Practical Bluetooth Traffic Sniffing: Systems and Privacy Implications

Wahhab Albazrqaoe[†‡], Jun Huang[†], Guoliang Xing[†]
[†]Department of Computer Science and Enginnering, Michigan State University, USA.
[‡]University of Karbala, Karbala City, Iraq.
{albazrqa, huangjun, glxing}@cse.msu.edu

ABSTRACT

With the prevalence of personal Bluetooth devices, potential breach of user privacy has been an increasing concern. To date, sniffing Bluetooth traffic has been widely considered an extremely intricate task due to Bluetooth's indiscoverable mode, vendor-dependent adaptive hopping behavior, and the interference in the open 2.4 GHz band. In this paper, we present *BlueEar* – a practical Bluetooth traffic sniffer. BlueEar features a novel *dual-radio architecture* where two Bluetooth-compliant radios coordinate with each other on learning the hopping sequence of indiscoverable Bluetooth networks, predicting adaptive hopping behavior, and mitigating the impacts of RF interference. Experiment results show that BlueEar can maintain a packet capture rate higher than 90% consistently in real-world environments, where the target Bluetooth network exhibits diverse hopping behaviors in the presence of dynamic interference from coexisting 802.11 devices. In addition, we discuss the privacy implications of the BlueEar system, and present a practical countermeasure that effectively reduces the packet capture rate of the sniffer to 20%. The proposed countermeasure can be easily implemented on the Bluetooth master device while requiring no modification to slave devices like keyboards and headsets.

1. INTRODUCTION

In recent years, Bluetooth has enjoyed an unprecedented penetration rate in mobile devices. About three billions Bluetooth units will be shipped worldwide in 2017 [2]. In particular, Bluetooth has become the *de facto* connectivity interface for wireless accessories and smart devices including keyboard/mouse, headsets, wearables like fitness trackers and smart watches, smart appliances, and in-car telematic systems like Android Auto [1] and CarPlay [4]. Because the communication of these devices is privacy-sensitive in nature, Bluetooth employs a two-level stream cipher to encrypt packets at the link-layer [32]. Unfortunately, recent studies have revealed many critical flaws of this encryption scheme [17] [10] [24] [26] [25] [31] [9] [28]. 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 [9] [31] [24].

Despite the well-documented flaws of Bluetooth encryption, to date, sniffing Bluetooth traffic has been considered an extremely intricate task due to the following reasons. *(i)* Bluetooth employs frequency hopping spread spectrum, where carrier frequency is rapidly switched following a pseudo-random hopping sequence. The hopping sequence is hidden when Bluetooth is in indiscoverable mode, making it difficult for a sniffer to hop following the target. *(ii)* When coexisting with other wireless devices on overlapping frequencies, Bluetooth performs adaptive hopping, where the hopping sequence is frequently modified to adapt spectrum use. Such an adaptive hopping behavior is vendor-dependent, and may differ significantly across different devices. *(iii)* In the crowded 2.4 GHz band, the sniffer may experience intensive interference from coexisting wireless devices, especially the prevalent 802.11 based WLANs, resulting in poor sniffing performance.

There exist a few proprietary off-the-shelf Bluetooth sniffers [12] [13], which are primarily designed for protocol diagnosis in controlled wireless settings. In addition, to sniff the traffic of frequency hopping Bluetooth, they rely on specialized radio to monitor all Bluetooth subchannels in parallel, which makes them costly. As an example, off-the-shelf sniffers manufactured by Frontline Test Equipment [11] – the leading provider of Bluetooth protocol analyzer – cost \$10K to \$25K per unit. Recently, a few low-cost Bluetooth packet sniffers have been developed based on open-source wireless platforms [6] [5] [29] [33] [30] [7] [20] [3]. These systems are exclusively designed for sniffing Bluetooth traffic in basic hopping mode. In practice, they suffer prohibitively poor sniffing performance due to the misprediction of adaptive hopping behavior, as well as excessive packet corruptions caused by the interference in the open 2.4 GHz band.

As Bluetooth becomes increasingly popular worldwide, an in-depth study on its resistance to traffic sniffing becomes imperative. In this paper, we explore the feasibility and privacy implications of sniffing Bluetooth traffic in practical environments using general, inexpensive wireless platforms. 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 of only inexpensive, Bluetooth-compliant radios. BlueEar features a novel *dual-*

MobiSys'16, June 25-30, 2016, Singapore, Singapore
© 2016 ACM. ISBN 978-1-4503-4269-8/16/06…\$15.00
DOI: http://dx.doi.org/10.1145/2906388.2906403

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 determine 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 Low Energy (BLE). The prototype employs two Ubertooths [6] 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 BLE 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 paper is organized as follows. Section 2 reviews related work. In section 3, we introduce the background on Bluetooth system in general. We present system overview and architecture in section 4. In section 5, we discuss BlueEar system design in detail. Evaluation of BlueEar system performance is presented in section 7. We discuss the impacts of BlueEar on privacy breach for Bluetooth system, including BLE, in section 8. Section 9 concludes the paper.

