On Fast and Accurate Detection of Unauthorized Wireless Access Points Using Clock Skews

Suman Jana and Sneha K. Kasera

Abstract—We explore the use of clock skew of a wireless local area network access point (AP) as its fingerprint to detect unauthorized APs quickly and accurately. The main goal behind using clock skews is to overcome one of the major limitations of existing solutions—the inability to effectively detect Medium Access Control (MAC) address spoofing. We calculate the clock skew of an AP from the IEEE 802.11 Time Synchronization Function (TSF) time stamps sent out in the beacon/probe response frames. We use two different methods for this purpose—one based on linear programming and the other based on least-square fit. We supplement these methods with a heuristic for differentiating original packets from those sent by the fake APs. We collect TSF time stamp data from several APs in three different residential settings. Using our measurement data as well as data obtained from a large conference setting, we find that clock skews remain consistent over time for the same AP but vary significantly across APs. Furthermore, we improve the resolution of received time stamp of the frames and show that with this enhancement, our methodology can find clock skews very quickly, using 50-100 packets in most of the cases. We also discuss and quantify the impact of various external factors including temperature variation, virtualization, clock source selection, and NTP synchronization on clock skews. Our results indicate that the use of clock skews appears to be an efficient and robust method for detecting fake APs in wireless local area networks.

Index Terms—IEEE 802.11, fingerprint, MAC address spoofing, fake access point, time stamp.

1 Introduction

With advances in microtechnology and wireless networks, networked mobile systems are becoming increasingly prevalent. There is also an ever growing demand for ubiquitous services. These two factors are fueling a wide-scale deployment of wireless networks including the IEEE 802.11 wireless local area networks. However, because of their importance in providing ubiquitous services and their inherent vulnerability due to broadcast nature of the wireless medium, the wireless local area networks (WLANs) are also becoming targets of a variety of attacks. One of the ways in which a WLAN can be attacked is by introducing one or more unauthorized *fake* Access Points (APs) in the network [1], [2], [3], [4]. A fake AP can be set up by a malicious attacker (Fig. 1) to masquerade as an authorized AP by spoofing the authorized AP's medium access control (MAC) address. This fake AP is used to fool a wireless node in the WLAN into accessing the network through the fake AP instead of the authorized one. The fake AP can then launch a variety of attacks thereby compromising the security of the wireless communication. Setting up fake APs is not hard. Public domain programs including *rglueap* [5] sniff 802.11 probe request frames to find out the default AP of the probing wireless node, and then, impersonate the default AP. Therefore, detecting unauthorized APs is a very important task of WLAN intrusion detection systems (WIDSs).

The new wireless security enhancement 802.11i RSNA (Robust Security Network Association) uses traditional cryptographic methods (i.e., digital certificates) to provide strong mutual authentication between wireless clients and the APs. Although this solution, if implemented properly, will make the fake AP attack less likely, the following practical issues can still make wireless networks using 802.11i RSNA vulnerable. First, management and verification of digital certificates across different domains are known to be cumbersome. Second, as the current AP selection algorithms use signal strength as the only criteria for AP selection, users can be fooled to connect to the fake AP that has a higher signal strength compared to the original one but does not support any security measures such as RSNA.[1] Third, an attacker can also set up fake APs having the same identifiers (MAC address, basic service set identifier (BSSID) and service set identifier (SSID)) as the original AP and evade detection by using different physical channel characteristics (by using short/long preambles, operating in a different channel, etc.). These facts motivate us to find a viable noncryptographic solution to the fake AP attack. We emphasize that this solution is not meant to replace existing cryptographic methods. Rather, it should be used in conjunction with the cryptographic methods to achieve a higher level of security in WLANs. The current state-of-the-art noncrypto methods for unauthorized AP detection [1], [7], [3], [8], [4] cannot detect fake APs.

In this paper, we explore a passive online scheme that can detect fake APs with high accuracy and minimum overhead.

• *The authors are with the School of Computing, University of Utah, Salt Lake City, UT 84112. E-mail: sumanj@gmail.com, kasera@cs.utah.edu.*

Manuscript received 23 Oct. 2008; revised 28 May 2009; accepted 4 Aug. 2009; published online 12 Aug. 2009.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-2008-10-0425. Digital Object Identifier no. 10.1109/TMC.2009.145.

1. This security rollback/downgrade attack is possible in 802.11i RSNA networks [6].

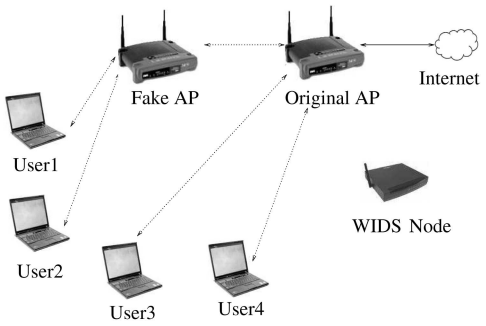

Fig. 1. A fake AP attack scenario.

This scheme, like the one proposed by Kohno et al. for fingerprinting personal computers and servers [9], is based on estimating clock skews of APs. An AP's clock skew acts as its fingerprint. Kohno and coauthors [9], [10] have shown that the clock skew of a device remains fairly consistent over time but the clock skews vary significantly across devices thereby arguing that the clock skew of a device can be used as its reliable fingerprint. However, Kohno's scheme focused on wide-area wired networks. Its application in a local area setting can result in higher accuracy. Unlike Kohno's scheme that uses TCP/ICMP time stamps, in our scheme, we use the Time Synchronization Function (TSF) time stamps in the IEEE 802.11 beacon/probe response messages sent by the AP, to determine its clock skew. The use of beacons has several advantages. First, beacons are sent all the time and at a fast rate (typically, 10 to 100 frames per second) independent of any application. Second, the granularity of 802.11 TSF timer is 1 microsecond which is much higher than that of TCP time stamp clocks. Third, as the beacon time stamp is the actual time when an AP sends a frame (i.e., the time after the channel is sensed to be free) rather than the time when it is scheduled to send the frame, we do not need to consider any significant unpredictable delays incurred by the network as in the case of TCP time stamps. Therefore, our scheme estimates more accurate clock skews and much faster compared to the TCP/ICMP time stamp approach [9]. We also improve upon the time taken for estimating the clock skew by using high-precision timers, at the fingerprinting node, that have resolutions in the order of microseconds to measure the arrival time of beacon frames.

We examine two different methods for estimating the clock skew of an AP. Our first method is based on the linear programming approach, first proposed by Moon et al. [11] and later used by Kohno et al. [9]. This method finds a line that upper bounds all the time offsets calculated from the time stamps in the AP beacons and the time of arrival of those beacons at fingerprinting node. The slope of this line is our clock skew estimate. Our second method is based on finding a line that is at the least-square distance from all the time offsets. The slope of the line represents our estimate of clock skew. As we show later in Section 5, both of these methods perform their tasks fairly well. However, in the special case when the frames transmitted by the fake AP are interspersed with the frames transmitted from the authorized AP that is being faked, both of these methods fail to determine clock skews accurately. These methods are not even designed to handle this scenario. To achieve separation of frames with the same identifiers but from different APs, we develop a novel heuristic for differentiating frames sent by the fake AP and the authorized one that is being faked. Our heuristic exploits differences both in the beacon time stamp values of different APs as well as the different rate of increment of those values. We also use our clock-skew-based fingerprinting technique in a wireless ad hoc setting to identify individual nodes but find that it is very difficult to estimate a node's clock skew accurately in this setting because of periodic clock synchronization among the nodes.

For our experimentation and evaluation, we implement our methodology on laptops running Linux and measure the clock skews of a wide range of APs from different manufacturers in three different residential settings. We also use WLAN traces from the 2004 ACM Sigcomm conference to compute the clock skews of the APs used at the conference venue. From our experiments, we find that an AP's clock skew remains consistent over time but the clock skew varies across APs. Therefore, an AP's clock skew can be used as its fingerprint. In our WLAN setting with predictable beacon delays and high-resolution time stamps, we can find clock skews very quickly, using 50-100 packets in most cases. We also discuss and quantify the impact of various external factors including temperature variation, virtualization, and NTP synchronization on clock skew. Very importantly, we also explore the possibility of engineering clock skews to allow a fake AP to generate the clock skew of the original one. Our exploration results indicate that the use of clock skews appears to be an efficient and robust method for detecting fake APs in WLANs.

In a real deployment, we expect our methodology to be implemented on the WIDS nodes. In order to verify whether or not an AP is genuine, a WIDS node can compute the clock skew of the AP and compare with the precomputed clock skew of the AP with the same identity (e.g., MAC address).

The rest of this paper is organized as follows: Section 2 describes the threat model that we address in this paper. We describe our clock skew estimation methodology in Section 3. Section 4 contains a description of our implementation, and Section 5 contains our experimental results. Fabrication of clock skews is discussed in Section 6. In Section 7, we explore the utility of our scheme in wireless ad hoc networks to identify participating devices uniquely. We survey the existing work on detecting unauthorized APs in Section 8. We conclude the paper in Section 9 by summarizing our work and indicating directions for future work.

