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

Suman Jana
School of Computing
University of Utah
suman.jana@utah.edu

Sneha K. Kasera
School of Computing
University of Utah
kasera@cs.utah.edu

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) timestamps 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 timestamp data from several APs in two 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 timestamp 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, 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.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Measurement, Security

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

MobiCom'08, September 14–19, 2008, San Francisco, California, USA.

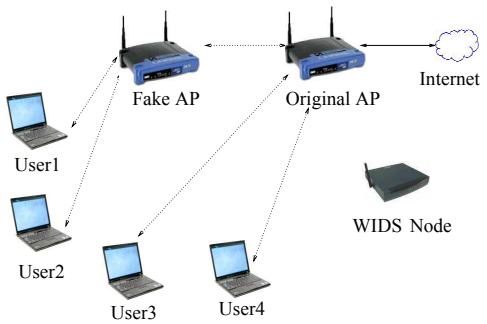

Figure 1: A Fake AP attack scenario

Keywords

IEEE 802.11, Fingerprint, MAC address spoofing, Fake access point, Timestamp

1. INTRODUCTION

With advances in micro-technology 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 [2, 3, 10, 15]. A fake AP can be set up by a malicious attacker (Figure 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* [7] 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 (WIDS).

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 is 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 non cryptographic 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 non-crypto methods for unauthorized AP detection [2, 8, 10, 14, 15] cannot detect fake APs.

In this paper, we explore a passive online scheme that can detect fake APs with high accuracy and minimum overhead. This scheme, like the one proposed by Kohno *et al.* for fingerprinting personal computers and servers [25], is based on estimating clock skews of APs. An AP's clock skew acts as its fingerprint. Kohno [25, 28] has 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 timestamps, in our scheme, we use the Time Synchronization Function (TSF) timestamps 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 one microsecond which is much higher than that of TCP timestamp clocks. Third, as the beacon timestamp 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 timestamps. Therefore our scheme estimates more accurate clock skews and much faster compared to the TCP/ICMP timestamp approach [25]. 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.* in [27] and later used by Kohno in [25]. This method finds a line that upperbounds all the time offsets calculated from the timestamps 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 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 timestamp values of different APs as well as the different rate of increment of those values.

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 two 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 timestamps, 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 pre-computed 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. We survey the existing work on detecting unauthorized APs in Section 7. We conclude the paper in Section 8 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

[1]This security rollback/downgrade attack is possible in 802.11i RSNA networks [23].

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 timestamps at a very fine granularity. We discuss this issue in more detail in Section 6.

3. METHODOLOGY

In an IEEE 802.11 infrastructure wireless local area network (WLAN), there are two methods that a client station (STA) may use to find an AP in the WLAN [1].

- **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 timestamp as shown in Figure 2. The timestamp field contains the value of Timer Synchronization Function (TSF) timer of the AP when it sends the frame. The beacons are scheduled to be sent at periodic intervals by the APs. The timestamps in the beacon frames do not get affected by the random medium access delays of the wireless medium as drivers set the timestamp 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 timestamps in beacon/probe response fra-
mes to estimate the clock skew of an AP and uses the clock skew as the AP's fingerprint.

Let us assume that a fingerprinter node (a WIDS node), has received n beacon frames from a particular AP. Let the timestamp in the i^{th} beacon frame be T_i and let t_i be the time in microsecond when the fingerprinter receives the i^{th} beacon frame. Let S_i be the size of the i^{th} beacon frame and let R_i be the data rate at which i^{th} 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 i^{th} frame be denoted o_i and the time difference between the first received frame and the i^{th} frame according to the fingerprinter's clock be x_i . Then,

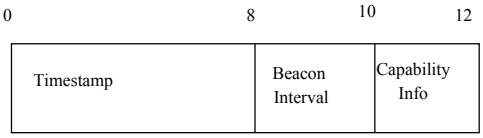

Figure 2: Structure of Fixed fields in Beacon/Probe Response Frame

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

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

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 [1]. 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 [27] and later used by [25] 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 2 variables.

The LPM method minimizes the effect of any unpredictable delays as it has higher tolerance towards outliers. The clock skew estimate remains stable even if there are significant number of outliers. However, in our case the number of outliers are very less because no significant random delay is involved in the communication path and the TSF clocks have a higher precision than TCP timestamp 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 towards 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 [25] 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 make 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 using multiple fake APs to fake one authorized AP, we can detect it 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) [24, 19] 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) [32] 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 [21]. However, the EM algorithm requires the initial parameters to be guessed and the accuracy of the results depend on the values of those 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 above mentioned 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.

