left[\left| y - X^T w \right|^2 \right] \quad , \tag{4}$$

where the mean value (that cannot be generated due to the lack of statistical data), is replaced by samples obtained during experimentation. The weight vector is therefore obtained according to the following rule:

$$w(k) = w(k-1) + \rho_k X_k (y_k - X_k^T w(k-1)) \quad , \tag{5}$$

where ρ_k is a learning coefficient.

- Sum of Errors Squares Estimation (SOE). Similar to LMS, here the cost function takes the form of the sum of quadratic error for each of the N training vectors X:

$$J(w) = \sum_{k=1}^{N} (y_k - X_k^T w)^2 = \sum_{k=1}^{N} e_k^2 \quad , \tag{6}$$

where the calculation of weight vectors is the simple matrix operation [8] shown below:

$$\sum_{k=1}^{N} X_k (y_k - X_k^T w) = 0 \Rightarrow (\sum_{k=1}^{N} X_k X_k^T) w = \sum_{k=1}^{N} (X_k y_k) \tag{7}$$

In order to compute both optimal w and desired values y, the Ho-Kashyap algorithm was used [10].

V. EXPERIMENTATION

A. Training set and feature extraction

As described in Section III, in the Wi-Fi case, six (6) 1000-packet captures formed the training set. In the Bluetooth case, simulated MATLAB captures consisted in two 6000-packet sequences corresponding to Scenarios 1 and 2.

The first proposed feature is the time interval between PPDUs, defined in [1] as Short Inter Frame Space (SIFS) corresponding to silence gaps on the medium when DATA-ACK procedures are in play. Most IFS timings are fixed and independent of the bitrate at the PHY [1]. Of all existing IFS types, SIFS has a nominal value of 10μs for the ISM 2.4GHz band, and is the most likely to occur in a scenario with medium to high traffic; it is usually used by a node responding to any polling, and always prior to: a) transmission of an ACK frame; b) a CTS frame; c) a second or subsequent PPDU of a fragment burst. For automatically extracting the SIFS and estimating its statistical behavior, SIFS was differentiated from a non-SIFS when two consecutive PPDU durations were such that: $0.6*PPDU_{ith} > PPDU_{ith+1}$.

The second proposed feature is the duration of the longest packet considering all the packets between two consecutive silence gaps, previously identified as SIFS.

Note that both proposed features are extremely simple and easy to extract thanks to simplest hardware such as an energy detector.

Figure 2 illustrates the feature plane for both Wi-Fi (real traffic and Bluetooth (simulated traffic) training set. The Bluetooth data correspond to Scenario 1 (single-slot case).

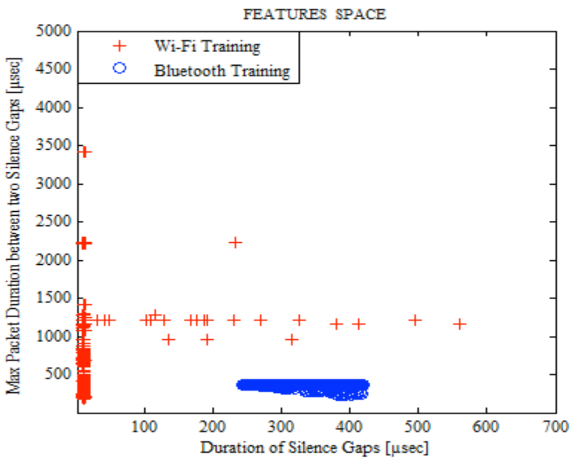


Figure 2 - Feature Plane for Wi-Fi and Bluetooth single-slot

Figure 3 shows the feature plane in the multi-slot Bluetooth Communication scenario (Scenario 2).

Note the presence of a few Wi-Fi points invading the Bluetooth "zone". Capture file revision indicated that these corresponded to non-SIFS, i.e. erroneously estimated SIFS. These points were however less than 1% of total.