2 Threat Model

An adversary can set up an unauthorized AP to masquerade as an authorized one.

There are two scenarios in which a fake AP can operate:

- The fake AP and the authorized AP that is being faked are both active at the same time. As the current AP selection mechanisms use signal strength as the only selection criteria, the user will select the fake

AP if he measures the fake AP's signal strength to be higher than the original AP.
- Only the fake AP is active and the authorized AP being faked is inactive. This can happen when the authorized AP has failed on its own or due to a Denial-of-Service attack from the adversary, or when the user moves to a location where only the fake AP is reachable. The adversary can facilitate this by tracking and following the user.

In our threat model, the adversary is powerful enough to modify any of the MAC address, BSSID, and SSID fields of any frame he wants. The adversary can also capture, collect, and analyze any amount of data without being detected even before actually trying to break into the network. If the packets are sent across the network in encrypted form, the adversary can gather enough packets needed to launch password guessing attacks. It can also decrypt the packets once it succeeds in guessing the password.

Our methodology will address the detection of unauthorized APs in all these cases. As our method is based on a physical characteristics of the AP (i.e., the clock skew), it can detect MAC, BSSID, and SSID spoofing, whether the authorized AP is active or not. We expect the clock-skew-based methodology to be deployed in the WIDS nodes for detecting unauthorized APs in WLANs. We assume that the adversary cannot break into WIDS nodes. We also assume that the attacker does not have access to any custom hardware that can generate fake time stamps at a very fine granularity. We discuss this issue in more detail in Section 6.

3 Methodology

In an IEEE 802.11 infrastructure WLAN, there are two methods that a client station (STA) may use to find an AP in the WLAN [12].

- **Active scanning:** The STA sends a probe request frame to determine which APs are within range. The APs in the vicinity then reply back with probe response frames.
- **Passive scanning:** The STA learns about the APs in the WLAN by listening to the beacon frames broadcast by the APs.

The probe response and the beacon frames both have an 8 byte time stamp. The time stamp field contains the value of TSF timer of the AP when it sends the frame. The beacons are scheduled to be sent at periodic intervals by the APs. The time stamps in the beacon frames do not get affected by the random medium access delays of the wireless medium as hardware sets the time stamp value just before actual transmission. The TSF timer is a 64 bit timer which is initialized at the time of starting the AP and incremented once every microsecond.

Our solution uses TSF time stamps in beacon/probe response frames to estimate the clock skew of an AP and uses the clock skew as the AP's fingerprint. An access point's (or regular computer's) clock consists of primarily two parts:

- **Oscillator:** An oscillator is controlled by a crystal and it ticks at a fixed frequency.
- **Counter:** A counter keeps track of the number of ticks produced by the oscillator.

The exact frequency of a crystal depends primarily on the type of the crystal and the angle at which the crystal was cut relative to its axes. However, in reality, even two crystals of the same type and the same cut will have slightly different frequencies due to the limited mechanical accuracy of the cutting process [13]. This is the primary reason behind the existence of clock skew even between seemingly similar clocks.

Let us assume that a fingerprinter node (a WIDS node) has received n beacon frames from a particular AP. Let the time stamp in the ith beacon frame be T_i and let t_i be the time in microsecond when the fingerprinter receives the ith beacon frame. Let S_i be the size of the ith beacon frame and R_i be the data rate at which ith beacon frame is sent. Therefore, the time, according to the AP's clock, when the fingerprinter receives the packet is $T_i + S_i/R_i$. Let our estimated offset for the ith frame be denoted o_i and the time difference between the first received frame and the ith frame according to the fingerprinter's clock be x_i . Then,

$$x_i = t_i - t_1, \tag{1}$$

$$o_i = ((T_i + S_i/R_i) - (T_1 + S_1/R_1)) - (t_i - t_1). \tag{2}$$

In most of the cases, the beacon frames are sent at a fixed-data rate and the size of the beacon frames remain fixed as well [12]. So, we can assume that $S_i/R_i = S_1/R_1$. This yields,

$$o_i = (T_i - T_1) - (t_i - t_1). \tag{3}$$

Now, if the clock skew of a particular device remains constant and if we plot (x_i, o_i), we will get an approximately linear pattern. The clock skew can be estimated as the slope of this linear pattern. Let us call the set of points $(x_1, o_1), \ldots, (x_n, o_n)$ the clock offset-set of the AP.

We evaluate two different methods for estimating the clock skew from the offset-set—a linear-programming-based method (LPM) that was proposed in [11] and later used by [9] and a least-square fit (LSF) method.

3.1 Linear Programming Method (LPM)

LPM finds a line $\delta x + \phi$, where δ is the slope of the line and ϕ is the y-axis intercept, that upper bounds the points in the clock offset-set of the AP and outputs the slope of the line, δ, as the clock skew estimate. So, our clock skew estimation δ is such that $\forall i = 1 \ldots n$

$$\delta.x_i + \phi \geq o_i, \tag{4}$$

and the following function is minimized:

$$1/n. \sum_{i=1}^{n} (\delta.x_i + \phi - o_i). \tag{5}$$

This problem can be solved using linear programming methods for two variables.

The LPM method minimizes the effect of any unpredictable delays as it has higher tolerance toward outliers. The clock skew estimate remains stable even if there is significant number of outliers. However, in our case, the

number of outliers is very less because no significant random delay is involved in the communication path and the TSF clocks have a higher precision than TCP time stamp clocks.

Interestingly, LPM's nature to tolerate the outliers may cause a serious security problem in our context. If an adversary is able to mix small number of beacons from a fake AP with the beacons of the authorized one, it is faking, and if the clock skew of the fake AP is close to the clock skew of authorized one, then this method might consider the fake AP frames as outliers and estimate the clock skew of the authorized AP as the clock skew of the set. In this case, it will be difficult to detect the fake AP by comparing the clock skews.

3.2 Least-Square Fitting (LSF)

We can also use LSF to estimate the clock skew of an AP from its clock offset-set. Given an offset-set $(x_1, o_1), \ldots, (x_n, o_n)$, LSF finds a line $\delta x + \phi$, where δ is the slope of the line and ϕ is the y-axis intercept such that

$$\sum_{i=1}^{n} (o_i - (\delta.x_i + \phi))^2 \quad (6)$$

remains minimum. The slope of the line δ is estimated as the clock skew of the clock offset-set.

One of the major differences of LSF from LPM is its lack of tolerance toward outliers. Even if there are only a very few outliers, the clock skew estimated by LSF will vary significantly from the clock skew determined by the majority of the points. This can cause problems while estimating clock skew from noisy data. Kohno et al. [9] decided not to use LSF for estimation of TCP clock skew because TCP segments can undergo random delays in the network which can affect the accuracy of the clock skew estimate. However, as mentioned earlier, in our case, the absence of any unpredictable delays makes the number of outliers insignificant. Therefore, we can use the LSF to estimate clock skews effectively. LSF has an advantage over LPM in the scenario where an adversary tries to avoid detection by interspersing frames from a fake AP with the frames from the authorized one as described above. LSF's sensitivity to the presence of even a small number of outliers will help determining the fake AP in the above scenario more effectively than LPM. So, it will be difficult for the adversary to masquerade frames from the fake AP as outlying data when LSF is used to estimate the clock skew.

We measure and compare the effectiveness of these two methods in estimating clock skews in Section 5.

3.3 Differentiating Frames of Fake APs

Separation of the clock offset-sets of the fake AP(s) and the authorized AP (if present) helps us to gain insight about the fake APs. For example, if the attacker uses multiple fake APs to fake one authorized AP, we can detect the fake APs by separating the clock offset-sets.

The general problem of fitting multiple lines to a data set is not new. In the domain of computer vision and image processing, Generalized Hough Transform (GHT) [14], [15] is a well-known technique that can be used for this purpose. However, the main drawback of GHT is that it is computationally intensive and requires a large amount of storage. Even though techniques like Randomized Hough Transform (RHT) [16] try to minimize these effects, the time required by these techniques is still quite high. The use of GHT is justified in the domain of image processing and computer graphics because images normally contain large number of edges and GHT can detect all of them together. However, in our case, we expect to have very few lines.

Another approach to solve the problem of fitting multiple lines to a data set is to model the data as a mixture model and apply the well-known statistical method of expectation maximization (EM) to separate the data [17]. However, the EM algorithm requires the initial parameters to be guessed and the accuracy of the results depends on the values of these initial parameters. Furthermore, EM requires multiple iterations to converge.

We note that the complexities and the computation intensiveness of these algorithms arise from their attempt to solve a general problem without any domain-specific assumptions. In our problem domain, we have some specific characteristics of the data that help us to create a less complex and lightweight solution. We know the following facts about the clock offset-sets:

- The thickness of the lines in the clock offset-set plot (i.e., the variance of the points in the set) remains mostly constant across the APs.
- The amount of noise in the data is negligible.

