BlueID: A Practical System for Bluetooth Device Identification

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

***Abstract*—Despite the widespread use of Bluetooth technology, identity management of Bluetooth devices remains a significant challenge because the MAC address and name of Bluetooth device are easy to forge. In this paper, we present *BlueID* – a practical system that identifies Bluetooth devices by fingerprinting their clocks. Previous approaches to clock fingerprinting exclusively rely on the timestamps carried by packet headers, which can be easily spoofed by hacking the user-space device driver. In comparison, BlueID performs clock fingerprinting based on the temporal feature of Bluetooth frequency hopping, which is impossible to forge without a customized baseband. Due to the proprietary nature of chipset firmware that implements baseband on commodity Bluetooth devices, BlueID will significantly raise the bar of identity spoofing. Moreover, BlueID employs simple yet efficient techniques to detect and differentiate low power Bluetooth transmissions from a distance, making it suitable for mobile applications like energy efficient localization and tracking. BlueID is implemented on a low cost wireless development platform and extensively evaluated based on 56 commodity devices. We show that BlueID can detect Bluetooth radios from 100m away, and identify different devices with high accuracy, short delay, and low computational overhead. Although this paper focuses on Bluetooth, the design of BlueID is general and can be applied to other frequency hopping based wireless systems.**

I. Introduction

Bluetooth [5] is the de facto wireless technology for personal area networking of smart devices. It is reported that more than two billion Bluetooth-ready devices were shipped during 2012 – over 50 millions every day [3]. With the widespread use of Bluetooth technology, identity management of Bluetooth devices is becoming increasingly important. First, reliable identification of Bluetooth device is critical to the privacy of Bluetooth users. Recent studies show that Bluetooth users are susceptible to identity spoofing based attacks [1] [2]. For example, by forging the MAC address of Bluetooth radio, a malicious device can pretend to be the Bluetooth headset of a smartphone and thereby trick the victim phone to call back the attacker, allowing for eavesdropping on conversations. Second, device identification is the basic building block for a wide range of mobile applications [12] [13] [17]. For instance, by deploying cheap Bluetooth detectors in a mall, service providers can track users by actively probing and identifying their Bluetooth radios, enabling energy efficient consumer counting and personalized advertising. Similarly, Bluetooth radios can be pre-deployed as localization anchors in a large building, allowing mobile users to locate themselves with low energy cost.

In this paper, we present *BlueID* – a practical system that identifies Bluetooth devices by fingerprinting their clocks. Due to manufacture imperfections, clocks designed to run at the same rate may drift from each other. The difference of real running rates between clocks is defined as *clock skew*. Previous studies show that clock skews can be used to fingerprint PCs [19] and 802.11 access points [11]. However, existing approaches to clock fingerprinting exclusively rely on the timestamps carried by packet headers, which can be easily spoofed by hacking the user-space device driver. In comparison, BlueID performs clock fingerprinting based on the temporal feature of Bluetooth baseband. Specifically, Bluetooth baseband divides the wireless channel into short time slots, and switches the center frequency of communication every slot. Packet transmission is aligned with the start of time slot. Exploiting this feature, BlueID can extract transmitter clock skew by measuring the rate at which received packets drift away from the slot boundary. Due to the proprietary nature of chipset firmware that implements Bluetooth baseband on commodity devices, BlueID will significantly raise the bar of identity spoofing. Although this paper focuses on Bluetooth, the idea of BlueID is general and can be applied to other frequency hopping based wireless systems.

To realize BlueID, several practical challenges must be addressed. First, Bluetooth is designed for short range communications. The difficulty in identifying low power Bluetooth radios from a distance hinders a wide range of applications that track mobile users using Bluetooth. To address this issue, BlueID fingerprints Bluetooth transmitters by timestamping their packet preambles, which can be done at an extremely low signal level, and hence significantly extends the detection range. Second, Bluetooth preamble detections suffer substantial false positives, where noises are wrongly identified as packet preambles. To avoid false detections affecting fingerprinting accuracy, BlueID employs an efficient filtering algorithm, which leverages the knowledge from Bluetooth specification to remove false detections. Third, when many Bluetooth users exist in the same environment, it is challenging to differentiate their packets based on preambles. To separate preambles transmitted by different devices, BlueID employs a powerful classification algorithm [10], and makes it practical by optimizing its computation and storage overhead. Finally, to improve the accuracy of device identification, BlueID employs a novel fingerprinting algorithm, which enables fine-grained characterization of fingerprint accuracy based on the probabilistic distribution of measurement errors.

We have implemented BlueID in BlueZ [4] – an open source Bluetooth stack of Linux. To timestamp Bluetooth preambles, BlueID is interfaced with Ubertooth [6], a low cost Bluetooth development platform, which reports its microcontroller time to BlueID when a preamble is detected. BlueID is extensively evaluated based on 56 commodity devices of different types, including headsets, USB adapters, and built-in Bluetooth radios in laptops, tablets and smartphones. We find that clock skews remain consistent over time on the same Bluetooth device, and vary significantly on different devices. These results validate the use of clock skew as a reliable fingerprint for Bluetooth device identification. Our results show that BlueID can detect Bluetooth radios from 100m away, and identifies different devices with high accuracy, low delay and computational overhead.

The rest of this paper is organized as follows. Section II discusses related work. Section III introduces the background of Bluetooth and gives an overview of BlueID. Section IV and Section V introduce the design and implementation of BlueID in detail. Section VI discusses important issues. Evaluation results are reported in Section VII. Section VIII concludes this paper.

II. Related Work

Previous work shows that traffic or driver level features can be used for fingerprinting users or operating systems. Pang et al. [18] show that most anonymous users can be identified using the pattern of network traffic, such as the websites visited and the distributions of packet lengths. However, traffic based user fingerprinting requires hours of traffic history to achieve satisfactory accuracy. To fingerprint device drivers, Mirza et al. [15] analyze the bit rates of overheard packets to identify 802.11 rate adaptation algorithms. Similarly, the pattern of access point scanning can be exploited to fingerprint the drivers of WLAN client [8] [22]. Unfortunately, driver based approaches cannot differentiate devices that use the same driver. Moreover, since both traffic and driver based fingerprinting rely on signatures extracted from software, they can be easily circumvented by changing the configuration or behavior of device.