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 timestamp resolution) is enough to separate a wide variety of datasets. Our results show that the value of threshold estimated by the test data depends on the method we use to generate the receiver's timestamps. 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 timestamp then the value of threshold becomes 0.05.

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

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

```
accumulator[0].dataset ⇐ [(x_1, y_1)]
accumulator[0].current_point ⇐ (x_1, y_1)
accumulator[0].count ⇐ 1
for i = 2 to n do
  for each entry j in accumulator do
    (tempx_k, tempy_k) ⇐ accumulator[k].current_point
    if Δ_ik ≤ threshold then
      add (x_i, y_i) to data set of accumulator entry j
      accumulator[j].count ⇐ accumulator[j].count + 1
      accumulator[j].current_point ⇐ (x_i, y_i)
    end if
  end for
  if none of the entry in accumulator satisfies (Δ_ik ≤ threshold) then
    add a new accumulator entry p
    p.dataset ⇐ [(x_i, y_i)]
    p.count ⇐ 1
    p.current_point ⇐ (x_i, y_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 ⇐ 0
for each test data set do
  threshold ⇐ Δ_12
  for i=3 to n do
    if Δ_i(i−1) ≥ threshold then
      threshold ⇐ Δ_i(i−1)
    end if
  end for
  if threshold ≥ finalthreshold then
    finalthreshold ⇐ threshold
  end if
end for
output finalthreshold as final calculated threshold
```

Once the datasets 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 timestamps, 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' source code is available. 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* [16], to find the arrival time of a frame. Even though this mechanism does not require any changes in the system, we note that the timestamp generated by tcpdump includes variable processing time of Operating System, interrupt latency. Therefore, use of tcpdump timestamp is not suitable for our purpose.

Next, we explore using the Prism monitoring headers in the MadWifi driver [5] and the Intel 3945ABG driver [4]. These drivers allow additional Prism monitoring headers to be added to frames arriving at the wireless card which has a 4 byte timestamp 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 timestamp 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 [13]. 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 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 timestamp. However, a microsecond precision timestamp 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 timestamp 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 timestamp field of the Radiotap header [5, 4]. Both these drivers maintain a microsecond resolution TSF timer. However, this TSF timer is synchronized to the timestamps 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 store the timestamp in the 8 byte timestamp 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 timestamps 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 are from data collected during the ACM Sigcomm 2004 conference [30]. 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 [30]. As the Sigcomm dataset 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 [25] 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 two different residential settings each with multiple APs operating simultaneously. One residential setting (residential setting A) has 8 APs and another (residential setting B) has 21 APs from different manufacturers. We use two laptops that implement our wireless drivers, 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.

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

5.1 Results from the Sigcomm Trace

Each packet in the Sigcomm traces has a prism header which contains receive timestamp of that packet. As stated in Section 4, the timestamps 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 does 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.

Table 1: Skew estimates for samples (collected by *chihuahua*) with different sample sizes. The samples contain beacon frames sent by sigcomm-nat.

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

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 51.09-51.37 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 we can see 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 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 timestamps. 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. Figure 3 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 Figure 3).

Thus, we can see that even using lower resolution timestamps (i.e., jiffies) we can estimate clock skews fairly accurately. However we require 300 or more packets. In Section

[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.

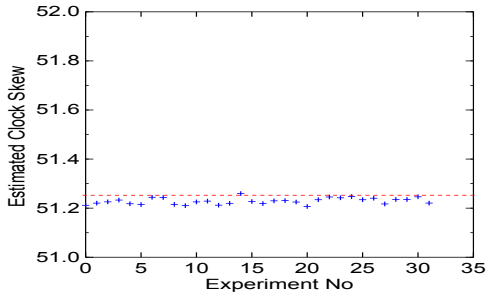

Figure 3: Skew estimates of samples containing 300 packets taken at different times by *chihuahua*. The samples contain beacon frames sent by sigcomm-nat.

Table 2: Skew estimates from Chihuahua. sigcomm-nat has 20,000 packets and sigcomm-public-2 has 7,500 packets.