Keeping these facts in mind and borrowing ideas from both of the aforementioned methods, we design a lightweight heuristic that solves our problem efficiently.

Our heuristic relies on the fact that if a clock offset-set is calculated from the beacons received from different APs, then the clock offset-set will contain certain *jumps* (i.e., sudden big changes in the value) at the boundary where one packet is from one AP and the successive packet is from another AP. Our heuristic identifies these jumps and differentiates the data based on it.

We exploit this fact to differentiate packets from different APs. Let Δ_{ij} be the *relative skew* between two samples in the clock offset-set (x_i, y_i) and (x_j, y_j). Δ_{ij} is defined as follows:

$$\Delta_{ij} = |y_i - y_j|/|x_i - x_j|, \quad (7)$$

where $|x|$ is the absolute value of x.

We introduce a tunable parameter called *threshold* to differentiate between *jump* and *consistent increment*. Thus, two consecutive points (x_i, y_i) and (x_j, y_j) in the clock offset-set are considered to be a jump if and only if $\Delta_{ij} > threshold$. Using this definition of jump, we can segregate the clock offset-set data into separate groups based on the jumps taken.

In Algorithm 1, $threshold$ is the only parameter that can be and must be tuned. A limit can also be imposed on the count field to filter out small number of outliers that are not part of any data set. However, we do not expect the WLAN samples to contain outliers that are not part of any sample, and hence, we do not set any limit on or tune the count field. The value of $threshold$ can be estimated empirically from the clock offset-set of a single AP. Algorithm 2 describes the algorithm for finding the $threshold$ value from the test data.

Algorithm 1. Separate clock offset-set points based on originating AP

$accumulator[0].dataset \Leftarrow [(x_1, y_1)]$
$accumulator[0].current\_point \Leftarrow (x_1, y_1)$
$accumulator[0].current\_offset \Leftarrow 1$
$accumulator[0].count \Leftarrow 1$
for $i = 2$ to n **do**
  for each entry j in accumulator **do**
    $k \Leftarrow accumulator[j].current\_offset$
    if $\Delta_{ik} \leq threshold$ **then**
      add (x_i, y_i) to data set of accumulator entry j
      $accumulator[j].count \Leftarrow accumulator[j].count + 1$
      $accumulator[j].current\_point \Leftarrow (x_i, y_i)$
      $accumulator[j].current\_offset \Leftarrow i$
    end if
  end for
  if none of the entry in accumulator satisfies ($\Delta_{ik} \leq threshold$) **then**
    add a new accumulator entry p
    $p.dataset \Leftarrow [(x_i, y_i)]$
    $p.count \Leftarrow 1$
    $p.current\_point \Leftarrow (x_i, y_i)$
    $p.current\_offset \Leftarrow i$
  end if
end for
output number of entries in accumulator as number of different data sets
output the data sets of accumulator entries as different data sets from the APs

Algorithm 2. Calculate $threshold$ from test clock offset-set

$finalthreshold \Leftarrow 0$
for each test data set **do**
  $threshold \Leftarrow \Delta_{12}$
  for $i = 3$ to n **do**
    if $\Delta_{i(i-1)} \geq threshold$ **then**
      $threshold \Leftarrow \Delta_{i(i-1)}$
    end if
  end for
  if $threshold \geq finalthreshold$ **then**
    $finalthreshold \Leftarrow threshold$
  end if
end for
output $finalthreshold$ as final calculated *threshold*

We estimate the $threshold$ using the above algorithm from different test data sets. We find that the $threshold$ estimated from a very small amount of data (i.e., 50-100 packets depending on the received time stamp resolution) is enough to separate a wide variety of data sets. From our experimental results, we find that the threshold value depends on the fingerprinter as well as on the AP which is being fingerprinted. However, these variations are very small compared to the variations in *relative skew* caused by mixing beacons from different APs. Our results suggest that we can use the same threshold to separate beacon packets from diverse set of APs. Our results also show that the value of threshold estimated by the test data depends on the method we use to generate the receiver's time stamps. For example, if we use our modified MadWifi driver, described later in Section 4, then the $threshold$ is estimated as 0.003, whereas if we use *jiffies*[2] to estimate the time stamp, then the value of threshold becomes 0.05.

Once the data sets are separated using the above heuristic, we can use either LPM- or LSF-based methods to estimate the exact clock skew of different fake APs.

4 Implementation

We implement our methodology for capturing beacon frames, recording time stamps, and computing clock skews of APs, presented in the last section, on two laptops—an Acer TravelMate 2303 NLC running Ubuntu Linux 7.4 and an Acer Aspire running SUSE Linux 10.1. We use two wireless cards—a Linksys WPC 55AG and a Intel PRO/Wireless 3945ABG. The Linksys card uses an Atheros chipset that works with the MadWifi driver. We chose these cards because they both support the monitor mode and also because their drivers are open source. The availability of the source code allows us to modify the drivers to measure the arrival time of beacon frames with higher resolution as described below. As the success of our methodology is closely tied to how precisely we can measure time, most of our implementation effort targets obtaining high-precision time measurements and we will focus on this very aspect of our implementation in the rest of this section.

In order to accurately estimate the clock skew of an AP, we need to precisely measure the time when a beacon frame reaches the wireless LAN card of the fingerprinter. We will now describe and discuss three different mechanisms that we explore for the purpose of accurately measuring the arrival time of a beacon frame at the fingerprinter. We first explore the use of sniffers such as *tcpdump* [18], to find the arrival time of a frame. Even though this mechanism does not require any changes in the system, we note that the time stamp generated by tcpdump includes variable processing time of the operating system. Therefore, use of tcpdump time stamp is not suitable for our purpose.

Next, we explore using the Prism monitoring headers in the MadWifi driver [19] and the Intel 3945ABG driver [20]. These drivers allow additional Prism monitoring headers to be added to frames arriving at the wireless card which has a 4-byte time stamp field. The drivers use it to report the time when the packet is received by the wireless cards. However, we find that the precision of the time reported by MadWifi is not accurate enough to detect the clock skew quickly and accurately. In Linux, MadWifi driver puts the current value of the *jiffies* variable in the time stamp field. *jiffies* is a counter incremented at regular intervals by the Linux kernel through timer interrupts. By default, it is incremented once every 4 ms in recent Linux kernels (newer than 2.6.13). This interval is a configurable parameter that can also be set to 1 ms or 10 ms [21]. Therefore, the highest resolution available for incrementing *jiffies* in Linux is 1 ms. Making the jiffies counter arbitrarily small is not desirable because the number of timer interrupts being invoked per second depends on this value and will increase

2. *jiffies* is a variable maintained and incremented once in every 4 ms by the Linux kernel.

significantly. High timer interrupt overhead can lead to unstable system behavior. Now, as noted before in Section 3, the TSF counter in the AP is incremented once every microsecond. Therefore, the clock skew of an AP cannot be estimated quickly and accurately with a 1 ms precision clock at fingerprinter's end.

The 1 ms resolution limitation of jiffies leads us to explore a third mechanism. Here, our goal is to use a microsecond precision time stamp. However, a microsecond precision time stamp will quickly overflow a 4 byte field that the Prism header allows and has room for. To deal with this problem, we use another header called the Radiotap header that has an 8 byte time stamp field. Normally, when the MadWifi or the IPW 3945 ABG drivers receive a frame, the current value of the TSF timer of the fingerprinter is stored in the time stamp field of the Radiotap header [19], [20]. Both these drivers maintain a microsecond resolution TSF timer. However, this TSF timer is synchronized to the time stamps of the received beacon frames. Therefore, it cannot be used for an accurate measurement of the clock skew. We modify both the drivers to call *do_gettimeofday*, which supports microsecond resolution, each time a frame is received and stores the time stamp in the 8 byte time stamp field of the Radiotap header. We show in Section 5 that using this improvement clock skews can be estimated accurately by examining 50-100 packets in most of the cases. We end this section by analyzing the overheads caused by our monitoring scheme.

The use of *do_gettimeofday* in our scheme does not add any significant performance overhead because time stamps are recorded only when a wireless card is in the monitor mode and the Radiotap headers are enabled. Moreover, we also introduce an *ioctl* system call to turn this feature on or off allowing us to turn off this feature when we are not measuring clock skews. As the packet capture for measuring skew only takes small amount of time (2-3 minutes), the overhead due to enabling this feature only for that duration is not significant.

5 Experimental Results

We use experimental traces from two very different settings to test our methodology for detecting unauthorized APs. Our first set of traces is from data collected during the ACM Sigcomm 2004 conference [22]. The Sigcomm conference network comprised 5 different APs. Five PCs, each with three Netgear WAG 311 wireless adapters, were used for wireless sniffing. The details of the data collection settings can be found in [22]. As the Sigcomm data set represents a heavily used 802.11 wireless network, we use it to estimate the number of frames needed to estimate the clock skew accurately in a loaded network. Kohno et al. [9] performed extensive measurements to show that clock skews of networked devices remained consistent over a long time. Our main goal here is to verify that this observation holds in case of APs as well, and estimate how quickly and accurately we can estimate the clock skew of APs.