2. RELATED WORK

Bluetooth 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. Recent studies have revealed many critical flaws of this encryption scheme [31] [17] [10] [24] [26] [25]. First, the security of E_0 relies on the strength of PIN – which is often too weak and can be easily cracked [31]. Moreover, recent cryptanalysis studies have shown that the 128-bit link key of E_0 is considerably weaker than what is originally intended [17]. It can be cracked in 2^{65} operations, instead of 2^{128} [10]. Furthermore, the effective security of the link key will further degrade when a packet sniffer is employed by the attacker [24] [26] [25]. In particular, after capturing 44.3 MB of data transferred by the target, the attacker can reverse the link key in 2^{38} operations, which makes fast key recovery possible in practice [24].

Recent studies have demonstrated the feasibility of circumventing Bluetooth encryption using expensive, proprietary packet sniffers. 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 [9]. Moreover, due to computation and power constraints, many Bluetooth peripherals – including most Bluetooth mice manufactured by major vendors like Logitech [23] – do not support encryption, which may result in user privacy breach.

Despite the well-documented flaws of Bluetooth encryption, sniffing Bluetooth traffic has been widely considered an extremely intricate task. 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 [7] [20] [3]. However, the sniffing device must pair with the target, which makes it incapable of passive sniffing. The GNURadio/USRP [5] [29] platform can be programmed to decode Bluetooth packets [33]. Due to large signal processing overhead and frequency switching delay, they are limited to sniffing one subchannel at a time. There exist several proprietary Bluetooth packet sniffers designed for protocol diagnosis in controlled wireless settings [12] [13]. They rely on specialized radio to monitor all subchannels in parallel, which makes them extremely costly. For instance, off-the-shelf sniffers manufactured by Frontline Test Equipment [11] – the leading provider of Bluetooth protocol analyzer – cost \$10K to \$25K per unit.

Recently, several low-cost Bluetooth packet sniffers [6] [30] have been developed based on Ubertooth [6] – an open-source Bluetooth development platform. However, existing Ubertooth-based sniffers 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 [14] [22] [34] [19]. 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.

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. BLUETOOTH BACKGROUND

Piconet. Bluetooth networks employ a master-slave structure called *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.

The hopping protocol. We now introduce the hopping protocol of Bluetooth Classic, which is more complex than that of BLE. The hopping protocol of BLE is discussed in Section 8. In Bluetooth Classic, the hopping protocol is defined by a *physical channel*, which is characterized by pseudo-random hopping over 79 subchannels from 2.4 to 2.48 GHz. The carrier frequency is switched every 625 μs, resulting in a maximum hopping rate of 1600 hops/s. 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 [32], A is the piconet address, 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 take 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 $i_n = \mathcal{H}(A, n)$. The hopping phase, which is determined by the piconet clock, is the index of the current hop.

(ii) Adapted channel. When coexisting with other wireless devices on overlapping frequencies, 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. 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.

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 [33], a Bluetooth packet sniffer cannot hop following the target unless it acquires all hidden parameters.

4. BLUEEAR OVERVIEW

4.1 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 implementation, 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 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. 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 traffics based on the piconet address of captured packets. Specifically, the preamble of each Bluetooth packet carries a synchronization word, which is derived from the piconet address using an encoding function specified by the standard [32]. BlueEar employs the brute force algorithm proposed in [33] 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.

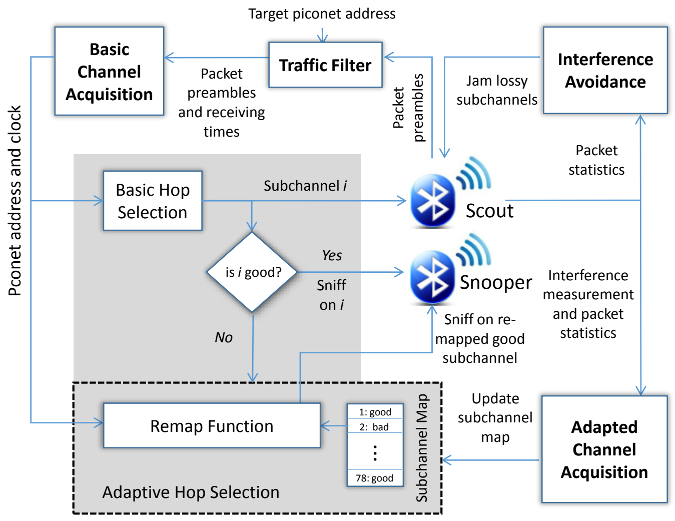
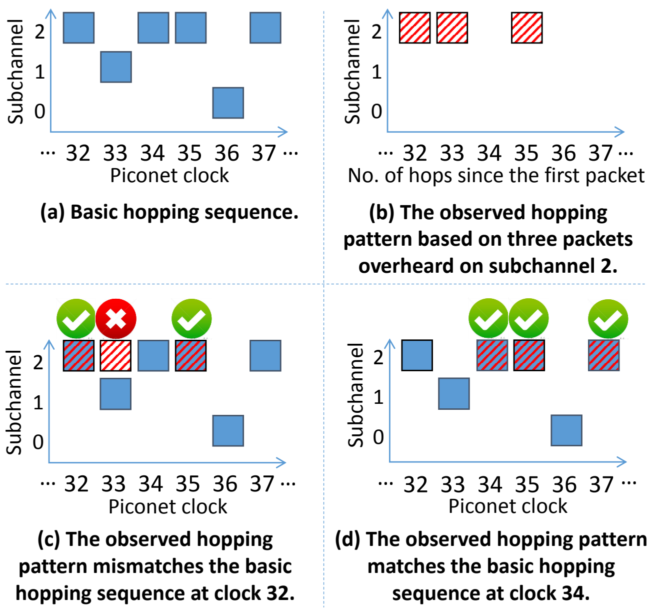


Figure 1: The dual-radio architecture of BlueEar. Components in the gray area comprise a standard-compliant hop selection kernel.