AP	skew(using LPM)	skew(using LSF)
sigcomm-nat	51.25 ppm	51.20 ppm
sigcomm-public-2	48.82 ppm	48.90 ppm

Table 3: Skew estimates from Kalahari. sigcomm-nat-foyer has 30,000 packets, sigcomm-public-2 has 20,000 packets and sigcomm-public-1 has 1,200 packets.

AP	skew(using LPM)	skew(using LSF)
sigcomm-nat-foyer	44.91 ppm	45.00 ppm
sigcomm-public-2	42.69 ppm	42.98 ppm
sigcomm-public-1	49.94 ppm	49.34 ppm

Table 4: Skew estimates from Sonoran. Both APs have around 10,000 packets.

AP	skew(using LPM)	skew(using LSF)
sigcomm-nat-foyer	34.99 ppm	35.29 ppm
sigcomm-public-2	32.59 ppm	32.62 ppm

Table 5: Skew estimates from Mojave. Both APs have around 10,000 packets.

AP	skew(using LPM)	skew(using LSF)
sigcomm-nat-foyer	40.30 ppm	40.39 ppm
sigcomm-public-1	48.16 ppm	48.21 ppm

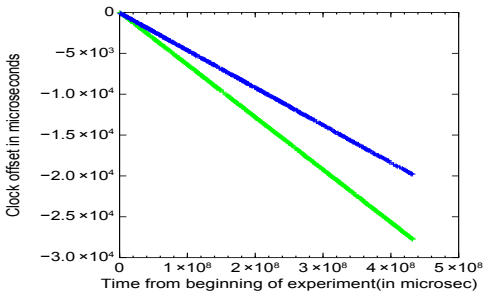

Figure 4: TSF clock offset-sets for two different Linksys APs. Clock skew estimations are -64.23 ppm and -45.69 ppm

5.2 we show that using higher resolution timestamp 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 Tables 2, 3, 4 and 5. 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*. Figure 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 8 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 6 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 setting B are shown in Table 7.

In both Table 6 and 7, we were able to estimate the clock skews accurately by analyzing 50-100 packets in most of the cases. Therefore, we find that microseconds resolution receiver timestamps, that we use in our methodology, results in a big improvement over millisecond resolution receiver timestamps 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 [25] where on an average, 1000-2000 packets were needed for a correct skew estimation. If we consider average time taken to estimate the skew, using higher precision timestamps in a more predictable WLAN setting takes

[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 6: Clock Skew estimates in residential setting A as measured from *laptop2*

AP	1st Measurement(LPM)	1st Measurement(LSF)	2nd Measurement(LPM)	2nd Measurement(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

Table 7: 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		

only 2-3 minutes whereas Kohno's clock skew estimates performed in a wide area setting with coarser timestamps [25] 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, the fingerprinter has no way of separating the packets. We analyze the effect of this intermingling on our estimation methods. We also test the efficiency of our algorithm for separating the packets using these synthetic data sets.

Table 8 shows that in some cases (e.g., case 1, 2 and 4), the skew estimated using LPM is same as the skew of one of the APs whose packets are intermingled. 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 timestamps LSF alone can be used to detect fake APs. However, if the receive timestamps are of low resolution, both LPM and LSF should be used. This is because LPM uses

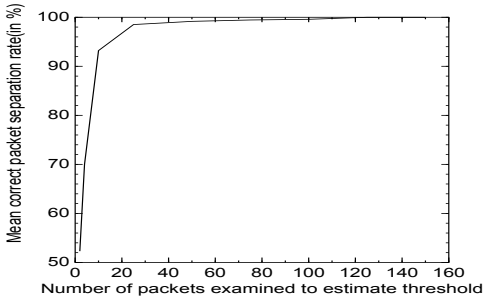

Figure 5: Mean correct packet separation rate versus number of packets examined to estimate threshold.

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 8 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. Figure 5

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% accurate packet separation on average (over the five synthetic traces used in Table 8). 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.