We obtain our second set of traces by collecting wireless data in three different residential settings each with multiple APs operating simultaneously. One residential setting (residential setting A) has 8 APs and two other ones (residential setting B and residential setting C) have 21 APs and 12 APs, respectively, from different manufacturers. We use two laptops that implement our measurement methodology, as described in the last section, to collect the packet traces. We collect the packet traces on multiple days in same residential settings to verify the consistency of AP clock skews over time.

TABLE 1
Skew Estimates for Samples (Collected by *Chihuahua*) with Different Sample Sizes

Packets examined	skew(using LPM)	skew(using LSF)
100	49.36 ppm	42.73 ppm
200	50.69 ppm	46.14 ppm
300	51.21 ppm	47.98 ppm
400	51.21 ppm	48.42 ppm
500	51.21 ppm	49.06 ppm
600	51.21 ppm	49.32 ppm

The samples contain beacon frames sent by sigcomm-nat.

We use the measure parts per million, essentially $\mu s/s$, denoted as *ppm*, to quantify clock skew. We describe the results of our experiments with the Sigcomm and the residential traces in the following sections.

5.1 Results from the Sigcomm Trace

Each packet in the Sigcomm traces has a prism header which contains receive time stamp of that packet. As stated in Section 4, the time stamps in Prism headers are in terms of jiffies. We also note in Section 4 that the resolution obtained with jiffies is in milliseconds.[3] Therefore, the Sigcomm data do not contain very precise time measurements in comparison to the data we collect with microsecond resolution. However, the Sigcomm data can still be used for estimating clock skews, albeit using more samples.

First, to check the consistency of the AP clock skew over time, we create 20 equal sized sample data sets by selecting blocks of packets starting from random offset from the trace collected by the machine *chihuahua* and measure the clock skew of a particular AP, with SSID *sigcomm-nat*, for each data set. We find that the clock skew estimate remains around 51.25 ppm (using LPM) and between 51.09 and 51.37 ppm (using LSF) for each of the sets. This reaffirms that the clock skew of an AP remains consistent over time. Next, we try to figure out the speed of convergence of our procedure, i.e., what is the minimum number of packets that we need to examine to get a close skew estimate. We start with the skew estimates for the first 100 packets, and then, increment the number of packets by 100 and measure the clock skew in each of the cases. The skew estimate results are shown in Table 1.

As can be seen in Table 1, the minimum number of packets needed to converge to a clock skew is 300 (using LPM). However, when we use LSF, even 600 packets are not enough to converge to a small range of clock skews. In fact, 900 packets (not shown in the table) are required to converge to the 51.09-51.37 range. Later, we will show in

3. As the Sigcomm trace was collected in 2004 (when 2.4 Linux kernels were latest ones), we assume that the resolution of jiffies is 10 ms. However, this assumption does not have any effect on the consistency of an AP clock skew or on the comparison between the clock skews of different APs. It only helps us in estimating absolute values of the skews which are easier to comprehend than comparing them using their ratio.

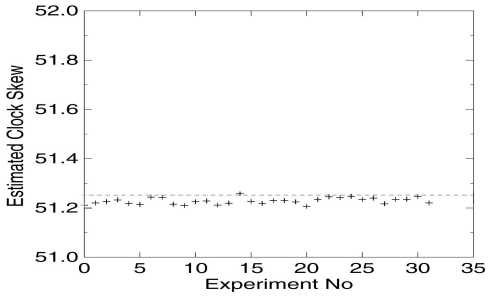

Fig. 2. Skew estimates of samples containing 300 packets taken at different times by *chihuahua*. The samples contain beacon frames sent by sigcomm-nat.

AP	Fingerprinter							
	Chihuahua		Mojave		Sonoran		Kalahari	
	skew (LPM)	skew (LSF)	skew (LPM)	skew (LSF)	skew (LPM)	skew (LSF)	skew (LPM)	skew (LSF)
sigcomm−nat	51.25	51.20						
sigcomm−nat−foyer			40.30	40.39	34.99	35.29	44.91	45.00
sigcomm−public−1			48.16	48.21			49.94	49.34
sigcomm−public−2	48.82	48.90			32.59	32.62	42.69	42.98

Fig. 3. Skew estimates of different APs by chihuahua, Kalahari, Mojave, and Sonoran. All skew estimates are in part per million.

Section 5.2 that LSF can also estimate clock skews accurately using the same number of packets as LPM if we use the higher resolution receiver time stamps. To verify if the clock skew estimated by monitoring 300 packets using LPM remains consistent over time, we take 32 random samples, each of size 300 packets, from the trace and we estimate the clock skew for each sample. Fig. 2 shows the estimated clock skew as a function of the experiment number. We find that all the estimates remain very close to 51.25 ppm which is the actual estimate of the skew made over all the packets (shown by the dashed line in Fig. 2).

Thus, we can see that even using lower resolution time stamps (i.e., jiffies), we can estimate clock skews fairly accurately. However, we require 300 or more packets. In Section 5.2, we show that using higher resolution time stamp, we can estimate skews much faster.

We also examine the skew estimates for different APs based on the time measurement data collected at different machines. The skew estimate results based on data from four different machines are shown in Fig. 3. We note that the clock skew estimates differ across different measurement nodes. This observation suggests that we must compare clock skews only from the same measuring node.

5.2 Results from the Residential Traces

In this section, we will refer to the Acer TravelMate 2303 NLC laptop as *laptop1* and Acer Aspire laptop as *laptop2*. We use the monitor mode supported by the wireless cards in both the laptops for capturing beacon frames and also enable the Radiotap headers in the packets (as described in Section 4) that we capture.

First, we measure the clock skew of two different Linksys APs (Linksys1 and Linksys2). The packets for this trace are collected using *laptop2*. Fig. 4 plots the offset-sets for the APs. Next, in order to study the consistency of the clock skews of different APs over time, we collect offset-sets from eight different APs (including Linksys1 and Linksys2) in residential setting A on two different days while keeping all the other parameters (i.e., the time span of capture, etc.) same.[4] Table 2 shows the skew estimates of all APs in residential setting A on two different days using LPM and LSF. As we did not have control over all the APs, manufacturer name is predicted based on the manufacturer specific first 3 bytes of the MAC address. The clock skew estimates measured in residential settings B and C are shown in Tables 3 and 4, respectively.

For all the three tables, we are able to estimate the clock skews accurately by analyzing 50-100 packets in most of the cases. Therefore, we find that microseconds resolution receiver time stamps that we use in our methodology result in a big improvement over millisecond resolution receiver time stamps that needed about 300 packets (or more for LSF) for accurate estimation of the clock skew (as shown in Section 5.1). This provides almost 20 times improvement over Kohno's results [9] where, on average, 1,000-2,000 packets were needed for a correct skew estimation. If we consider average time taken to estimate the skew, using higher precision time stamps in a more predictable WLAN setting takes only 2-3 minutes, whereas Kohno's clock skew estimates performed in a wide-area setting with coarser time stamps [9] take about 30 minutes-1 hour to converge. This makes our use of clock skew in the WLAN settings 15-20 times faster. We also make other important observations from these tables. First, clock skews are different for different APs. Second, the clock skew for a given AP is consistent over the two measurements. Third, clock skews obtained using LPM closely match those obtained using LSF.

5.3 Differentiating Frames of Fake APs

To simulate the attack scenarios where a fake AP and an authorized AP are active at the same time, we construct synthetic data sets by mixing beacon packets collected in real packet captures from multiple APs. While creating this data sets, we preserve the order in which the packets were received by the fingerprinter. As the fake AP and the authorized AP, both have the same MAC address, and the fingerprinter has no way of separating the packets. We analyze the effect of this intermingling on our estimation methods. We also test the

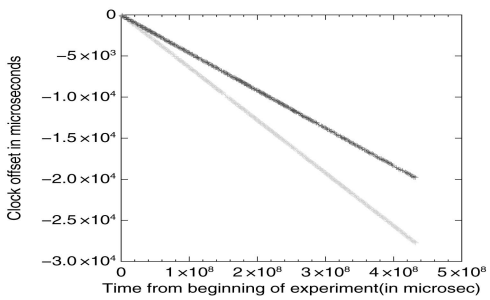

Fig. 4. TSF clock offset-sets for two different Linksys APs. Clock skew estimations are -64.23 and -45.69 ppm.

4. We do not have any control over the amount of wireless traffic generated in these experiments. However, the traffic variation does not affect our results.

