CAPTURING BLUETOOTH TRAFFIC IN THE WILD: PRACTICAL SYSTEMS AND PRIVACY IMPLICATIONS

By

Wahhab Albazrqaoe

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science - Doctor of Philosophy

2018

ABSTRACT

CAPTURING BLUETOOTH TRAFFIC IN THE WILD: PRACTICAL SYSTEMS AND PRIVACY IMPLICATIONS

By

Wahhab Albazrqaoe

Bluetooth wireless technology is today present in billions of smartphones, mobile devices, and portable electronics. With the prevalence of personal Bluetooth devices, a practical Bluetooth traffic sniffer is of increasing interest due to the following. First, it has been reported that a traffic sniffer is an essential, day-to-day tool for Bluetooth engineers and applications developers [4] [14]; and second, as the communication between Bluetooth devices is privacy-sensitive in nature, exploring the possibility of Bluetooth traffic sniffing in practical settings sheds lights into potential user privacy leakage. To date, sniffing Bluetooth traffic has been widely considered an extremely intricate task due to wideband spread spectrum of Bluetooth, pseudo-random frequency hopping adopted by Bluetooth at baseband, and the interference in the open 2.4 GHz band.

This thesis addresses these challenges by introducing novel traffic sniffers that capture Bluetooth packets in practical environments. In particular, we present the following systems. (i) BlueEar, the first practical Bluetooth traffic sniffing system only using general, inexpensive wireless platforms. BlueEar features a novel dual-radio architecture where two inexpensive, Bluetooth-compliant radios coordinate with each other to eavesdrop on hopping subchannels in indiscoverable mode. Statistic models and lightweight machine learning tools are integrated to learn the adaptive hopping behavior of the target. Our results show that BlueEar maintains a packet capture rate higher than 90% consistently in dynamic settings. In addition, we discuss the implications of the BlueEar approach on

Bluetooth LE sniffing and present a practical countermeasure that effectively reduces the packet capture rate of sniffer by 70%, which can be easily implemented on the Bluetooth master while requiring no modification to slave devices like keyboards and headsets. And *(ii) BlueFunnel*, the first low-power, wideband traffic sniffer that monitors Bluetooth spectrum in parallel and captures packet in realtime. BlueFunnel tackles the challenge of wideband spread spectrum based on low speed, low cost ADC (2 Msamples/sec) to subsample Bluetooth spectrum. Further, it leverages a suite of novel signal processing algorithms to demodulate Bluetooth signal in realtime. We implement BlueFunnel prototype based on USRP2 devices. Specifically, we employ two USRR2 devices, each is equipped with SBX daughterboard, to build a customized software radio platform. The customized SDR platform is interfaced to the controller, which implements the digital signal processing algorithms on a personal laptop. We evaluate the system performance based on packet capture rates in a variety of interference conditions, mainly introduce by the 802.11-based WLANs. BlueFunnel maintains good levels of packet capture rates in all settings. Further, we introduce two scenarios of attacks against Bluetooth, where BlueFunnel successfully reveals sensitive information about the target link.

ACKNOWLEDGMENTS

The work presented in this thesis would not have been possible without the help and support of a large group of people to whom I owe a lot of gratitude. First and foremost, I am really thankful for my advisor Dr. Guoliang Xing who has given me this tremendous opportunity to come and work with him. For more than five years, he has supported me and worked with me. From my early days, despite my unfamiliarity with research, he encouraged me to tackle real research problems and perform high quality research. I am deeply indebted to Dr. Xing and hope to be as good an advisor when guiding my own students.

I am also really grateful for Jun Huang who was like a second advisor to me. He has supported me and guided me through a lot problems. His decision to work with me on problems related to Bluetooth and WiFi has changed and shaped my entire PhD career. I truly admire and respect him.

I would like to thank my sponsor Iraqi culture office at Washington D.C. who offered me this opportunity and supported me through all of my Ph.D. journey.

I would like to thank Dr. Eric Torng and Dr. Abdol-Hossein Esfahanian, from the department of Computer Science and Engineering, for their help and support. I would like to thank Dennis Phillips for his valuable discussions, comments, and advises about electronic devices. Dennis also helped me to print electronic circuits boards, where I learned how to design printed circuits. I had great pleasure of working closely and run

discussions with students at graduate school Alireza Ameli, Husain Khalifeh, Qiu Chen, and Mohammad Mahdi Moazzami.

I would like to express my earnest gratitude to my family, my parents, my wife Zainab Alkaabi, and my children, all of whom made countless sacrifices to support me and give me the best possible opportunity for education. They have always supporting, encouraging, loving me.

TABLE OF CONTENTS

LIST OF FIGURES . . . . . **ix**

KEY TO ABBREVIATIONS . . . . . **xii**

Chapter 1 Introduction **1**
1.1 Challenges 2
1.2 Contributions 3
1.2.1 The BlueEar Bluetooth Sniffer 3
1.2.2 The BlueFunnel Traffic Sniffer 4
1.3 Organization of Thesis 5

Chapter 2 Background **6**
2.1 Bluetooth Background 6
2.1.1 Bluetooth PHY layer 6
2.1.2 Physical Channel Access 7
2.1.2.1 Bluetooth Classic Hopping Protocol 8
2.1.2.2 Bluetooth LE Hopping Protocol 10
2.1.3 Bluetooth Network 11
2.1.4 Encryption 12
2.1.4.1 Bluetooth LE Encryption 12
2.1.4.2 Bluetooth Classic 13

Chapter 3 Review of Related Work **14**
3.1 Sniffing on Bluetooth Traffic 14
3.1.1 Narrowband sniffers 14
3.1.2 Wideband sniffers 16
3.2 Cracking Bluetooth Encryption 17
3.3 Potential Privacy Leakage 18

Chapter 4 BlueEar: A Practical Bluetooth Traffic Sniffing System Based on Bluetooth Compliant Radios **20**
4.1 Introduction 20
4.2 BlueEar Overview 22
4.2.1 Objectives and Challenges 22
4.2.2 System Architecture 23
4.3 Design of BlueEar 26
4.3.1 Clock Acquisition 26
4.3.1.1 Brute Force Clock Acquisition 26
4.3.1.2 Probabilistic Clock Matching (PCM) 28
4.3.1.3 Accelerating Clock Acquisition 29
4.3.2 Subchannel Classification 31

4.3.2.1 Packet-based Classifier . . . 32
4.3.2.2 Spectrum Sensing-based Classifier . . . 34
4.3.2.3 Hybrid Classifier . . . 35
4.3.3 Selective Jamming . . . 37
4.4 Implementation . . . 39
4.4.1 Ubertooth-End Implementation . . . 40
4.4.2 Controller Implementation . . . 41
4.5 BlueEar Performance . . . 42
4.5.1 Experimental Methodology . . . 43
4.5.2 Synchronization Delay . . . 44
4.5.2.1 Delay based on PCM algorithm . . . 45
4.5.2.2 Delay based on ML algorithm . . . 45
4.5.3 Fast Varying Spectrum Context . . . 47
4.5.4 Hidden and Exposed Interference . . . 48
4.5.5 Crowded Spectrum . . . 51
4.5.6 Ambient Interference . . . 52
4.6 Implications of BlueEar . . . 52
4.6.1 Implications on Bluetooth LE Privacy . . . 52
4.6.2 Impacts on Privacy Breach . . . 54
4.6.2.1 Eavesdropping on Data Traffic . . . 55
4.6.2.2 Eavesdropping on Audio Traffic . . . 56
4.6.3 Practical Countermeasure . . . 57
4.7 Summary of Chapter . . . 58

Chapter 5 BlueFunnel: Enabling Low-Power, Wideband Sniffing of Bluetooth Traffic . . . 59
5.1 Introduction . . . 59
5.2 BlueFunnel Design . . . 61
5.2.1 Objectives and Challenges . . . 61
5.2.2 System architecture . . . 63
5.2.2.1 Analog filtering and down conversion . . . 63
5.2.2.2 Packet detection and demodulation. . . . 64
5.3 Design . . . 65
5.3.1 Bluetooth Demodulator . . . 65
5.3.1.1 Demodulating Bluetooth LE signal . . . 66
5.3.1.2 Demodulating Bluetooth Classic signals . . . 68
5.4 Implementation . . . 71
5.5 System Performance . . . 73
5.5.1 Experimental Methodology . . . 73
5.5.2 Evaluation . . . 74
5.5.2.1 Packet Capture Rate in Controlled Settings . . . 74
5.5.2.2 Packet Capture Rate in Practical Environment . . . 75
5.6 Attack Scenarios . . . 76
5.6.1 Sniffing on Pairing Phase of Bluetooth LE . . . 77
5.6.2 Sniffing on Bluetooth Mouse Traffic . . . 78

5.7 Summary of Chapter 80

Chapter 6 Conclusion and Future Work 81
6.1 Contributions 81
6.2 Future work, and Conclusion 83

BIBLIOGRAPHY 85

LIST OF FIGURES

Figure 2.1: An illustrative figure presents Bluetooth Classic subchannels that overlap with 802.11-based channels in the open 2.4 GHz ISM band. 7

Figure 2.2: Bluetooth modulated signal based on GFSK modulation scheme. The figure is adopted from [51]. 8

Figure 2.3: An illustrative example of frequency hopping channel access scheme. 9

Figure 2.4: An illustrative example of adaptive frequency hopping under the effect of 802.11-based interference in the 2.4 GHz band. 10

Figure 3.1: Narrowband sniffers utilize Bluetooth-compliant radios to capture Bluetooth packets. This figure presents three samples including (a) Ubertooth one, (b) Texas Instrument CC2540 USB Dongle Bluetooth LE sniffer, and (c) Bluefruit Bluetooth LE sniffer. 15

Figure 3.2: Wideband sniffers utilize specialized wideband radios to monitor Bluetooth spectrum in parallel. This figure presents two samples including (a) Frontline Sodora wideband sniffer, and (b) Ellisys Bluetooth Explorer. 16

Figure 4.1: The dual-radio architecture of BlueEar. Components in the gray area comprise a standard-compliant hop selection kernel. 24

Figure 4.2: An illustrative example of brute force search for clock acquisition. 27

Figure 4.3: The effect of subchannel remapping on brute force search based on the same example shown in Fig. 4.2. 28

Figure 4.4: Estimation of a piconet clock based on the ML algorithm and loss function $L(c_i) = d_i$. 31

Figure 4.5: Running examples of packet-based subchannel classification for data and voice traffic under in different spectrum contexts. The packet-based classifier calculates a probability score based on Eq. (4.1) to determine subchannel status. A subchannel is classified as bad if the probability score is below the pre-defined threshold. 33

Figure 4.6: Probability densities of interference power measured on clean and dirty subchannels. 34

Figure 4.7: Time-domain illustration of run-time training of the spectrum sensing-based classifier. 36

Figure 4.8: BlueEar prototype that consists of two Ubertooths [31]. 39

Figure 4.9: Clock acquisition delay when sniffing data and audio traffics in different spectrum contexts (characterized by the percentage of bad subchannels at the target piconet). 44

Figure 4.10: Clock acquisition delay based on the ML algorithm, which is evaluated in different spectrum contexts (characterized by the percentage of bad subchannels at the target piconet). 46

Figure 4.11: Subchannel classification accuracy in fast varying spectrum context. 47

Figure 4.12: Packet capture rate in fast varying spectrum context. 48

Figure 4.13: The gain of selective jamming under hidden interference. 48

Figure 4.14: Packet capture rates in crowded spectrum characterized by the percentage of bad subchannels at the target piconet. 49

Figure 4.15: Packet capture rates under exposed interference condition. 49

Figure 4.16: Subchannel classification accuracy in crowded spectrum characterized by the percentage of bad subchannels at the target piconet. 50

Figure 4.17: Subchannel classification accuracy under exposed interference condition. . . . 51

Figure 4.18: Packet capture rates under the ambient interference: (a) different locations in an office building (interference of a large-scale 802.11-based WLANs); (b) different distances. 51

Figure 4.19: Seriousness of privacy leakage evaluated: (a) standard; and (b) countermeasure. 55

Figure 4.20: Audio quality observed by BlueEar (without countermeasure). 56

Figure 4.21: Audio quality as the target implements countermeasure. 57

Figure 4.22: BlueEar pkt capture rates: standard and countermeasure. 58

Figure 5.1: BlueFunnel system architecture. 63

Figure 5.2: Frequency down conversion of Bluetooth signal. 64

Figure 5.3: Bluetooth real-time complex signal. 65

Figure 5.4: An illustrative example where the center frequency of Bluetooth transmitter and BlueFunnel are matched. 66

Figure 5.5: An illustrative example where the center frequency of Bluetooth LE transmitter and BlueFunnel are not matched. 67

Figure 5.6: An illustrative example where the center frequency of Bluetooth Classic transmitter and BlueFunnel are not matched. 69

Figure 5.7: A prototype of BlueFunnel sniffer, where we built a customized Software Defined Radio (SDR) platform based on two USRP2 devices as shown in (a). The software components, including packet header detector and Bluetooth signal demodulator, are implemented on the Linux-based Lenovo T530 laptop as shown in (b). The USRP2 devices are synchronized via a MIMO-Cable and each USRP2 is connected to the host (Lenovo laptop) via Gigabit Ethernet cable. 72

Figure 5.8: Packet capture rate achieved by BlueFunnel under various interference conditions in a controlled setting. 74

Figure 5.9: Packet capture rate achieved by BlueFunnel at different from the target. 75

Figure 5.10: Packet capture rate achieved by BlueFunnel at different locations in office Building. 76

Figure 5.11: Bluetooth LE encryption establishment parameters (marked with red). 77

Figure 5.12: Mouse coordinates obtained from captured Bluetooth packets. 79

KEY TO ABBREVIATIONS

AP	Access Point
AES	Advanced Encryption Standard
ADC	Analog to Digital Converter
CCM	Cipher block Chaining Message authentication
FCC	Federal Communications Commission
FEC	Forward Error Correction
ISM	Industrial, Scientific, and Medical
CSR	Cambridge Silicon Radio
CRC	Cyclic Redundancy Check
DK	Development Kit
DMA	Direct Memory Access
RAM	Random Access Memory
dBi	Decibels relative to an isotropic antenna
dBm	Decibel (referenced to milliwatts)
EDIV	Encrypted Diversifier
GB	Giga Byte
MB	Mega Byte
KB	Kilo Byte
PHY	Physical Layer
SS	Spectrum Sensing based Classifier
SSD	Solid State Drive
SVM	Support Vector Machine
PCM	Probabilistic Clock Matching
ML	Maximum Likelihood
LE	Low Energy

LNA	Low Noise Amplifier
log	Logarithm
Msamples	Million samples
MSymbols	Million Symbols
Mbps	Mega Bit Per Second
MOS	Mean Opinion Score
MIMO	Multiple Input and Multiple Output
PSNR	Peak Signal to Noise Ratio
SDR	Software Defined Radio
GFSK	Gaussian Frequency-Shift Keying
DPSK	Differential Phase-Shift Keying
OFDM	Orthogonal Frequency Division Multiplexing
FP	False Positive
FN	False Negative
RF	Radio Frequency
PC	Personal Computer
PIN	Personal Identification Number
Pkt	Packet-based Classifier
WLAN	Wireless Local Area Network
STK	Short Terms Key
LTK	Long Terms Key
USRP	Universal Software Radio Peripheral
UHD	USRP Hardware Driver
USB	Universal Serial Bus
10K	10000 US$
SKDm	Session Key Diversifier for Master
IVm	Initialization Vector for Master
μ sec	Micro Second
msec	Millisecond

Chapter 1

Introduction

Recent years have witnessed the phenomenal penetration rate of Bluetooth technology in our daily lives. About 4 billion Bluetooth units are expected to be shipped worldwide in 2018 [15]. Due to it's attractive features, such as low cost and low power consumption, Bluetooth has become the de facto wireless connectivity for wireless accessories, smart devices including smartphones, wearables including smart watches and fitness trackers, sensor-based healthcare systems [57], consumer electronic devices, in-car telematic systems like Android Auto [18] and CarPlay [5], and many other applications.

With the widespread use of Bluetooth technology, including Bluetooth Classic and Bluetooth Low Energy (LE)[1], sniffing Bluetooth traffic becomes of increasing interest due to the following. First, a passive Bluetooth traffic sniffer/analyzer has become one of the essential, day-to-day tool for Bluetooth engineers and application developers, as reported by [4] [14]. Second, as the communications between Bluetooth devices are privacy-sensitive in nature, an in-depth study on Bluetooth's resilience to traffic sniffing and potential privacy leakages become imperative.

In the following, we discuss the challenges of sniffing Bluetooth traffic and introduce our contributions to tackle the challenges.

[1] In this thesis, we use Bluetooth LE to refer to Bluetooth Low Energy unless otherwise specified.

1.1 Challenges

Sniffing Bluetooth traffic has been considered an extremely intricate task due to the following,

(*i*) *Wideband Spread Spectrum.* Bluetooth operates in the free 2.4 GHz band and defines several subchannels that spread over 80 MHz of the spectrum. Therefore, a sniffer would need a wideband radio to monitor the entire spectrum and capture all Bluetooth packets in realtime. However, this wideband radio solution is challenging because it requires a high-speed (about twice the spectrum, according to Nyquist-Shannon rate) analog-to-digital-converters (ADCs) to capture Bluetooth signal. These ADCs are costly, power hungry, and require high computational power system to support such high sample rate.

(*ii*) *Pseudo-random frequency hopping.* As Bluetooth occupies one subchannel, that is 1-2 MHz, at a time for data exchange, a sniffer could employ a low-power, off-the-shelf Bluetooth-compliant radio to capture data packets. At the baseband, however, Bluetooth adopts frequency hopping spread spectrum, which is a challenge for this kind of sniffers. That is, frequency hopping sequence is known to legitimate Bluetooth devices but not other parties. Therefore, a sniffer can listen on an arbitrary subchannel, and capture packets that are randomly transmitted on that subchannel. However, the number of packets captured in this way represents a tiny portion of the traffic. Moreover, Bluetooth sessions may last for a very short period of time and involve only tens of packets.

(*iii*) *Interference in the crowded 2.4 GHz band.* When coexisting with other wireless devices in the crawdad 2.4 GHz band, Bluetooth may experience various interference conditions, including hidden and exposed interference, from 802.11-based WLANs and other ambient radios. Therefore, Bluetooth adopts adaptive frequency scheme to adapt spectrum utilization and improve performance. To tackle these challenges, a sniffier needs to learn adaptive frequency hopping behavior, and provide some mechanism to null the interference in the case of collision with Bluetooth packets in realtime.

1.2 Contributions

This thesis tackles the challenges of sniffing Bluetooth traffic in practical environment. The contributions of the thesis are summarized as follows.

1.2.1 The BlueEar Bluetooth Sniffer

We present the design, implementation, and evaluation of *BlueEar* – the first practical Bluetooth traffic sniffer that consists only of inexpensive, Bluetooth-compliant radios. BlueEar features a novel *dual-radio architecture*, where two radios – named as *scout* and *snooper* – coordinate with each other on learning the hopping sequence of indiscoverable Bluetooth, predicting adaptive hopping behavior, and handling complex interference conditions. We implemented a prototype of BlueEar for sniffing the traffic of Bluetooth Classic, which offers enhanced data rates and a more complex hopping protocol than Bluetooth LE. The prototype employs two Ubertooths [31] to implement the scout and the snooper, and interfaces them to a controller running on a Linux laptop. Extensive

experiments results show that BlueEar can maintain a packet capture rate higher than 90% consistently in practical environments, where the target Bluetooth network exhibits diverse hopping behaviors in the presence of interference from coexisting 802.11-based WLANs. Further, we discuss privacy implications of BlueEar on Bluetooth system and propose a practical countermeasure approach that effectively reduces the average packet capture rate of a sniffer to 20%.

1.2.2 The BlueFunnel Traffic Sniffer

We introduce the design, implementation, and evaluation of *BlueFunnel* –a low-power, wideband traffic sniffer that monitors Bluetooth spectrum in parallel and captures packet in realtime. BlueFunnel leverages low speed ADC to subsample Bluetooth spectrum, which addresses the challenge of high-speed ADC. In particular, BlueFunnel consists of two main components, including, *(i)* a software defined radio (SDR), that integrates a wideband radio, and a low-speed ADC. The ADC is used to subsample Bluetooth spectrum based on 2 Msamples/sec, which is below the Nyquist-Shannon rate; and *(ii)* a controller, that implements digital signal processing packages, which are responsible for detecting, demodulating, and analyzing Bluetooth packets in realtime. We implement BlueFunnel prototype based on USRP2 devices. Specifically, we utilize two USRR2 devices, each is equipped with SBX daughterboard, to build a customized software radio platform. The customized SDR platform is interfaced to the controller, which implements the digital signal processing algorithms on a personal laptop. We evaluate the system performance based on packet capture rates in variety of settings. Specifically, we evaluated the system in a controlled setting, where we intentionally introduce 802.11-based traffic as a form of interference on overlapped Bluetooth subchannels. We test BlueFunnel

when there is no interference, 25% and 50% of the subchannels are interfered with 802.11-based traffic, respectively. Moreover, we evaluate the system in an office building, where dynamic interference conditions are introduced by enterprise 802.11-based WLANs. In all settings, BlueFunnel maintains good levels of packet capture rates. Further, we introduce two scenarios of attacks against Bluetooth, where BlueFunnel successfully reveals sensitive information about the target link.

1.3 Organization of Thesis

The rest of this thesis is organized as follows. Chapter 2 introduces the basic characteristics of Bluetooth system, and discusses the challenges of sniffing Bluetooth traffic. Chapter 3 provides a review of related works. Chapter 4 presents BlueEar, the first practical system that consists of only inexpensive, Bluetooth-compliant off-the-shelf radios. Chapter 5 introduces BlueFunnel, a novel low-power, wideband Bluetooth traffic sniffer. Chapter 6 concludes this thesis and discusses future work.

Chapter 2

Background

In this chapter, we present the background for the thesis including the basic characteristics of Bluetooth baseband.