Recent studies exploit signal level signatures to fingerprint wireless transmitters. A representative example is PARADIS [7], which fingerprints 802.11 transmitters based on their modulation imperfections. Similarly, Danev et al. [20] uses transient based fingerprinting to identify wireless sensor nodes. Unfortunately, measuring signal level fingerprints requires specialized signal analyzer, which is prohibitively expensive to deploy in large quantities.

Compared with the aforementioned approaches, clock skew based device identification is both robust (i.e., exploiting hardware signature) and cost effective (i.e., requiring no specialized equipment for measurement). Previous approaches to clock fingerprinting exclusively rely on the timestamps of packet headers. For instance, Kohno et al. [14] measure the clock skew

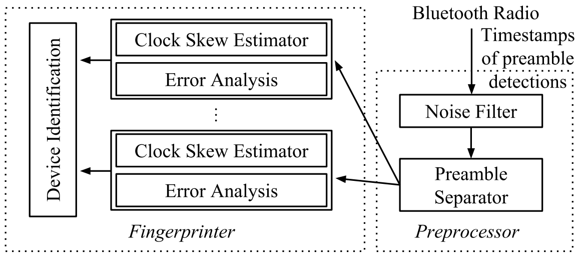


Fig. 1. System architecture of BlueID.

of a remote PC based on the time drifts observed in TCP/ICMP timestamps. Similarly, Jana et al. [11] exploit the timestamp of periodic beacon to measure the clock skew of WLAN access point. Unfortunately, in real-life network traffic, only a few control packets (e.g., TCP/ICMP [14] and beacons of WLAN access points [11]) carry timestamp in their headers. Moreover, packet headers can be easily spoofed by hacking the user-space device driver. Compared with previous approaches, a key novelty of BlueID is to perform clock fingerprinting based on the temporal feature of Bluetooth baseband, which is implemented by manufacturers in chipset firmware due to its time criticality. Due to the proprietary nature of chipset firmware on commodity Bluetooth devices, BlueID will significantly raise the bar of identity spoofing.

Recently, several systems such as AirShark [21] and DOF [9] have been developed to detect and classify wireless devices. The goal is to distinguish wireless radios of different technologies (i.e., WiFi, ZigBee, and Bluetooth), thereby improving coexistence by efficiently managing cross-standard interference. In comparison, the goal of BlueID is to fingerprint low power Bluetooth radios. BlueID employs simple yet efficient techniques to address practical challenges, including detecting and separating low power Bluetooth transmissions and filtering false positive detections.

III. Background and System Overview

In this section, we introduce the background of Bluetooth, and then give an overview on the design of BlueID.

A. Bluetooth Background

Bluetooth [5] is a wireless standard developed for short range communications. Bluetooth adopts *frequence hopping spread spectrum* (FHSS) at the baseband. In Bluetooth FHSS, wireless channel is divided into time *slots* of $625\mu s$. The center frequency of communication is switched every time slot over 79 channels in the unlicensed 2.4 GHz band. Frequency hopping sequence is defined by a pseudorandom number generator known to both transmitter and receiver. Packet transmission is aligned with the start of time slot.

Bluetooth LAN, also known as *piconet*, is formed by devices that follow the same frequency hopping sequence. Devices in a piconet are synchronized to the same clock. The device that defines this clock is called *master*. All other devices are *slaves*. As defined in Bluetooth specification [5], the clock skew of commodity Bluetooth device must be within ± 20 $parts\ per\ million$ (ppm). To compensate for the time drift caused by clock skew, the baseband controller of slave calibrates its clock using master packets periodically.

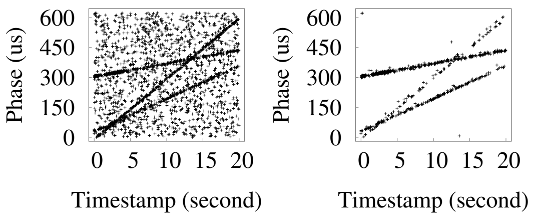
B. BlueID Overview

BlueID fingerprints the clock of Bluetooth device based on the temporal feature of Bluetooth frequency hopping. In a piconet, all packet transmissions are scheduled by the master device based on its own clock. Ideally, the packet interval should be $625k\mu s$, where k is a positive integer. However, due to the effect of clock skew, the packet intervals observed by other Bluetooth devices are always drifted from $625k\mu s$. By measuring the rate of such drift, BlueID can estimate the clock skew of piconet master, extracting the device fingerprint. However, fingerprinting slave devices is more challenging because slaves continuously calibrate their clocks to keep synchronization once paired with the master. Although the current design of BlueID does not incorporate slave fingerprinting, it will not affect the utility of BlueID. We will discuss the implications of BlueID for master and slave modes in Section VI-A.

To perform clock fingerprinting, BlueID employs a Bluetooth radio for traffic sniffing, and then extracts clock skews to identify the master devices of nearby piconets. BlueID is not required to pair with or hop along with the target devices. Instead, BlueID listens on one of the 79 Bluetooth channels, and then sniffs packets when other devices switches to the monitored channel during frequency hopping.

Fig. 1 shows the system architecture of BlueID. BlueID is composed of two major components, including *preprocessor* and *fingerprinter*. During the preprocessing stage, BlueID timestamps overheard Bluetooth transmissions by logging the time when a packet preamble is detected. Because preamble detection can be done at an extremely low signal level, timestamping preambles rather than the overheard packets can significantly extend the detection range of BlueID. However, such design also raises two challenges for accurate fingerprinting. First, Bluetooth preamble detection yields high false positive rate, as noises are often wrongly identified as preambles. Second, when many Bluetooth users exist in the same environment, it is difficult to differentiate their packets based on preambles. To address these issues, BlueID employs a simple yet efficient *noise filter* to remove the timestamps of false positive detections. The filtered detections are then processed by the *preamble separator* to separate preambles of different piconets.

BlueID employs an accuracy-aware fingerprinter, which estimates the clock skew for each piconet master, estimates the fingerprinting error, and then identifies the master devices. First, the *clock skew estimator* derives master clock skews based on the preprocessed clock samples. For fine-grained characterization of fingerprint accuracy, BlueID employs the *error analyzer* to estimate the distribution of measurement errors. Based on the results, the *device identification* component calculates the difference between the estimated clock skew and a fingerprinting reference to determine whether the piconet master matches a given device. The fingerprinting reference can be profiled if BlueID was paired with the device before (e.g., for the application to authenticate a known device), or can be estimated when the device is first detected (e.g., for tracking an unknown Bluetooth user).