Subchannel
2
1
0
··· 32 33 34 35 36 37 ···
Piconet clock
(a) Basic hopping sequence.
No. of hops since the first packet
(b) The observed hopping pattern based on three packets overheard on subchannel 2.
(c) The observed hopping pattern mismatches the basic hopping sequence at clock 32.
(d) The observed hopping pattern matches the basic hopping sequence at clock 34.

Figure 2: An illustrative example of brute force clock acquisition.

(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 hopping 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.

5. DESIGN OF BLUEEAR

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

5.1 Clock Acquisition

In the following, we first introduce an algorithm that can reverse the piconet clock when the target hops in the basic mode, and then augment it with probabilistic matching to determine the piconet clock in the presence of interference.

5.1.1 Brute Force Clock Acquisition

Because the entire hopping sequence is known after the piconet address is extracted from packet preamble [33], it is possible to reverse the piconet clock through a simple brute force search, which compares the observed hopping pattern with the entire hopping sequence at all phases to search for a match. Fig. 2 shows an illustrative example of brute force clock acquisition. At the beginning, the scout listens on a single subchannel to monitor the target's packet transmissions. As shown in the example given in Fig. 2, the scout listens on subchannel 2 where three packets are captured. The receiving times of these packets compose a hopping pattern that describes how the target visits the monitored subchannel. As shown in Fig. 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 exponentially fast as the number of captured packets increases. As a result, the target piconet clock can be determined in a short time period, which we will evaluate in detail in Section 7.2.

5.1.2 *Probabilistic Clock Matching*

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 [14] [22] [34] [19]. 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. 3 illustrates the impact of subchannel remapping using the same example shown in Fig. 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 in the presence of interference, BlueEar leverages the following key observation. When comparing the observed hopping pattern with the basic hopping sequence 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 [32]. 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 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. When a new packet is captured, BlueEar updates the mismatch ratios for remaining clock candidates. A clock is identified as incorrect if the 95% confidence interval of its mismatch ratio exceeds $\frac{59}{79}$. Specifically, let d_c be the number of mismatches when comparing the observed hopping pattern with the basic hopping sequence at clock c, and n be the total number of overheard packets. If c is the correct clock, the ratio of mismatches $\mu = \frac{d_c}{n}$ should be close to the ratio of unused subchannels. Based on the central limit theory, the distribution of $\sqrt{n}(\frac{d_c}{n} - \mu)$ should approach normality when n is reasonably large. The 95% confidence interval of μ can be estimated as $\frac{d_c}{n} \pm 2\frac{\sigma}{\sqrt{n}}$, where σ^2 is the variance. Therefore, clock c is determined as incorrect if

$$\frac{d_c}{n} - 2\frac{\sigma}{\sqrt{n}} \geq \frac{59}{79}.$$

5.2 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.

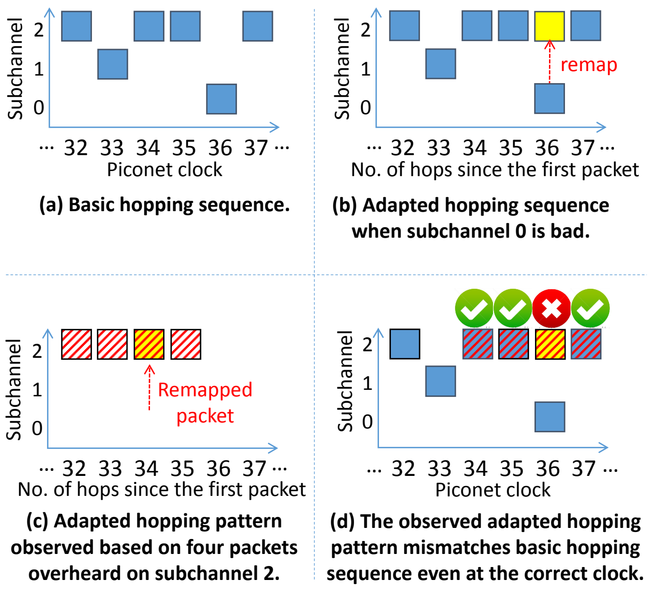

Subchannel
2
1
0
··· 32 33 34 35 36 37 ···
Piconet clock
(a) Basic hopping sequence.
remap
No. of hops since the first packet
(b) Adapted hopping sequence when subchannel 0 is bad.
Remapped packet
(c) Adapted hopping pattern observed based on four packets overheard on subchannel 2.
(d) The observed adapted hopping pattern mismatches basic hopping sequence even at the correct clock.

Figure 3: The effect of subchannel remapping on brute force search based on the same example shown in Fig. 2.

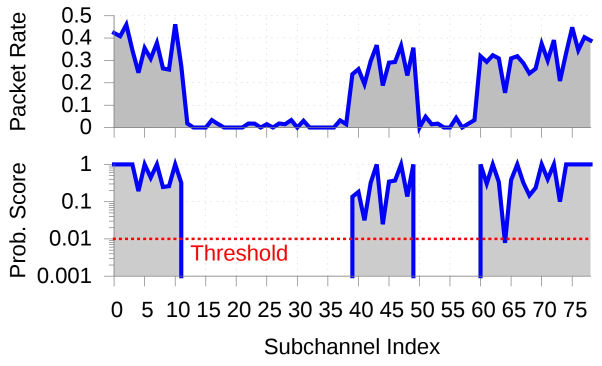
In the following, we present two complementary subchannel classification algorithms, and discuss their advantages and limitations in achieving the above goals. We then discuss how BlueEar integrates the two algorithms for subchannel classification.