2.1 Bluetooth Background

In this section, we introduce a general background about Bluetooth physical layer, including modulation scheme, frequency hopping protocol, encryption techniques adopted by Bluetooth system, including Bluetooth Classic and Bluetooth LE.

2.1.1 Bluetooth PHY layer

Bluetooth operates in the free 2.4 GHz ISM band. Bluetooth classic defines 79 adjacent subchannels, where each subchannel occupies 1 MHz of bandwidth, as illustrated in Fig. 2.1. On the other hand, Bluetooth LE defines 40 adjacent subchannels, where each subchannel occupies 2 MHz of bandwidth. Bluetooth subchannels occupy about 80 MHz of the spectrum starting from 2402 MHz up to 2480 MHz.

At the baseband, Bluetooth adopts the symbol transmission rate of 1 Million Symbols/sec, where each symbol is modulated based on Gaussian Frequency-Shift Keying (GFSK) modulation scheme. The transmitted signal can be described as a cosine wave

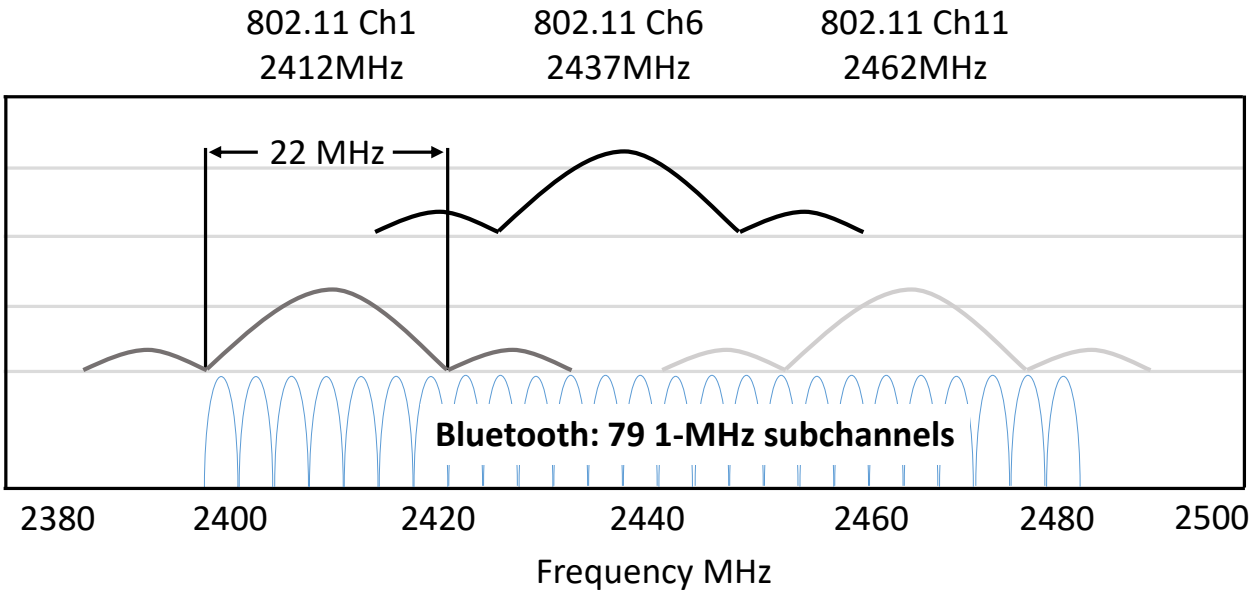


Figure 2.1: An illustrative figure presents Bluetooth Classic subchannels that overlap with 802.11-based channels in the open 2.4 GHz ISM band.

[53],

$$x(t) = A\cos(2\pi(f_c + \delta)t) \tag{2.1}$$

where f_c is the carrier frequency, δ is a frequency shift introduced by the GFSK modulation, and $0 < \delta < 0.5$ represents '1', while $-0.5 < \delta < 0$ represents '0', as depicted in Fig. 2.2. There is a special case for Bluetooth Classic to enhance data rates, where Differential Phase-Shift Keying (DPSK) modulation scheme is also supported. However, this modulation is beyond the scope of this thesis and may be considered as a future work.

2.1.2 Physical Channel Access

At the baseband, Bluetooth utilizes *frequency hopping spread spectrum* to access subchannels. That is, the carrier frequency f_c is switched every a period of time. In the following, we introduce a general description about the hopping protocols of Bluetooth Classic and Bluetooth LE, respectively.

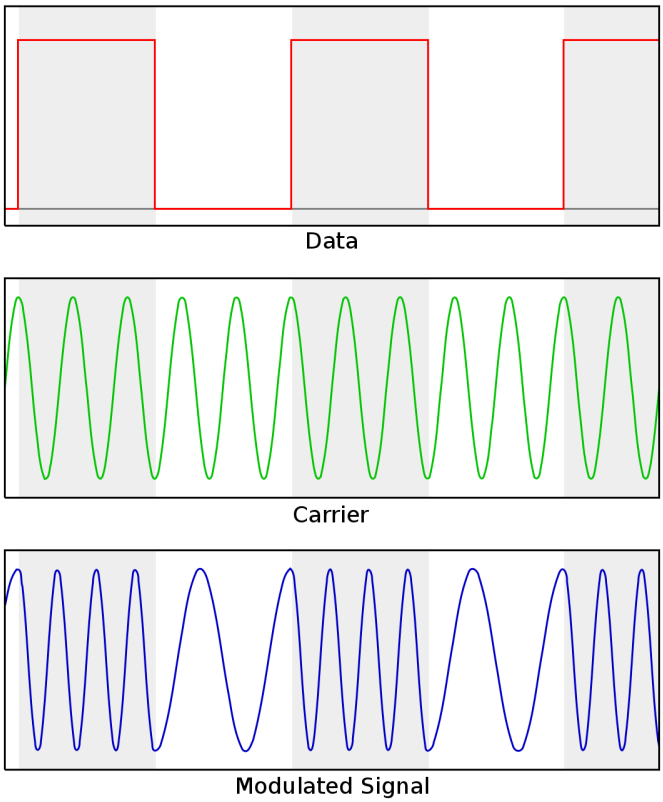


Figure 2.2: Bluetooth modulated signal based on GFSK modulation scheme. The figure is adopted from [51].

2.1.2.1 Bluetooth Classic Hopping Protocol

In Bluetooth Classic, the carrier frequency f_c is switched every 625 µsec, resulting in a maximum hopping rate of 1600 hops/sec, as depicted in Fig. 2.3. The carrier frequency is calculated as $f_c = 2402 + k$ MHz, where $k = 0, 1, 2, ..., 78$ is the subchannel index. Now the question is how to calculate k? The subchannel index is calculated in two different ways based on the two physical channel defined by Bluetooth Classic. Specifically, there are two types of physical channel for data communication, including,

(*i*) *Basic channel*. In each hop of the basic channel, the subchannel index is calculated by $\mathcal{H}(A, c)$, where $\mathcal{H}(\cdot)$ denotes the basic hop selection kernel specified by the Bluetooth standard [47], A is the piconet address (i.e. the MAC address of the piconet

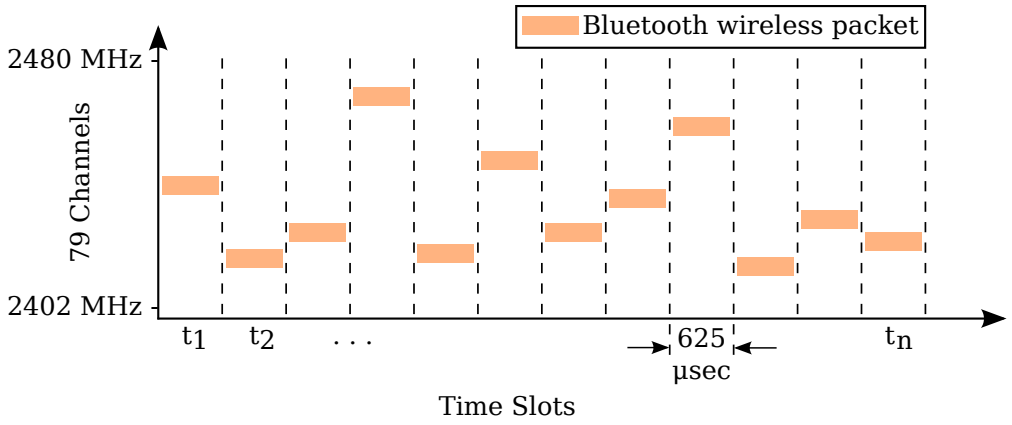


Figure 2.3: An illustrative example of frequency hopping channel access scheme.

master), and c is the piconet clock. In Bluetooth Classic, the piconet clock is a 27-bit logic counter that ticks every hop. Because piconet clock wraps around every 2^{27} ticks, the basic channel repeats itself every 134,217,728 hops, which takes approximately 24 hours. In other words, the basic channel can be characterized by a *basic hopping sequence* and a *hopping phase*. The basic hopping sequence, which is determined by the piconet address, is composed of 2^{27} subchannel indices $\{i_0, ..., i_{2^{27}-1}\}$, where $i_c = \mathcal{H}(A, c)$. The hopping phase, which is determined by the piconet clock, is the index of the current hop.

(*ii*) *Adapted channel*. Operating in the free 2.4 GHz ISM band, Bluetooth may coexist with other wireless devices on overlapped frequencies. Therefore, Bluetooth performs adaptive hopping where the basic channel is frequently modified to adapt spectrum use. The adaptation is guided by a *subchannel map*, which classifies subchannels as good and bad based on interference conditions. When the subchannel selected in a hop is bad, a *remap* function is called to compute a pseudo-random index i based on piconet address and clock. The bad subchannel is then replaced by the i-th good subchannel.

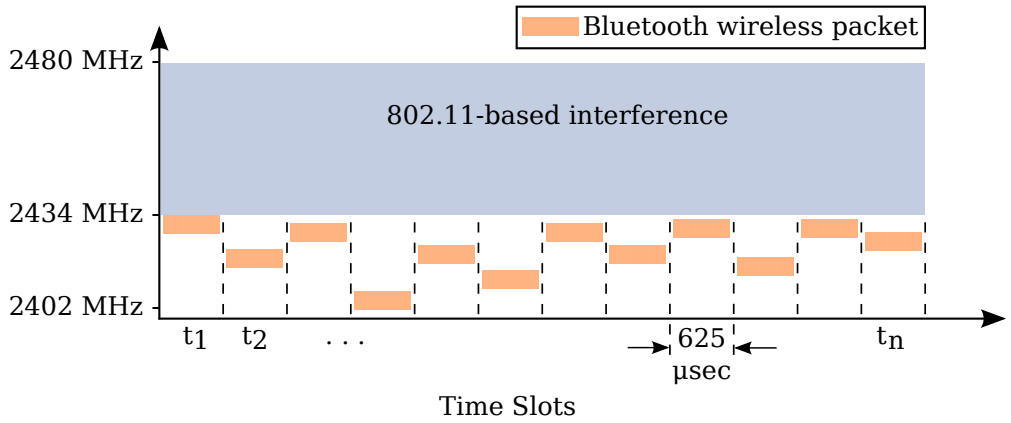


Figure 2.4: An illustrative example of adaptive frequency hopping under the effect of 802.11-based interference in the 2.4 GHz band.

Fig. 2.4 shows an illustrative example of adaptive frequency hopping, where Bluetooth avoids interfered subchannels (gray). Although adaptive hopping is the *de facto* scheme used by off-the-shelf Bluetooth devices, the Bluetooth standard does not specify the implementation of subchannel classification, resulting in different adaptive hopping behaviors across devices manufactured by different vendors.

2.1.2.2 Bluetooth LE Hopping Protocol

Bluetooth LE defines two types of physical channel, namely *(i) advertisement*, and *(ii) connection*. For advertisement, there are three subchannels, namely 37, 38, and 39, where Bluetooth LE operates in a connectionless mode. That is, Bluetooth LE utilizes the advertisement subchannels to broadcast information for other Bluetooth LE devices. Further, the three subchannels may be used to initiate connections. For connection state, Bluetooth LE utilizes the remaining 37 subchannels to establish data connections between Bluetooth LE devices. The connection state of Bluetooth LE is defined based on connection events, where an event represents a short session to exchange data packets. For each event, Bluetooth LE switches to a new subchannel.

Similar to Bluetooth Classic, Bluetooth LE defines a *basic* and *adaptive* hopping schemes. For the basic channel hopping, the carrier frequency f_c is calculated as $f_c = 2402 + 2k$ MHz, where $k = 0, 1, 2, ..., 39$ is the subchannel index. The index is calculated based on $\mathcal{K}(c_i, inc)$, where $\mathcal{K}(.)$ is the hop selection kernel, and c_i is the index of current subchannel. The subchannel index c_i is initialized to zero for the first connection event. Therefore, the hopping sequence repeats whenever the index becomes $c_i = 0$ [46]. During pairing phase (i.e. connection establishment), the master defines connection configuration parameters. This includes *(i)* connection interval –a multiple of $1.25ms$ ranging from $7.5ms$ to $4.0s$ that defines the event lifetime; *(ii)* transmission window size –a multiple of $1.25ms$ that defines the size of transmit window, i.e. packet size; and *(iii)* hop increment inc –a random value ranges from 5 to 16.

The adaptive channel is defined based on a *subchannel map* that classifies the connection subchannels into good and bad. When a bad subchannel is selected by the selection kernel $\mathcal{K}(c, inc)$, a remapping procedure is invoked to calculate a *remapped index*. The master maintains the subchannel map and it notifies slave(s) about any updates [46].

2.1.3 Bluetooth Network

Bluetooth networks employ a master-slave structure called a *piconet*. The device that has the least computation and power constraints is usually selected as the master to manage communication. Other devices are slaves. Bluetooth piconets use the MAC address of the master device as the *piconet address*. All devices from the same piconet are synchronized to the *piconet clock* – a clock signal generated by the master.

Indiscoverable mode. When establishing the piconet, Bluetooth devices authenticate each other through a *pairing* process. To enhance privacy, Bluetooth piconets can be put

in indiscoverable mode to hide key technical parameters from unpaired devices. These parameters – including piconet address, piconet clock, and subchannel map – determines hopping behavior. Although recent study has demonstrated the possibility of extracting piconet address from packet preambles [49], a Bluetooth packet sniffer cannot hop following the target unless it acquires all hidden parameters.

2.1.4 Encryption

2.1.4.1 Bluetooth LE Encryption

To preserve user's privacy, Bluetooth LE employs AES encryption –a popular symmetric-key encryption algorithm. This means the piconet master and slave(s) must share the same encryption key, that is known as Long Term Key (LTK), to establish encryption session. Key distribution mechanism of Bluetooth LE is different from that adopted by Bluetooth Classic; this is because of computational and storage limitations of low power Bluetooth LE slaves. Unlike Bluetooth Classic, which employs Secure Simple Pairing (SSP) for key agreement, a Bluetooth LE master generates and distributes LTK as encrypted message, based on a temporary key aka Short Term Key (STK), to other slaves. Now to decrypt the LTK, all piconet participants needs to regenerate the STK. Bluetooth LE slaves base on some seeds and random numbers provided by the master device. Unfortunately, these seeds are sent over the air as plaintext. As a result, an eavesdropper, who can capture pairing phase of Bluetooth LE, may be able to determine LTK and, hence, compromise Bluetooth LE privacy [6] [7] [8] [12] [36] [43] [45].

2.1.4.2 Bluetooth Classic

As communication of Bluetooth devices is privacy-sensitive in nature, Bluetooth Classic employs E_0 –a symmetric-key stream cipher algorithm– to encrypt user's data. Unfortunately, recent studies have revealed many critical flaws of this encryption scheme [55] [56] [13] [38]. In particular, it is shown that Bluetooth is subject to several practical attacks that can circumvent, compromise, or even crack the link-layer encryption, leading to potential user privacy breach [13] [56].

Chapter 3

Review of Related Work

Despite the well-documented flaws of Bluetooth encryption, sniffing Bluetooth traffic has been widely considered an extremely intricate task. This mainly due to wideband spread spectrum, pseudo-random frequency hopping employed by Bluetooth at baseband, and presence of interference in the open 2.4 GHz band, which is mainly introduced by 802.11-based WLANs. In this chapter, we review related work from the literature, which includes Bluetooth sniffers, cryptanalysis of Bluetooth encryption, and potential privacy leakage in the light of successful traffic sniffing.

3.1 Sniffing on Bluetooth Traffic

In this section, we review Bluetooth traffic sniffers, highlight their drawbacks, and compare our practical systems, namely BlueEar and BlueFunnel, that significantly address the challenges and improve traffic sniffing performance. Bluetooth traffic sniffers can be categorized into two main classes including:

3.1.1 Narrowband sniffers

To sniff on Bluetooth traffic, narrowband sniffers employ Bluetooth-compliant radios to capture Bluetooth traffic. Fig. 3.1 presents samples of narrowband Bluetooth sniffers.

(a) Ubertooth one. (b) Texas Ins. CC2540 Dongle. (c) Bluefruit Bluetooth LE Sniffer.

Figure 3.1: Narrowband sniffers utilize Bluetooth-compliant radios to capture Bluetooth packets. This figure presents three samples including (a) Ubertooth one, (b) Texas Instrument CC2540 USB Dongle Bluetooth LE sniffer, and (c) Bluefruit Bluetooth LE sniffer.

There exist several proprietary and open source systems for sniffing Bluetooth traffic. For example, the firmware of a few Bluetooth chipsets allow the radio to report packet-level diagnosis by working in sniffing mode [1] [24] [33]. However, the sniffing device must pair with the target, which makes it incapable of passive sniffing.

Recently, several proprietary low-cost Bluetooth packet sniffers [23] [44] [2] [34] have been proposed based on inexpensive, Bluetooth compliant Bluetooth platform. Similarly, low cost, open source sniffers [31] [44] have been developed based on Ubertooth [31] – an open-source Bluetooth development platform. In [50], a traffic sniffer is developed based on GNURadio/USRP platform. However, existing low-cost sniffers, including Ubertooth-based one, are exclusively designed for sniffing Bluetooth traffic in basic hopping mode. In the crowded 2.4 GHz band, Bluetooth rarely hops in basic hopping mode because of the interference from coexisting wireless devices, especially the prevalent 802.11 based WLANs [17] [28] [54] [20]. As a result, existing low-cost sniffers suffer prohibitively poor sniffing performance in practice due to the misprediction of adaptive hopping behavior, as well as excessive packet corruptions caused by interference.

(a) Sodora wideband Sniffer. (b) Bluetooth Explorer.

Figure 3.2: Wideband sniffers utilize specialized wideband radios to monitor Bluetooth spectrum in parallel. This figure presents two samples including (a) Frontline Sodora wideband sniffer, and (b) Ellisys Bluetooth Explorer.

Compared with existing Ubertooth-based systems, BlueEar is designed for sniffing Bluetooth traffic in practical environments where both the sniffer and the target may suffer from intensive interference from coexisting wireless devices. To achieve this goal, we address the key challenges posed by the indiscoverable mode of Bluetooth, the vendor-dependent adaptive hopping behavior, and the difficulties in maintaining consistent sniffing performance in the crowded 2.4 GHz band. In addition, we identify various critical issues in Ubertooth firmware that significantly degrade its performance during frequency hopping, and present solutions to address these flaws. We note that although our prototype is developed based on Ubertooth, the design of BlueEar is platform-independent and can be easily ported to other systems.

3.1.2 Wideband sniffers

Wideband traffic sniffers employ a wideband radios to monitor the entire Bluetooth spectrum at the same time. Fig. 3.2 shows examples of wideband Bluetooth sniffers. There are a few proprietary Bluetooth sniffers designed for testing and protocol analysis [27]

[14]. These sniffers utilize specialized radios to monitor all subchannels in parallel which makes them extremely costly and power hungry. For instance, protocol analyzers manufactured by Teledyne [27] cost $10k to $25K per unit. Moreover, these sniffers require explicit pairing with the target Bluetooth, which makes it impossible to sniff on Bluetooth traffic passively.

In comparison with the above systems, BlueFunnel is designed as a passive traffic sniffer that is based on wideband, low-power, inexpensive devices along with a suit of digital signal processing methods to capture Bluetooth traffic in the wild. Moreover, BlueFunnel incurs no delay and no packet loss as it requires no statistical learning phase.

3.2 Cracking Bluetooth Encryption

Bluetooth Classic employs E_0 – a two-level stream cipher based on 128-bit link key – to encrypt packet payloads. The link key is established through *pairing*, where Bluetooth devices authenticate each other using a secret PIN. In the literature, studies have revealed many critical flaws of this encryption scheme [55] [56] [13] [38] [37] [40] [11].

In particular, several studies on E_0 cryptanalysis have shown that the 128-bit link key of E_0 is considerably weaker than what is originally intended [55] [56] [37]. The encryption key can be cracked with 2^{27} online operations and $2^{21.1}$ offline operations instead of 2^{128}. Specifically, given the headers of $2^{22.7}$ Bluetooth packets, the attack takes a few seconds to restore the target encryption key. Such attack is implemented on a single core PC and requires 4 MB RAM of memory. Furthermore, the effective security of the link key will further degrade when a packet sniffer is employed by the attacker.

3.3 Potential Privacy Leakage

In the light of successful traffic sniffing, we explore potential privacy leakages of Bluetooth system, including,

(i) Cracking Bluetooth encryption. Bluetooth classic employs E_0 – a two-level stream cipher based on 128-bit link key – to encrypt packet payloads. Recent studies on E_0 cryptanalysis shows that the 128-bit link key of E_0 is considerably weaker than what is originally intended [55] [56] [37]. The encryption key can be cracked with 2^{27} online operations and $2^{21.1}$ offline operations instead of 2^{128}. Specifically, given the headers of $2^{22.7}$ Bluetooth packets, the attack takes a few seconds to restore the target encryption key. Therefore, the effective security of the link key will further degrade when a packet sniffer is employed by the attacker. Moreover, due to computation and power constraints, many Bluetooth peripherals including most Bluetooth mice manufactured by major vendors, like Logitech [29], do not support encryption, leading to user privacy breach.