(a) Before noise filtering. (b) After noise filtering.

Fig. 2. Effect of noise filtering on Bluetooth preamble detections.

IV. Design of BlueID

In this section, we introduce the design of BlueID in detail.

A. Filtering False Positive Preamble Detections

During preamble detections, BlueID may falsely recognize noises as preambles, causing false positive detections. Fig. 2(a) illustrates the effect of noise on preamble detections. Each point in the figure represents a detected preamble. The *phase* of a preamble detected at time t is defined as the offset of t to the Bluetooth time slot, calculated as $p = t \bmod 625\ \mu s$, where t is measured using the clock of BlueID. For Bluetooth devices synchronized to the same master clock, the phases of their preambles will be linearly increasing or decreasing with time, depending on the clock skew between BlueID and the master. As shown in the figure, three Bluetooth links are observable during our measurements. However, the detection is heavily disturbed by significant noise.

To address this problem, BlueID leverages the prior knowledge of Bluetooth specification to filter out false positive detections. The basic idea is as follows. Suppose two preambles are detected at time t_i and t_j, respectively. If both preambles are transmitted by devices synchronized to the same master clock, then their phase difference will be upper-bounded by $40\times|t_i-t_j|\ \mu s$. This is due to the *clock skew constraint* imposed by Bluetooth specification [5], which requires all commercial off-the-shelf Bluetooth devices to limit their clock skews within ± 20 ppm. Due to the burstiness of application traffic, true preambles will be detected with small phase differences within a short time window. While for false preambles, their phases are randomly distributed within $[0, 625)\mu s$. Consequently, given a false preamble detected at t, the probability of observing a number of preambles with small phase differences to t is small.

Based on this observation, we propose a simple yet efficient noise removal algorithm. Specifically, the detector buffers the timestamps of detected preambles in a moving time window of one second. Let t_0 be the smallest timestamp in the window, and n be the number of timestamps whose phase differences to t_0 are within $\pm 20\ \mu s$. The detector removes t_0 if n is less than a pre-defined threshold N.

We now discuss how to choose the threshold N. Assuming a false preamble is detected at time t, we use $\mathbf{d}_t$ to denote the set of preamble detections within the one second time window of $(t, t+1]$. Let n be the number of false detections in $\mathbf{d}_t$ whose

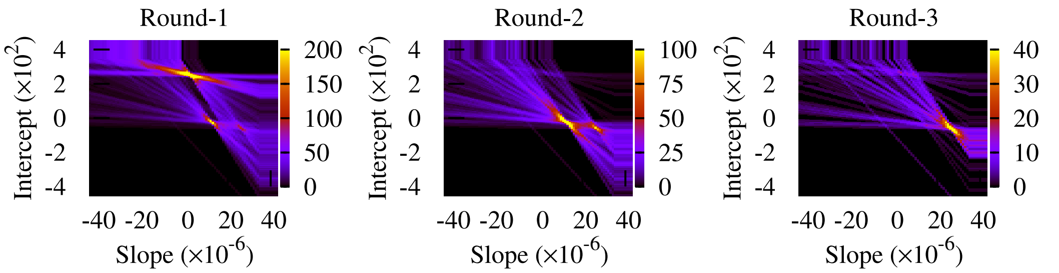


Fig. 3. The results of iteratively applying GHT on the filtered preamble detections shown in Fig. 2(b). Each bin in the figure represents a line featured by the intercept and slope. Bin brightness indicates the number of votes, i.e., number of preamble detections falling into the bin. In each round of GHT, the bin of most votes are separated out and removed from the data set. The separated points are shown in Fig. 4.

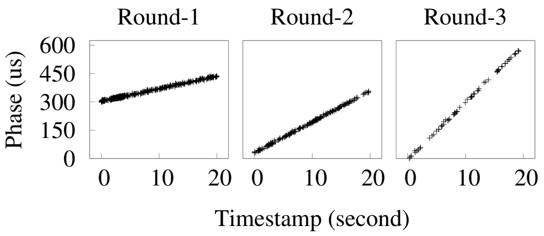


Fig. 4. Preamble detections separated based on the GHT shown in Fig. 3.

phases satisfy the clock skew constraint. n follows the Binomial distribution, i.e., $n \sim \mathcal{B}(r, p \times q)$, where r is the Bluetooth bit rate measured in *bits per second*, and $p \times q$ computes the probability that a false detection satisfies the clock skew constraint. Specifically, p is the probability that a sequence of noise bits is identified as a preamble, which can be computed as $\frac{1}{2^k}$, where $k = 32$ is the number of bits in a Bluetooth preamble. q is the probability that the phase of false detection to t is within $\pm 20\mu s$. Since the phase of false detection is randomly distributed between 0 to 625 μs, q can be computed as $40/625 = 0.064$. To remove P percent of false preamble detections, we can set the threshold N to the P-th percentile of the Binomial distribution, such that $\Pr(n \leq N) = P$. In BlueID, we adopt a threshold of $N = 7$. Theoretically, this removes 99.7% false preambles. Fig. 2(b) shows the remaining timestamps after noise filtering.

B. Separating Preambles of Different Piconets

The purpose of preamble separation is to differentiate the preambles transmitted by devices of different piconets. This is challenging because Bluetooth preamble contains no identity information, making it difficult to achieve this goal. In a similar context, previous work [11] adopts simple heuristic to separate 802.11 beacons transmitted by authorized and fake access points. Separation is achieved based on the different rates of increment observed in beacon timestamps, which works efficiently when there are very few active access points (i.e., two or three). In BlueID, we target at a scenario where significantly more transmitters may be active at the same time.

To separate preambles, we propose an algorithm based on the Generalized Hough Tranform (GHT) [10], which is a powerful technique used for detecting lines in a data set. Specifically, points in a data set are divided into bins in a two dimensional space defined by the intercept and slope of the line. Line detection is achieved through a voting procedure, which counts the number of points fallen into each bin. Bins of high densities imply the existence of lines.