5.2.1 *Packet-based Classifier*

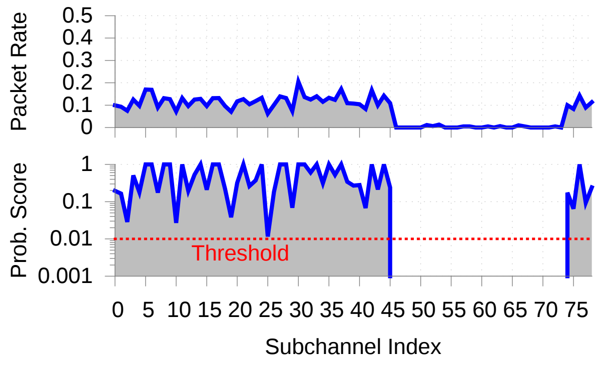
Design. As Bluetooth only transmits on good subchannels, it is possible to infer the status of a subchannel based on its *packet rate*, which indicates how frequently the target transmits on a subchannel. To measure packet rates, the scout hops over all 79 subchannels on the acquired basic channel to monitor the target's packet transmissions. For each subchannel i, BlueEar computes its packet rate as $r_i = \frac{p_i}{v_i}$, where p_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 [32]. 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: 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. (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.

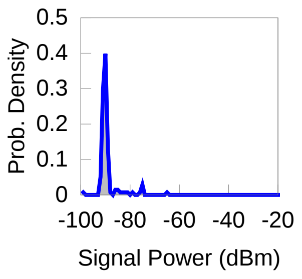
We determine if $r_i = \frac{p_i}{n_i}$ is an outlier of $\mathcal{R}_g$ as follows. Let r_g be the average packet rate of $\mathcal{R}_g$. Assuming subchannel i is good, probability that the target transmits less than p_i packets after v_i visits can be computed as,

$$\rho_i = \sum_{n=0}^{p_i} \binom{v_i}{n} (1 - r_g)^{v_i - n} r_g^n \qquad (1)$$

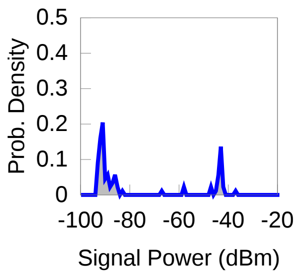
We identify outliers under a given confidence level, denoted as θ. Subchannel i is classified as bad if $\rho_i \leq 1 - \theta$.

Fig. 4 gives two examples of packet-based subchannel classification for data and audio traffics in different spectrum contexts. The upper figure shows the packet rates measured on 79 subchannels. Low packet rates are observed on bad subchannels subjected to interference. The bottom figure shows the probability scores ρ_i for each subchannel i calculated using Eq. (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.



(a) Clean subchannel.



(b) Dirty subchannel.

Figure 5: Probability densities of interference power measured on clean and dirty subchannels.

5.2.2 *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. 5 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. 5(b). 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, achiev-

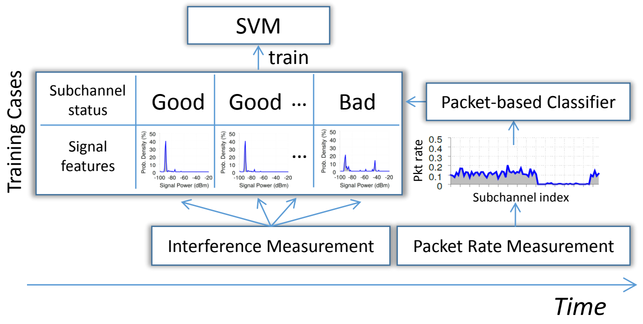


Figure 6: Time-domain illustration of run-time training of the spectrum sensing-based classifier.

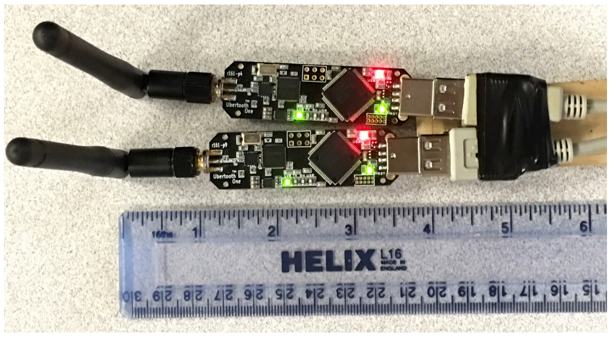


Figure 7: A prototype of BlueEar that consists of two Ubertooths [6].

ing 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.

5.2.3 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 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. 6 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.

5.3 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.

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*, BlueEar only jams a selected subchannel when the target is staying on that subchannel during hopping. *Second*, 802.11 is robust against narrow-band, short-period interference thanks to the design of OFDM and channel coding [8]. Moreover, previous study has shown that wireless devices are sensitive to strategically engineered interference patterns generated using the same communication technology [15]. By incorporating such patterns to generate low-power, narrow-band interference, the impact of selective jamming on 802.11 can be further reduced.