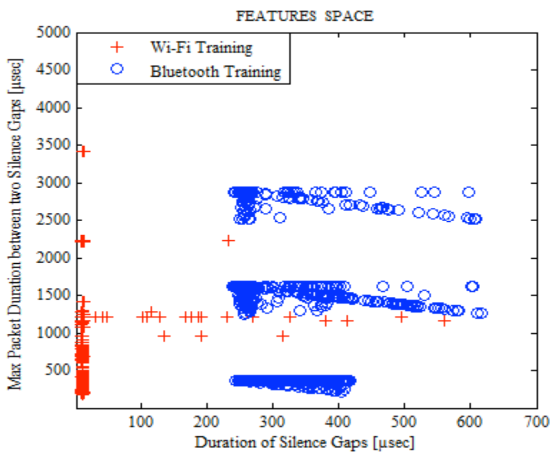


Figure 3 - Feature Plane for Wi-Fi and Bluetooth multi-slot

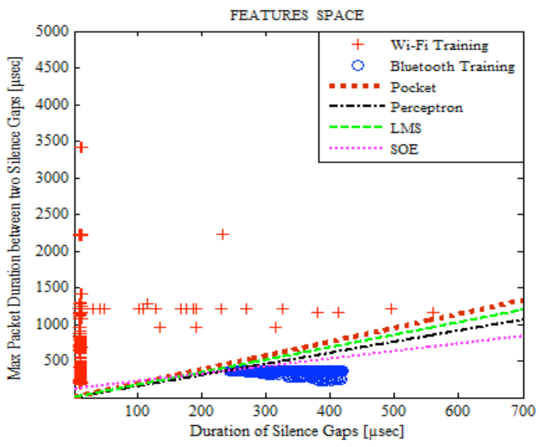


Figure 4 - Automatic classification of Wi-Fi vs. Bluetooth single-slot

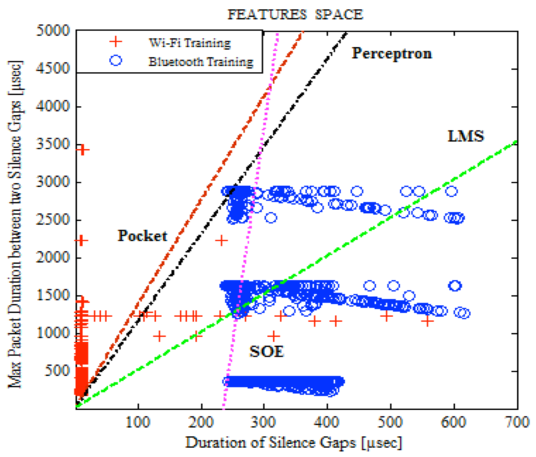


Figure 5 - Automatic classification of Wi-Fi vs. Bluetooth multi-slot

B. Results of automatic classification

The four classification algorithms were run over the training sets of Figs. 2 and 3. Results are reported on Figs. 4 and 5, for the single-slot vs. multi-slot Bluetooth cases. The classifiers were then applied to data not belonging to the training sets, i.e. a new 1000-packet Wi-Fi capture (1.4 seconds capture duration), and two new 1000-packets Bluetooth simulations (Scenarios 1 and 2) were generated (each around 0.7 seconds long). Results of classification percentage of Wi-Fi vs. Bluetooth (single-slot case), when the input to the classifier is formed by either Wi-Fi captures or Bluetooth sequences of packets are reported in Tables II and III, for the single vs. multi-slot Bluetooth, respectively.

TABLE II
CLASSIFICATION RESULTS

	Classifier Input Network	Classification into Wi-Fi	Classification into single-slot Bluetooth
Pocket	Bluetooth	0% [0/456]	100% [456/456]
Pocket	Wi-Fi	100% [352/352]	0% [0/352]
Perceptron	Bluetooth	0% [0/456]	100% [456/456]
Perceptron	Wi-Fi	100% [352/352]	0% [0/352]
LMS	Bluetooth	0% [0/456]	100% [456/456]
LMS	Wi-Fi	100% [352/352]	0% [0/352]
SOE	Bluetooth	0% [0/456]	100% [456/456]
SOE	Wi-Fi	100% [352/352]	0% [0/352]

A mixed input to the classifiers (multi-network environment) was then considered. Given that the Wi-Fi capture is on real traffic, while the Bluetooth streams were simulated, the mixture could be controlled by software. In particular, three different mixs were generated: a) predominant Wi-Fi (1000 Wi-Fi packets vs. 200 Bluetooth packets); b) balanced (1000 Wi-Fi packets vs. 1000 bluetooth packets); c) Bluetooth predominant (1000 Wi-Fi vs. 2000 Bluetooth packets). All Bluetooth sequences were multi-slot. Wi-Fi captures were 1.4 seconds long, while Bluetooth simulations lasted 0.16, 0.75 and 1.6 seconds. Due to differences in the duration of captures only partial overlapping in the combined packet sequences was achieved. Results for this test are displayed on Table IV.

TABLE III
CLASSIFICATION RESULTS

	Classifier Input Network	Classification into Wi-Fi	Classification into multi-slot Bluetooth
Pocket	Bluetooth	0% [0/462]	100% [462/462]
Pocket	Wi-Fi	98.86% [348/352]	1.14% [4/352]
Perceptron	Bluetooth	0.43% [2/462]	99.57% [460/462]
Perceptron	Wi-Fi	98.86% [348/352]	1.14% [4/352]
LMS	Bluetooth	34.85% [161/462]	65.15% [301/462]
LMS	Wi-Fi	99.43% [350/352]	0.57% [2/352]
SOE	Bluetooth	29.87% [138/462]	70.13% [324/462]
SOE	Wi-Fi	99.72% [351/352]	0.28% [1/352]