Leveraging GHT in line detections, BlueID separates detected preambles as follows. Given a detected preamble i, we characterize it using $< t_i, p_i >$, where t_i is the time when i is detected, and p_i is the phase of t_i. As shown in Fig. 2(b), preambles transmitted by different piconets are clustered on straight lines in the two dimensional space defined by time and phase, where line slope describes the clock skew, and the intercept the initial phase. As a result, we can apply GHT on the filtered data to classify piconet transmissions. To determine whether a line exists or not, a conventional approach is to apply a threshold on the votes received by the corresponding bin. The optimal threshold depends on the number of points in the data set [10]. Unfortunately, in our case of separating detected preambles, it is impossible to find an oracle threshold working consistently well over time. This is because the number of detected preambles varies significantly as Bluetooth links are dynamically formed and disconnected. To this end, we employ a threshold-free iterative approach. Specifically, BlueID first performs GHT on a given set of points. Let $\mathbf{d}$ be the set of points located in the bin of highest votes. The separator first reports $\mathbf{d}$ to the fingerprinting component, and then removes it from the data set. This procedure repeats until all points are removed. In BlueID, we use a bin size of $1ppm \times 5\mu s$ for GHT, which is sufficient to separate preambles when more than ten piconets coexist in the same environment (see Section VII). Fig. 3 gives an example after iteratively applying GHT on the data shown in Fig. 2(b). The separated preamble detections are plotted in Fig. 4.

GHT is known to be computationally intensive and requires a large amount of storage. BlueID addresses these issues by optimizing GHT using the knowledge of Bluetooth specification. Specifically, since the maximum clock skew between two commodity Bluetooth devices is 40 ppm, slopes of lines are within the range of $[-0.00004, +0.00004]$. Moreover, as the length of Bluetooth time slot is $625\mu s$, intercepts of lines are limited within $\pm 625\mu s$. Accordingly, we reduce the search space of GHT to avoid unnecessary computation and storage overhead.

C. Clock Skew Estimation

Although the algorithm described above can separate detected preambles based on their initial phases and clock skews, it cannot measure transmitter clock skew in a fine granularity. We now discuss how to extract clock skew after preamble separation. The clock skew between two clocks is defined as the difference between their actual running rates. Let $C_x(t)$ be the amount of time reported by clock x for a groundtruth time period t. The clock skew between clocks x and y can be computed as $\phi_{x,y} = C_x(t)/C_y(t)$, where $\phi_{x,y}$ is typically expressed in *microseconds per second* ($\mu s/s$), or *parts per million (ppm)*.

Let t be the time when a packet transmission is scheduled on a Bluetooth device, and t' the time when BlueID detects its preamble. We have $t' = t + d$, where d is a random delay that includes the signal propagation delay and the delay incurred in the transmitter and receiver hardwares. Given a pair of packets, their time interval measured at BlueID is $\Delta t' = \Delta t + \Delta d$, where Δd is a random jitter with mean zero. Let x and y be the clocks of the piconet master and BlueID. Their clock skew can be expressed as $\phi_{x,y} = \frac{C_x(\Delta t)}{C_y(\Delta t')} + \frac{C_x(\Delta d)}{C_y(\Delta t')}$. Given n packet pairs, the sum of random jitter Δd approaches zero when n goes large. As a result, we can estimate the clock skew $\phi'_{x,y}$ and the corresponding estimation error e as follows,

$$\phi'_{x,y} = \frac{1}{n} \sum_{1 \le i \le n} \frac{C_x(\Delta t_i)}{C_y(\Delta t'_i)}, \quad e = \frac{1}{n} \sum_{1 \le i \le n} \frac{C_x(\Delta d_i)}{C_y(\Delta t'_i)} \tag{1}$$

where $\sum C_y(\Delta t'_i)$ can be obtained using the clock of BlueID.

To estimate the clock skew using Eq. (1), a key problem is how to reverse $C_x(\Delta t_i)$, the clock value of piconet master. BlueID achieves this goal by exploiting the temporal feature of Bluetooth FHSS. Because Bluetooth transmission always begins at the start of time slot, we have,

$$C_x(\Delta t_i) = k \times 625\mu s, \tag{2}$$

where k is a positive integer. Let z be the groundtruth clock, i.e., $C_z(t) = t$, k can be derived as follows,

$$k_i = \frac{C_x(\Delta t'_i) - C_x(\Delta d_i)}{625\mu s} = \frac{(1 - \phi_{x,y}) C_y(\Delta t'_i) - (1 - \phi_{x,z})\Delta d_i}{625\mu s}, \tag{3}$$

where $\phi_{x,z}$ is the skew between clock x and the groundtruth clock z. As defined in Bluetooth specification [5], the random delay d_i must be smaller than $20\mu s$. Thus we have $|\Delta d| \le 20\mu s$. Moreover, based on the clock skew constraint, we have $|\phi_{x,z}| \le 20ppm$ and $|\phi_{x,y}| \le 40ppm$. Putting them together, we can compute the range of k using the BlueID clock,

$$k_i \in \left[\frac{0.99996 C_y(\Delta t'_i)}{625\mu s} - 0.032, \frac{1.00004 C_y(\Delta t'_i)}{625\mu s} + 0.032 \right], \tag{4}$$

When $C_y(\Delta t'_i) \le 7.3s$, k can be determined because there is at most one interger in the region defined in Eq. (4).

Therefore, for the i-th pair of detected preambles whose interval is shorter than $7.3s$, we can reverse the clock value of piconet master for this interval by finding the integer k_i in Eq. (4), and then take it into Eq. (2) to compute $C_x(\Delta t_i)$. Finally, we can use Eq. (1) to estimate clock skew, where $C_y(\Delta t'_i)$ can be obtained by reading the clock value of BlueID.

D. Analysis of Estimation Error

The accuracy of clock skew estimation could be affected by several error sources, such as the random delays occurred inside the hardwares of transmitter and BlueID, the small time deviations of slave packet transmissions induced by imperfect synchronization, as well as the false positive preamble detections that accidentally survive noise filtering. Although estimation accuracy could be improved using long time measurement, this is often impractical due to the prohibitive delay incurred. Therefore it is important for BlueID to analyze the accuracy of clock skew estimation, and then account for its impact in device identification.

In the following, we introduce the fine-grained characterization of clock skew estimation error. Specifically, we prove the Lyapunov central limit theorem (CLT) for the estimation error e given in Eq. (1). Therefore, the range of e can be estimated using normal distribution.