6. IMPLEMENTATION

In this section, we present the implementation of BlueEar in detail. As shown in Fig. 7, we employ two Ubertooths [6] 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.

6.1 Ubertooth-End Implementation

Ubertooth is an open source 2.4 GHz wireless development board that costs around \$80 per unit [6]. 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 $50\mu s$ 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 [18], 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.

6.2 Controller implementation

The controller implements compute intensive tasks, including packet decoding, clock acquisition, subchannel classification, and jamming subchannel selection. In addition, it interacts 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 5.2.1 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 [21]. 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 5.2.2 and Fig. 6), 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%.

7. 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.

7.1 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 WLAN, 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 target master using a script written based on hcitool. The host of BlueEar is connected with the target 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

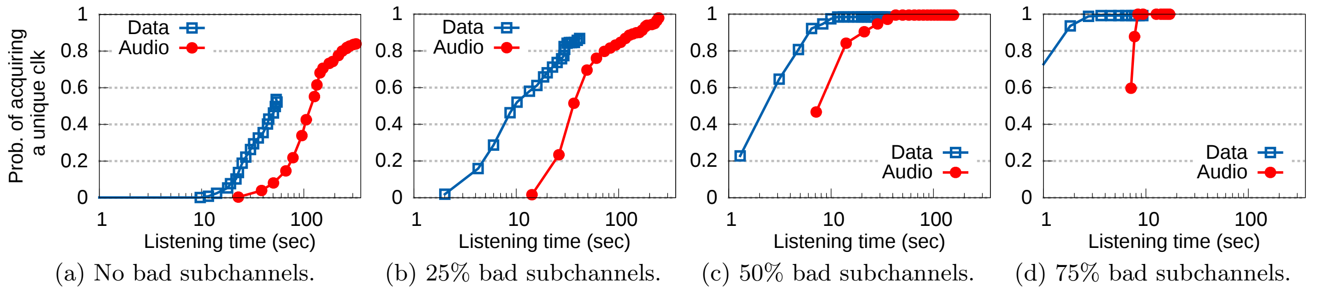


Figure 8: Clock acquisition delay when sniffing data and audio traffics in different spectrum contexts (characterized by the percentage of bad subchannels at the target piconet).

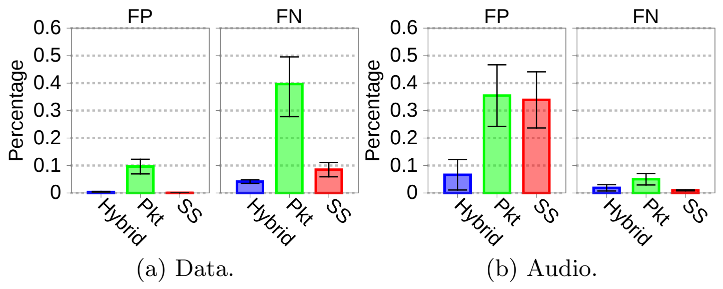


Figure 9: Subchannel classification accuracy in fast varying spectrum context.

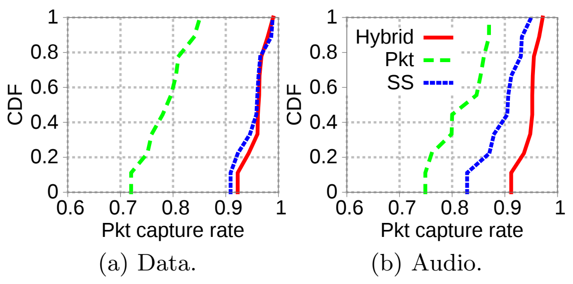


Figure 10: Packet capture rate in fast varying spectrum context.

compare the hybrid subchannel classifier proposed in Section 5.2.3 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 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 [6], which operates in the basic hopping mode, and is oblivious to the adaptive hopping behavior and the impacts of interference.

7.2 Synchronization Delay

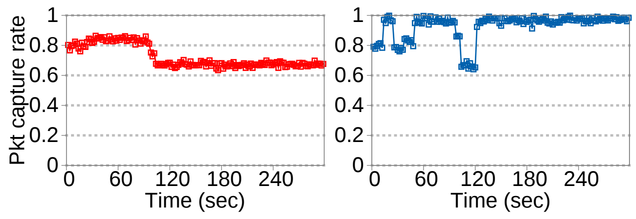
We first evaluate the delay incurred when synchronizing BlueEar with the target piconet. The dominant component of this delay is introduced by clock acquisition, during which the scout listens on a single subchannel until it captures enough packets to reverse the piconet clock. We benchmark clock acquisition delay in different spectrum contexts where the targets 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. 8 shows the probability of successfully determining the piconet clock 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.

7.3 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. 9 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



(a) Selective jam disabled. (b) Selective jam enabled.

Figure 11: The gain of selective jamming in the presence of hidden interference.

performs best among the three classifiers. In particular, the FP and FN rates are lower than 8% in both data and audio tests. Fig. 10 further compares the packet capture rates when BlueEar uses the three classifiers to predict adaptive hopping. Similar with the results shown in Fig. 9, the hybrid classifier is able to maintain the best packet capture rate, which is higher than 90% in both data and audio tests.