TABLE 2
Clock Skew Estimates in Residential Setting A as Measured from *laptop2*

AP	1st Measure(LPM)	1st Measure(LSF)	2nd Measure(LPM)	2nd Measure(LSF)
Linksys1	-64.23 ppm	-64.10 ppm	-64.90 ppm	-64.77 ppm
Linksys2	-45.69 ppm	-45.96ppm	-46.94 ppm	-46.71 ppm
Linksys3	-62.05 ppm	-61.84 ppm	-62.77 ppm	-62.64 ppm
Belkin1	-56.37 ppm	-56.57 ppm	-56.71 ppm	-56.85 ppm
Belkin2	-1105.50 ppm	-1105.69 ppm	-1106.29 ppm	-1106.06 ppm
Netgear1	-58.08 ppm	-57.78 ppm	-58.86 ppm	-59.25 ppm
Dlink1	-47.27 ppm	-47.17 ppm	-47.80 ppm	-48.14 ppm
Unknown1	-40.91 ppm	-40.99 ppm	-41.61 ppm	-41.47 ppm

efficiency of our algorithm for separating the packets using these synthetic data sets.

Table 5 shows that in some cases (e.g., cases 1, 2, and 4), the skew estimated using LPM is same as the skew of one of the APs whose packets are intermingled.[5] These results suggest that when we use LPM, we might miss a fake AP operating at the same time as the authorized AP. This points to a serious problem in using LPM. On the contrary, the skews estimated by LSF are exceptionally large than the actual clock skews of each of the contributing AP. So, by just observing the skew value, we can conclude that some fake APs are active. Therefore, when using higher resolution receive time stamps LSF alone can be used to detect fake APs. However, if the receive time stamps are of low resolution, both LPM and LSF should be used. This is because LPM uses fewer packets than LSF to estimate the clock skew accurately. On the other hand, LSF detects the mixing of packets from different sources with a higher success rate than LPM.

We apply our packet separation algorithm (Algorithm 1), as described in Section 3, to all the five synthetic data sets that we use for Table 5 as well as to 10 other synthetic data sets created from traces collected by *laptop1*. Recall that Algorithm 1 requires a threshold that is used to differentiate between the jumps and the consistent increments of the clock offsets. We calculate this threshold using Algorithm 2 for each data set. Once this threshold has been determined, we use Algorithm 1 to separate out the beacon packets of the fake APs from the ones sent by the authentic ones. We find that for all data sets, our algorithm accurately predicts the number of APs generating the data and correctly separates the offset-set corresponding to each AP. Algorithm 1 can also be used to separate packets in real time. Fig. 6 shows how the accuracy of separation increases with increase in the number of packets used to estimate the threshold. We observe that 75 packets are needed to estimate a threshold that achieves 99 percent accurate packet separation on average (over the five synthetic traces used in Table 5). These separated packets from the fake APs must be ignored by the wireless users. These packets can also be used to fingerprint the fake APs and determine their locations.

5.4 Impact of External Factors on Clock Skews

We now discuss the impact of external factors on clock skews.

5. In some cases (e.g., cases 3 and 5), LPM estimates the skew to be 0 ppm because some of the clock offset-set values become extremely large due to the intermingling of packets. As LPM tries to use the highest values in the clock offset-set to estimate clock skew, it finds that the differences between these large values are negligible compared to the values themselves. Therefore, in these cases, LPM approximates the clock skews as 0 ppm. An example is shown in Fig. 5.

5.4.1 Effect of Virtual APs on Clock Skew

Virtual APs use single wireless hardware to simulate multiple APs with different MAC Addresses, SSIDs, and BSSIDs. In this aspect, virtual APs are not much different

TABLE 3
Clock Skew Estimates (Using LPM) in Residential Setting B as Measured from *laptop1*

AP	Clock Skew	AP	Clock Skew
Linksys1	22.53 ppm	MeruNetworks1	28.14 ppm
Linksys2	17.51 ppm	MeruNetworks2	32.53 ppm
Unknown	31.66 ppm	Trapeze Networks1	23.66 ppm
Linksys3	20.67 ppm	Trapeze Networks2	11.50 ppm
Linksys4	24.95 ppm	Dlink2	30.50 ppm
Linksys5	23.54 ppm	Linksys6	23.21 ppm
Unknown1	42.33 ppm	Trendwar	34.28 ppm
Unknown2	36.22 ppm	Dlink3	12.84 ppm
Unknown3	39.28 ppm	Unknown5	35.5 ppm
DLink1	30.85 ppm	Linksys7	27.70 ppm
Unknown4	33.26 ppm		

TABLE 4
Clock Skew Estimates (Using LPM) in Residential Setting C as Measured from *laptop2*

AP	Clock Skew	AP	Clock Skew
Linksys1	-42.01ppm	Apple1	-33.35 ppm
Linksys2	-21.21 ppm	Unknown1	-34.56ppm
Linksys3	-35.16 ppm	ActionTec1	-32.77ppm
Linksys4	-28.04 ppm	Microsoft1	-7.93 ppm
Unknown4	-37.54 ppm	Unknown2	-31.48 ppm
Unknown5	-46.34 ppm	Unknown3	-36.08 ppm

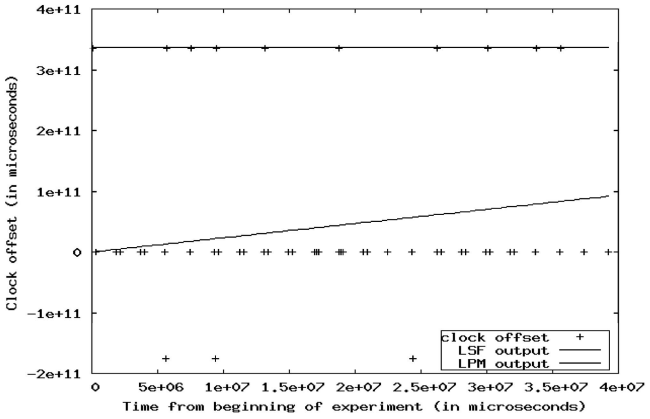

Fig. 5. LPM and LSF output using clock offset-set calculated from mixed beacon packets of three different APs (Case 5 in Table 5). The skew estimated by LPM is 0 ppm and LSF skew estimate is 4,256,390,000 ppm.

TABLE 5
Measure of Skew from the Synthetic Data Set

Case	Data Sets mixed	original skews	skew(using LPM)	skew(using LSF)	Data sets estimated
1	2	62.05,62.47	62.47	4614750000	2
2	2	40.91,48.60	40.91	363843000	2
3	2	60.03,45.69	0	406340000	2
4	2	60.61,1106.31	1106.31	4729570	2
5	3	55.14,60.61,1106.31	0	4256390000	3

All skews are absolute values. Note that the skews estimated by LSF are extremely large because of the mixing which helps us to detect the presence of fake APs much faster than LPM.

from virtual machines where multiple machines are simulated on the same hardware. However, from our experiments, we find that unlike the virtual machine clocks which normally have higher skew than real machines, as shown by Kohno et al. [9], all virtual APs being emulated on a particular hardware have the same clock skew, and the clock skew is in the same range as the real AP clock skews. This happens because while sending the time stamp, all virtual APs read from the same hardware timer and send the value unaltered. Virtual APs do not maintain separate virtual clocks. Therefore, all virtual APs using the real hardware clock will have the same clock skew as the real hardware clock. We test with five different APs (three Trapeze networks APs running their default firmware and two Linksys WRT54G APs running the DD-WRT firmware [23]). We simulate four virtual APs on each of the five real APs. Our results, as shown in Table 6, confirm the above argument. This implies that our methodology can also be used to distinguish virtual APs from real APs.

5.4.2 Effect of Temperature on Clock Skew

It has been shown in existing work [9], [10] that under normal PC operating temperatures, the clock skew of a device remains constant within ± 1 ppm. It has also been noted [10] that this temperature change can also occur due to varying processor load. However, Pásztor and Veitch [24] have shown that for small time periods (less than 1,000 seconds), the clock skew variance remains less than ± 0.1 ppm. The results presented in another existing work [10] also support this observation as the change of clock skew due to temperature variance in their results occurs gradually. Therefore, in order to be able to track any changes in the clock skew of genuine APs and for detecting fake ones in the presence of clock skew variation with temperature, we propose using a "rolling signature" scheme described in Algorithm 3. We propose that an AP's clock skew must be updated to a new value if the difference between the new measured value and the old value is within a threshold. The nodes that measure clock skews (e.g., WIDS nodes) should collect packets from different APs and execute Algorithm 3 over each 50-100 beacon frame block. Since collection of 50-100 beacon frames typically takes much less than 1,000s, we can assume that the clock skew variance due to temperature will cause the consecutive clock skew estimates to differ only by approximately ± 0.1 ppm rather than ± 1 ppm. This method thus enables our scheme to compare measured clock skews with a higher precision in comparison to the one used by Kohno et al. [9].

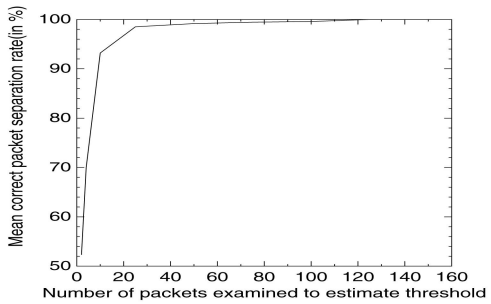

Fig. 6. Mean correct packet separation rate versus number of packets examined to estimate threshold.