Given n pairs of detected preambles, we define the random variable $X = C_x(\Delta d)$, and $Y_i = X/\alpha_i$, where $\alpha_i = C_y(\Delta t'_i) \gg 1\mu s$. Let $\sigma_n^2 = \sum \mathbb{E}(Y_i^2)$. The order-4 Lyapunov's condition can be proved as follows,

$$\lim_{n \to \infty} \frac{1}{\sigma_n^4} \sum_{1 \le i \le n} \mathbb{E}(Y_i^4) = \lim_{n \to \infty} A \times \frac{\sum_{1 \le i \le n} \alpha_i^4}{(\sum_{1 \le i \le n} \alpha_i^2)^2} = 0, \tag{5}$$

where $A = \frac{\mathbb{E}(X^4)}{\mathbb{E}^2(X^2)}$ is a constant. Therefore, we have $\frac{1}{\sigma_n} \sum Y_i \sim N(0,1)$. As a result, the clock skew estimation error e follows normal distribution, $e \sim N(0, \frac{\sigma_n^2}{n})$, where σ_n can be estimated as follows,

$$\sigma_n^2 = \sum_{i=1}^{n} \left(1 - \frac{625 k_i}{C_y(\Delta t'_i)} - \phi'_{x,y} \right)^2, \tag{6}$$

where $C_y(\Delta t'_i)$ can be obtained using the BlueID clock, k_i is the only integer in Eq. (4), and $\phi'_{x,y}$ is estimated in Eq. (1).

E. Device identification

Based on the results derived in Section IV-C and Section IV-D, the device identification component of BlueID calculates the difference between the estimated clock skew and a given fingerprint reference. We represent the fingerprint as $\lambda =< \phi, s^2 >$, where ϕ is the estimated clock skew, and $s^2 = \sigma_n^2/n$ can be computed using Eq. (6). n is the number of preambles used for clock skew estimation. Given two fingerprints, denoted as $\lambda_x =< \phi_x, s_x^2 >$ and $\lambda_y =< \phi_y, s_y^2 >$, BlueID derives their difference using the Bhattacharyya distance, which is a common measure for the similarity of two distributions. Specifically, it is computed as,

$$\mathcal{D}_B(\lambda_x, \lambda_y) = \frac{1}{4} \ln \left(\frac{1}{4} (\frac{s_x^2}{s_y^2} + \frac{s_y^2}{s_x^2} + 2) \right) + \frac{1}{4} \left(\frac{(\phi_x - \phi_y)^2}{s_x^2 + s_y^2} \right). \tag{7}$$

In the application of BlueID, we can impose a threshold on Bhattacharyya distance for device identification. If the clock skews between two devices is greater than the threshold, then they are identified as different devices. We will evaluate the impacts of threshold selection on the accuracy of device identification in Section VII-C.

V. BlueID Implementation

We have implemented BlueID with 619 lines of C code in BlueZ [4] – an open source user-space Bluetooth driver of Linux. Preamble filtering and separation are performed based on a time window of one second. To separate preambles of different piconets, the bin size of GHT is set to $1ppm \times 5\mu s$.

In order to timestamp the preambles of Bluetooth packets, BlueID is interfaced to Ubertooth [6] – an open source 2.4 GHz wireless development platform that costs about $90. Ubertooth is mainly composed of a LPC1700 microcontroller and a low-power CC2400 wireless transceiver, which is compliant with the radio layer of latest Bluetooth specification [5]. To timestamp preambles, we employ the sniffing tool provided by Ubertooth firmware. The sniffer records the value of a 10 MHz clock whenever a Bluetooth preamble is received, and then reports the clock value to a laptop running BlueID through a high speed USB port.

For efficient preamble detection, the listening channel of Ubertooth is carefully configured by BlueID to address the cross-standard interference of coexisted devices. For example, when WiFi transmitters are active in the same environment, *Adaptive Frequency Hopping* (AFH) enabled Bluetooth piconets will avoid the channels used by WiFi during frequency hopping. In this case, listening on those channels will decrease the success rate of preamble detection, leading to longer fingerprinting delay. In real-life WLANs, most access points use one of three orthogonal 802.11 channels, including channel 1, 6, and 11, which overlap with 60 out of 79 Bluetooth channels. Based on this observation, BlueID initially configures Ubertooth to use one of the 19 channels that are not interfered by WiFi. When strong un-decodable signal is detected, BlueID quickly switches the listening channel until a clear channel is found.

VI. Discussion

In this section, we discuss important issues related to the design, application, and deployment of BlueID.

A. Security Implications for Master and Slave Modes

The ability of BlueID to identify the piconet masters based on their clock skews may raise security concerns. In fact, knowing the identity of a target device is the prerequisite of many Bluetooth based attacks [1]. A common practice to secure Bluetooth is to put the device into *invisible* mode, where the device identity is never exposed to untrusted parties. During packet transmission, encryption is often enabled to conceal the device MAC address from eavesdropper. However, fingerprinting systems like BlueID could compromise such design. In particular, BlueID is able to extract device fingerprints by simply timestamping their packet preambles, even without decoding their packets.

To address such security concerns, a simple yet efficient approach is to put privacy critical Bluetooth devices into slave mode during communication, which will effectively avoid the issue of clock exposure. Even if the clock skew is known to an attacker, the security risk is controllable at the authentication stage. Specifically, to enable BlueID-assisted authentication, Bluetooth devices should insist on using slave mode when receiving a connection request, allowing BlueID to fingerprint and authenticate the requesting device when negotiating the connection.

B. Fabrication of Clock Skews

We now discuss the feasibility of fabricating clock skews using an unauthorized Bluetooth device through driver modification. As we discussed in previous section, compared with existing clock fingerprinting approaches [14] [11], a key advantage of BlueID is that it extracts clock skews from Bluetooth baseband, which must be implemented in the chipset firmware due to the time criticality of FHSS. In Bluetooth, the device driver cannot control the time when a packet is actually transmitted. This is because the baseband controller will forcibly align the packet on the start of a time slot defined by the chipset clock. Although the baseband controller of slave device will calibrate its clock to synchronize itself with master, the master-slave connection cannot be established without authentication. As a result, clock skew fabrication is impossible for unauthorized devices without customizing the chipset firmware. Because of the proprietary nature of chipset firmware, BlueID will significantly lift the bar of Bluetooth identity spoofing.