7.4 Hidden and Exposed Interference

We now evaluate the packet capture rate of BlueEar in the presence of hidden and exposed interference. Fig. 11 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 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 rate, 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 interference 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. 12 compares the subchannel classification accuracy of the hybrid, the packet-based, and the spectrum sensing-based classifiers. Different from what we observed in Fig. 9, 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. 13.

7.5 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. 14. 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. 15 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.

7.6 Ambient Interference

We further evaluates the performance of BlueEar in an office building under the ambient interference from a large-scale 802.11 based WLAN. Fig. 16(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 [6] 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. 16(b) evaluates the packet capture rate at site D when BlueEar is deployed at different distances to 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.

8. IMPLICATIONS OF BLUEEAR

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

8.1 Implications on BLE privacy

In the following, we briefly introduce the hopping protocol of BLE, highlight the essential differences with the hopping protocol of Bluetooth Classic, and elaborate on the impacts of BlueEar on BLE privacy breach.

The hopping protocol of BLE defines the physical channel, which 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 BLE can be characterized as a set of connection events. At initialization of a connection event, 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 inc, a random value ranges from 5 to 16. The *basic channel* hopping is characterized by $\mathcal{K}(c, inc)$, where $\mathcal{K}(.)$ is the hop selection kernel, c is the index of current subchannel, and inc which is the hop increment. At the first connection event, the first subchannel is defined to be zero [32], and we note that the basic channel sequence repeats itself whenever subchannel zero is visited. On the other hand, BLE defines *adapted channel* hop mode just as Bluetooth Classic. BLE might modify the basic channel to adapt spectrum use in the presence of ambient interference. The adaptive channel is defined by a subchannel map, which classifies the data subchannels into good and bad. If the basic kernel $\mathcal{K}(c, inc)$ 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 [32].

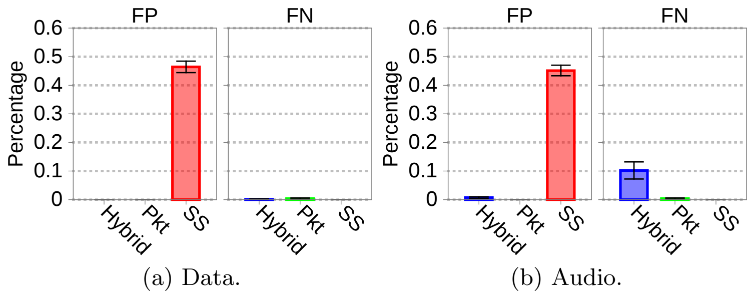


Figure 12: Subchannel classification accuracy under exposed interference.

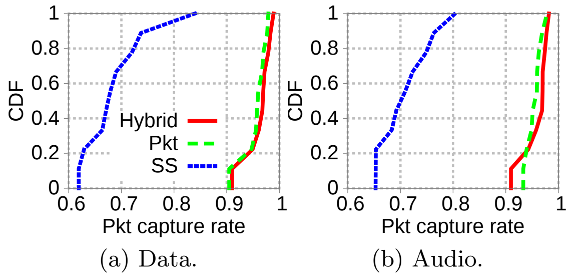


Figure 13: Packet capture rate under exposed interference.

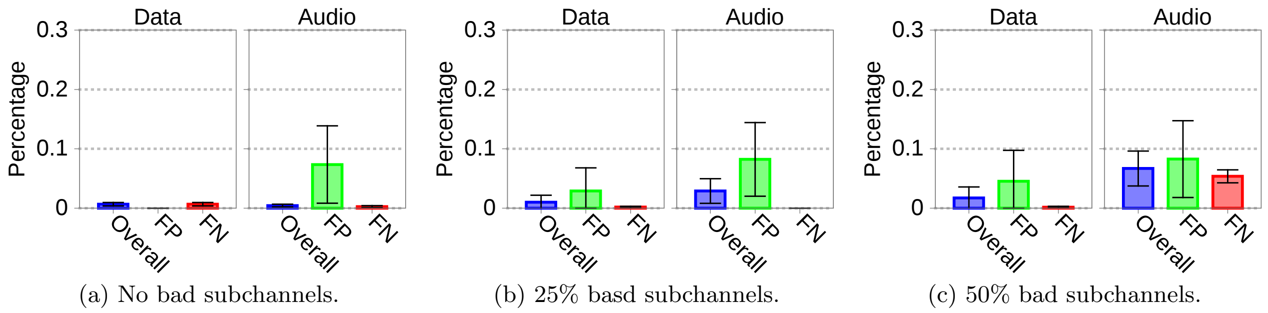


Figure 14: Subchannel classification accuracy in crowded spectrum (characterized by the percentage of bad subchannels at the target piconet).

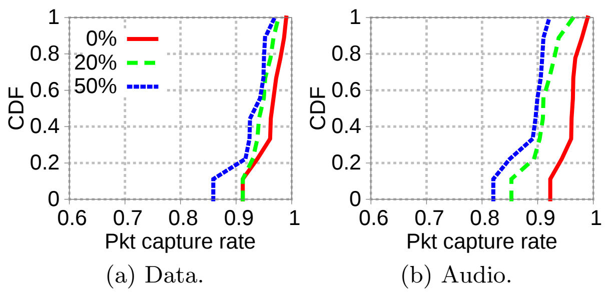


Figure 15: Packet capture rates in crowded spectrum (characterized by the perctange of bad subchannels at the target piconet)