5.4.1 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 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 [25], all virtual APs being emulated on a particular hardware have the same clock skew, and that the clock skew is in the same range as the real AP clock skews. This happens because while sending the timestamp, 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 5 different APs (3 Trapeze networks APs running their default firmware and 2 Linksys WRT54G APs running the DD-WRT firmware [11]). We simulate 4 virtual APs on each of the 5 real APs. Our results, shown in Table 9, 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 [25, 28] that under normal PC operating temperatures the clock skew of a device remains constant within ±1 ppm. It has also been noted [28] that this temperature change can also occur due to varying processor load. However, Pasztor *et al* [29] have shown that for small time periods (less than 1000 seconds) the clock skew variance remains less than ±0.1 ppm. The results presented in another existing work [28] also supports 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 executes Algorithm 3 over each 50-100 beacon frame block. Since collection of 50-100 beacon frames typically takes much less than 1000s, 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

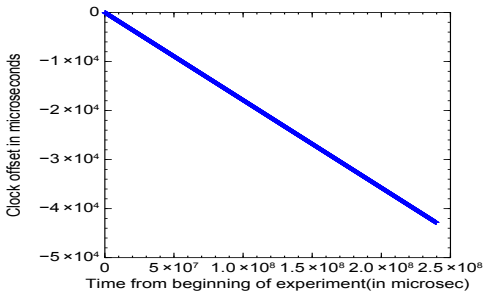

Figure 6: TSF clock offset-sets for the original AP. Clock skew estimation for this AP is -178.83 ppm (using LPM)

with a higher precision in comparison to the one used by Kohno [25].

Algorithm 3 Fake AP detection algorithm

```
Calculate newskew
if (newskew - currentskew) ≤ max skew variance then
    currentskew ⇐ newskew
    AP is original
else
    Fake AP detected.
end if
```

As, we measure relative skew between two physical clocks, extrapolating the findings of [29], we can set *max skew variance* to ±0.2 ppm. In our high precision residential traces, all but one pair of access points (Linksys5 and Trapeze Networks 1 in Table 7) 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 [25], 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 timestamps are measured in microseconds. NTPv4 is accurate within 10 milliseconds over the wide-area Internet and within 200 microseconds over a LAN. The default minimum polling interval for NTP is 64 seconds [12]. However in our case, as the timestamps 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.

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

Table 8: Measure of skew from the synthetic data set. All skews are absolute values. Please note that the skews estimated by LSF is extremely large because of the mixing which helps us to detect the presence of fake APs much faster than LPM.

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

Table 9: Skew of virtual APs

AP	Virtual AP1	Virtual AP2	Virtual AP3	Virtual 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

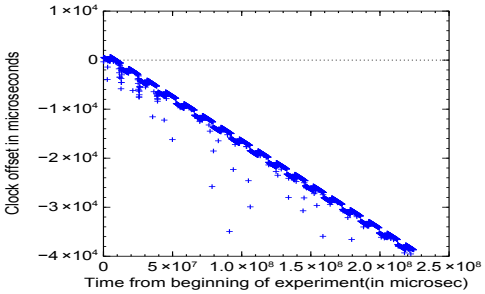

Figure 7: TSF clock offset-sets for the attacker with forged timestamps. Clock skew estimation is -35 ppm (using LPM)

timestamps by adding proper offsets (those calculated from the measured skew) to its own timestamp. Let S denote the relative skew of the original AP as calculated by the attacker. Now the attacker can read its own timestamp T_i and try to generate fake sequence of timestamps 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 8) and the attacker can be detected[5].

Now, consider the scenario where only the attacker is active and it is fabricating timestamps 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 timestamps, 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 performance reasons. Furthermore, the timestamp 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 timestamp 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 timestamps 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 Inter-Frame 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 the driver and when it is actually sent unpredictable. Therefore, the forged timestamp 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 (as we expect the attacker to do so because enabling it will cause even more random delays). We also modify the *rfakeap* program [6] to send beacon packets with forged timestamps created by offsetting the attacker's timestamp with the skew of the original AP measured by the attacker. We shut down

[5]Additionally, the sequence number of the received beacons will not increase monotonically as it should if only one AP is active as shown by [18].

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 timestamps in the rfkeap beacons. We show the results of our clock skew calculations in Figure 6 and 7. As expected, we see that the attacker's clock skew using forged timestamps differs significantly from the skew of the original AP.

One might be able to design a wireless card in the future that allows beacon timestamps 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 timestamps wishes to fake the clock skew of an AP, it must calculate the actual offset by performing a floating point multiplication and an addition/substraction 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 [9], 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 timestamps to be directly set by software does not exist. Hence, we cannot verify our argument in a real implementation.