C. Deployment on Commodity Device

Major operations of BlueID, including preprocessing and fingerprinting, can be implemented in user-space device driver. To deploy BlueID on commodity Bluetooth device, the only modification needed in chipset firmware is to have the chipset report its clock value when detecting a Bluetooth preamble. As we discussed earlier, deploying BlueID on commodity devices will enable a wide range of mobile applications for low energy localization and user tracking. Moreover, BlueID will significantly benefits existing authentication protocols. Meanwhile, the security risk of BlueID is easily controllable.

VII. Evaluation

In this section, we evaluate BlueID based on a set of 56 commodity Bluetooth devices, including two wireless headsets, 37 USB adapters, and built-in radios in four laptops, eight smartphones, and five tablets. Among these devices, 32 USB adapters are of the identical model. Devices of the same model are typically more challenging to differentiate because of the similarity in the characteristics of their hardware clocks. Our evaluation focuses on three aspects of BlueID performance, including: (1) How effectively does BlueID detect and separate Bluetooth preambles? (2) What is the overhead of BlueID? and (3) How efficient is clock fingerprinting for Bluetooth device identification?

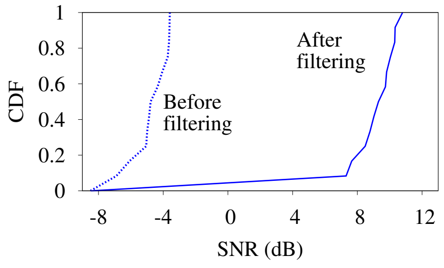


Fig. 5. Amount of false preamble detections before and after noise filtering. Noise is quantified using signal-to-noise ratio (SNR), which is the ratio between the numbers of true and false preamble detections calculated in dB.

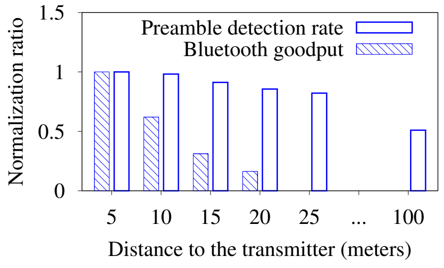


Fig. 6. Comparison of BlueID detection range and Bluetooth communication range. The preamble detection rate and Bluetooth goodput are normalized to the results measured at 5m.

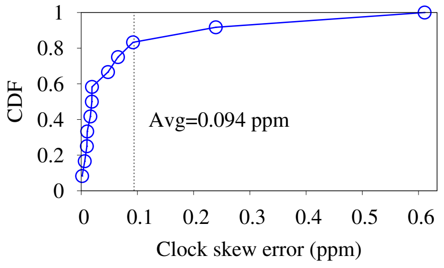


Fig. 7. Error of clock skew measurement when multiple piconexts coexist in the same environment.

A. *Preamble Detection and Separation*

Filtering false preamble detections. We first evaluate how efficiently BlueID removes false preamble detections. Our experiments are conducted on 30 piconets composed of different Bluetooth devices. In each piconet, we deploy a Ubertooth node to detect preambles, and then apply the filtering algorithm described in Section IV-A to remove false preamble detections. To obtain the ground truth of whether a detected preamble is noise, we use Ubertooth [6] to decode the received signal, and then extract the sender MAC address. The amount of noise is quantified using signal-to-noise ratio (SNR), which is calculated as $10 \log_{10} \frac{n_T}{n_F}$, where n_T and n_F are the numbers of true and false preamble detections, respectively. Fig. 5 compares the SNRs before and after noise filtering. As shown in the figure, the SNR is significantly improved after filtering. Specifically, the average SNRs before and after filtering are -4.9dB and 7.9dB, respectively. Filtering boosts SNR by 12.8 dB.

Although false positive preamble detections cannot be completely removed by the filter component, its impact on the performance of device identification is limited. First, BlueID employs a line detection based algorithm to separate preambles (see Section IV-B). Noise preambles not clustered in the line will be ignored after separation. Second, BlueID employs an accuracy-aware fingerprinting scheme, which characterize measurement errors, and accounts for their impacts in device identification.

Detection range of BlueID. To measure the clock skew of a transmitter, BlueID sniffs and timestamps packet preambles, which can be done at an extremely low signal level. A key motivation of this design is to extend the detection range of BlueID, which is particularly crucial when identifying low-power Bluetooth radios.

Fig. 6 compares the detection range of BlueID with the communication range of a commodity Bluetooth device. Our experiment is conducted in an outdoor soccer field, where a Lenovo T430S laptop is deployed to transmit a large file over Bluetooth. The receiving sensitivity of Ubertooth is comparable to Class-1 Bluetooth radios. For a fair comparison, we compare the detection range of Ubertooth to a LinkSys Bluetooth adapter, which is a Class-1 device equipped with an external antenna. The preamble detection rate is defined as the number of detected preambles per second after filtering. In Fig. 6, both preamble detection rate and Bluetooth goodput are normalized to the results measured at 5m. We observe that Bluetooth goodput drops to zero when the LinkSys adapter is about 25m away from the latptop. In comparison, BlueID reliably detects Bluetooth preambles when it is 100m away from the laptop. This result clearly demonstrates the advantage of preamble-based detection adopted by BlueID. The long detection range can also lead to practical Bluetooth-based localization and tracking applications.

Separating preambles of different piconets. Next, we evaluate how accurately BlueID separates the preambles of different piconets. We construct a synthetic data set by mixing the preambles detected in 12 piconets, and then apply the algorithm introduced in Section IV-B for preamble separation. We observe that the average error rate of separation is only 4.3%. We further study the impact of separation error on clock skew measurement. We first measure the clock skew for each piconet master before mixing the detected preambles. The results are then used as the ground truth to evaluate the clock skews measured in the synthetic data set. Fig. 7 plots the distribution of measurement error. As shown in the figure, the average error is only 0.094ppm.

B. *Overhead of BlueID*

We now evaluate the overhead of BlueID, including the measurement delay and the computational overhead.

Measurement delay. We first study the measurement delay of BlueID, i.e., the time needed by BlueID to achieve accurate clock skew measurement. We characterize measurement accuracy using the length of 95% confidence interval. Clearly, measurement delay depends on the packet rate of Bluetooth application. High packet rate typically yields short measurement delay. Our experiments are conducted based on two representative types of Bluetooth traffic, including TCP-based data traffic and UDP-based voice traffic.