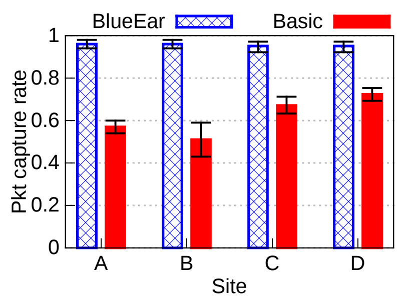
The hopping protocol of BLE is different from the hopping protocol of Bluetooth Classic in the following ways. First, basic channel sequence of BLE is characterized by a random value of the hop increment *inc*. In contrast, basic channel sequence of Bluetooth Classic is characterized by the piconet address. The second difference between BLE and Bluetooth Classic, is hopping phase, which is not defined in BLE hopping protocol. Unlike Bluetooth Classic basic channel sequence, which repeats itself every about 23 hours, BLE basic sequence repeats itself whenever subchannel zero is visited. Due to power constraints, the hopping protocol of BLE is much simpler than that of Bluetooth Classic, making BLE basic sequence easier to compromise.

As BLE becomes pervasive, the privacy leakage of BLE devices has been an increasing concern. Although BlueEar is designed for Bluetooth Classic, it has significant impacts on the privacy leakage of BLE devices and this calls for research to further investigate and enhance the privacy of BLE. 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 BLE 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 BLE.

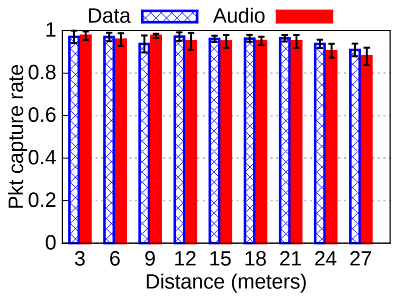
8.2 Impacts on privacy breach

Previous research has shown the possibilities of cracking Bluetooth encryption and compromising user privacy [17] [10] [24] [26] [25] [28] [9]. A prerequisite of these attacks is to passively sniff Bluetooth traffic. Existing attacks [28] [9] 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.

We conduct an experiment to further understand the impacts of BlueEar on privacy leakage. In particular, we study if the packet capture rate of BlueEar can result in successful eavesdropping on speech conversation, which is known to be challenging because audio streams are extremely sensitive to packet loss. This experiment precedes as follows: (1) We collect real traces of packet loss rates. Bluetooth piconet is first established to stream speech. BlueEar is then deployed to sniff the speech stream and we obtain a detailed packet loss trace, which logs the locations of all packets missed by BlueEar. (2) We write a computer script to simulate the



(a) Different locations in a building.



(b) Different distances to the target.

Figure 16: Packet capture rate under the ambient interference at: (a) different locations in an office building under the interference of a large-scale 802.11 based WLAN; (b) different distances to the target.

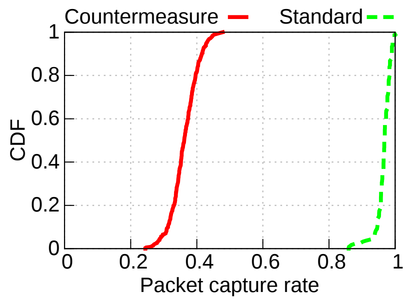
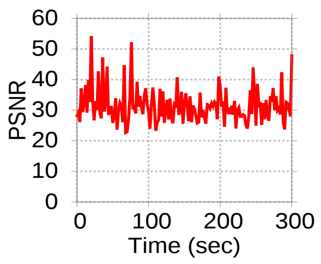
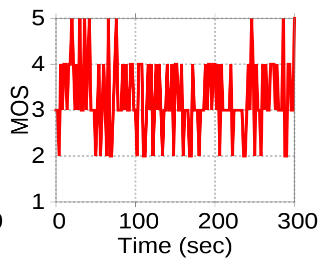


Figure 17: The packet capture rate of BlueEar with and without the countermeasure.

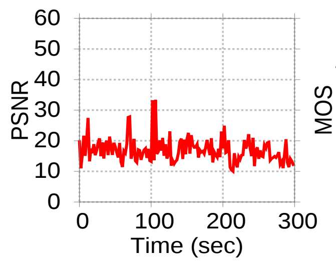


(a) PSNR.

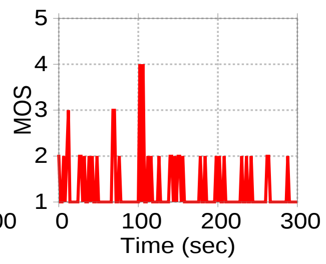


(b) Mean opinion score.

Figure 18: Subchannel classification accuracy in fast varying spectrum context.



(a) PSNR.



(b) Mean opinion score.

Figure 19: Audio quality when the target piconet implements the countermeasure approach.

audio stream without encryption. (3) We then synthesize the real packet loss trace with the simulated audio stream. If a packet is found loss in the real trace, the equivalent packet is dropped from the simulated audio stream. For each dropped packet, the previous packet (in the simulated audio stream) is replayed for loss concealment. (4) We use the peak signal to noise ratio (PSNR) to quantify the quality of the simulated audio stream, which should be equivalent to an eavesdropped audio file by BlueEar. As shown in Fig. 18(a), the PSNR is calculated every 2 seconds time window. (5) Finally, we map the PSNR values to mean opinion scores (MOS), as shown in Fig. 18(b), using the method proposed by [27] [16]. As in Fig. 18, MOS is categorized into five voice qualities, including excellent, good, fair, poor, and bad. We find that BlueEar can maintain a voice quality equal to or higher than fair in 81% of time.