(ii) Privacy leakage based on traffic patterns. Without the pain of cracking Bluetooth encryption, previous studies demonstrate the possibility of Bluetooth privacy breach by observing traffic patterns. For instance, the traffic pattern of popular fitness trackers is found to be strongly correlated with the user's activity, making it possible to track user gait and identity. As a result, a passive traffic sniffer can uncover important private information about the user, even without decrypting packet payloads [13].

(iii) Leakage of encryption key. Bluetooth LE employs AES-CCM block cipher –a popular symmetric-key encryption algorithm– to decrypt packets payload. To establish an encrypted session, the piconet master shares a Long Term Key (LTK) with slave(s). The key distribution mechanism of Bluetooth LE is different from that adopted by Bluetooth

Classic, due to computational and storage limitations of low power slaves. In practice, the master generates and distributes LTK as encrypted messages to other slaves. The master encrypts LTK based on a temporary key that is known as Short Term Key (STK). Unfortunately, Bluetooth LE sends STK over the air as plain text. Therefore, a successful eavesdropping on Bluetooth LE pairing phase could potentially lead to determining LTK; and, hence, compromise Bluetooth LE privacy [8] [12] [36]. Furthermore, a study on privacy of ZigBee network [43] demonstrates the feasibility of circumventing AES-CCM encryption without the need to know link encryption key. Although, the study considers ZigBee encryption, the proposed methods can also be applied to Bluetooth LE as both ZigBee and Bluetooth LE employ the AES-CCM block cipher.

Chapter 4

BlueEar: A Practical Bluetooth Traffic Sniffing System Based on Bluetooth Compliant Radios

4.1 Introduction

In this chapter, we explore the feasibility and privacy implications of sniffing Bluetooth traffic in practical environments. Unlike prior Bluetooth traffic sniffers that rely on specialized power-hungry wideband radios, our sniffing system consists of only commodity Bluetooth-compliant radios. Our major contribution is two-fold.

The BlueEar system. We present the design, implementation, and evaluation of *BlueEar* – the first practical Bluetooth traffic sniffer that consists only of inexpensive, Bluetooth-compliant radios. BlueEar features a novel *dual-radio architecture*, where two radios – named as *scout* and *snooper* – coordinate with each other on learning the hopping sequence of indiscoverable Bluetooth, predicting adaptive hopping behavior, and handling complex interference conditions. Specifically, the scout hops over all Bluetooth subchannels to survey interference conditions and monitors the target's packet transmissions. By fusing these measurements, BlueEar uses a probabilistic clock matching algorithm to deter-

mine the basic hopping sequence, and then integrates statistical models and a lightweight machine learning algorithm to predict the adaptive hopping behavior, which allows the snooper to hop following the target. To maintain reliable sniffing performance in complex interference conditions, the scout runs a selective jamming algorithm, which manipulates the hopping of the target to mitigate the impacts of interference. We have implemented a prototype of BlueEar for sniffing the traffic of Bluetooth Classic, which offers enhanced data rates and a more complex hopping protocol than Bluetooth LE. Our prototype can be expended to sniff on Bluetooth LE traffic. The prototype employs two Ubertooths [31] to implement the scout and the snooper, and interfaces them to a controller running on a Linux laptop. We identify critical issues in Ubertooth firmware that severely degrades its performance during frequency hopping, and present novel solutions to address these flaws. Extensive experiments results show that BlueEar can maintain a packet capture rate higher than 90% consistently in practical environments, where the target Bluetooth network exhibits diverse hopping behaviors in the presence of interference from coexisting 802.11 based WLANs.

Privacy implications. We discuss the implication of the BlueEar system on Bluetooth privacy. Moreover, we evaluate the performance of BlueEar when eavesdropping on audio traffic, which is known to be extremely sensitive to packet loss. We show that the packet capture rate achieved by BlueEar translates to a high audio quality with a mean opinion score (MOS) of 3.5, which is translated into 'Fair' and 'Good' from the listener's perspective. Furthermore, we present a practical countermeasure, that can be easily implemented in the user-space driver of the Bluetooth master device, while requiring no modification to existing slave devices like keyboards and headsets. The countermeasure

effectively reduces the packet capture rate of the sniffer to 20%, and degrades the MOS of eavesdropped audio to 1.5, which is between 'Bad' and 'Poor'.

The rest of the chapter is organized as follows. We present system overview and architecture in section 4.2. In section 4.3, we discuss BlueEar system design in detail. Evaluation of BlueEar system performance is presented in section 4.5. We discuss the impacts of BlueEar on privacy breach for Bluetooth system, including Bluetooth LE, in section 4.6. Section 4.7 concludes the chapter.

4.2 BlueEar Overview

4.2.1 Objectives and Challenges

We study Bluetooth privacy by exploring the feasibility of sniffing Bluetooth traffic in practical environments. To this end, BlueEar is designed as a *passive* traffic sniffer that leverages general, inexpensive wireless platform to capture Bluetooth packets without pairing with the target piconet. To achieve this goal, we tackle the following challenges.

(*i*) *Secret hopping phase*. In indiscoverable mode, the piconet clock that indicates the hopping phase is hidden from BlueEar. In adaptive hopping mode, determining the hopping phase is particularly challenging because the basic hopping sequence of the target is subject to frequent modifications.

(*ii*) *Vendor-dependent adaptive hopping*. The adaptive hopping of Bluetooth is guided by a subchannel map, which classifies subchannels as good and bad based on dynamic interference conditions. However, the Bluetooth standard does not specify the implementation of subchannel classification, resulting in vendor-dependent implemen-

tation, where Bluetooth chipsets manufactured by different vendors may hop over different subchannels even in the same spectrum context.

(*iii*) *Interference in the crowded 2.4 GHz band*. When coexisting with other wireless devices, BlueEar may experience hidden and exposed interference. When an RF signal that interferes with the target is too weak to be measured at BlueEar, the spectrum contexts observed by BlueEar and the target may differ, making it difficult to predict adaptive hopping behavior. When an RF source interferes with the target but BlueEar, significant degradation of sniffing performance may occur. While authorized devices can maintain packet reception performance by coordinating their hopping, designed as a passive sniffer, BlueEar cannot coordinate with the target, which may lead to substantial packet corruptions.

4.2.2 System Architecture

Instead of using specialized radio to monitor all subchannels in parallel, BlueEar tackles the above challenges based on a simple *dual-radio architecture* that consists of two inexpensive, Bluetooth-compliant radios. The two radios – named as *scout* and *snooper* – are interfaced to a controller, which employs a suite of novel algorithms to coordinate the two radios, gluing them as a powerful passive traffic sniffer. Fig. 4.1 illustrates the architecture of BlueEar. Specifically, the working flow of BlueEar can be divided into the following steps.

(*i*) *Traffic filtering*. When multiple piconets coexist in the same environment, BlueEar separates their traffic based on the piconet address of captured packets. Specifically, the preamble of each Bluetooth packet carries a synchronization word, which

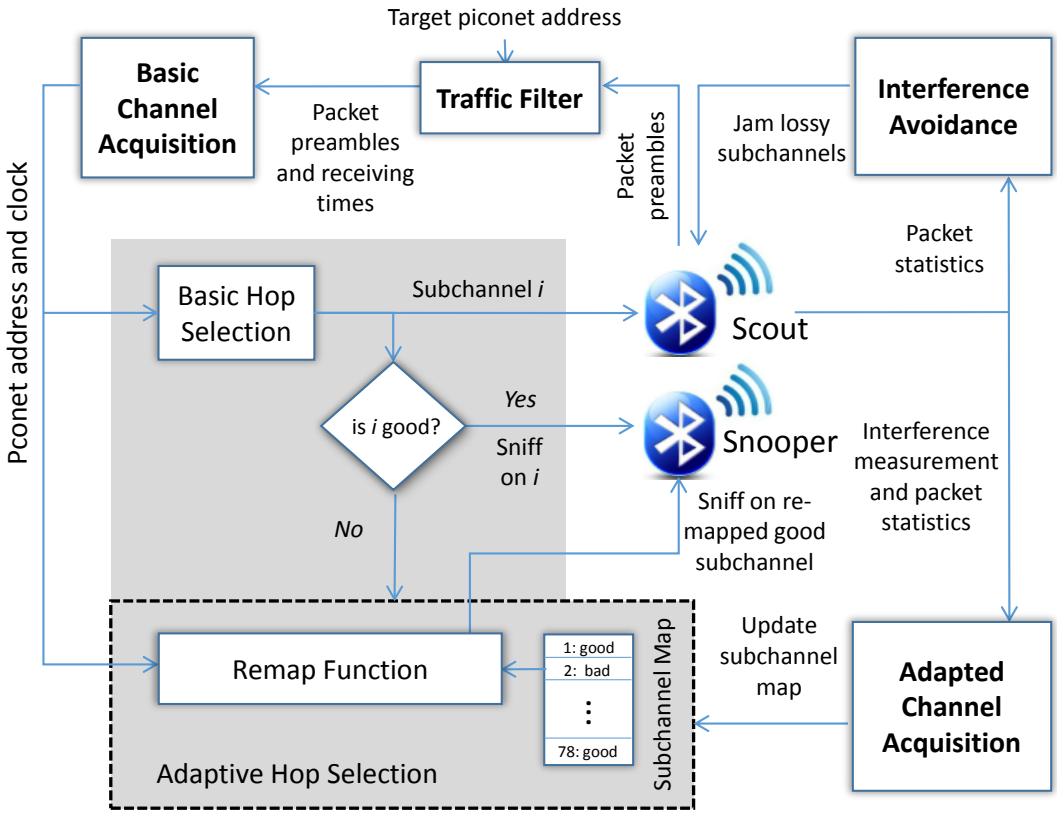


Figure 4.1: The dual-radio architecture of BlueEar. Components in the gray area comprise a standard-compliant hop selection kernel.

is derived from the piconet address using an encoding function specified by the standard [47]. BlueEar employs the brute force algorithm proposed in [49] to extract piconet address from the synchronization word, and then filters out the packets whose piconet address mismatches the target piconet address specified by the BlueEar user.

(*ii*) *Basic channel acquisition.* To acquire the basic channel of the target piconet, the scout listens on an arbitrary subchannel to monitor the target's packet transmissions. After extracting piconet address from packet preamble, BlueEar derives the entire basic hopping sequence, and then reverses the piconet clock using an interference resilient, probabilistic clock matching algorithm. Specifically, the receiving times of captured packets are compared with the basic hopping sequence at different hop-

ping phases, until a correct phase is found. The acquired piconet address and clock are then fed to a standard-compliant basic hop selection kernel to compute the basic channel.

(*iii*) *Adapted channel acquisition.* To acquire the adapted channel, BlueEar needs to predict how the target reacts in dynamic spectrum context. To this end, the scout hops over all subchannels on the acquired basic channel to survey interference conditions, and monitors packet transmissions to infer the target's subchannel utilization. By fusing these measurements, BlueEar trains a subchannel classifier at run-time, which accurately derives the target's subchannel map despite vendor-dependent adaptive hopping behavior and the possible disparity between the spectrum contexts of the scout and the target. The subchannel map is then fed to a standard-compliant adaptive hop selection kernel for computing the adapted channel.

(*iv*) *Interference avoidance.* The snooper hops following the target on the adapted channel to sniff traffic. BlueEar handles complex interference in the crowded 2.4 GHz band using a selective jamming algorithm. Specifically, a loss detector is employed to monitor the sniffing performance of all subchannels. When substantial packet corruptions are detected on a subchannel i, the scout deliberately generates interference on i while the target visits i during hopping. Because of adaptive hopping, the target will be driven away from lossy subchannels where BlueEar observes poor sniffing performance. The objective is to manipulate the target's hopping to enforce implicit coordination.

4.3 Design of BlueEar

In this section, we present the design of key BlueEar components in detail.

4.3.1 Clock Acquisition

In the following, we define some terminology and present key concepts of clock acquisition.

Definition 1 Basic hopping sequence $\beta = (i_0, .., i_c, .., i_N)$ is an n-tuple of integers $i \in \{0, .., 78\}$ that represents subchannel index, c refers to the piconet clock, i.e. hopping phase, and $N = 2^{27} - 1$.

Definition 2 Observed hopping pattern $P = \{p_0, p_1, .., p_n\}$ is a set of captured packets, where a packet is identified based on system time-stamp at reception time.

Next, we introduce our basic idea of reversing the piconet clock, which involves three fundamental algorithms, including, first reversing the piconet clock when the target hops in the basic mode; second, we present a probabilistic matching approach to determine the piconet clock in the presence of interference. And finally, we introduce a maximum-likelihood (ML) based algorithm, which accelerates clock acquisition and significantly reduces clock acquisition delay.

4.3.1.1 Brute Force Clock Acquisition

Because the entire hopping sequence β is known after the piconet address is extracted from packet preamble [49], it is possible to reverse the piconet clock c through a simple brute force search, which compares an observed hopping pattern with the entire hopping sequence at all phases to search for a match. Fig. 4.2 shows an illustrative example.

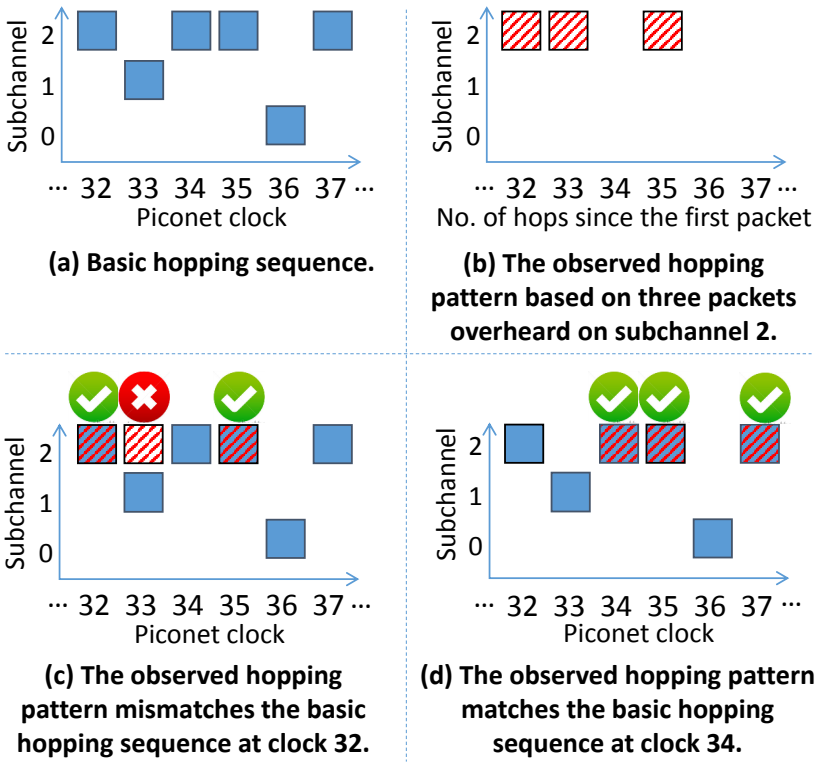


Figure 4.2: An illustrative example of brute force search for clock acquisition.

At the beginning, the scout listens on a single subchannel to monitor the target's packet transmissions. As shown in Fig. 4.2, the scout listens on subchannel 2 where three packets are captured, which defines P. The receiving times of these packets compose a hopping pattern that describes how the target visits the monitored subchannel. As shown in Fig. 4.2(c) and (d), BlueEar then compares the observed hopping pattern with the entire basic hopping sequence at all phases. A match is found at clock 34.

Before capturing a sufficient number of packets, the observed hopping pattern may happen to match the basic hopping sequence at multiple clock values, resulting in clock ambiguity. Because the basic hopping sequence is pseudo-random, probability that an observed hopping pattern that comprises n packets matches the basic hopping sequence at an arbitrary clock can be computed as $\frac{1}{79^n}$. Therefore, clock ambiguity decreases as the number of captured packets increases.

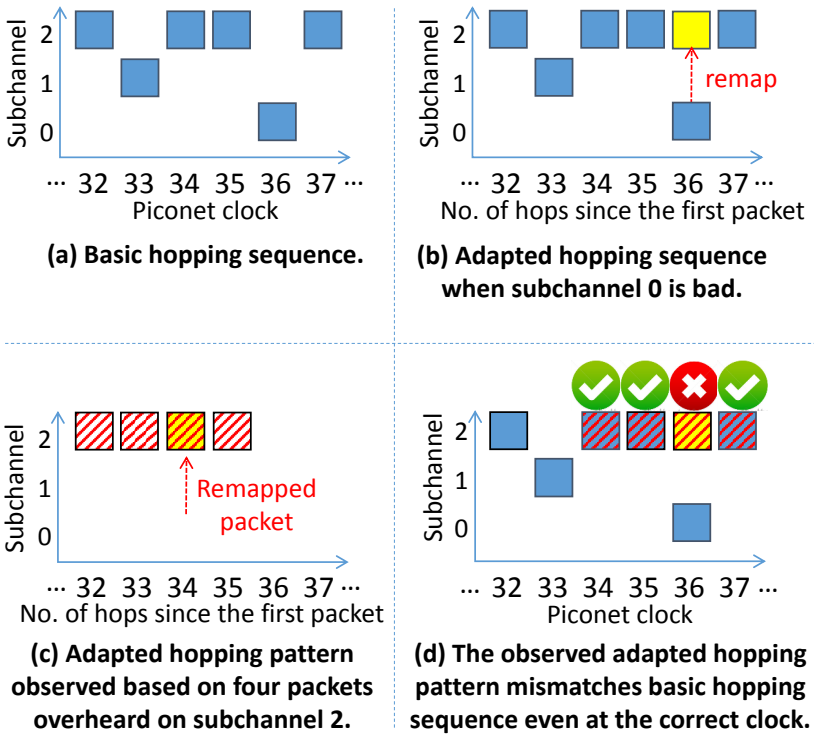


Figure 4.3: The effect of subchannel remapping on brute force search based on the same example shown in Fig. 4.2.

4.3.1.2 Probabilistic Clock Matching (PCM)

In the crowded 2.4 GHz band, Bluetooth devices rarely hop in the basic mode because of the impacts of interference from coexisting wireless devices [17] [28] [54] [20]. To adapt spectrum use, Bluetooth modifies the basic channel by remapping bad subchannels subjected to interference with pseudo-randomly selected good subchannels. Since the adapted channel may differ from the basic channel in various phases, the hopping pattern observed by the scout may mismatch the basic hopping sequence even at the correct clock. Fig. 4.3 illustrates the impact of subchannel remapping using the same example shown in Fig. 4.2. As illustrated in the figure, brute force search may fail to find a perfect match due to the packet transmitted on the remapped subchannel.

To acquire the piconet clock c in the presence of interference, BlueEar leverages the

following key observation. When comparing an observed hopping pattern P with the basic hopping sequence β at the correct clock, the ratio of mismatches should equal the ratio of remapped subchannels. As required by FCC, Bluetooth Classic must use at least 20 subchannels for frequency hopping [47]. Therefore, the ratio of remapped subchannels is at most $\frac{59}{79}$. Hence, if the ratio of mismatches at a clock c is significantly larger than $\frac{59}{79}$, then c is likely an incorrect clock.

Driven by the above observation, BlueEar employs a probabilistic clock matching (PCM) algorithm for clock acquisition. Instead of seeking a perfectly matched clock, BlueEar determines the correct clock by eliminating incorrect clocks based on the number of mismatches. A clock is identified as incorrect if the 95% confidence interval of its mismatch ratio exceeds $\frac{59}{79}$. Specifically, let β be the basic hopping sequence, $P = \{p_0, .., p_n\}$ be a set of observed packets, and $C = \{c_0, c_1, ..c_j\}$ and $D = \{d_0, d_1, ..d_j\}$ be the sets of clock candidates and the corresponding mismatches numbers, respectively. When comparing P with β, the PCM algorithm returns C and D, where d_i is the number of mismatches at clock candidate c_i. If c_i is the correct clock, the ratio of mismatches $\mu = \frac{d_i}{n}$, $n = |P|$, should be close to the ratio of unused subchannels. Based on the central limit theory, the distribution of $\sqrt{n}(\frac{d_i}{n} - \mu)$ should approach normality when n is reasonably large. The 95% confidence interval of μ can be estimated as $\frac{d_i}{n} \pm 2\frac{\sigma}{\sqrt{n}}$, where σ^2 is the variance. Therefore, clock c_i is determined as incorrect if $\frac{d_i}{n} - 2\frac{\sigma}{\sqrt{n}} \geq \frac{59}{79}$. When a new packet is captured, the algorithm updates the mismatch ratios for remaining clock candidates.

4.3.1.3 Accelerating Clock Acquisition

The PCM algorithm acquires the correct clock by gradually eliminating incorrect candidates; this may incur some time delay while waiting for new packets to be captured. To

reduce clock acquisition delay, BlueEar resorts to estimate the piconet clock. That is, given a set of observed packets P, the ML algorithm derives several subsets $P_1, .., P_z \subset P$, and invokes the PCM algorithm to obtain corresponding sets of clock candidates $C_1, .., C_z$. From the later sets, the algorithm finds probability distribution of candidates and identifies a candidate that maximizes the distribution function. In particular, the algorithm proceeds with the following steps.

Step 1: *Input*: $P = \{p_0, .., p_n\}$ and $C = \{c_0, .., c_j\}$, and $b < n$

Step 2: Define a counter $a = 1$,

Step 3: Find the subset $P_a \subset P$, $P_a = \{p_a, .., p_b\}$

Step 4: Invoke the PCM algorithm with P_a and obtains C_a

Step 5: Subtract q from all elements in C_a. Intuitively, if c was the piconet clock at the time of transmitting p_0, then $c + q$ must be the piconet clock at the time of transmitting p_a, where q is the number of clock ticks between p_0 and p_a.