Algorithm 3. Fake AP detection algorithm
Calculate $newskew$
if ($newskew$ - $currentskew$) $\leq$ $max\ skew\ variance$ **then**
$currentskew \Leftarrow newskew$
AP is original
else
Fake AP detected.
end if

As, we measure relative skew between two physical clocks, extrapolating the findings of [24], we can set *max skew variance* to ± 0.2 ppm. In our high precision residential traces, when using the same fingerprinter, all but one pair of access points (Linksys5 and Trapeze Networks 1 in Table 3) differ by more than 0.2 ppm.

5.4.3 Effect of NTP Synchronization of Fingerprinter's Clock on Skew Estimate

Unlike the approach used by Kohno et al. [9], we do not synchronize the fingerprinter's clock using the Network Time Protocol (NTP) or any other clock synchronization mechanism. Rather, we measure clock skew of an AP relative to the fingerprinter. Our measurement times are expected to be small (2-3 minutes) and the time stamps are measured in microseconds. NTPv4 is accurate within 10 milliseconds over the wide-area Internet and within 200 microseconds over an LAN. The default minimum polling interval for NTP is 64 seconds [25]. However, in our

TABLE 6
Skew of Virtual APs

AP	Virt. AP1	Virt. AP2	Virt. AP3	Virt. AP4
1	23.66	23.66	23.66	23.66
2	17.53	17.54	17.17	17.34
3	28.55	28.56	28.56	28.55
4	32.45	32.46	32.45	32.45
5	21.24	21.28	21.27	21.24

TABLE 7
Comparison of Clock Skew Estimates of Same APs Measured from *laptop2* Running on AC Power and Battery Power

AP	Skew (AC power)	Skew (battery power)
Linksys1	-64.23 ppm	272.62 ppm
Linksys2	-45.69 ppm	254.10 ppm

The measurements were taken in residential setting A.

case, as the time stamps are measured in microseconds and the estimates of the clock skews are in the range of 100 ppm, enabling NTPv4 will not provide enough accuracy to make the clock skew estimates independent of the fingerprinter's own clock skew. However, in our problem definition, the fingerprinter (a WIDS node in a WLAN environment) remains the same. So, this dependence on the fingerprinter's clock is not an issue in our scheme.

5.4.4 Selection of Fingerprinter's Clock Source

As mentioned earlier in Section 4, in a PC running Linux, *gettimeofday* system call provides microsecond resolution time stamps. gettimeofday internally uses PC's internal clock source to generate microsecond granularity time stamps. However, any modern PC normally has more than one clock source. The actual number and type of the clock sources depend on the particular model of the processor and the motherboard being used in the PC. The Linux kernel chooses the best available clock source in the PC for tracking time stamps that are reported by the gettimeofday system call. Some common clock sources are [26]—Programmable Interval Timer (PIT), Time Stamp Counter (TSC), Advanced Configuration and Power Interface Power Management Timer (ACPI PMT), and High-Precision Event Timer (HPET).

As all these internal clock sources are physically different, they will have different clock skew. As described earlier, our estimates of the AP's clock skew are also dependent on the fingerprinter's clock skew. Therefore, while using time stamps from gettimeofday to measure clock skew of an AP, we must check whether the same clock source is being used by the kernel for all the measurements. The Linux kernel dynamically selects the most accurate clock source available as the internal clock source for the kernel. The accuracy of certain clock sources can change depending on different conditions. For example, in a particular device, TSC might be initially selected as the clock source for gettimeofday. However, after some time if that device switches to battery power from AC power, the Linux kernel will decrease the frequency of the processor (assuming that the processor supports frequency scaling that has been enabled) to save power. This will cause the TSC to go slower and might result in inconsistent time values. In this situation, the kernel will select some other clock source instead of TSC. Table 7 shows the change in clock skew estimates caused by the change of power source. To avoid these scenarios, for all our measurements, we use ACPI PMT clock source as this clock source is available in almost all modern laptops and its frequency does not get affected by external events including a switch to battery power.

6 Fabrication of Clock Skews

Our approach to detect a malicious AP is based on the clock skew of the AP. As an AP broadcasts beacon packets, an attacker can also listen to those packets, and then, calculate the relative clock skew of the AP with respect to its own clock skew. Using this clock skew estimate, an attacker can try to masquerade as the original AP by generating fake time stamps by adding proper offsets (those calculated from the measured skew) to its own time stamp. Let S denote the relative skew of the original AP as calculated by the attacker. Now the attacker can read its own time stamp T_i and try to generate fake sequence of time stamps TF_i using the following equation:

$$TF_i = T_i + S * T_i. \tag{8}$$

There can be two scenarios where an attacker can try to fake an original AP based on whether the original AP is active at the time of attack or not. If the attacker and the original AP are both active at the same time, the attacker's beacon frames will get mixed with the beacons sent by the original AP. As the attacker cannot control the time when the original AP sends its beacons, some of the beacons from the attacker might reach the receiver earlier than the beacons from the original one and some might reach later. As a result, the calculated skew will differ from the skew of the original AP (as shown in Table 5) and the attacker can be detected.[6]

Now, consider the scenario where only the attacker is active and it is fabricating time stamps by using the relative skew of the original AP that it calculated when the original AP was active. In order to test how accurately the attacker can fabricate time stamps, we examine systems that use the open-source MadWifi and Intel 3945ABG drivers. In these systems, channel sensing is done by the wireless hardware for the performance reasons. Furthermore, the time stamp in the beacon packets is set by the hardware when it actually transmits the packet. None of the wireless hardware supported by these drivers allow the time stamp to be set by software. However, these drivers support a mode called the *raw packet injection* mode, where the drivers can transmit any byte stream as a link layer frame without any modification. Thus, an attacker can send beacon frames with forged time stamps using this mode. Even with this capability, an attacker cannot fabricate the original APs clocks skew as we explain below.

In an IEEE 802.11 wireless network medium access control, before sending any frame, the sender is required to sense the channel for any other ongoing communication. If the sender finds the channel to be idle for the Distributed Interframe Sequence (DIFS) duration, the sender delays its transmission by a number of random time slots. The length of each time slot is chosen from the interval [0, CW] (where CW is the contention window size). If the channel is still idle after the random delay, depending on configuration, either the sender does the Request to Send (RTS)/Clear to Send (CTS) handshake, and then, sends the data, or directly sends the data bypassing RTS/CTS handshake. These two random delays, waiting time for the medium to be free and random back off time before actual transmission, make the exact time between when a wireless frame is handed over to

6. Additionally, the sequence number of the received beacons will not increase monotonically as it should if only one AP is active as shown by [27].

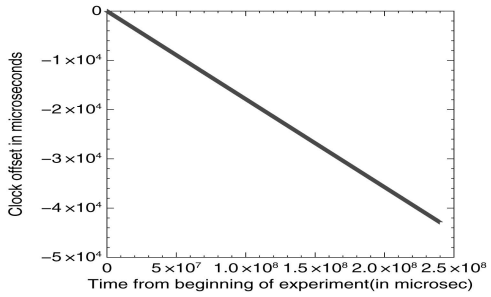

Fig. 7. TSF clock offset-sets for the original AP. Clock skew estimation for this AP is -178.83 ppm (using LPM).

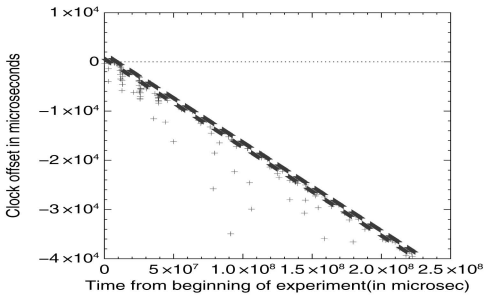

Fig. 8. TSF clock offset-sets for the attacker with forged time stamps. Clock skew estimation is -35 ppm (using LPM).

the driver and when it is actually sent unpredictable. Therefore, the forged time stamp used by the attacker will not reflect the actual time of transmission, and thus, will not result in the same clock skew as that of the original AP.

To test the effectiveness of clock skew fabrication quantitatively, we first measure the clock skew of an AP from an attacker PC. The RTS/CTS mechanism is disabled. We also modify the *rfakeap* program [28] to send beacon packets with forged time stamps created by offsetting the attacker's time stamp with the skew of the original AP measured by the attacker. We shut down the original AP and run this modified rfakeap program on the attacker PC. We calculate the clock skew of the attacker PC based on the time stamps in the rfkeap beacons. We show the results of our clock skew calculations in Figs. 7 and 8. As expected, we see that the attacker's clock skew using forged time stamps differs significantly from the skew of the original AP.