Fig. 8 shows the measurement time needed to achieve different levels of clock skew accuracy. To limit the measurement error within 0.1ppm, BlueID needs only 21 seconds of data traffic monitoring, and 65 seconds for voice traffic.

Such delay can be further reduced. Current design of BlueID uses a single detector to monitor one of 79 Bluetooth channels,

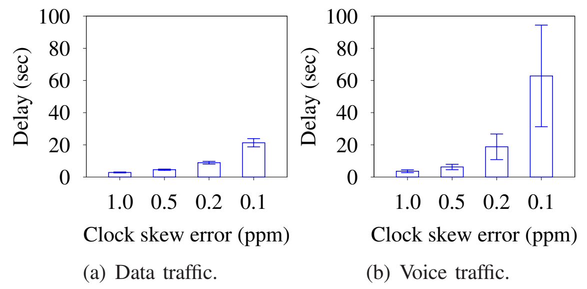


Fig. 8. Delay needed to achieve different levels of measurement accuracy.

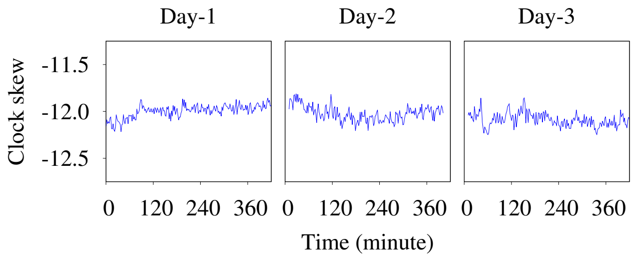


Fig. 9. Temporal variation of clock skew on a Bluetooth USB adapter.

thus will only capture $\frac{1}{79}$ Bluetooth traffic. A simple way to address this issue is to use several BlueID detectors to monitor multiple channels at the same time, which will reduce the measurement delay by several times. Moreover, in the application of authenticating a device, BlueID will be synchronized with the target device, capturing all packets. This will reduce the delay by 79 times.

Computational cost. We evaluate the computational cost of BlueID by measuring its CPU usage. During our evaluation, BlueID is loaded with a one-hour synthetic data set constructed by mixing the preambles detected in 12 piconets. BlueID performs noise filtering every second, and then separates the filtered preambles iteratively. We use `emstat` to measure the average CPU usage every minute. The experiment runs on a 2.3GHz Intel Core i5 laptop. We observe that the CPU usage consumed by BlueID ranges from 1.2% to 10.3%. The average CPU usage is only 2.4%.

C. Clock Fingerprinting based Device Identification

In the following, we evaluate the efficiency of clock fingerprinting for Bluetooth device identification. Our aim is to validate the use of clock skew as a Bluetooth device fingerprint, and evaluate the performance of BlueID in device identification. Our experiments answer the following key questions, including: (1) How consistent is the clock skew on individual Bluetooth devices? (2) How different are clock skews across different devices? 3) What is the impact of temperature on the clock of Bluetooth devices? and (4) How accurately BlueID identifies Bluetooth devices through clock fingerprinting?

Variations of clock skews on individual devices. We first study how clock skews vary on individual devices over time. Fig. 9 plots the trace of clock skews measured on a Broadcom chipset-based USB adapter in three days. In each day, we measure the clock skew every two minutes for about six to seven hours. We find that the clock skew varies between -11.79ppm and -12.25ppm during our measurement. The average clock skews are -12.00ppm, -12.03ppm, and -12.05ppm, respectively.

We further conduct a large-scale measurement to study the clock skew variations in our device set. Fig. 10 shows the distribution of clock skew variations within one hour. We observe that the variation is within 0.5ppm for all devices. The average variation is only 0.205ppm. As we will show later, such variations will not affect the performance of device identification.

Difference of clock skews across devices. Next, to further validate the consistency of clock skew on Bluetooth devices, we compare the clock skew variations observed on individual devices with that across devices. Clearly, whether two Bluetooth devices can be differentiated based on clock skews depends on the accuracy of clock skew measurments. Although collecting more preamble detections will improve measurement accuracy, it will incur larger delay. To this end, we evaluate clock skew difference based on our device set using different numbers of preamble detections. Clock skew difference is quantified using Bhattacharyya distance, which is computed in Eq. (7). *Cross-distance* is computed between clock skews of different devices. For comparison, *self-distance* is the distance between clock skews measured on the same device at different times.

The results are shown in Fig. 11. We find that, for clock skews of different devices, significant distance is observable even using a small number of samples. Specifically, when using 50 preamble detections to measure clock skew, the average self-distance and cross-distance are 2.8 and 1960.5, respectively. The observed difference increases with the number of preamble detections. For example, when using 300 preamble detections, self-distance and cross-distance are 18.5 and 27903.2, respectively. Cross-distance is four orders higher than self-distance.

Impact of temperature. Previous study [16] shows that clock skews of PCs may experience small variations with the change of temperature. To study the temperature effect on Bluetooth clocks, we conduct experiments based on two types of chipsets manufactured by Broadcom and Cambridge Silicon Radio (CSR), which are two major Bluetooth chipset vendors on the market. The tested devices include one CSR USB adapter, and one Broadcom radio in a laptop. The clock skews of these devices are first measured under normal room temperature of about $70°F$. Before the second measurement, the environment is cooled to $35°F$. Both measurements last about 15 minutes. We observe that the difference between two measurements are 0.46ppm for CSR chipset, and 0.35ppm for Broadcom chipset, respectively. In comparison, the average absolute clock skew difference measured in our device set is 8.77ppm, which is significantly higher than the variation caused by temperature change.

To further reduce the impact of temperature on device identification, applications can track the gradual changes of clock skews for Bluetooth devices, and notify BlueID when needed. Previous work [16] shows that the variation of clock skew remains ± 0.1ppm within 1000s in the presence of temperature change. Thus the overhead of tracking the change of clock skew

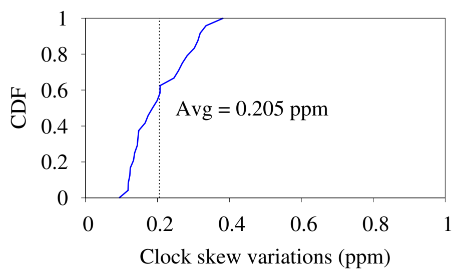


Fig. 10. CDF of clock skew variations.