TABLE IV
CLASSIFICATION RESULTS
MULTI-NETWORK ENVIRONMENT

Classifier	Input Network	Classification into Wi-Fi	Classification into multi-slot Bluetooth
Pocket	Bluetooth predominant	17.10% [133/778]	82.90% [645/778]
Pocket	Wi-Fi predominant	86.07% [315/366]	13.93% [51/366]
Pocket	Balanced	41.34% [210/508]	58.66% [298/508]
Perceptron	Bluetooth predominant	17.22% [134/778]	82.78% [644/778]
Perceptron	Wi-Fi predominant	86.07% [315/366]	13.93% [51/366]
Perceptron	Balanced	41.53% [211/508]	58.47% [297/508]
LMS	Bluetooth predominant	37.79% [294/778]	62.21% [484/778]
LMS	Wi-Fi predominant	90.16% [330/366]	9.84% [36/366]
LMS	Balanced	56.89% [289/508]	43.11% [219/508]
SOE	Bluetooth predominant	36.89% [287/778]	63.11% [491/778]
SOE	Wi-Fi predominant	90.71% [332/366]	9.29% [34/366]
SOE	Balanced	56.10% [285/508]	43.90% [223/508]

Given that construction of the decision hyperplane is a one-time occurrence, algorithms timely performance was not deemed relevant in the context of this work. Once features are extracted, classification is automatically computed through a scalar product of feature vector and weight vector *w, the normal of the above-mentioned hyperplane*.

VI. DISCUSSION OF RESULTS AND FUTURE DIRECTIONS

As described in the above Section, network classification of Wi-Fi vs. Bluetooth was attempted based on the definition of two features: the maximum packet duration between two silence gaps, and duration of silence gaps. Four different classification algorithms were used: Pocket, Perceptron, LMS, and SOE
Results of classification showed that:
1) For the Wi-Fi vs. single-slot Bluetooth case (Table II), all proposed classifiers achieved perfect classification into the two classes, when one traffic stream (either Wi-Fi or Bluetooth) was given as input to the classifier. This result shows that the selected features were appropriate since they completely identify these two classes.
2) For the Wi-Fi vs. multi-slot Bluetooth case (Table III), classification is not as perfect as in the previous case, and depends upon classification algorithm as well as input data to the classifier. Among all the proposed classification strategies, Pocket and Perceptron emerge as the most successful and reliable, leading to a correct classification rate greater than 98%.
3) Data in Table IV speak to the adequacy of the classifiers in environments with heavy predominance of one technology, by their ability to reveal both technologies in each case. This ability is shown by comparing results of Pocket reported by Tables III and IV. As shown by tables, only Pocket and Perceptron are capable of performing a reliable classification. Note that these classifiers were always capable of providing as output, the pre-dominant network, and moreover, the rate of classification follows the trend in the proportion between both technologies packets in the observation sequence. When the traffic flows are balanced, the classifier seems to follow a “50-50” “win-win” rule, by outputting balanced classification decisions.

Future work will focus on investigating whether the selected features extend beyond the present case of two technologies in the ISM band. In particular, the AIR-AWARE project will proceed by incorporating the IEEE 802.15.4 technology (ZigBee) [11] into the set of possible classes. Preliminary investigations, based on the analysis of the 802.15.4 standard specifications, show that SIFS is also defined for ZigBee networks, with a nominal value of $192\mu s$ [11] in the ISM 2.4 *GHZ* band. This value compared to extracted features on this paper experiments, should allow the classification algorithms to obtain good separation for all three classes (Wi-Fi vs. Bluetooth vs. ZigBee).

REFERENCES

[1] IEEE Std 802.11 – 2007, 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, 12 June 2007
[2] IEEE Std 802.15.1 – 2005, IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 15.1: Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs), 14 June 2005.
[3] Gandetto M. and Regazzoni C., “Spectrum Sensing: A Dsitributed Approach for Cognitive Terminals,” *IEEE Journal on selected areas in communications*, Vol.25 (3), 2007.
[4] http://netresearch.ics.uci.edu/kfujii/jpcap/doc/
[5] http://en.wikipedia.org/wiki/Monitor_mode/
[6] http://www.radiotap.org/
[7] Duda R.O., Hart P. E., and Stork D.G., *Pattern classification*, 2° Ed., Wiley-Interscience, 2004.
[8] Theodoridis S. and Koutroumbas K., *Pattern recognition*, 4° Ed., Elsevier Inc., 2009.
[9] Gallant S. I., Perceptron-Based Learning Algorithms, *IEEE Transactions on neural networks*, Vol. 1(2), 1990.
[10] Ho Y.H. and Kashyap R.L. “An algorithm for linear inequalities and its applications,” *IEEE Transactions on Electronic Computers*,Vol.14(5), 1965.
[11] IEEE Std 802.15.4 – 2006, IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs), 8 September 2006.