Step 6: Finds $C \leftarrow C \cup C_a$

Step 7: Increments a and b, while $b < n$, goto *Step 3*

BlueEar evaluates optimality of the ML estimator based on some *loss function L*, which we define as follows: $L(c_i) = d_i$, where c_i is an approximation for the piconet clock c, and d_i is the number of mismatches at clock c_i as defined earlier. According to the PCM algorithm, a clock candidate c_i, that is aligned with high mismatches d_i, is most likely to be eliminated from C as sufficient number of packets are overheard. Based on this observation, the ML estimator c_{winner} is considered if it is aligned with the minimum mismatches number d_m; otherwise, the ML algorithm waits for new packets to be captured.

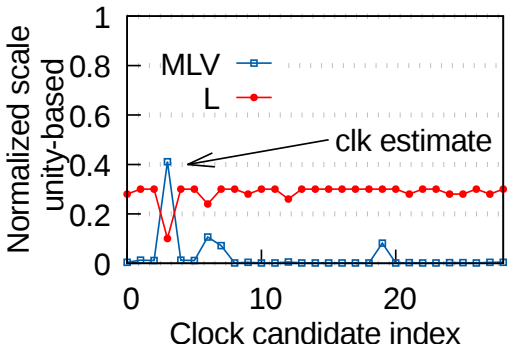


Figure 4.4: Estimation of a piconet clock based on the ML algorithm and loss function $L(c_i) = d_i$.

Fig. 4.4 shows an example of ML algorithm, where the algorithm successfully identifies the piconet clock based on a set of 20 observed packets. In this figure, it is clear that the winner candidate aligns with the minimum value of the loss function L.

4.3.2 Subchannel Classification

The adaptive hopping of Bluetooth is guided by a subchannel map that classifies subchannels as good and bad based on dynamic interference conditions. To acquire the adapted channel, BlueEar employs a subchannel classifier, which infers how the target classifies subchannels. The subchannel classifier must meet the following requirements.

- *Accuracy*. When a subchannel is misclassified, the snooper may hop to a wrong subchannel different from the one used by the target. Poor subchannel classification accuracy may result in substantial packet misses.

- *Responsiveness*. In dynamic spectrum contexts, the subchannel classifier must be responsive to the change of interference conditions.

In the following, we present two complementary subchannel classification algorithms, and discuss their advantages and limitations in achieving the above goals. We then dis-

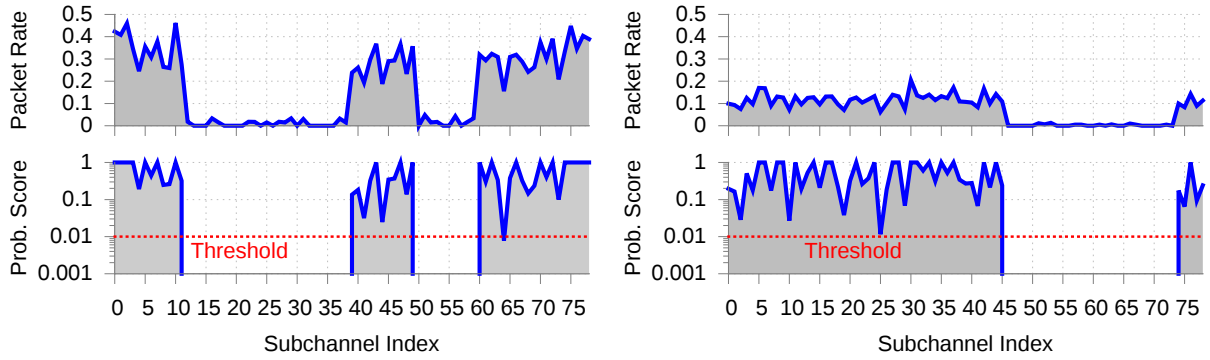
cuss how BlueEar integrates the two algorithms for subchannel classification.

4.3.2.1 Packet-based Classifier

Design. As Bluetooth only transmits on good subchannels, it is possible to infer the status of a subchannel based on its *packet rate*, which indicates how frequently the target transmits on a subchannel. To measure packet rates, the scout hops over all 79 subchannels on the acquired basic channel to monitor the target's packet transmissions. For each subchannel i, BlueEar computes its packet rate as $r_i = \frac{q_i}{v_i}$, where q_i is the number of packets received on i, and v_i is the number of times the scout visits i. When a subchannel is classified as bad by the target, the packet rate measured by the scout will be substantially lower than that of good subchannels.

A key challenge to achieve accurate packet-based classification is that packet rates differ significantly across different applications (e.g., data transfer, audio, and keystroking, etc.), and may vary with time depending on the traffic dynamics in the target piconet. To address this challenge, we leverage the fact that, as required by FCC, Bluetooth Classic uses at least 20 subchannels for frequency hopping [47]. As a result, the 20 subchannels that have the highest packet rates can be used as a reference to estimate the current packet rate of the target piconet. Driven by this observation, the packet-based classifier identifies bad subchannels using the following algorithm.

- *Step 1*: Initially, the 20 subchannels that have the highest packet rates are classified as good. Let $\mathcal{R}_g = \{r_{i_1}, ..., r_{i_{20}}\}$ be the set of their packet rates.
- *Step 2*: In remaining unclassified subchannels, the classifier searches for the one with the highest packet rate. Denote this subchannel as i, and its packet rate as r_i.



(a) Data. (b) Audio.

Figure 4.5: Running examples of packet-based subchannel classification for data and voice traffic under in different spectrum contexts. The packet-based classifier calculates a probability score based on Eq. (4.1) to determine subchannel status. A subchannel is classified as bad if the probability score is below the pre-defined threshold.

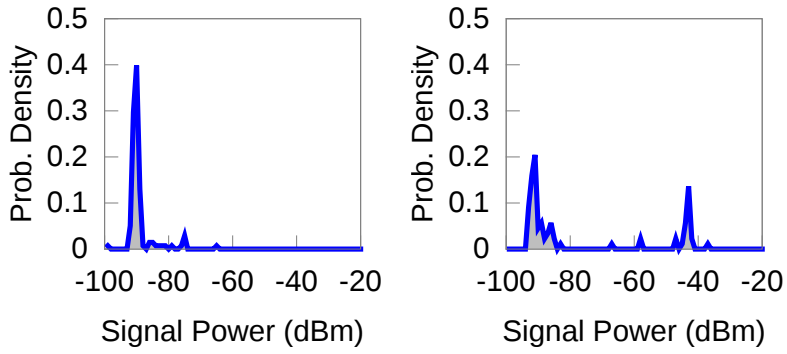
- *Step 3*: The packet-based classifier determines whether subchannel i is bad by checking if r_i is an outlier of $\mathcal{R}_g$. If r_i is an outlier, i and all remaining subchannels of even lower packet rates are classified as bad. Otherwise, i is classified as good and its packet rate r_i is inserted to $\mathcal{R}_g$. The algorithm then goes back to step 2 until a bad subchannel is identified.

We determine if $r_i = \frac{q_i}{n_i}$ is an outlier of $\mathcal{R}_g$ as follows. Let r_g be the average packet rate of $\mathcal{R}_g$. Assuming subchannel i is good, probability that the target transmits less than q_i packets after v_i visits can be computed as,

$$\rho_i = \sum_{n=0}^{q_i} \binom{v_i}{n} (1 - r_g)^{v_i - n} r_g^n \tag{4.1}$$

We identify outliers under a given confidence level, denoted as θ. Subchannel i is classified as bad if $\rho_i \leq 1 - \theta$.

Fig. 4.5 gives two examples of packet-based subchannel classification for data and audio traffic in different spectrum contexts. The upper figure shows the packet rates measured on 79 subchannels. Low packet rates are observed on bad subchannels subjected



(a) Clean subchannel. (b) Dirty subchannel.

Figure 4.6: Probability densities of interference power measured on clean and dirty subchannels.

to interference. The bottom figure shows the probability scores ρ_i for each subchannel i calculated using Eq. (4.1). As shown in the figure, the packet-based classifier reliably identifies bad subchannels despite the significant variation of packet rates across different applications.

Discussion. By classifying subchannels based on packet rates, packet-based classifier offers two advantages: *(i)* it works efficiently across different Bluetooth devices despite vendor-dependent subchannel classification methods, *(ii)* the classification is not affected by the disparity between the spectrum contexts of the target and BlueEar. However, packet-based classifier is less responsive in dynamic spectrum context because a subchannel cannot be classified as good or bad before overhearing a sufficient number of packets. As a result, packet-based classifier may perform poorly when subchannel status changes fast with time-varying interference.

4.3.2.2 Spectrum Sensing-based Classifier

Design. Since Bluetooth piconet classifies subchannel based on interference conditions, subchannel i is likely bad if strong interference is measured on i. Driven by this observation, spectrum sensing-based classifier infers subchannel status based on interference

measurements. When hopping on the basic channel, the scout measures interference power on each subchannel. The interference condition on a given subchannel is characterized using the probability density of interference power. Fig. 4.6 illustrates two examples measured by the scout on clean and dirty subchannels. On both subchannels, the environment noise floor is found at -95 dBm. An interference source whose signal power ranges from -45 dBm and -40 dBm can be observed in Fig. 4.6b. The interference source is active in about 15% of time.

Based on interference measurement, the spectrum sensing based classifier employs an SVM to determine subchannel status. The SVM takes as input a feature vector obtained by discretizing the probability density of interference power based on $\mathcal{X}_i = \{x_{-100,i}, x_{-99,i}..., x_{-20,i}\}$, where $x_{s,i}$ is the probability of observing an interfering signal power of s dBm on subchannel i. $\mathcal{X}_i$ characterizes interference condition between -100 dBm and -20 dBm, which is sufficient to capture the activities of all interfering sources in practice.

Discussion. Although spectrum sensing-based classifier is responsive to time-varying interference conditions, achieving satisfactory accuracy is difficult because *(i)* depending on the location of interference source, the spectrum measured by the scout may differ from the one observed by the target; *(ii)* the subchannel classification method adopted by the target is vendor-dependent and may differ across different devices. To address these limitations, the spectrum sensing-based classifier must be trained at run-time.

4.3.2.3 Hybrid Classifier

For accurate and responsive classification of subchannel status, BlueEar employs a hybrid classifier that combines the complementary packet- and spectrum sensing-based classifiers. At run-time, the hybrid classifier uses packet-based classification results to train a

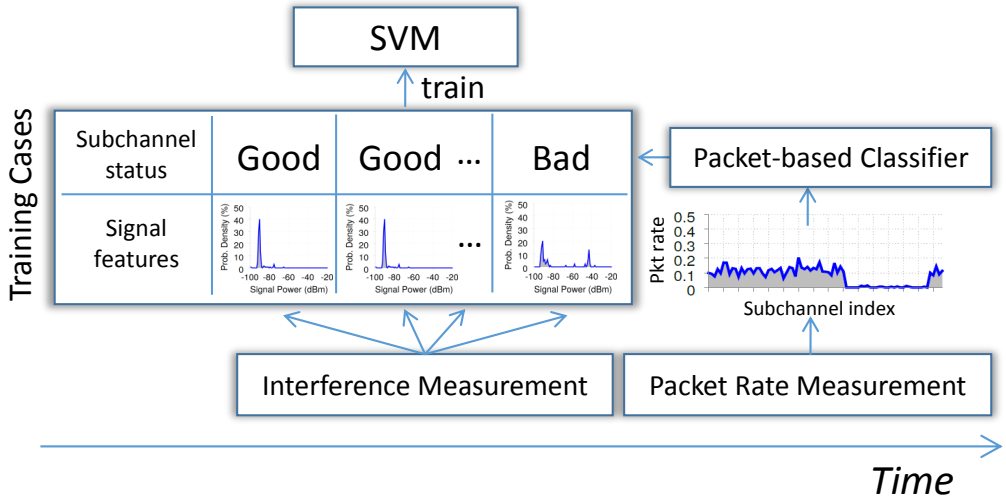


Figure 4.7: Time-domain illustration of run-time training of the spectrum sensing-based classifier.

spectrum sensing-based classifier, which learns the vendor-dependent subchannel classification method of the target. After training, BlueEar fuses the outputs of packet- and spectrum sensing-based classifiers to infer subchannel status.

To train the spectrum sensing-based classifier, BlueEar labels interference conditions measured by the scout using the outputs of packet-based classification. Note that packet-based classifier infers subchannel status based on packet rates derived from the history of overheard packets, therefore its result may lag behind the true subchannel status. To compensate this delay, BlueEar composes training cases by labeling $\mathcal{X}_f$ using packet-based classification results obtained at a later time point. Fig. 4.7 illustrates this training procedure in time-domain.

For subchannel classification, the hybrid classifier fuses the outputs of packet- and spectrum sensing-based classifier based on the responsiveness and confidence of results. The soft-output of SVM is utilized to characterize the confidence of classification. The soft-output of SVM is a log-likelihood ratio computed as $\lambda_i = \log \frac{\rho_i}{1-\rho_i}$, where ρ_i is the probability that i is good, and $|\lambda_i|$ indicates confidence. Because spectrum sensing-based

classifier is more responsive to dynamic spectrum context, the hybrid classifier adopts the output of SVM as the final classification result if its confidence $|\lambda_i|$ is higher than a predefined threshold. Otherwise, the output of packet-based classifier is adopted.

4.3.3 Selective Jamming

In the crowded 2.4 GHz frequency band, BlueEar is subjected to the interference of coexisting wireless devices, especially the prevalent 802.11 based WLANs. Unlike authorized Bluetooth devices that can handle such interference by coordinating their hopping, designed as a passive packet sniffer, BlueEar cannot coordinate with the target, which may result in poor sniffing performance. BlueEar mitigates the impacts of such interference using a selective jamming algorithm. In the following, we present the algorithm design in detail, and then discuss the impact of jamming on 802.11 devices.

When the interference causes substantial packet corruptions on a subchannel i, the scout deliberately generates interference on i while the target visits i. Because of adaptive hopping, the target will be driven away from subchannels i, resulting in implicit coordination. To this end, BlueEar employs a loss detector to identify subchannels subjected to hidden interference. Whenever the scout overhears a packet, it checks packet integrity using CRC, and then sends the result to the loss detector. For each subchannel, the loss detector employs a moving window to compute the ratio of corrupted packets. The scout is commanded to jam a subchannel if the packet corruption ratio is higher than a predefined threshold. To effectively drive the target, a class one Bluetooth radio capable of high power transmission is employed to implement the scout.

Discussion. Despite deliberately generating interference in the 2.4 GHz band, the impacts of selective jamming on 802.11 devices is very limited because of two reasons. First, there

are considerably low chances for collisions between 802.11-based frames and selective jamming frames. As explained earlier, selective jamming is equivalent to Bluetooth class 1 transmission, and hence, probability of collision with 802.11-based WLANs is equivalent to that of Bluetooth. A previous study [48] highlights that in worst case scenario, where the three 802.11 non-overlapped channels are consistently occupied, the probability of collision with Bluetooth is less than 0.2. However, this probability significantly reduces if we consider the following factors: *(i)* the scenario of occupying the three 802.11 non-overlapped channels is rare; *(ii)* the study [48] considers a low data rate (5.5Mbps) for 802.11 link. With adoption of OFDM higher data rates, frame-in-air-time becomes shorter than that of 5.5Mbps rate. Accordingly, abundant white space between 802.11 transmissions is expected. A recent study [20] confirms this and shows that 802.11 traffic is highly bursty and frames are clustered together with short intervals of time (typically less than $1ms$), while periods between clusters are significantly longer; *(iii)* life time of a jamming session is 2 sec in average, where our experiments show that Bluetooth reacts to interference within 4 sec at most; and finally *(iv)* BlueEar only jams in the presence of hidden interference, which is a special scenario for BlueEar. Second, assuming the case of collision with 802.11-based frame, a previous study [10] shows that 802.11-based WLANs is robust against narrow band (the scout occupies 1MHz vs. 20MHz for 802.11), short-lived interference. Therefore, we conclude that selective jamming has very limited effect on 802.11-based WLANs.

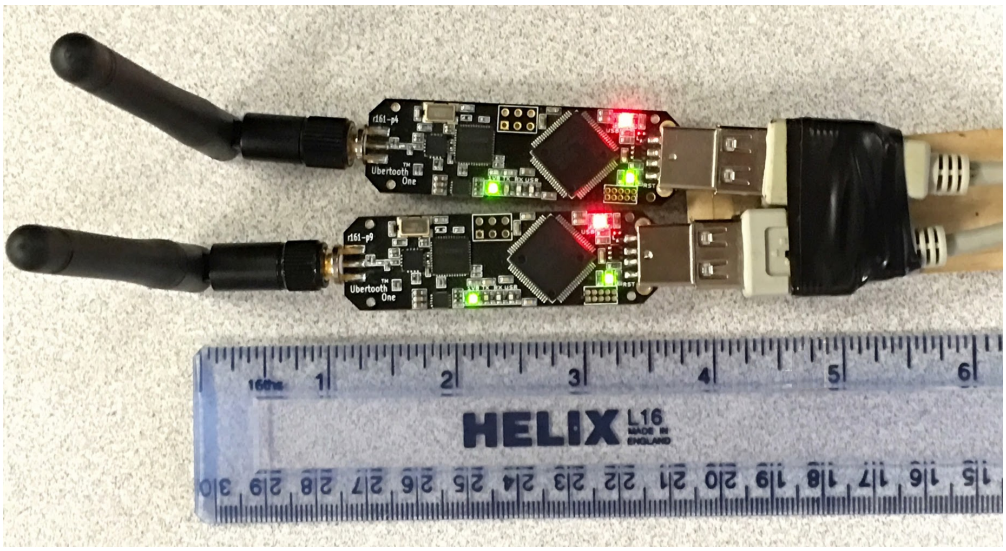


Figure 4.8: BlueEar prototype that consists of two Ubertooths [31].

4.4 Implementation

In this section, we present the implementation of BlueEar in detail. As shown in Fig. 4.8, we employ two Ubertooths [31] to implement the scout and the snooper, and then interface them to a controller running on a Linux laptop. Computation intensive tasks like clock acquisition and subchannel classification are implemented on the laptop. Time-sensitive components like basic and adaptive hop selection are implemented by extending the firmware of Ubertooth. In addition, we identify critical issues in Ubertooth firmware that severely degrade its performance during frequency hopping, and present novel solutions to address these flaws. We note that although our current prototype is built based on Ubertooth, the design of BlueEar is platform-independent and can be easily ported to other systems.

4.4.1 Ubertooth-End Implementation

Ubertooth is an open source 2.4 GHz wireless development board that costs around $80 per unit [31]. Each Ubertooth is equipped with an LPC17xx microcontroller, and a low-power Bluetooth-compliant CC2400 transceiver connected to a 4-inch 2.2 dBi antenna. Ubertooth is capable of transmitting at 22 dBm, which assures the effectiveness of selective jamming.

The original firmware of Ubertooth is implemented in 823 lines of C code, which implements DMA management, basic hop selection, and carrier sense, etc. Data in DMA buffer is framed into USB packets and forwarded to the host. However, the original firmware lacks support for adaptive hop selection and run-time clock synchronization. In addition, we find that the firmware is poorly optimized for real-time frequency hopping. In particular, because of resource contention among multiple tasks, subchannel switching may be improperly delayed (e.g., by USB packet streaming, which typically takes around 50μs according to our measurements). Such delay will break the hop synchronization between BlueEar and the target. We extend the firmware of Ubertooth using 400 lines of C code, which implement the following functions.

(*i*) *Run-time clock synchronization*. To hop following the basic and the adapted channel of the target, the scout and the snooper must synchronize their native clocks with the target's piconet clock, i.e. their clocks must have the same value and tick at the same time. Run-time clock synchronization is imperative because the clocks of the scout, the snooper, and the target may have clock skews [19], which make them run at different rates, accumulating a drift that breaks hop synchronization. The extended firmware accomplishes clock synchronization as follows. After clock

acquisition, the firmware receives a piconet clock value from the clock acquisition component. The clock value is used as the initial value of a native clock, which is obtained by programming a 10MHz timer provided by LPC17xx into a 27-bit counter that ticks every hop. To assure that the native clock and the piconet clock tick at the same time, the extended firmware leverages the fact that Bluetooth packets are always transmitted immediately after clock ticks. Therefore, the receiving times of overheard packets can be utilized as a clock reference to correct clock drift. To avoid packet miss caused by remaining clock drift, the native clocks of the scout and the snooper are programmed to tick 1 μs earlier than the target.

(*ii*) *Adaptive hop selection*. The firmware of the snooper implements a standard-compliant adaptive hop selection kernel. The kernel takes three inputs, including the inferred subchannel map, the piconet address obtained from the controller, and the value of the native clock. The inferred subchannel map is updated every second.

(*iii*) *Task scheduler*. To assure real-time hopping performance, the extended firmware schedules tasks based on their time sensitivities. Hop selection and subchannel switching are given the highest priority to assure the right hop synchronization. USB packet streaming and carrier sense are given the second and lowest priority, respectively. Tasks are executed in the interval between subchannel switching in the order of their priorities.

4.4.2 Controller Implementation

The controller implements compute intensive tasks, including packet decoding, clock acquisition, subchannel classification, and jamming subchannel selection. In addition, it in-

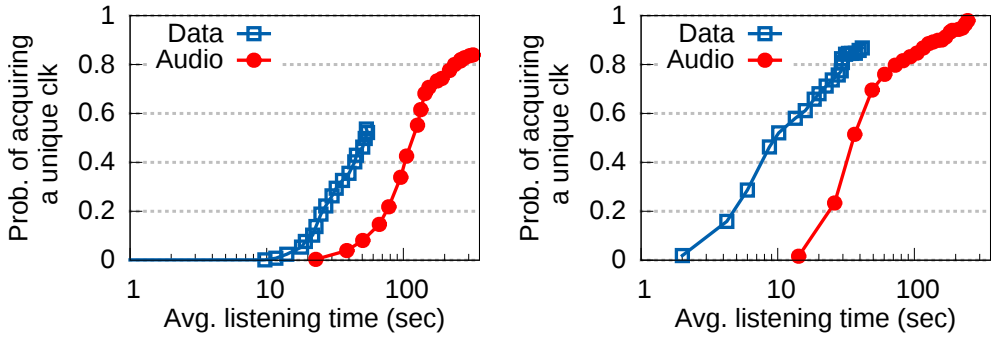
teracts with the scout and the snooper via high-speed USB. These tasks are implemented as multiple processes, which share a 3 KB of memory for coordination and parameter exchange. For packet-based subchannel classification, the controller implements the algorithm described in Section V-B1 in 53 lines of C code. A confidence level of 99% is used to assure accurate identification of bad subchannels. The spectrum sensing-based classifier is implemented based on SVM^{light}, which is an open-source computation-efficient SVM library [25]. The spectrum sensing-based classifier takes about 51.2 KB of memory at run-time. To compensate the delay of packet-based classification when training the SVM (as explained in Section V-B2 and Fig. 4.7), the controller uses packet-based classification results obtained in $t + 4s$ to label the signal features measured at t. This choice is motivated by our empirical measurements, which show that most Bluetooth devices update subchannel map every $4s$. The hybrid classifier chooses the result of SVM as output if the confidence of SVM is higher than 90%. Otherwise the output of packet-based classifier is adopted. The controller is responsible for decoding the raw bit stream received from Ubertooth. Packet integrity is examined by checking the received CRC. A subchannel is jammed if the ratio of corrupted packets is higher than 10%.