8.3 Practical countermeasure

To counteract sniffing systems like BlueEar, we discuss a practical countermeasure, which can be implemented as a user-space script on the Bluetooth master, and requires no modifications to existing slaves. The key idea is to flip the status of a set of randomly selected good subchannels, thereby interfering the subchannel classifier, making it hard for the sniffer to learn the adaptive hopping sequence. Specifically, to comply with the FCC rule that requires Bluetooth to use at least 20 subchannels for hopping, the countermeasure randomly selects $n-20$ good subchannels to flip, where n is the total number of good subchannels. The subchannel map is updated every 200ms using the user-space script, which is composed using the interface provided by BlueZ –the open source Bluetooth stack. To evaluate the performance of the proposed countermeasure, we conduct the following experiment. The setup of this experiment is similar to the setup of the previous experiment presented in section 8.2 except that the target piconet implements the countermeasure approach.

Fig. 17 and 19 evaluate the effectiveness of the countermeasure. As expected, the average PSNR is about 15 due to high packet loss rate caused by the countermeasure, as in Fig. 19(a). As a result, the lower PSNR rates degrade the quality of the eavesdropped audio to poor in 95% of the time, as shown in Fig. 19(b).

9. CONCLUSION

This paper presents BlueEar, the first Bluetooth packet sniffer that only uses cheap, Bluetooth-compliant radios. Bluetooth has 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%.

10. ACKNOWLEDGMENTS

This work was supported in part by U.S. National Science Foundation under grant CNS1423221. We would like to thank the anonymous reviewers and our shepherd, Jeremy Gummeson, for their insightful feedback.

11. REFERENCES

[1] Android auto. https://www.android.com/auto/.

[2] Bluetooth technology website. https://www.bluetooth.com/.

[3] Busting the bluetooth myth – getting raw access. http://www.remote-exploit.org/content/busting_bluetooth_myth.pdf.

[4] Carplay – apple. http://www.apple.com/ios/carplay/.

[5] Gnuradio. https://gnuradio.org/.

[6] Ubertooth. http://ubertooth.sourceforge.net/.

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

[8] 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.

[9] 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 Internation Workshop on Mobile Computing Systems and Applications*, HotMobile '16. ACM, 2016.

[10] S. R. Fluhrer and C. S. Inc. Improved key recovery of level 1 of the bluetooth encryption system. Cambridge University Press, 2002.

[11] FTE. Frontline test equipments. http://www.fte.com/.

[12] FTE. Fte comprobe bpa 600. http://www.fte.com/products/BPA600.aspx.

[13] FTE. Fte comprobe sodera. http://www.fte.com/products/sodera.aspx.

[14] 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, pages 170–181, New York, NY, USA, 2011. ACM.

[15] R. Gummadi, D. Wetherall, B. Greenstein, and S. Seshan. Understanding and mitigating the impact of rf interference on 802.11 networks. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07. ACM, 2007.

[16] C. heng Ke, C. kuen Shieh, W. shyang Hwang, and A. Ziviani. An evaluation framework for more realistic simulations of mpeg video transmission. *Journal of INFORMATION SCIENCE AND ENGINEERING 24, 425-440 (2008)*, 2008.

[17] M. Hermelin and K. Nyberg. Correlation properties of the bluetooth combiner generator. In *Proceedings of the Second International Conference on Information Security and Cryptology*, ICISC '99. Springer-Verlag.

[18] J. Huang, W. Albazrqaoe, and G. Xing. Blueid: A practical system for bluetooth device identification. In *INFOCOM, 2014 Proceedings IEEE*, 2014.

[19] J. Huang, G. Xing, G. Zhou, and R. Zhou. Beyond co-existence: Exploiting wifi white space for zigbee performance assurance. In *Network Protocols (ICNP), 2010 18th IEEE International Conference on*, pages 305–314, Oct 2010.

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

[21] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.

[22] 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*, pages 309–322. ACM, 2010.

[23] Logitech. Logitec advanced 2.4 ghz technology. http://www.logitech.com/images/pdf/roem/Logitech_Adv_24_Ghz_Whitepaper_BPG2009.pdf.

[24] Y. Lu, W. Meier, and S. Vaudenay. The conditional correlation attack: A practical attack on bluetooth encryption. In *Proceedings of the 25th Annual International Conference on Advances in Cryptology*, CRYPTO'05. Springer-Verlag, 2005.

[25] Y. Lu and S. Vaudenay. Cryptanalysis of bluetooth keystream generator two-level e0. In *in Advances in Cryptology-ASIACRYPT 2004, Lecture Notes in Computer Science*. Springer, 2004.

[26] Y. Lu and S. Vaudenay. Faster correlation attack on bluetooth keystream generator e0. In *Advances on Cryptography - CRYPTO 2004, Lecture Notes in Computer Science*, 2004.

[27] 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*, pages 1–7, June 2014.

[28] 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.

[29] E. Researches. Ettus research. https://www.ettus.com.

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

[31] Y. Shaked and A. Wool. Cracking the bluetooth pin. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, MobiSys '05. ACM, 2005.

[32] B. SIG. Bluetooth core specification v4.0. https://www.bluetooth.org/.

[33] 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.

[34] 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 &#38; Networking*, MobiCom '13, pages 465–476, New York, NY, USA, 2013. ACM.