One might be able to design a wireless card in the future that allows beacon time stamps to be directly set by software. We now argue that even when armed with such a wireless card, it will be hard for an attacker fabricating the clock skews to go undetected. In an IEEE 802.11 network, an AP schedules transmission of a beacon frame every beacon interval. The time instant at which an AP schedules transmission of a beacon is called the Target Beacon Transmission Time (TBTT). IEEE 802.11 defines time zero as a TBTT. The subsequent TBTT values are multiples of the beacon interval. Now, even though each beacon is scheduled to be sent at a TBTT, the actual time at which a beacon is transmitted depends on the time to process the beacon and the time to acquire the shared medium. The actual time at which the beacon is transmitted is included in the beacon. Therefore, based on the beacon number and the beacon interval and the actual time of beacon transmission, a receiver (e.g., a WIDS node) can determine the delay between scheduling a beacon and the actual transmission of the beacon. Let T denote this delay. Let T_B be the beacon processing delay and T_C be the contention delay in acquiring the wireless medium. Then, $T = T_B + T_C$. Note that in systems running the MadWifi and the Intel 3945ABG drivers, the beacon frames are prioritized over data frames. The beacon frames and the data frames have separate hardware queues. Thus, the number of data frames in the data queue has no impact on the actual beacon transmission time.

A WIDS node that observes a large number of beacon frames can find the minimum values of T. This minimum value corresponds to the situation where the medium contention time T_C is minimum. Now, when an attacker armed with the capability to directly set beacon time stamps wishes to fake the clock skew of an AP, it must calculate the actual offset by performing a floating point multiplication and an addition/subtraction operation (as shown in (8)). These operations must be performed by the embedded processor in the wireless card which will increase the T_B value thereby increasing the minimum value of T. For the typical 150-250 MHz processors [29], T_B will increase at least by a few microseconds. This increase in the minimum value of T can be detected at the WIDS node. Currently, a special wireless card that allows beacons time stamps to be directly set by software does not exist. Hence, we cannot verify our argument in a real implementation.

7 Use of Clock Skew in Wireless Ad Hoc Networks to Identify Individual Nodes

In this section, we explore the possibility of using clock skews to uniquely identify different devices participating in a wireless ad hoc network. According to the IEEE 802.11 protocol specifications, all nodes in an ad hoc network must broadcast beacon packets periodically containing time stamps according to their own clock. The time stamps in these beacon packets are meant for synchronizing the clocks of all nodes. Each participating device periodically synchronizes its clock using the beacon time stamps it receives, by applying a clock synchronization algorithm that ensures the monotonicity of each node's clock. As mentioned in Section 6, in 802.11 infrastructure networks, beacons are only sent at TBTT. Similarly, in an IEEE 802.11 ad hoc network, to avoid collision while sending these beacon frames, each node waits for TBTT before attempting to send a beacon packet. At TBTT, each node backs off for a random amount of time before sending the beacon. During this random time interval, if a node detects any other node transmitting a beacon, it cancels its transmission. If a node does not detect any other node transmitting beacon packets during the entire random time interval, it sends its own beacon packet. After receiving a beacon packet, each node updates its clock according to Algorithm 4:

Algorithm 4. Beacon generation and Clock synchronization in IEEE 802.11 ad hoc networks

At each TBTT calculate a random delay and wait for that period.

if a beacon arrives within the delay **then**

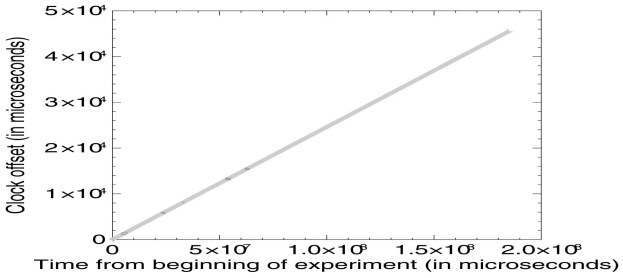

Fig. 9. Clock offset-sets calculated from beacon packets sent by two nodes participating in our ad hoc network. Clock skew estimates (214.89 and 215.11 ppm) are very close due to fast synchronization.

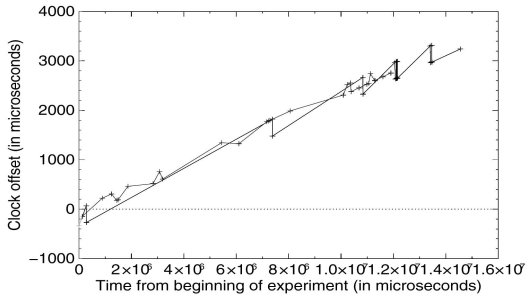

Fig. 10. Clock offset-sets calculated from probe response packets sent by two nodes participating in our ad hoc network.

if beacon's time stamp > local clock's time stamp **then**
  Update local clock's time stamp to the beacon's time stamp
end if
else
  Send a beacon with local clock's time stamp
end if

This algorithm ensures that over a period of time, the clock of each node will catch up to the fastest clock. This frequent synchronization makes it very difficult to estimate the accurate clock skew from beacon time stamps as all the clock skew estimates tend to be close to the clock skew of the fastest clock. To test the effect of the synchronization mechanism on our algorithms that we describe in earlier sections of this paper, we use a simple two-node IEEE 802.11 wireless ad hoc network. We collect beacon frames sent by each of the two nodes at the other node. According to Algorithm 4, a node's clock will synchronize with the time stamps sent by the other node with the faster clock. Our two-node ad hoc network will result in a fast synchronization of the slower clock. In larger networks, all but the node with the slowest clock will synchronize slowly because for each of these nodes, there will be some slower nodes whose time stamps will be ignored by that particular node. Furthermore, in larger networks, the opportunities to transmit beacon packets might be missed due to higher wireless medium contention. We wish to study the effect of fast synchronization on our algorithms in order to understand their applicability in ad hoc settings. Our two-node testbed suffices for this purpose. Fig. 9 shows the effect of fast synchronization on our clock-skew-based scheme. We observer that the clock offset-sets of two nodes almost overlap each other. The estimated clock skews of the two nodes are thus very close to each other. This shows that in wireless ad hoc networks, it is very difficult to calculate accurate clock skews of participating nodes using beacon packet time stamps.

One of the possible ways to solve this problem is to collect time stamp samples from a node's clock more frequently. However, beacon packets are only sent after a beacon interval. If we decrease the beacon interval to increase the frequency of time stamp samples, the synchronization process will also become faster which will not help our cause. To address this problem, we explore the use of *probe response* packets instead of beacon packets, which also contain the same TSF timer time stamps as the beacon packets. However, unlike beacon packets, a node in an ad hoc network sends a probe response packet whenever it receives a probe request packet. Therefore, we can send probe request frames and can get time stamp values from the resulting probe response packets at a faster rate than using the beacon packets. However, we also note here that the fingerprinter's clock does not have unlimited accuracy. Therefore, if we send probe request packets too fast causing the probe response time stamps to be very close to each other, the estimated clock offset-set will not be accurate due to the errors caused by the measurement process. We test this phenomenon in our two-node ad hoc network by allowing probe requests to be sent by the two nodes as fast as the hardware and the medium allow. Fig. 10 shows the clock offsets we obtain from probe response packets. We find that the slower clock (i.e., the clock with lower clock skew) gets periodically synchronized with the faster clock. However, the offset-set of the faster clock also shows irregularities unlike the offset-set calculated from the beacon packets. These irregularities are caused by the errors introduced by the limited accuracy of the measurement process as mentioned earlier. Our results show that probe requests should be sent at a rate that is low enough to minimize the measurement errors compared to the measured clock offset-set but high enough to generate enough probe response frames before the clock gets synchronized, to allow us to estimate the clock skew accurately.

In this paper, we only show the results from a two-node ad hoc network. However, for larger networks, with the increase in the number of nodes, we expect the synchronization interval for most of the nodes to increase due to the monotonic synchronization algorithm (Algorithm 4), and higher medium contention time. Therefore, in larger ad hoc networks, it might be possible to gather enough probe response packets from participating nodes to estimate their clock skews accurately before their clocks get synchronized. We plan to devise a practical algorithm to estimate a node's clock skew accurately in larger ad hoc networks in the future as an extension to our current work.

8 Related Work

For understanding the related work on detecting unauthorized APs, we first distinguish between *rogue* APs and *fake* APs. A rogue AP is set up by some naive user for convenience and higher productivity [1], [2], [3], [4]. If this AP's security is not carefully managed, this seemingly innocuous practice opens up the network to unauthorized wireless hosts, who can now become part of the network and launch different types of attacks. In contrast, a fake AP

is set up by a malicious attacker to masquerade as an authorized AP. In this paper, we focus on fake APs. Currently, there are two main methods for detecting rogue APs—one that monitors wireless networks either manually or in an automated fashion by sniffing wireless frames to detect rogue APs based on MAC address, BSSID, and SSID-based filtering [1], [7], [3], [8], [4], [30], and the other that monitors IP traffic to differentiate wireless network access from wired access using interpacket delay patterns [31], [32], [33]. However, these approaches are ineffective in detecting fake APs mainly because all of the identity fields (e.g., MAC address) can be easily spoofed.