4.5 BlueEar Performance

In this section, we present a thorough evaluation of BlueEar performance. In the following, we first introduce our experimental methodology, and then discuss experiment results in detail.

4.5.1 Experimental Methodology

We study BlueEar performance when sniffing data transfer and audio streaming, which are representative Bluetooth traffics that have distinct packet rates. Data traffic is generated by transferring data files between two laptops equipped with Broadcom dongles. Audio traffic is generated by playing an audio file on a laptop equipped with CSR dongle, and a Samsung Bluetooth headset is set as the audio sink. We conduct experiments in an office building under the interference of a large-scale 802.11-based WLANs, as well as in various controlled settings to benchmark BlueEar performance under specific interference patterns. We evaluate the synchronization delay, the subchannel classification accuracy, and the packet capture rate of BlueEar. The synchronization delay is measured as the time needed to determine the correct piconet clock. To measure subchannel classification accuracy and packet capture rate, we log the groundtruth subchannel map and packet rates at the piconet master using a script written based on hcitool. The host of BlueEar is connected with the piconet master via an Ethernet link. The instantaneous readings of groundtruth subchannel map and packet rates are transferred to the BlueEar host using UDP. We compare BlueEar with a set of baselines. First, we compare the hybrid subchannel classifier proposed in Section V-B3 with pure packet-, and spectrum sensing-based classifiers (abbreviated as `Pkt' and `SS' in figures). The SVM of pure spectrum sensing-based classifier is trained offline in a controlled setting consisting of an 802.11 access point (AP) and a Broadcom piconet. During training, we tune the power and temporal pattern of 802.11 transmissions to introduce different interference conditions, which enables extensive profiling of the adaptive hopping behavior of the Broadcom device. We then use the trained classifier to predict the subchannel maps in data and audio tests, where the



(a) No bad subchannels.

(b) 25% bad subchannels.

(c) 50% bad subchannels.

(d) 75% bad subchannels.

Figure 4.9: Clock acquisition delay when sniffing data and audio traffics in different spectrum contexts (characterized by the percentage of bad subchannels at the target piconet).

piconets are formed using different Bluetooth devices. Second, to evaluate the gain of selective jamming, we compare BlueEar with a baseline where the selective jamming is disabled. Third, we compare the packet capture rate of BlueEar with that of an existing Ubertooth-based sniffer [31], which operates in the basic hopping mode, and is oblivious to the adaptive hopping behavior and the impacts of interference.

4.5.2 Synchronization Delay

We first evaluate the delay incurred when synchronizing BlueEar with the target piconet. The dominant component of this delay is introduced by clock acquisition, during which the scout listens on a single subchannel until it captures enough packets to reverse the piconet clock. In the following, we evaluate clock acquisition delay, first based on the

PCM algorithm, and then based on ML algorithm.

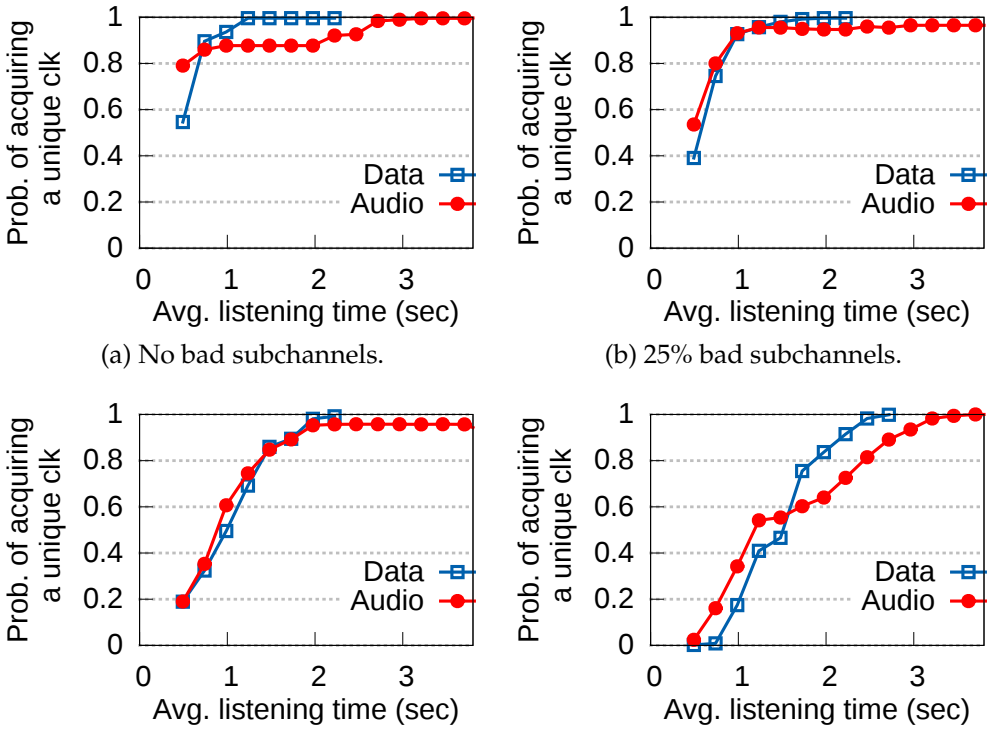
4.5.2.1 Delay based on PCM algorithm

We benchmark clock acquisition delay in different spectrum contexts where the target exhibits diverse hopping behaviors. Our experiments are conducted in a controlled setting where three 802.11 access points (APs) are deployed around the target. Each AP occupies one of the three non-overlapping 20 MHz channels. When all APs are active, they create a crowded spectrum where about 75% subchannels of the target piconet are bad.

Fig. 4.9 shows the probability of successfully determining the piconet clock, based on the PCM algorithm, as the listening time of the scout increases. We observe that clock acquisition delay when sniffing audio traffic is higher than that when sniffing data traffic. This is mainly because of the lower packet rate of audio traffic. Interestingly, the delay substantially reduces when the spectrum becomes more crowded. This is because, when more subchannels are occupied by 802.11 APs, the target piconet has to use fewer subchannels for packet transmissions, resulting in an increased packet rate on the subchannel monitored by the scout. Specifically, when 75% of subchannels are occupied by 802.11 APs, the clock acquisition delay is less than 10s in both data and audio tests. The result implies that Bluetooth traffic sniffers can substantially reduce its synchronization delay using deliberately planned interference.

4.5.2.2 Delay based on ML algorithm

Now we evaluate the ML algorithm performance. We consider clock acquisition delay as an evaluation metric. We compare the ML algorithm performance with that of the PCM algorithm when acts alone.



(a) No bad subchannels.

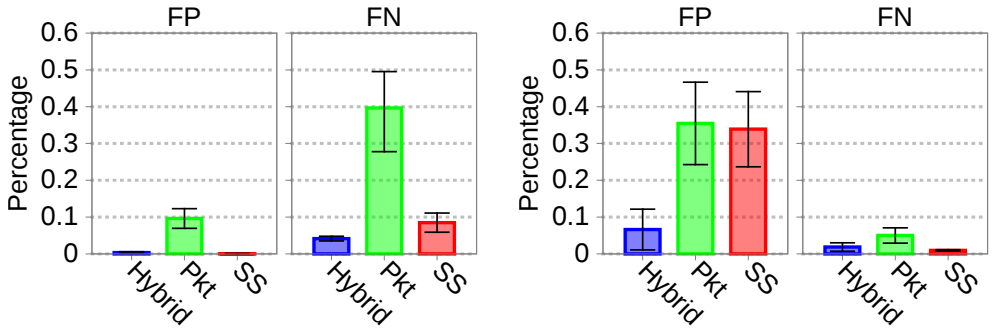
(b) 25% bad subchannels.

(c) 50% bad subchannels.

(d) 75% bad subchannels.

Figure 4.10: Clock acquisition delay based on the ML algorithm, which is evaluated in different spectrum contexts (characterized by the percentage of bad subchannels at the target piconet).

Fig. 4.10 shows the probability of successfully estimating the piconet clock as the listening time increases. To compare, we observe that the ML algorithm outperforms the PCM algorithm, as in Fig. 4.9. The ML algorithm significantly reduces clock acquisition delay to less than 4*s*. Similar to the PCM algorithm, we observe that clock acquisition delay when sniffing on audio traffic is higher than that when sniffing on data traffic. In contrast, the ML algorithm requires more time to estimate the piconet clock when more subchannels are occupied by 802.11 traffic, which in turn maximizes number of remapped packets. Specifically, when a set of observed packets P starts with a remapped packet, there is a low chance that the true clock is presented in the set of candidates C, i.e. all candidates are false. Alternatively, when P starts with a packet that belongs to the basic hopping sequence, the true clock must appear in C. As a result, the ML algorithm requires



(a) Data. (b) Audio.

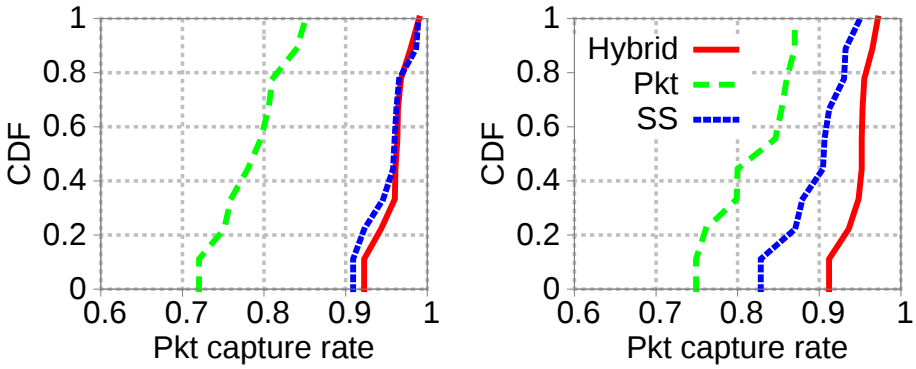
Figure 4.11: Subchannel classification accuracy in fast varying spectrum context.

more packets, i.e. more listening time, if the number of remapped packets is increased. Based on this finding, the design of a countermeasure should consider increasing number of remapped packets in Bluetooth traffic to prevent passive attacker, like BlueEar, from sniffing on Bluetooth link.

4.5.3 Fast Varying Spectrum Context

We now evaluate BlueEar in dynamic spectrum contexts where the subchannel map of the target is modified frequently. The transmission of AP is turned on/off every a couple of seconds to create a fast varying spectrum context, which causes the target to modify its subchannel map every update period.

Fig. 4.11 evaluates subchannel classification accuracy based on false positive (FP) and false negative (FN) rates. As expected, the packet-based classifier performs the worst because of its poor responsiveness. In comparison, the spectrum sensing-based classifier offers better performance when sniffing data traffic, but fails to maintain its accuracy when sniffing audio. This is because the spectrum sensing-based classifier is trained offline against Broadcom devices, and it fails to predict the adaptive hopping of the CSR devices used in the audio test. We also observe that the hybrid classifier performs best



(a) Data. (b) Audio.

Figure 4.12: Packet capture rate in fast varying spectrum context.

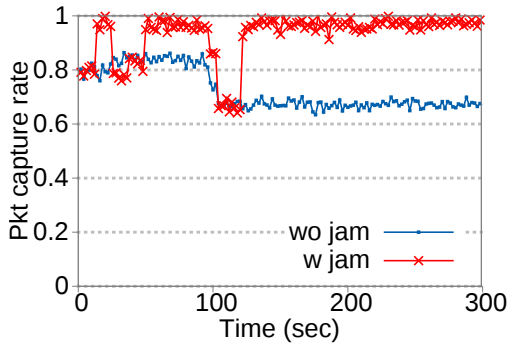
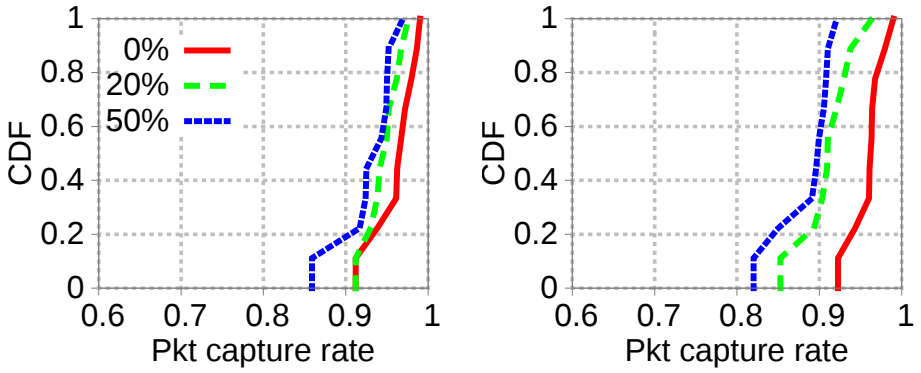


Figure 4.13: The gain of selective jamming under hidden interference.

among the three classifiers. In particular, the FP and FN rates are lower than 8% in both data and audio tests. Fig. 4.12 further compares the packet capture rates when BlueEar uses the three classifiers to predict adaptive hopping. Similar with the results shown in Fig. 4.11, the hybrid classifier is able to maintain the best packet capture rate, which is higher than 90% in both data and audio tests.

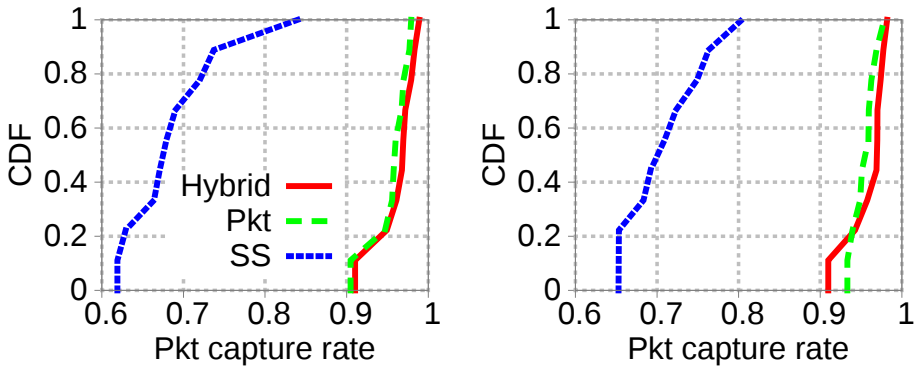
4.5.4 Hidden and Exposed Interference

We now evaluate the packet capture rate of BlueEar in the presence of hidden and exposed interference. Fig. 4.13 evaluates the gain of selective jamming in the presence of hidden interference, where an RF signal does not interfere with the target, but collide with



(a) Data. (b) Audio.

Figure 4.14: Packet capture rates in crowded spectrum characterized by the percentage of bad subchannels at the target piconet.

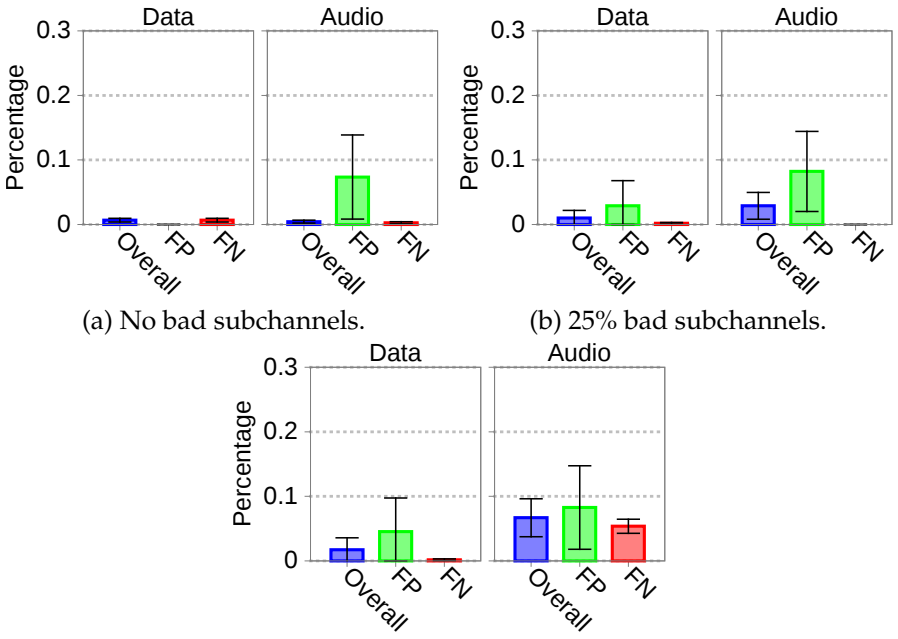


(a) Data. (b) Audio.

Figure 4.15: Packet capture rates under exposed interference condition.

captured packets at BlueEar. The experiments are conducted in a controlled setting where an 802.11 device generates hidden interference starting from the 100-th second. When selective jamming is enabled, BlueEar is able to maintain high packet capture rates, despite a short period of performance drop before the target piconet reacts to the generated interference. In comparison, when selective jamming is disabled, BlueEar suffers substantial performance degradation, where the packet capture rate is reduced to about 60% from higher than 95%.

We further evaluate BlueEar in the presence of exposed interference, where an RF signal interferes the target, but is too weak to be measurable at the scout. Exposed in-



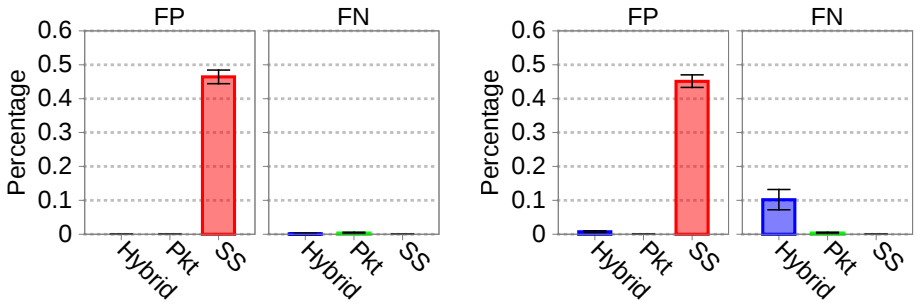
(a) No bad subchannels.

(b) 25% bad subchannels.

(c) 50% bad subchannels.

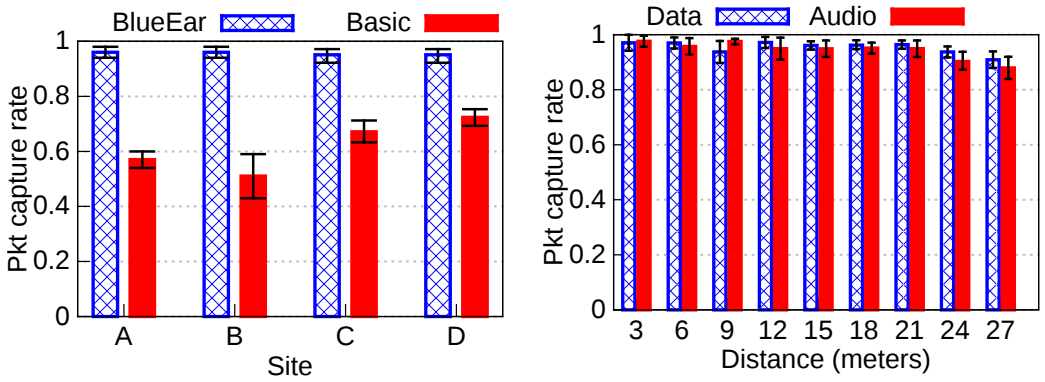
Figure 4.16: Subchannel classification accuracy in crowded spectrum characterized by the percentage of bad subchannels at the target piconet.

terference results in significant disparity between the spectrum contexts at BlueEar and the target. We conduct experiments in a controlled setting where an 802.11 device is deployed to generate exposed interference. During our experiment, the 802.11 device keeps active, and interferes with 20 of 79 subchannels of the target piconet. Fig. 4.14 compares the subchannel classification accuracy of the hybrid, the packet-based, and the spectrum sensing-based classifiers. Different from what we observed in Fig. 4.11, the spectrum sensing-based classifier suffers high FP in both tests. This is because spectrum sensing-based classifier relies on the interference measurements of the scout to identify bad subchannels, which works poorly when the interference signal cannot be detected by the scout. In comparison, hybrid and packet-based classifiers are able to maintain extremely low FP and FN rates and high packet sniffing performance, as shown in Fig. 4.15.



(a) Data. (b) Audio.

Figure 4.17: Subchannel classification accuracy under exposed interference condition.



(a) Different building locations (b) Different distances.

Figure 4.18: Packet capture rates under the ambient interference: (a) different locations in an office building (interference of a large-scale 802.11-based WLANs); (b) different distances.

4.5.5 Crowded Spectrum

We then evaluate the misclassification rate of the hybrid classifier in spectrum contexts with different levels of crowdedness. The FPs, FNs, and overall misclassification rates are shown in Fig. 4.16. We observe that the hybrid classifier maintains high accuracy despite the increasingly crowded spectrum. In particular, when 50% of subchannels are bad, the overall misclassification rate is below 8% in both data and audio tests. Fig. 4.17 shows the packet capture rates measured in the same experiment. As shown in the figure, BlueEar captures more than 90% packets in both data and audio tests.