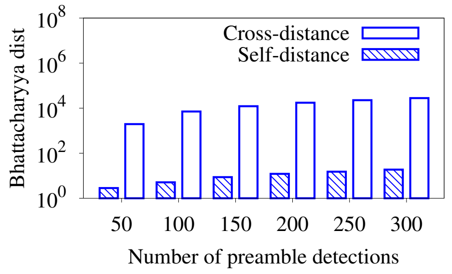


Fig. 11. Self and cross-distance of clock skews.

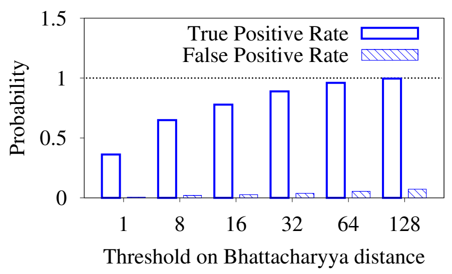


Fig. 12. Performance of device identification using different distance thresholds.

is extremely low.

Accuracy of device identification. We now evaluate a simple threshold-based device identification approach. Specifically, we apply a threshold on the Bhattacharyya distance of clock skews. Suppose the clock skew of $\mathcal{A}$ is known to be $s_{\mathcal{A}}$. A device is identified as $\mathcal{A}$, if the Bhattacharyya distance between its clock skew and $s_{\mathcal{A}}$ is smaller than the threshold.

For each clock skew measurement, BlueID uses 200 detected preambles, which is only 25% of the packets transmitted by Bluetooth per second. Our evaluation focuses on two metrics of performance. *True positive rate* (TPR) is the probability that a device is identified as itself. *False positive rate* (FPR) is the probability that different devices are identified as the same. An efficient identification system should maintain high TPR, while controlling FPR at a very low level to distinguish between different devices.

Fig. 12 shows the TPRs and FPRs when using different thresholds of Bhattacharyya distance. TPR is obtained by comparing the clock skews measured on the same devices at different times. As shown in Fig. 12, using a larger threshold leads to higher FPR, while using small threshold yields a low TPR due to the temporal variations of clock skews on individual devices. By tuning the threshold, BlueID can achieve different trade-offs between TPR and FPR. We also observe that BlueID can accurately identify different Bluetooth devices while maintaining high TPR. For example, using a threshold of 128, the resulted TPR and FPR are 100% and 5.7%, respectively. This means that BlueID can accurately differentiate 94.3% Bluetooth devices, without falsely rejecting device identities.

VIII. Conclusion

We presented BlueID, a novel system that identifies Bluetooth devices using the clock skews of their chipsets. Previous clock skew based approaches rely on the timestamps of packet headers, making them vulnerable to packet injection based identity spoofing. In comparison, BlueID extracts clock skew based on the temporal feature of Bluetooth frequency hopping, which cannot be forged without a customized baseband. Because the proprietary chipset firmware that implements frequency hopping is not open on commercial off-the-shelf Bluetooth devices, BlueID will significantly lift the bar of identity spoofing. To realize BlueID, we developed novel techniques, including noise filtering, preamble separation, and accuracy-aware fingerprinting, to enable efficient and accurate fingerprinting in a noisy environment. Our extensive evaluations show that BlueID can detect low-power Bluetooth radios from 100m away, and identifies different devices with high accuracy, short delay, and low computation overhead.

IX. Acknowledgment

This work was supported in part by U.S. National Science Foundation under grants CNS-1218475, and CNS-0954039 (CAREER).

References

[1] Bluebugging. http://trifinite.org/trifinite_stuff_bluebug.html.
[2] Bluesnarf. http://www.bluesnarf.blogspot.com/.
[3] Bluetooth technology. http://www.bluetooth.com.
[4] Bluez. http://www.bluez.org/.
[5] Ieee std 802.15.1-2005. http://standards.ieee.org/.
[6] Ubertooth. http://ubertooth.sourceforge.net/.
[7] V. Brik, S. Banerjee, M. Gruteser, and S. Oh. Wireless device identification with radiometric signatures. In *ACM MobiCom*, 2008.
[8] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. Van Randwyk, and D. Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In *USENIX Security Symposium*, 2006.
[9] S. S. Hong and S. R. Katti. Dof: a local wireless information plane. In *ACM SIGCOMM*, 2011.
[10] P. Hough. Method and Means for Recognizing Complex Patterns. U.S. Patent 3.069.654, 1962.
[11] S. Jana and S. K. Kasera. On fast and accurate detection of unauthorized wireless access points using clock skews. In *ACM MobiCom*, 2008.
[12] S. Jung, U. Lee, A. Chang, D.-K. Cho, and M. Gerla. Bluetorrent: Cooperative content sharing for bluetooth users. In *IEEE PerCom*, 2007.
[13] A. J. Khan, V. Subbaraju, A. Misra, and S. Seshan. Mitigating the true cost of advertisement-supported "free" mobile applications. In *HotMobile*, 2012.
[14] T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2005.
[15] M. Mirza, P. Barford, X. Zhu, S. Banerjee, and M. Blodgett. Fingerprinting 802.11 rate adaption algorithms. In *IEEE INFOCOM*, 2011.
[16] S. J. Murdoch. Hot or not: revealing hidden services by their clock skew. In *ACM CCS*, 2006.
[17] S. Nath, F. X. Lin, L. Ravindranath, and J. Padhye. Smartads: bringing contextual ads to mobile apps. In *ACM MobiSys*, 2013.
[18] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 user fingerprinting. In *ACM MobiCom*, 2007.
[19] A. Pásztor and D. Veitch. Pc based precision timing without gps. In *ACM SIGMETRICS*, 2002.
[20] K. B. Rasmussen and S. Capkun. Transient-based identification of wireless sensor nodes. In *ACM IPSN*, 2009.
[21] S. Rayanchu, A. Patro, and S. Banerjee. Airshark: detecting non-wifi rf devices using commodity wifi hardware. In *IMC*, 2011.
[22] R. Zhou, G. Xing, X. Xu, J. Wang, and L. Gu. Wiznet: A zigbee-based sensor system for distributed wireless lan performance monitoring. In *IEEE PerCom*, pages 123–131, 2013.