Bahl et al. [27] proposed a method to detect fake APs by monitoring the anomaly in the monotonicity of the "sequence number" field of beacon frames sent by the authorized AP and the fake AP which is masquerading as the authorized one. However, this method can only detect the presence of a fake access point; on the contrary, our scheme can detect and separate out packets from fake AP. Another serious drawback of this method is that it will only work if both the authorized AP and the fake AP are active at the same time. Bahl et al. [27] also suggested the use of a location detection algorithm to detect the fake AP if the authorized AP is inactive at the time of detection. The accuracy of this method depends on the accuracy of the location detection algorithm. If the fake AP operates at a location that is very close to the authorized AP's working location, then this location detection method will be ineffective. Our solution removes these constraints and detects unauthorized APs in realistic scenarios. Yin et al. proposed a method for detecting rogue APs that also act as layer 3 routers. However, this work is also vulnerable to MAC spoofing. Franklin et al. [34] introduced a technique to fingerprint wireless device drivers. However, an attacker can also use fake APs with the same wireless device drivers by choosing the same model and the same manufacturer as the original one to evade detection.

Our use of clock skew to fingerprint a remote device is not new. Kohno et al. [9] have already shown that clock skew can be used as a reliable fingerprint for a device. However, our contribution is significant because we apply the clock-skew-based fingerprinting to a scenario where the detections are much faster, accurate, and less vulnerable to spoofing attacks compared to Kohno's original scenario that uses TCP time stamps.

9 Conclusions and Future Work

In this paper, we explored the use of clock skews to detect unauthorized access points in wireless local area networks. We developed a methodology that benefits from higher precision time stamps and higher predictability in a local area setting. We evaluated this methodology using traces from the ACM Sigcomm 2004 conference and two different residence areas. We showed that our high-precision skew estimation is an order of magnitude faster and uses an order of magnitude less packets compared to the existing TCP-/ICMP-based techniques [9]. We also discussed and quantified the impact of various external factors including temperature variation, virtualization, and NTP synchronization on clock skew. We also explored the possibility of engineering clock skews to allow a fake AP to generate the clock skew of the original one. Our exploration results indicate that the use of clock skews appears to be an efficient and robust method for detecting fake APs in WLANs. We also used our clock-skew-based fingerprinting technique in wireless ad hoc setting to identify individual nodes and showed that it is more difficult to estimate a node's clock skew accurately due to periodic clock synchronization among the nodes. As part of future work, we plan to devise a practical algorithm to estimate a node's clock skew accurately in ad hoc wireless networks where the number of participating nodes is large enough to slow down the clock synchronization process. Our solution addresses the problem of detecting fake APs effectively, but the general problem of finding a noncrypto method to detect MAC address spoofing by any wireless host still remains an interesting open problem.

Acknowledgments

This research was supported in part by ONR/ARL MURI grant W911NF-07-1-0318.

References

[1] "AirDefense, Wireless Lan Security," http://airdefense.net, 2009.
[2] "AirWave Management Platform," http://airwave.com, 2009.
[3] "Cisco Wireless LAN Solution Engine (WLSE)," http://www.cisco.com, 2009.
[4] "Rogue Access Point Detection: Automatically Detect and Manage Wireless Threats to Your Network," http://www.proxim.com, 2009.
[5] "Raw Glue AP," http://rfakeap.tuxfamily.org, 2009.
[6] C. He and J.C. Mitchell, "Security Analysis and Improvements for IEEE 802.11i," *Proc. Ann. Network and Distributed System Security Symp. (NDSS)*, 2005.
[7] "AirMagnet," http://www.airmagnet.com, 2009.
[8] "NetStumbler," http://www.netstumbler.com, 2009.
[9] T. Kohno, A. Broido, and K.C. Claffy, "Remote Physical Device Fingerprinting," *IEEE Trans. Dependable Secure Computing*, vol. 2, no. 2, pp. 93-108, Apr.-June 2005.
[10] S.J. Murdoch, "Hot or Not: Revealing Hidden Services by Their Clock Skew," *Proc. Conf. Computer and Comm. Security (CCS '06)*, pp. 27-36, 2006.
[11] S.B. Moon, P. Skelly, and D. Towsley, "Estimation and Removal of Clock Skew from Network Delay Measurements," technical report, Univ. of Massachusetts at Amherst, 1998.
[12] *IEEE Standard 802.11—Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, The Institute of Electrical and Electronics Engineers, Inc., 1999.
[13] *IEEE Guide for Measurement of Environmental Sensitivities of Standard Frequency Generators*, IEEE Standards Coordinating Committee 27-SCC27- on Time and Frequency, 1995.
[14] P. Hough, *Method and Means for Recognizing Complex Patterns*, US Patent 3069654, 1962.
[15] D.H. Ballard, "Generalizing the Hough Transform to Detect Arbitrary Shapes," *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms*, pp. 714-725, Morgan Kaufmann, 1987.
[16] L. Xu and E. Oja, "Randomized Hough Transform (RHT): Basic Mechanisms, Algorithms, and Computational Complexities," *CVGIP: Image Understanding*, vol. 57, no. 2, pp. 131-154, 1993.
[17] A.P. Dempster, N.M. Laird, and D.B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *J. Royal Statistical Soc.*, vol. 39, no. 1, pp. 1-38, 1977.
[18] "tcpdump," http://www.tcpdump.org/, 2009.
[19] "MadWifi—Multiband Atheros Driver for WiFi," http://madwifi.org, 2009.
[20] "Intel PRO/Wireless 3945abg Driver for Linux," http://ipw3945.sourceforge.net, 2009.
[21] "Linux Kernel Source Code," http://www.kernel.org, 2009.
[22] M. Rodrig, C. Reis, R. Mahajan, D. Wetherall, J. Zahorjan, and E. Lazowska, "CRAWDAD Dataset of Wireless Network Measurement," *Proc. SIGCOMM '04*, Oct. 2006.
[23] "DD-WRT," http://www.dd-wrt.com, 2009.

[24] A. Pásztor and D. Veitch, "PC Based Precision Timing without GPS," *SIGMETRICS Performance Evaluation Rev.*, vol. 30, no. 1, pp. 1-10, 2002.
[25] "Network Time Protocol Version 4 Reference and Implementation Guide," http://www.eecis.udel.edu/emills/database/reports/ntp4/ntp4.pdf, 2009.
[26] D. Bovet and M. Cesati, *Understanding the Linux Kernel*, third ed. O'Reilly Media, Inc., Nov. 2005.
[27] P. Bahl et al., "Enhancing the Security of Corporate Wi-Fi Networks Using DAIR," *Proc. MobiSys*, pp. 1-14, 2006.
[28] "Raw Fake AP," http://rfakeap.tuxfamily.org, 2009.
[29] "Broadcom Product Brief BCM-5354," http://www.broadcom.com/collateral/pb/5354-PB01-R.pdf, 2009.
[30] A. Adya et al., "Architecture and Techniques for Diagnosing Faults in IEEE 802.11 Infrastructure Networks," *Proc. ACM MobiCom*, pp. 30-44, 2004.
[31] R. Beyah et al., "Rogue Access Point Detection Using Temporal Traffic Characteristics," *Proc. IEEE Global Telecomm. Conf. (GLOBECOM)*, Dec. 2004.
[32] C. Mano et al., "Ripps: Rogue Identifying Packet Payload Slicer Detecting Unauthorized Wireless Hosts through Network Traffic Conditioning," *ACM Trans. Information and System Security*, vol. 11, no. 2, 2007.
[33] W. Wei et al., "Passive Online Rogue Access Point Detection Using Sequential Hypothesis Testing with TCP ACK-Pairs," *Proc. Internet Measurement Conf. (IMC)*, pp. 93-108, 2007.
[34] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J.V. Randwyk, and D. Sicker, "Passive Data Link Layer 802.11 Wireless Device Driver Fingerprinting," *Proc. 15th Conf. USENIX Security Symp. (USENIX-SS '06)*, pp. 12-12, 2006.

Suman Jana received the bachelor's degree in computer science from Jadavpur University, India. He is currently working toward the master's degree at the School of Computing, University of Utah. His primary research interests are in the fields of network security and computer systems.

Sneha K. Kasera received the master's degree in electrical communication engineering from the Indian Institute of Science Bengaluru, and the PhD degree in computer science from the University of Massachusetts Amherst. He is an associate professor in the School of Computing at the University of Utah in Salt Lake City. From 1999 to 2003, he was a member of technical staff in the Mobile Networking Research Department of Bell Laboratories. He has held research and development positions at Wipro Infotech and Center for Development of Advanced Computing at Bengaluru, India. His research interests include computer networks and systems encompassing mobile and pervasive systems and wireless networks, network security and reliability, social network applications, overload and congestion control, multicast communication, Internet measurements, and inferencing. He is a recipient of the 2002 Bell Labs President's Gold Award for his contribution to wireless data solutions. He has served in many technical program committees including those of ACM Mobicom, ACM Sigmetrics, IEEE Infocom, IEEE ICNP, and IEEE SECON, and among others. He is an associate editor of ACM Sigmobile MC2R, and also serves in the editorial boards of ACM/Springer WINET and Elsevier COMNET Journals.