4.5.6 Ambient Interference

We further evaluates the performance of BlueEar in an office building under the ambient interference from a large-scale 802.11-based WLANs. Fig. 4.18(a) shows the packet capture rates measured at four randomly selected locations, where BlueEar is deployed at 10m away from the target piconet. In all of the four locations, the number of active 802.11 APs is higher than 20 during our experiments. We compare BlueEar with an existing Ubertooth-based sniffer [31] that hops following the basic channel of the target. Because the basic hopping sniffer is oblivious to the adaptive hopping behavior, it suffers 50% to 25% packet misses. In comparison, BlueEar is able to maintain a packet capture rate higher than 95% at all of the four locations.

Fig. 4.18(b) evaluates the packet capture rate at site D when BlueEar is deployed at different distances from the target. The disparity in spectrum contexts is expected to increase as BlueEar moves away from the target. However, thanks to the high-performance subchannel classifier, BlueEar is able to capture more than 85% packets even when it is 27m away from the target.

4.6 Implications of BlueEar

In this section, we discuss the privacy implications of BlueEar in detail.

4.6.1 Implications on Bluetooth LE Privacy

Although the current prototype of BlueEar is developed for sniffing classic Bluetooth, its methodology has significant implications on the privacy of Bluetooth LE. In the following, we first briefly introduce the hopping protocol of Bluetooth LE, and elaborate on the

impacts of BlueEar on Bluetooth LE privacy breach. The hopping protocol of Bluetooth LE defines a physical channel, that hops over 37 data subchannels in the open 2.4 GHz spectrum starting from 2.402 to 2.48 GHz. All subchannels are equally spaced with 2 MHz of bandwidth. The connection state of Bluetooth LE can be characterized as a set of connection events. During the initialization of a connection, the master defines *(i)* connection interval –a multiple of $1.25ms$ ranging from $7.5ms$ to $4.0s$ that defines the event lifetime; *(ii)* transmission window size –a multiple of $1.25ms$ that defines the size of transmit window, i.e. packet size; and *(iii)* hop increment $\mathtt{inc}$ –a random value ranges from 5 to 16. The basic channel hopping is characterized by $\mathcal{K}(\mathtt{c},\mathtt{inc})$, where $\mathcal{K}(.)$ is the hop selection kernel, and $\mathtt{c}$ is the index of current subchannel. At the first connection event, the first subchannel is defined to be zero [47], the channel sequence repeats itself whenever subchannel zero is visited. Similar with Bluetooth classic, Bluetooth LE adopts adaptive hopping mode, where the basic channel is modified to adapt spectrum use in the presence of ambient interference. The adaptive channel is defined by a subchannel map that classifies the data subchannels into good and bad. If the basic kernel $\mathcal{K}(\mathtt{c},\mathtt{inc})$ selects a bad subchannel, a remapping procedure is invoked to calculate a *remapped subchannel index*. The master maintains the subchannel map and it notifies slave(s) about any updates [47].

The hopping protocol of Bluetooth LE is different from that of Bluetooth Classic in two aspects. First, basic channel sequence of Bluetooth LE is characterized by a random value of the hop increment $\mathtt{inc}$. In contrast, basic channel sequence of Bluetooth Classic is characterized by the piconet address. The second difference between Bluetooth LE and Bluetooth Classic is hopping phase, which is not defined in Bluetooth LE hopping protocol. Unlike the basic channel sequence of Bluetooth Classic, which repeats itself every

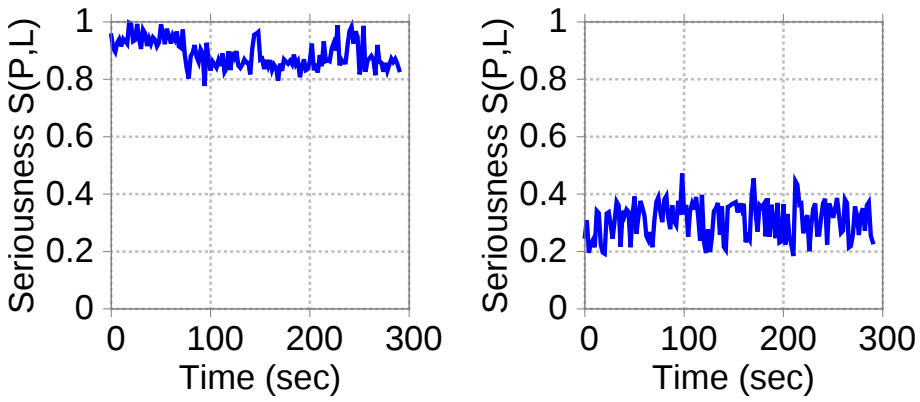
about 23 hours, Bluetooth LE basic sequence repeats itself whenever subchannel zero is visited. Due to power constraints, the hopping protocol of Bluetooth LE is much simpler than that of Bluetooth Classic, making Bluetooth LE basic sequence easier to compromise.

As Bluetooth LE becomes pervasive, the privacy leakage of Bluetooth LE devices is becoming an increasing concern. Although BlueEar is designed for Bluetooth Classic, it has significant impacts on the privacy leakage of Bluetooth LE devices, which calls for further research to further investigate and enhance the privacy of Bluetooth LE. In particular, the key components of BlueEar system, including subchannel classification and selective jamming, are independent of the hopping protocol. These techniques can be directly ported to Bluetooth LE as well as other adaptive hopping systems without modifications. Unfortunately, the clock acquisition component, the hop selection subsystem, and the packet decoder of our prototype are specifically engineered for Bluetooth classic, which make the current version of BlueEar incompatible with Bluetooth LE.

4.6.2 Impacts on Privacy Breach

Previous research has shown the possibilities of cracking Bluetooth encryption and compromising user privacy [55] [56] [13] [38] [37] [40] [11]. A prerequisite of these attacks is to passively sniff Bluetooth traffic. Existing attacks [38] [13] employ prohibitively expensive commodity sniffers, which limits their widespread distribution. The BlueEar system we demonstrated in this paper may unleash such attacks, making them a real issue for off-the-shelf Bluetooth devices.

To further understand the impacts of BlueEar on privacy leakage, we conduct the following two experiments.



(a) Standard. (b) Countermeasure.

Figure 4.19: Seriousness of privacy leakage evaluated: (a) standard; and (b) countermeasure.

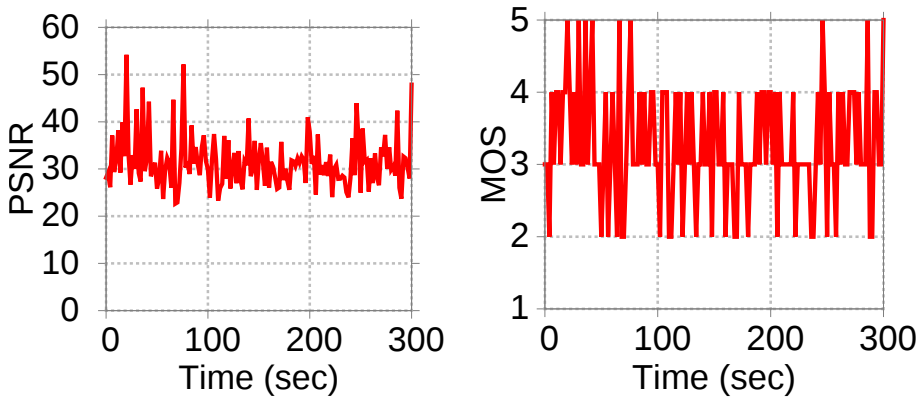
4.6.2.1 Eavesdropping on Data Traffic

We study the impacts of BlueEar when successfully eavesdropping on a Bluetooth data link. To quantify such privacy leakage, we adopt the *seriousness of privacy leakage* model. The model, which is proposed by [9], states,

$$S(P, L) = \Sigma w_i.p_i.l_i \tag{4.2}$$

where $p_i \in P = \{p_0, p_1, .., n\}$ is a *privacy unit*, w_i is the weight of p_i, and $l_i \in L = \{0, 1\}$, $l_i = 1$ when p_i is leaked, and 0 otherwise . As data packet carries potential sensitive information, rather than other packets, we define the components of privacy unit based on data packets and assign their weights as: data packet (weight=95%), and other types of packets (weight=5%).

Fig. 4.19(a) quantifies seriousness of privacy leakage, where S is calculated every 2 seconds based on number of packets captured by BlueEar. As shown in the figure, the seriousness of privacy based successful sniffing on Bluetooth data traffic is very high, where the average is about 90%.



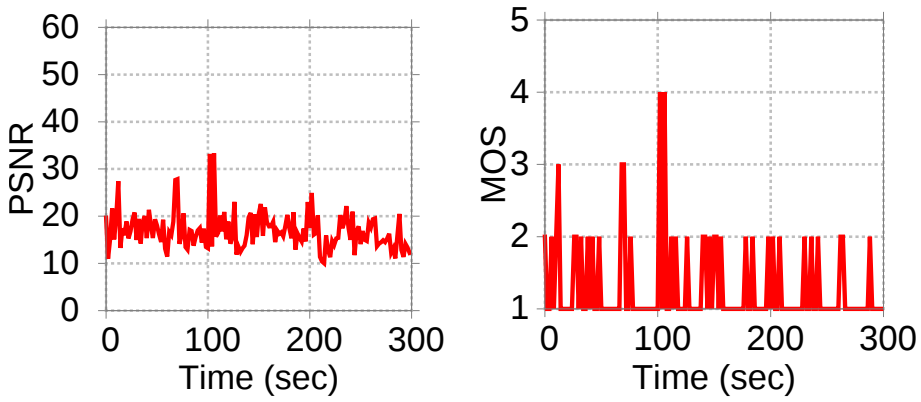
(a) PSNR (b) MOS

Figure 4.20: Audio quality observed by BlueEar (without countermeasure).

4.6.2.2 Eavesdropping on Audio Traffic

We study impacts of BlueEar when eavesdropping on audio traffic, like eavesdropping on a speech conversation, which is known to be challenging because audio streams are extremely sensitive to packet loss. The experiment proceeds as follows. *(i)* We generate audio traffic over a Bluetooth link and deployed BlueEar to sniff on traffic; *(ii)* as BlueEar collects real-time trace, it reports packet loss rates every 2 seconds and logs locations of missing packets; and *(iii)* we simulate the audio packet stream on a PC and replayed each missing packets with it's preceding one.

We quantify the quality of the simulated audio stream, which should be equivalent to the quality of the eavesdropped audio, based on peak signal to noise ratio (PSNR) and mean opinion scores (MOS). To obtain the later, we map PSNR values into MOS, which is categorized as five voice qualities including, excellent, good, fair, poor, and bad as proposed in [32]. Fig. 4.20(a) evaluates audio quality based on PSNR, that is calculated every 2 seconds, and MOS. We observe that BlueEar maintains high audio quality, where average PSNR is 35 and MOS scores are higher than fair in 81% of the time.



(a) PSNR (b) MOS

Figure 4.21: Audio quality as the target implements countermeasure.

4.6.3 Practical Countermeasure

To counteract sniffing systems, like BlueEar, we propose a practical countermeasure approach. We implement the proposed countermeasure as a user-space script on Bluetooth master and it requires no modifications to existing slaves. The key idea of the approach is to frequently flip status of randomly selected subchannels (good becomes bad, and vice versa). Such random flipping interferes the subchannel classifier, making it hard for the sniffer to learn the adaptive hopping sequence. Thus, the sniffer experiences poor data quality. To evaluate the effectiveness of the countermeasure, we run the following experiment. The countermeasure randomly selects $n - 20$ subchannels to be flipped, where n is the total number of good subchannels; this complies with the FCC rule, that requires at least 20 subchannels to be used by Bluetooth. We wrote a user-space script that updates the subchannel map is every $200ms$. The script interacts with BlueZ –the open source Bluetooth stack. We deploy BlueEar to eavesdrop on a data and audio traffic as in section 4.6.2.

Fig. 4.22 shows significant drop in packet capture rates due to the countermeasure. Fig. 4.21(a) evaluates PSNR, where the average drops to 15. Further MOS scores drop to

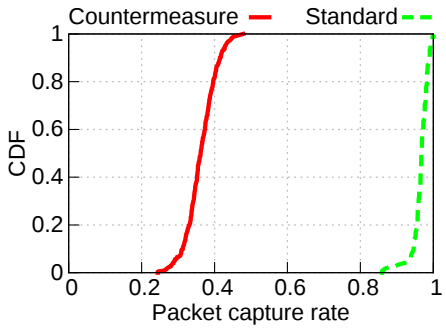


Figure 4.22: BlueEar pkt capture rates: standard and countermeasure.

poor in 95% of the time, as shown in Fig. 4.21(b). This confirms that the sniffer experience poor audio quality due to high packet loss rate, which is mainly caused by the countermeasure. Similarly, Fig. 4.19(b) evaluates seriousness of eavesdropped data packets, where average seriousness drops to 40% in 95% of the time.

4.7 Summary of Chapter

This chapter presents BlueEar, the first Bluetooth packet sniffer that only uses off-the-shelf, Bluetooth-compliant radios. BlueEar features a dual-radio architecture, where two radios are coordinated by a suite of novel algorithms to eavesdrop on an indiscoverable Bluetooth device, relieving the need of expensive specialized radios adopted by commodity sniffers. Extensive experiments show that BlueEar can maintain a high packet capture rate higher than 90% in dynamic settings. We discuss the privacy implications of BlueEar, and propose a practical countermeasure that can reduce the packet capture rate of the sniffer to 20%.

Chapter 5

BlueFunnel: Enabling Low-Power, Wideband Sniffing of Bluetooth Traffic

5.1 Introduction

Existing Bluetooth traffic sniffers can be divided into two categories, which suffer different technical limitations. Traditional wideband sniffers monitor all Bluetooth subchannels in parallel by sampling the entire 2.4 GHz band using a high-speed ADC, which is extremely power-hungry and expensive. Low-power sniffers use cheap, Bluetooth-compliant radios to hop following the target, which however requires knowing the pseudo-random hopping sequence. The BlueEar system we introduced in chapter 4 can passively crack the hopping sequence, but relies on a long training process that requires seconds of traffic monitoring. Unfortunately, except for a few applications like audio streaming, Bluetooth traffics are typically short-lived in nature. For example, during mobile payments, low-duty cycled sensing, device pairing and key establishment, one traffic session contains only a couple to tens of packets lasting at most tens of milliseconds, which effectively prevents a sniffer from learning the hopping sequence.

In this chapter, we present BlueFunnel – a low-power wideband sniffer that addresses the limitations of prior systems. Unlike traditional wideband sniffers, BlueFunnel enables

parallel monitoring of Bluetooth subchannels without using a high-speed ADC. Unlike prior low-power sniffers, BlueFunnel can decode Bluetooth packets without knowing the hopping sequence of the target. The technical approach of BlueFunnel is to first *subsample* the wideband spectrum using a low-speed ADC, and then decode Bluetooth signals using novel signal processing algorithms. By achieving this, BlueFunnel is of great importance to understanding and diagnosing Bluetooth networks and applications in the wild. In addition, by enabling practical sniffing of the pairing and key establishment process, BlueFunnel has important implications to the security and privacy of a wide range of Bluetooth applications.

We implement a prototype of BlueFunnel based on USRP platform [41]. Because commodity USRPs do not support sub-sampling, we emulate sub-sampling by first using the high-speed ADC of USRP to sample the 2.4 GHz band at full speed, and then down-sample the signals at a low rate before decoding using BlueFunnel. The signal processing of BlueFunnel is implemented as a software deployed on a host of the USRP.

We evaluate the system performance based on packet capture rates in a variety of settings. Our goal is to understand the performance of BlueFunnel in real wireless environments in the presence of ambient interference on the system performance, which may pollute the sub-sampled signal. Experiment results show that BlueFunnel can maintain good levels of packet capture rates in all settings.

We further explore the security and privacy implications of BlueFunnel by studying two eavesdropping-based attacks. In the first attack, we deploy BlueFunnel to sniff on the pairing phase of a Bluetooth Low Energy link, and shows that BlueFunnel successfully reveals EDIV, SKDm, and IVm that are used to establish the Short Term key (STK). In the second attack, we utilize BlueFunnel to sniff on Bluetooth mouse traffic. Our experiment

shows that BlueFunnel can reveal the mouse coordinates with high accuracy.

The rest of this chapter is organized as follows. Section 5.2 provides an overview for BlueFunnel sniffer, and system architecture. In section 5.3, we introduce the design details of BlueFunnel system. We present implementation details of a prototype of BlueFunnel in section 5.4. Section 5.5 evaluates the system performance. In section 5.6, we discuss possible attack scenarios based on BlueFunnel packet sniffer. Section 5.7 concludes the chapter.

5.2 BlueFunnel Design

5.2.1 Objectives and Challenges

We introduce *BlueFunnel* –a low-power, wideband Bluetooth traffic sniffer that leverages off-the-shelf devices to capture Bluetooth packets in the wild. Designed as a *passive* traffic sniffer, BlueFunnel employs a simple RF circuit along with a suit of novel digital signal processing algorithms to demodulate Bluetooth signal without the need of pairing or explicit coordination with the target Bluetooth network. To achieve these goals, we tackle the following challenges.

(*i*) *Wideband spectrum sampling.* Bluetooth operates in the free 2.4 GHz band and defines several subchannels that spread over 80 MHz of the spectrum. Therefore, a sniffer would need a wideband radio to monitor the entire spectrum and capture all Bluetooth packets in realtime. However, this wideband radio solution is challenging because it requires a high-speed (about twice the spectrum, according to Nyquist-Shannon rate) analog-to-digital-converters (ADCs) to capture Bluetooth

signal. These ADCs are costly, power hungry, and require high computational power system to support such high sample rate.

(*ii*) *Pseudo-random frequency hopping.* As Bluetooth occupies one subchannel, that is 1-2 MHz, at a time for data exchange, a sniffer could employ a low-power, off-the-shelf Bluetooth-compliant radio to capture data packets. At the baseband, however, Bluetooth adopts frequency hopping spread spectrum, which is a challenge for this kind of sniffers. That is, frequency hopping sequence is known to legitimate Bluetooth devices but not other parties. Therefore, a sniffer can listen on an arbitrary subchannel, and capture packets that are randomly transmitted on that subchannel. However, the number of packets captured in this way represents a tiny portion of the traffic. Moreover, Bluetooth sessions may last for a very short period of time and involve only tens of packets.

(*iii*) *Interference in the crowded 2.4 GHz band.* When coexisting with other wireless devices in the crawdad 2.4 GHz band, Bluetooth may experience various interference conditions, including hidden and exposed interference, from 802.11-based WLANs and other ambient radios. Therefore, Bluetooth adopts adaptive frequency scheme to adapt spectrum utilization and improve performance. To tackle these challenges, a sniffier needs to learn adaptive frequency hopping behavior, and provide some mechanism to null the interference in the case of collision with Bluetooth packets in realtime.

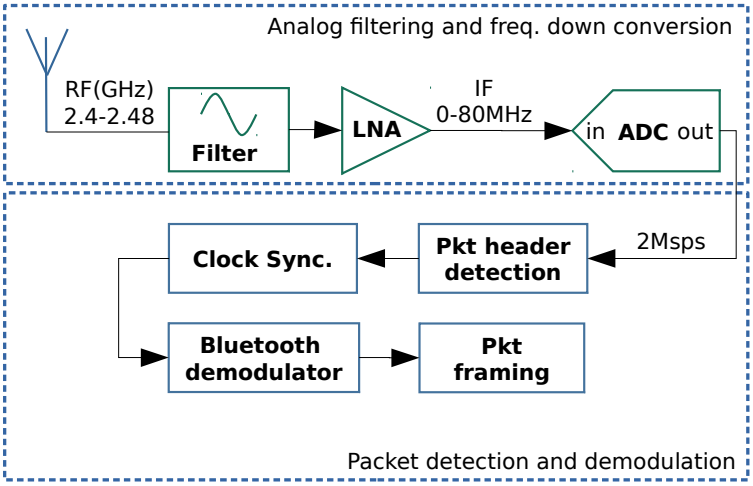


Figure 5.1: BlueFunnel system architecture.

5.2.2 System architecture

To tackle the above challenges, BlueFunnel leverages a low-power, wideband, customized software defined radio (SDR) platform to subsample Bluetooth spectrum in realtime. The SDR is interfaced to a controller which implements novel signal processing algorithms to detect, demodulate, and analyze Bluetooth packets in realtime. Fig. 5.1 illustrates the architecture of BlueFunnel. In particular, the working flow of BlueFunnel system can be divided into the following steps.

5.2.2.1 Analog filtering and down conversion

The SDR platform incorporates two main components, namely a wideband RF-Front-end circuit, and a low speed ADC. The RF-front-end circuit down converts a received 2.4 GHz signal to an intermediate frequency (IF) that ranges from 0-80 MHz, as depicted in Fig. 5.2. Further, it filters out undesired frequency components that are out of the 2.4 GHz ranges. The ADC subsamples the received signal based on 2 Msamples/sec, which is less than Nyquist-Shannon rate (i.e. 2×80 MHz). The resulting complex samples are then

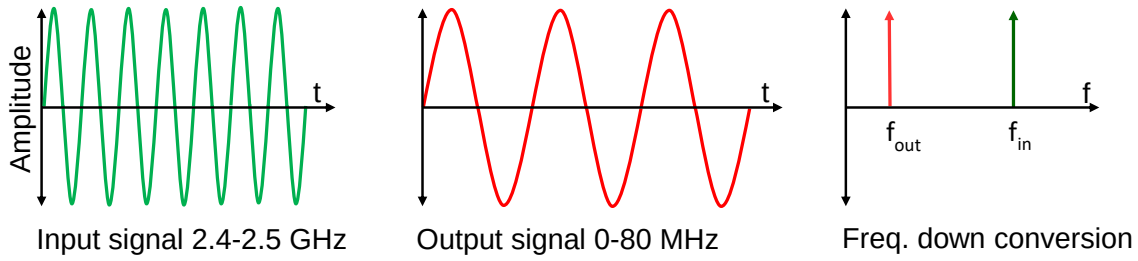


Figure 5.2: Frequency down conversion of Bluetooth signal.

forwarded to the host computer over a serial bus or Gigabit Ethernet connection.

5.2.2.2 Packet detection and demodulation.

On the host computer, the controller processes a received digital signal based on the following stages.

(*i*) *Packet detection.* To reduce complexity of system computations and power consumption, BlueFunnel implements a packet detector, that notifies the system only when a new packet is arrived. The detector is based on measuring the amplitude of received digital signals to indicate packets transmissions. In particular, BlueFunnel measures strong change in the amplitude of received signal. This is because a sharp change in the signal amplitude is highly correlated with packets transmissions. Fig.5.3 shows a realtime received signal.

(*ii*) *Clock synchronization.* Designed as a passive traffic sniffer, BlueFunnel cannot coordinate with the transmitter for clock synchronization at run time. Therefore, BlueFunnel implements a clock synchronization package in order to keep synchronized with Bluetooth signal. The task of this package is to recover samples from a received signal with the same frequency and phase as those used by the transmitter. This is essential task when BlueFunnel wants to extract symbols from an asynchronous

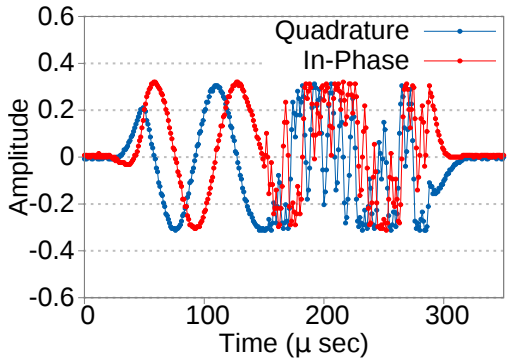


Figure 5.3: Bluetooth real-time complex signal.

digital signal. It allows BlueFunnel to synchronize sampling times with centers of ones and zeros present in the signal.

(*iii*) *Bluetooth demodulation.* As BlueFunnel subsamples Bluetooth spectrum based on 2 Msamples/sec ADC, which is less than the Nyquist-Shannon rate, a regular Bluetooth demodulator may no be able to demodulate a received signal. Therefore, BlueFunnel implements a novel algorithm to demodulate Bluetooth signal under these circumstances.

5.3 Design

In this section, we present the design of BlueFunnel sniffer in details.

5.3.1 Bluetooth Demodulator

As Bluetooth adopts the symbol rate of 1 MSymbols/sec at baseband, BlueFunnel leverages only a simple ADC with a sampling rate of 2 Msamples/sec, to demodulate Bluetooth signals without the pain of learning the target's frequency hopping or adoption of

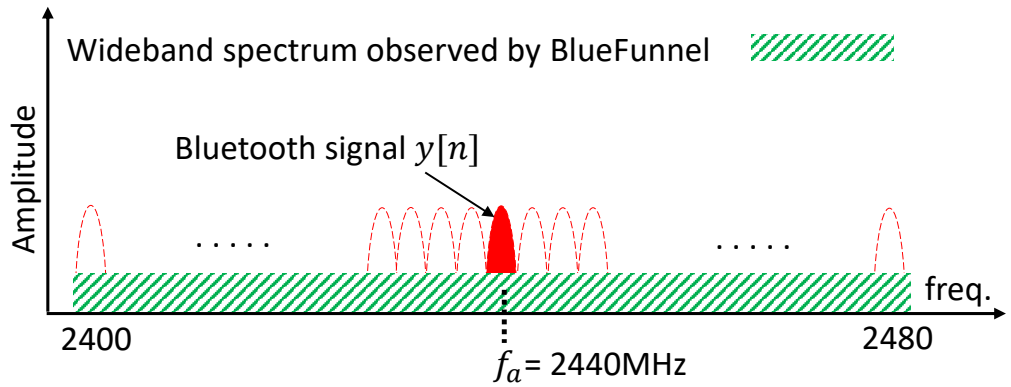


Figure 5.4: An illustrative example where the center frequency of Bluetooth transmitter and BlueFunnel are matched.

a high speed ADCs. This low sampling rate enables a low power, low computation complexity, and inexpensive system. Before diving into the design details of the demodulator, we assume here that Bluetooth signal is received with no interference. In the following, we first discuss demodulating Bluetooth Low Energy (BLE) signals and then consider the case of demodulating Bluetooth Classic signals.

5.3.1.1 Demodulating Bluetooth LE signal

To start with, BlueFunnel listens on a Bluetooth LE subchannel a, where f_a is the center frequency of the subchannel. Therefore, BlueFunnel can directly demodulates a Bluetooth LE signal $y[n]$ that is transmitted on f_a, as illustrated in Fig. 5.4.

Now we assume that $y[n]$ is transmitted on some other frequency f_b, where $a \neq b$. In this case, the received $y[n]$ experiences a frequency shift. Fig. 5.5 shows an illustrative example of Bluetooth LE signal that experiences a spectrum shift. The frequency shift can be expressed as $\delta_2 = f_a - f_b$, where we use δ_2 to denote a shift in the Bluetooth LE spectrum. To understand the effect of this frequency shift on the received time-domain signal $y[n]$, we leverage the frequency-shift property of Fourier transform, which is a

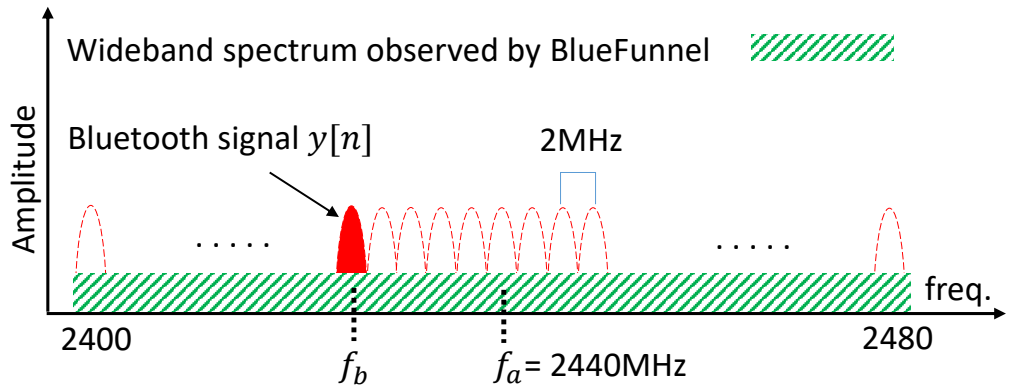


Figure 5.5: An illustrative example where the center frequency of Bluetooth LE transmitter and BlueFunnel are not matched.

fundamental mathematical tool relating frequency and time domains. The property states that a shift δ_2 in the spectrum of a signal $y[n]$ corresponds to multiplication of the signal $y[n]$ by a factor $e^{j\frac{2\pi}{N}\delta_2 n}$ [26]. This is mathematically expressed as,

$$\mathcal{F}^{-1}\{X(f-\delta_2)\} = y[n]e^{j\frac{2\pi}{N}\delta_2 n} \tag{5.1}$$

where $N = 2$ Msamples/sec is the sampling rate of $y[n]$, $n = 0, 1, 2, ..., N$ is a sample index, and δ_2 MHz is the amount of shift in the spectrum. To calculate δ_2, we leverage the following observation. Bluetooth specifications [46] states that Bluetooth LE adjacent subchannels are spaced with 2 MHz of spectrum. Specifically, a subchannel frequency is calculated as $f_{BLE} = 2402 + 2k$ MHz, where $k = 0, 1, 2, ..., 39$, as depicted in Fig. 5.5. Based on this observation, we conclude that δ_2 is basically a multiple of $2k$ MHz. That is, $\delta_2 = f_a - f_b = m \times 2k$ MHz, where m is an integer. Hence, we analyze the exponential term in Eq. (5.1) as follows,

$$e^{j\frac{2\pi}{2\times 10^6}(\delta_2 \times 10^6)n} = e^{j\pi\delta_2 n} \tag{5.2}$$

This can be further simplified based on Euler's formula, which relates the complex exponential function and trigonometric functions, as follows,

$$e^{j\pi\delta_2 n} = \cos(\pi\delta_2 n) + j\sin(\pi\delta_2 n) \tag{5.3}$$

Given that $\delta_2 = m \times 2k$ is always an even integer, the imaginary component cancels because $\sin(\pi\delta_2 n) = 0$, and the real component becomes $\cos(\pi\delta_2 n) = (-1)^{\delta_2 n} = 1$, according to the trigonometric functions facts. Therefore, the exponential term $e^{j\frac{2\pi}{N}\delta_2 n}$ is, in fact, a sequence of ones [30], and hence, it has no effect on the received Bluetooth LE signal $y[n]$. The same argument holds for $-\delta_2$ because the cosine is an even function, i.e. $\cos(-\pi\delta_2 n) = \cos(\pi\delta_2 n)$.

To conclude, BlueFunnel can directly demodulate received Bluetooth LE signals regardless of their transmission subchannels. In practice, BlueFunnel listens on the center of Bluetooth LE spectrum, i.e. subchannel 39 or 40, which guarantees that transmitted signals on other subchannels are shifted by $\delta_2 = m \times 2k$ MHz.

5.3.1.2 Demodulating Bluetooth Classic signals

Similar to the case of Bluetooth LE, BlueFunnel initially listens on subchannel a, where f_a is the center frequency of the subchannel, as shown in Fig. 5.4. Therefore, BlueFunnel can directly demodulate Bluetooth Classic signals $y[n]$ that are transmitted on f_a.

However, $y[n]$ could be transmitted on some other subchannel b, where $f_a \neq f_b$, as depicted in Fig. 5.6. In this case, received signal $y[n]$ experiences spectrum shift and demodulating $y[n]$ is challenging because of the following. Unlike the case of Bluetooth LE, Bluetooth Classic signal $y[n]$ could be multiplied by some values rather than a se-

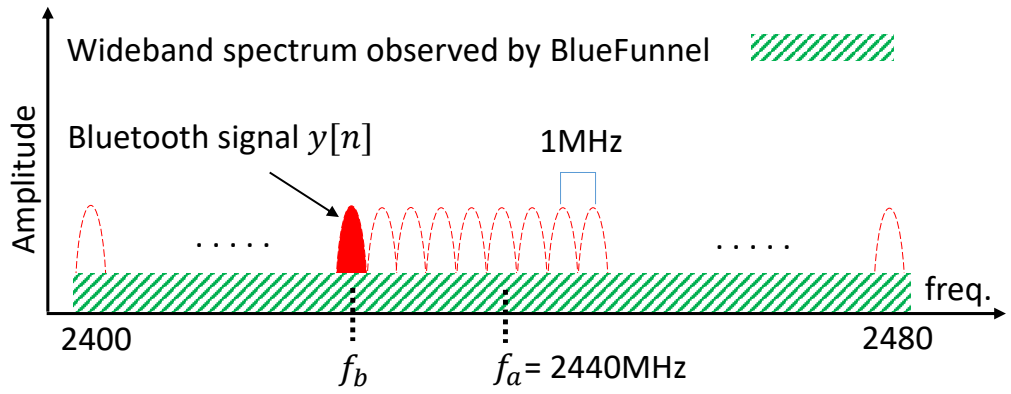


Figure 5.6: An illustrative example where the center frequency of Bluetooth Classic transmitter and BlueFunnel are not matched.

quence of ones. Specifically, Bluetooth Classic subchannels are spaced with 1 MHz of spectrum [46], where a subchannel frequency is calculated as $f_{Classic} = 2402 + k$ MHz, $k = 0, 1, 2, ..., 78$, as depicted in Fig. 5.6. We denote a shift in Bluetooth Classic spectrum as δ_1 to distinguish from the case of Bluetooth LE. Therefore, the spectrum shift can be calculated as $\delta_1 = f_a - f_b = m \times k$ MHz, where m is an integer. We also re-write Euler's formula for Bluetooth Classic as follows,

$$e^{j\pi\delta_1 n} = \cos(\pi\delta_1 n) + j\sin(\pi\delta_1 n) \tag{5.4}$$

Because δ_1 could be an even or odd integer, the product of $\delta_1 \times n$ could be an odd or even integer accordingly. In any way, the imaginary component cancels as $\sin(\pi\delta_1 n) = 0$, and the real component $\cos(\pi\delta_1 n) = (-1)^{\delta_1 n} = \pm 1$. This holds all the time according to the trigonometric functions facts.

Now we examine the two possibilities of the real component. First, when δ_1 is an even integer, $(\delta_1 \times n)$ is always an even integer. This leads to $\cos(\pi\delta_1 n) = 1$, meaning that the exponential term is basically a sequence of ones, and it has no effect on $y[n]$. In this case, BlueFunnel can directly demodulate $y[n]$ just like the case of Bluetooth LE.

Second, when δ_1 is an odd integer, $\delta_1 \times n$ is a sequence of even, odd integers as $n = 0, 1, .., N$. Thus, the exponential term is basically a sequence of $\{+1, -1, +1, -1, ...\}$, which multiplies $y[n]$. In this case, BlueFunnel cannot directly demodulate $y[n]$. To null the effect of the exponential term, BlueFunnel simply multiplies $y[n]$ by the same sequence of $\{+1, -1, +1, -1, ...\}$.

However, the key question is how to know whether δ_1 is an odd or even integer for a given signal $y[n]$? To answer this question, BlueFunnel relies on some features of Bluetooth Classic packets and conduct some statistics to infer the answer. Specifically, a Bluetooth Classic packet starts with an access code followed by a packet header. The header is protected by $1/3$ FEC error correction code. Given that the packet header is a string of 18 bits, the protected header is a string of 54 bits (i.e. 18×3) [46]. So it is common to see bit patterns of three ones, like '111', or three zeros in a row when demodulating packet header. These bit patterns, which are mandatory introduced by FEC, is a good indicator that a received signal $y[n]$ does not suffer from a shift.

To conclude, BlueFunnel infers whether δ_1 is odd or even based on the following steps. *(i)* Given a received signal $y[n]$, BlueFunnel shifts the signal based on multiplying it's symbols by $\{+1, -1, +1, -1\}$ (in fact, processing header bits is good enough here) to get $y'[n]$. *(ii)* BlueFunnel demodulates headers of $y[n]$ and $y'[n]$ and looks for specific bit patterns, like three zeros in a row '000', or three ones in a row, that are introduced by $1/3$ FEC error correction code. And *(iii)* once BlueFunnel identifies the shifted version of $y[n]$, that is either $y[n]$ or $y'[n]$, it can demodulate the received signal.

Moreover, the access code is a string of 72 bits, and is fixed for a given Bluetooth Classic piconet. This means all the piconet packets start with the same access code. BlueFunnel leverages this property as follows. BlueFunnel saves access codes and headers

of $y[n]$ and $y'[n]$ in a data base to identify shifted signals in future. That is, BlueFunnel compares incoming signals against access codes and headers of $y[n]$ and $y'[n]$ to identify shifted version of later received signals $y[n]$.

It is also worth to mention that this algorithm requires light-weight computations. This is because the processes of this algorithm, including shifting header of a signal, demodulation, comparing and saving signal header, does not require significant computations as the total number of bits in the access code and packet header is $72 + 54 = 126$.

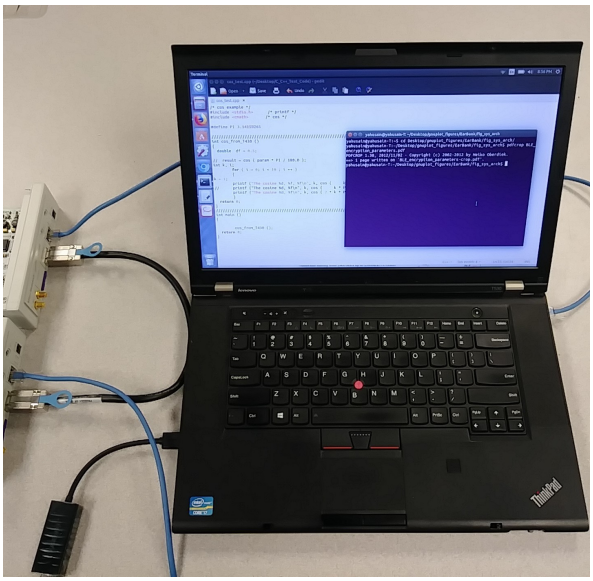
5.4 Implementation

In this section, we present implementation of BlueFunnel prototype in details.

To build a prototype for BlueFunnel, we opt to use off-the-shelf Universal Software Radio Peripheral (USRP), namely USRP2 [22], along with SBX daughterboard [16], which covers variety of bands including the 2.4 GHz ISM band. Due to the hardware limitations imposed by USRP2/SBX configuration, including the limitation of RF-front-end bandwidth, we have to use two USRP2 devices to observe the entire Bluetooth spectrum. In particular, the highest possible bandwidth supported by USRP2/SBX configuration is 50 MHz. Therefore, we built a customized software defined radio (SDR) based on two USRP2 devices to observe 100 MHz of the free 2.4 GHz spectrum, starting from 2400 MHz up to 2500 MHz. Each USRP2 is equipped with SBX daughterboard and two 2.4 GHz 3dBi antennas. The customized SDR is shown in Fig. 5.7(a).

Another limitation imposed by USRP2/SBX configuration is the difficulty of tuning the ADC sampling rate. However, this customized SDR provides BlueFunnel with the highest possible sampling rate, which is about 100 Msamples/sec. To mimic a low-speed

(a) Software Defined Radio (SDR).



(b) Linux-Based Lenovo T530 laptop.

Figure 5.7: A prototype of BlueFunnel sniffer, where we built a customized Software Defined Radio (SDR) platform based on two USRP2 devices as shown in (a). The software components, including packet header detector and Bluetooth signal demodulator, are implemented on the Linux-based Lenovo T530 laptop as shown in (b). The USRP2 devices are synchronized via a MIMO-Cable and each USRP2 is connected to the host (Lenovo laptop) via Gigabit Ethernet cable.

ADC, we implement a digital down converter on the host, which basically decimates the complex samples stream and makes the sample rate as 2 MSymbols/ses. Base on this simple trick, the customized SDR meets the BlueFunnel required sampling rate.

For the purpose of clock synchronization, we utilize a MIMO-Cable [42], which acts to synchronize the ADCs on both USRP2 devices to the same clock tick. Each USRP2 is interfaced to a Linux-based laptop via Gigabit Ethernet cable. We configure each USRP2 to provide complex real-time samples, where the resolution of a complex sample is 16 bits; that is, 8 bits are assigned for the real part and the other 8 bits are assigned for the imaginary part. The center frequency of the first USRP2 is set to 2420 MHz, while the center frequency for the second USRP2 device is set to 2460 MHz.

For the host computer, we utilizes a Linux-based Lenovo T530 laptop (Processor Corei7 2.4 GHz third generation, RAM 16 GB, Hard drive SSD 250 GB) as shown in Fig. 5.7(b).

We implement the software components for BlueFunnel, including packet header detector and Bluetooth signal demodulator, on the Linux-based laptop with 530 lines of C/C++ code. For better computational efficiency in real-time, BlueFunnel components are implemented as threats. BlueFunnel components are interfaced to UHD [21] – the celebrated driver for USRP hardware devices. As the software components need to interact with the UHD driver in real-time, we opt to use shared memory technique to provide an interface for such high speed, real-time interaction. In particular, the shared memory is used to pass data between UHD and BlueFunnel. We write a C/C++, Linux-based shared memory package that supports interaction between BlueFunnel and UHD driver. Accordingly, we modified the UHD driver so that it supports the shared memory interfaced.

5.5 System Performance

In this section, we evaluate BlueFunnel performance.

5.5.1 Experimental Methodology

The testbed for our experiment consists of the following.

(i) Bluetooth Classic testbed, which includes two wireless headsets, 37 USB adapters, and built-in Bluetooth interfaces in four laptops, eight smartphones, and five tablets. We also employs two Linux-Based Asus netbooks, to which we connect Bluetooth USB adapters and establish a link.

(ii) For Bluetooth Low Energy (BLE), we employ the following devices as piconet master: nRF52840 development kit (DK) [35], two Bluetooth dual-mode USB adapters, and three smartphones, including two Moto G and LG Stylo3+. For the piconet slave, we utilizes

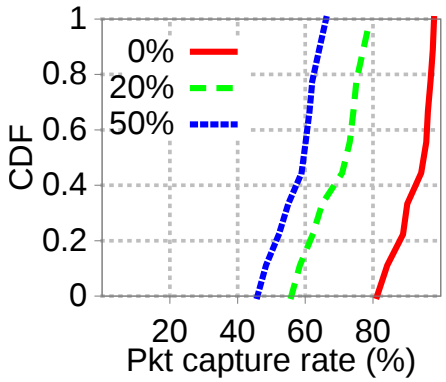


Figure 5.8: Packet capture rate achieved by BlueFunnel under various interference conditions in a controlled setting.

BlueFruit LE [2], and Bluetooth LE nRF51 USB Dongle [34]. We also employ Linux-based Lenovo T530, Windows-based Lenovo T430, and two Linux-based Asus netbooks. These laptops are used to connect and operate the above Bluetooth LE devices.

5.5.2 Evaluation

We evaluate BlueFunnel performance based on packet capture rate in a controlled environment. We also evaluate the system performance based on packet capture rate in a practical environment, where various interference patterns introduced by enterprise 802.11-based WLANs radios. We opt to conduct this experiment in the engineering building at Michigan State University.

5.5.2.1 Packet Capture Rate in Controlled Settings

In this experiment, we evaluate the system performance under the following conditions: *(i)* 802.11-based traffic interferes with 50% of Bluetooth subchannels; *(ii)* 802.11-based traffic interferes with 25% of Bluetooth subchannels; and *(iii)* No Bluetooth subchannel is interfered by 802.11-based traffic. The experiment setup consists of three Linux-based Asus

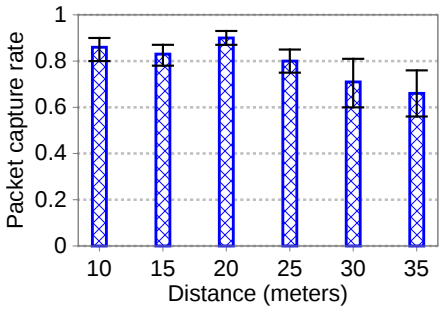


Figure 5.9: Packet capture rate achieved by BlueFunnel at different from the target.

netbooks, Bluetooth Classic USB adapter and Moto G smartphone. We utilizes two of the netbooks to generate 802.11-based traffic. We write a user-space script, which interacts with an open source 802.11 Linux driver, namely . The script configures the 802.11 WLAN adapter on the netbook to be in monitor mode and injects 802.11-based packets [52].

Fig. 5.8 evaluates system performance in terms of packet capture rates under three interference conditions. As shown in the figure, the system performance is the highest when no interference is introduced by 802.11-based WLANs. However, the packet capture rates degraded as the number of interfered subchannel is increased. Specifically, the system experience the worst performance when 50% of Bluetooth subchannels are interfered by 802.11-based traffic. This due to significant packets collisions, i.e. between 802.11-based packet and Bluetooth Classic packets, on the overlapped subchannels.

5.5.2.2 Packet Capture Rate in Practical Environment

In this experiment, we evaluate the system performance in a practical environment, which an office building. In this setting, different interference patterns are introduced by enterprise 802.11-based WLANs radios. We evaluate packet capture rate as: *(i)* BlueFunnel is placed at different distances from the target; and *(ii)* BlueFunnel captures Bluetooth packets at different sites in the building.

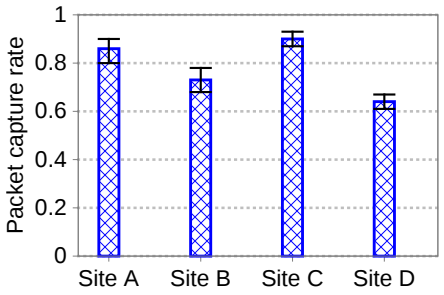


Figure 5.10: Packet capture rate achieved by BlueFunnel at different locations in office Building.

Fig. 5.9 compares the system performance in terms of packet capture rates, where BlueFunnel is placed at different distances from the target Bluetooth link. As shown in the figure, the system presents high performance when the target is less than 21 meters away. The packet capture rate starts to degrade as BlueFunnel moves away from the target. This mainly because signal attenuation, which degrades the signal-to-noise-ratio (SNR) at BlueFunnel receiver.

Fig. 5.10 compares the system performance at different sites in the building, where different interference conditions are expected. As shown in the figure, the packet capture rates vary from one site to other. This is because the system experiences various interference conditions, that is mainly introduced by 802.11-based WLANs, at different sites of the building.

5.6 Attack Scenarios

In this section, we introduce two scenarios of possible attacks against Bluetooth. As proof of the concept, we implement these attacks based on BlueFunnel traffic sniffer. As we discussed in chapter 3, there are several cryptanalysis attacks against Bluetooth encryption, including Bluetooth Classic and Bluetooth LE. A prerequisite for these kinds of attacks is a Bluetooth traffic sniffer, such as BlueFunnel. As a proof-of-concept, we present in this

```
%%%%%%%%%%%%%%%%%%%%%%%% pkt0 (adv)
time   8924, snr=35.7, BTLE index=37, AA=8e89bed6,
PDUType=0, TxAdd=1, RxAdd=0, Length=22
  AdvA=2f388ab4ecc2

  (char) AdvData= . . . . . N o r d i c _ U A R T
  (byte) AdvData=0201050c094e6f726469635f55415254
%%%%%%%%%%%%%%%%%%%%%%%% pkt1 (adv)
time   8926, snr=35.2, BTLE index=38, AA=8e89bed6,
PDUType=5, TxAdd=1, RxAdd=1, Length=34
  InitA=cc75a19cfa71
  AdvA=2f388ab4ecc2
  AA=295bd110, CRCInit=0178de, WinSize=2, WinOffset=4
  Interval=39, Latency=0, Timeout=2000,
  ChannelMap=1fffffffff, HopInc=15, SCA=5
%%%%%%%%%%%%%%%%%%%%%%%% pkt2 (Ctrldata)
time   9016, snr=39.5, BTLE index=08, AA=295bd110,
PDUType=5, TxAdd=1, RxAdd=1, Length=34
  EDIV = 0x5b0a
  SKDm = 0x87609adb102d09ba
  IVm = 0xdb239031
%%%%%%%%%%%%%%%%%%%%%%%% pkt3 (data)
```

Figure 5.11: Bluetooth LE encryption establishment parameters (marked with red).

section two scenarios to show that a successful traffic sniffing, based on a traffic sniffer like BlueFunnel, may compromise Bluetooth privacy. In particular, we implement one attack against Bluetooth Classic, where BlueFunnel reveals (x, y) coordinates out of capture packets. We also implement another attack against a Bluetooth LE target. In this case, BlueFunnel listens on the pairing phase of Bluetooth LE and reveals sensitive parameters that are used to generate link encryption key.

5.6.1 Sniffing on Pairing Phase of Bluetooth LE

The importance of sniffing on Bluetooth LE pairing phase is of increasing interest because of the following. First, Blueooth LE engineers and applications developers need to examine Bluetooth LE link establishment, and hence, monitoring these early stages of handshaking including pairing phase, is important. Second, the pairing phase involves the exchange of sensitive parameters between Bluetooth LE piconet participants in early stages of link establishment. These parameters, including encryption keys, frequency

hopping parameters, are if privacy concern with respect to Bluetooth LE users. However, the pairing phase is a short-lived session that may involve a few tens of Bluetooth LE packets. In this experiments, we deploy BlueFunnel to eavesdrop on pairing phase of a Bluetooth LE link. We show that BlueFunnel can successfully sniffing on Bluetooth LE paring and reveal Short Terms Key (STK), which and be used by an attacker to determine LTK of the Bluetooth LE victim. In particular, at the pairing phase, Bluetooth LE shares some random seeds that are used to generate STK [46]. As proof-of-concept application, we leverage our design to show that BlueFunnel can reveal STK seeds, which paves the way to compromise Bluetooth LE privacy.

The experimental setup. The testbed setup of this experiment consists of nRF52 board [35], that plays the role of Bluetooth LE master, and LG Stylo 3+ smartphone (Bluetooth LE slave). We deploy BlueFunnel system to sniff on pairing phase of Bluetooth LE. We run this experiment as follows. First, we run BlueFunnel to sniff on Bluetooth LE traffic. Second, we established Bluetooth LE link between LG Stylo 3+ smartphone and the nRF52 board.

As shown in Fig. 5.11, BlueFunnel reveals sensitive link parameters, including EDIV, SKDm, and IVm. These parameters are used as seeds by Bluetooth LE piconet participant to establish STK, which can be used by an attacker to reveal long term encryption key [36] [3].

5.6.2 Sniffing on Bluetooth Mouse Traffic

A recent research [39] reported that mouse movement data leakage is extremely privacy sensitive and may lead to reveal sensitive data like users passwords. In this scenario of attack, we target Bluetooth Classic link. That is, as proof-of-concept, we utilizes BlueFun-

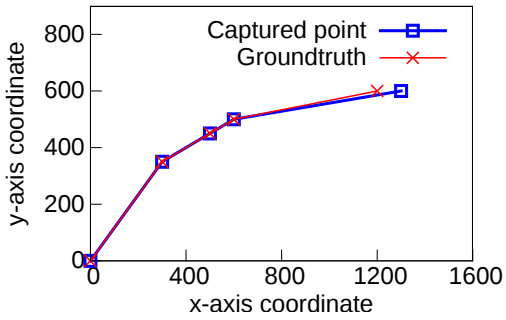


Figure 5.12: Mouse coordinates obtained from captured Bluetooth packets.

nel to sniff on Bluetooth mouse traffic and reveal the mouse coordinates. We opt to establish unencrypted Bluetooth Classic link with the mouse to facilitate extraction of data from packet payload. Although, the cryptanalysis of Bluetooth Classic cipher is studied [55] [56] in the literature, implementing cryptanalysis attack against captured packets is beyond the scope of this thesis.

The experiment setup. The setup of this experiment consists of an HP Bluetooth Mouse X4000B that is paired with a Linux-based Notebook. The Notebook is equipped with a USB Bluetooth adapter. To collect groundtruth coordinates of the mouse pointer, we wrote a python-based script, that runs in Linux user-space, to report the mouse coordinates, i.e. (x, y) pairs, in real-time. Finally, we deploy BlueFunnel to sniff on Bluetooth traffic. The mouse generates traffic as the user moves the mouse pointer.

Fig. 5.12 shows the revealed (x, y) coordinates out of captured packets. The figure compares the captured coordinates by BlueFunnel sniffer (in blue) against groundtruth (x, y) coordinates (in red), that are collected on the Linux-based laptop. The plot shows that the collected mouse trace experience some error points, which do not match with groundtruth trace; this is because BlueFunnel may experience some bit-level errors specially when demodulating Bluetooth signal in the presence of interference.

5.7 Summary of Chapter

In this chapter, we introduce BlueFunnel –a low-power, wideband Bluetooth traffic sniffer that monitors Bluetooth spectrum in realtime. BlueFunnel features a low-power software defined radio architecture, which integrates a low speed ADC, and suite of novel digital signal processing algorithms. We present a prototype implementation for BlueFunnel system that is based on USRP2 devices and SBX daughterboard. We evaluate the system performance in a controlled and practical environment, to understand the impacts of 802.11-based interference. The experiments show that the systems maintains good level of packet capture rates. Further, we discuss two scenarios of attacks based on a wideband traffic sniffer, like BlueFunnel system. The first scenario considers sniffing on Bluetooth LE pairing phase, where BlueFunnel successfully reveals sensitive security information including link encryption primitives. The second scenario employs BlueFunnel to eavesdrop on data traffic transmitted by Bluetooth mouse to a personal computer, where BlueFunnel successfully extracts the mouse coordinates from captured packets.

Chapter 6

Conclusion and Future Work

This thesis tackles the challenges of sniffing Bluetooth traffic, including wideband spread spectrum, pseudo-random frequency hopping adopted by Bluetooth at baseband, and the interference in the open 2.4 GHz band. Specifically, we introduce practical systems for sniffing Bluetooth traffic in the wild. In this chapter, we summarize the contributions of this thesis, and then present future work.

6.1 Contributions

The main contributions of this thesis are summarized as follows.

(*i*) *BlueEar Sniffer* We present the design, implementation, and evaluation of *BlueEar* – the first practical Bluetooth traffic sniffer that consists only of inexpensive, Bluetooth-compliant radios. BlueEar features a novel dual-radio architecture, where two radios – named as scout and snooper – coordinate with each other on learning the hopping sequence of indiscoverable Bluetooth, predicting adaptive hopping behavior, and handling complex interference conditions. We implemented a prototype of BlueEar for sniffing the traffic of Bluetooth Classic, which offers enhanced data rates and a more complex hopping protocol than Bluetooth LE. The prototype employs two Ubertooth devices to implement the scout and the snooper, and interfaces them to

a controller running on a Linux laptop. Extensive experiments results show that BlueEar can maintain a packet capture rate higher than 90% consistently in practical environments, where the target Bluetooth network exhibits diverse hopping behaviors in the presence of interference from coexisting 802.11 based WLANs. Further, we discuss privacy implications of BlueEar on Bluetooth system and propose a practical countermeasure approach that effectively reduces the average packet capture rate of a sniffer to 20%.

(*ii*) *BlueFunnel Sniffer* We introduce the design, implementation, and evaluation of *BlueFunnel* –a low-power, wideband traffic sniffer that monitors Bluetooth spectrum in parallel and captures packet in realtime. BlueFunnel leverages low speed ADC to subsample Bluetooth spectrum to address the challenge of high-speed ADC. In particular, BlueFunnel consists of two main components, including, *(i)* a software defined radio (SDR), that integrates a wideband radio, and a low-speed ADC, which subsamples Bluetooth spectrum based on 2 Msamples/sec, which is below the Nyquist-Shannon rate; and *(ii)* a controller, that implements digital signal processing packages, which are responsible for detecting, demodulating, and analyzing Bluetooth packets in realtime. We implement BlueFunnel prototype based on USRP2 devices. Specifically, we employ two USRR2 devices, each is equipped with SBX daughterboard, to build a customized software radio platform. The customized SDR platform is interfaced to the controller, which implements the digital signal processing algorithms on a personal laptop. We evaluate the system performance based on packet capture rates in variety of settings. Specifically, we evaluated the system in a controlled setting, where we intentionally introduce 802.11-based traffic as a form

of interference on overlapped Bluetooth subchannels. We test BlueFunnel when there is no interference, 25% and 50% of the subchannels are interfered with 802.11-based traffic, respectively. Moreover, we evaluate the system in an office building, where dynamic interference conditions are introduced by an enterprise 802.11-based WLAN. In all settings, BlueFunnel maintains good levels of packet capture rates. Further, we introduce two scenarios of attacks against Bluetooth, where BlueFunnel successfully reveals sensitive information about the target link.

6.2 Future work, and Conclusion

In this section, we present some future work for BlueFunnel system. This includes,

(*i*) *Interference cancellation.* In practical environment, BlueFunnel is expected to experience interference introduced by ambient radios, specially the prevalent 802.11-based WLANs. Therefore, the next step is to connect a small group of SDRs and synchronized such devices to performance interference nulling and enable demodulation of Bluetooth signal in the presence of interference.

(*ii*) *Aliasing and multiple packet reception.* As BlueFunnel subsamples of the wideband spread spectrum in time-domain, it causes aliasing in frequency domain. Therefore, Bluetooth packets from multiple piconets could collide at BlueFunnel's antenna. We design an algorithm to resolve collision and enable multiple Bluetooth reception by BlueFunnel.

(*iii*) *Demodulating enhance data rate packets*. To enhance BlueFunnel, we plan to introduce some algorithms to demodulate enhanced data rates packets, which are modulated

based on Differential Phase-shift Keying scheme.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Adafruit. Bluefruit le sniffer.

[2] Adafruit. Bluefruit le sniffer. https://www.adafruit.com/products/2269.

[3] M. Afaneh. The best bluetooth low energy sniffer tutorial (connections). https://www.novelbits.io/best-bluetooth-low-energy-sniffer-tutorial-connections/.

[4] M. Afaneh. Five essential tools for every bluetooth low energy developer. Online: goo.gl/C5sMK1.

[5] Apple. Apple carplay.

[6] A. Biryukov, D. Khovratovich, and I. Nikolić. Distinguisher and related-key attack on the full aes-256. In S. Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, pages 231–249, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[7] A. Bogdanov, D. Khovratovich, and C. Rechberger. Biclique cryptanalysis of the full aes. In *Proceedings of the 17th International Conference on The Theory and Application of Cryptology and Information Security*, ASIACRYPT'11, Berlin, Heidelberg, 2011. Springer.

[8] M. Bon. A basic introduction to ble security. https://pomcor.com/2015/06/03/has-bluetooth-become-secure/, Accessed: Oct, 2016.

[9] N. Cheng, X. O. Wang, W. Cheng, P. Mohapatra, and A. Seneviratne. Characterizing privacy leakage of public wifi networks for users on travel. In *2013 Proceedings IEEE INFOCOM*, April 2013.

[10] A. Cidon, K. Nagaraj, S. Katti, and P. Viswanath. Flashback: Decoupled lightweight wireless control. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12. ACM, 2012.

[11] P. Cope, J. Campbell, and T. Hayajneh. An investigation of bluetooth security vulnerabilities. In *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1–7, Jan 2017.

[12] F. Corella. Has bluetooth become secure? https://pomcor.com/2015/06/03/has-bluetooth-become-secure/, Accessed: Jun, 2015.

[13] A. K. Das, P. H. Pathak, C.-N. Chuah, and P. Mohapatra. Uncovering privacy leakage in ble network traffic of wearable fitness trackers. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, HotMobile '16. ACM, 2016.

[14] Ellisys. Ellisys protocol test solutions. `https://www.ellisys.com/products/bex400`.

[15] Ettus. Bluetooth technology website. `https://www.bluetooth.com/`.

[16] Ettus. Ettus research - sbx40. `https://www.ettus.com/product/details/SBX`.

[17] S. Gollakota, F. Adib, D. Katabi, and S. Seshan. Clearing the rf smog: Making 802.11n robust to cross-technology interference. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, New York, NY, USA, 2011. ACM.

[18] Google. Android auto. `https://www.android.com/auto`.

[19] J. Huang, W. Albazrqaoe, and G. Xing. Blueid: A practical system for bluetooth device identification. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 2849–2857, April 2014.

[20] J. Huang, G. Xing, G. Zhou, and R. Zhou. Beyond co-existence: Exploiting wifi white space for zigbee performance assurance. In *Network Protocols (ICNP), 18th IEEE International Conference on*, Oct 2010.

[21] N. Instruments. Usrp hardware driver. `https://www.ettus.com/sdr-software/detail/usrp-hardware-driver`.

[22] N. Instruments. Usrp2. `http://files.ettus.com/manual/page_usrp2.html`.

[23] T. Instruments. Ble sniffer. `http://processors.wiki.ti.com/index.php/BLE_sniffer_guide`.

[24] T. Instruments. Sniffer firmware of cc2540. `https://e2e.ti.com/support/wireless_connectivity/f/538/t/197748`.

[25] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, Cambridge, MA, 1999.

[26] B. P. Lathi. *Modern Digital and Analog Communication Systems*. Oxford University Press, Inc., New York, NY, USA, 2nd edition, 1995.

[27] T. LeCroy. Teledyne lecroy inc. `http://teledynelecroy.com/frontline`.

[28] C.-J. M. Liang, N. B. Priyantha, J. Liu, and A. Terzis. Surviving wi-fi interference in low power zigbee networks. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2010.

[29] Logitech. Logitech advanced 2.4 ghz technology.

[30] R. G. Lyons. *Understanding Digital Signal Processing (Second Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[31] D. S. Michael Ossmann et al. Ubertooth.

[32] A. N. Moldovan, I. Ghergulescu, and C. H. Muntean. A novel methodology for mapping objective video quality metrics to the subjective mos scale. In *Broadband Multimedia Systems and Broadcasting (BMSB), 2014 IEEE International Symposium on*, June 2014.

[33] M. Moser. Busting the bluetooth myth – getting raw access.

[34] Nordic. Ble sniffer. `https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF-Sniffer`.

[35] Nordic. nrf52 preview development kit. `https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF52-Preview-DK/(language)/eng-GB`.

[36] J. Padgette, J. Bahr, and et al. Guide to bluetooth security. `https://doi.org/10.6028/NIST.SP.800-121r2`, 2010.

[37] J. Padgette, J. Bahr, and et al. Guide to bluetooth security, 2017.

[38] X. Pan, Z. Ling, A. Pingley, W. Yu, N. Zhang, and X. Fu. How privacy leaks from bluetooth mouse? In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12. ACM, 2012.

[39] X. Pan, Z. Ling, A. Pingley, W. Yu, N. Zhang, K. Ren, and X. Fu. Password extraction via reconstructed wireless mouse trajectory. *IEEE Transactions on Dependable and Secure Computing*, 13(4):461–473, July 2016.

[40] Y. Qu and P. Chan. Assessing vulnerabilities in bluetooth low energy (ble) wireless network based iot systems. In *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, pages 42–48, April 2016.

[41] E. Research. Ettus research.

[42] E. Research. Mimo cable. https://www.ettus.com/product/details/MIMO-CBL.

[43] E. Ronen, A. Shamir, A. O. Weingarten, and C. OFlynn. Iot goes nuclear: Creating a zigbee chain reaction. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 195–212, May 2017.

[44] M. Ryan. Bluetooth: With low energy comes low security. In *Proceedings of the 7th USENIX Conference on Offensive Technologies*, WOOT'13. USENIX Association, 2013.

[45] K. Schramm, G. Leander, P. Felke, and C. Paar. A collision-attack on aes. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, pages 163–175, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[46] B. SIG. Bluetooth core specification v5.0. https://www.bluetooth.org/.

[47] B. SIG. Bluetooth technology.

[48] M. Song, S. Shetty, and D. Gopalpet. Coexistence of ieee 802.11b and bluetooth: An integrated performance analysis. *Mobile Networks and Applications*, 12(5):450–459, Dec 2007.

[49] D. Spill and A. Bittau. Bluesniff: Eve meets alice and bluetooth. In *Proceedings of the First USENIX Workshop on Offensive Technologies*, WOOT '07. USENIX Association, 2007.

[50] D. Spill and M. Ossmann. Gr-bluetooth. https://github.com/greatscottgadgets/gr-bluetooth.

[51] Wikipedia. Frequency-shift keying (fsk). https://en.wikipedia.org/wiki/Frequency-shift_keying.

[52] L. Wireless. 802.11 wlan driver. https://wireless.wiki.kernel.org/en/users/drivers/ath9k.

[53] B. Yu, L. Yang, and C. C. Chong. Optimized differential gfsk demodulator. *IEEE Transactions on Communications*, 59(6):1497–1501, June 2011.

[54] Y. Yubo, Y. Panlong, L. Xiangyang, T. Yue, Z. Lan, and Y. Lizhao. Zimo: Building cross-technology mimo to harmonize zigbee smog with wifi flash without intervention. In *Proceedings of the 19th Annual International Conference on Mobile Computing and Networking*, MobiCom '13, New York, NY, USA, 2013. ACM.

[55] B. Zhang, C. Xu, and D. Feng. Real time cryptanalysis of bluetooth encryption with condition masking. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[56] B. Zhang, C. Xu, and D. Feng. Practical cryptanalysis of bluetooth encryption with condition masking. In *Journal of Cryptology*, Jul, 2017.

[57] T. Zhang, J. Lu, F. Hu, and Q. Hao. Bluetooth low energy for wearable sensor-based healthcare systems. In *2014 IEEE Healthcare Innovation Conference (HIC)*, pages 251–254, Oct 2014.