7. 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 [2, 3, 10, 15]. 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 [2, 8, 10, 14, 15, 17], and the other that monitors IP traffic to differentiate wireless network access from wired access using inter-packet delay patterns [20, 26, 31]. 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.* [18] 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 [18] 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.* [22] 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.* [25] 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 timestamps.

8. 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 timestamps 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 [25]. 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. Our solution addresses the problem of detecting fake APs effectively, but the general problem of finding a non-crypto method to de-

tect MAC address spoofing by any wireless host still remains an interesting open problem.

9. REFERENCES

[1] IEEE Standard 802.11 - wireless LAN medium access control (MAC) and physical layer (PHY) specifications. The Institute of Electrical and Electronics Engineers, Inc., 1999.

[2] AirDefense, wireless lan security, http://airdefense.net.

[3] AirWave management platform, http://airwave.com.

[4] Intel PRO/Wireless 3945ABG Driver for Linux, http://ipw3945.sourceforge.net/.

[5] MadWifi- multiband atheros driver for WiFi, http://madwifi.org/.

[6] Raw Fake AP, http://rfakeap.tuxfamily.org/.

[7] Raw Glue AP, http://rfakeap.tuxfamily.org/.

[8] AirMagnet, http://www.airmagnet.com.

[9] Broadcom Product Brief BCM-5354, http://www.broadcom.com/collateral/pb/5354-PB01-R.pdf.

[10] Cisco wireless LAN solution engine(WLSE), http://www.cisco.com.

[11] DD-WRT, http://www.dd-wrt.com.

[12] Network Time Protocol version 4 reference and implementation guide, http://www.eecis.udel.edu/ emills/database /reports/ntp4/ntp4.pdf.

[13] Linux kernel source code, http://www.kernel.org/.

[14] NetStumbler, http://www.netstumbler.com.

[15] Rogue access point detection: Automatically detect and manage wireless threats to your network, http://www.proxim.com.

[16] tcpdump, http://www.tcpdump.org/.

[17] A. Adya, P. Bahl, and R. C. et al. Architecture and techniques for diagnosing faults in IEEE 802.11 infrastructure networks. In *MobiCom '04*, pages 30–44, 2004.

[18] P. Bahl, R. Chandra, and J. P. et al. Enhancing the security of corporate Wi-Fi networks using DAIR. In *MobiSys '06*, pages 1–14, 2006.

[19] D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Readings in computer vision: issues, problems, principles, and paradigms*, pages 714–725, 1987.

[20] R. Beyah, S. Kangude, and G. Y. et al. Rogue access point detection using temporal traffic characteristics. In *Proceedings of IEEE GLOBECOM*, December 2004.

[21] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[22] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. V. Randwyk, and D. Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, pages 12–12, Berkeley, CA, USA, 2006. USENIX Association.

[23] C. He and J. C. Mitchell. Security analysis and improvements for IEEE 802.11i. In *NDSS*, 2005.

[24] P. Hough. Method and means for recognizing complex patterns. U.S. Patent 3069654, 1962.

[25] T. Kohno, A. Broido, and K. C. Claffy. Remote physical device fingerprinting. *IEEE Trans. Dependable Secur. Comput.*, 2(2):93–108, 2005.

[26] C. Mano, A. Blaich, and Q. L. et al. Ripps: Rogue identifying packet payload slicer detecting unauthorized wireless hosts through network traffic conditioning. *ACM Transactions on Information and System Security*, 2007.

[27] S. B. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. Technical report, Amherst, MA, USA, 1998.

[28] S. J. Murdoch. Hot or not: revealing hidden services by their clock skew. In *CCS '06*, pages 27–36, New York, NY, USA, 2006. ACM.

[29] A. Pásztor and D. Veitch. PC based precision timing without GPS. *SIGMETRICS Perform. Eval. Rev.*, 30(1):1–10, 2002.

[30] M. Rodrig, C. Reis, and R. M. et al. CRAWDAD data set uw/sigcomm2004 (v. 2006-10-17). http://crawdad.cs.dartmouth.edu/uw/sigcomm2004, Oct. 2006.

[31] W. Wei, K. Suh, and B. W. et al. Passive online rogue access point detection using sequential hypothesis testing with TCP ACK-pairs. In *IMC*, pages 93–108, 2007.

[32] L. Xu and E. Oja. Randomized Hough transform (RHT): basic mechanisms, algorithms, and computational complexities. *CVGIP: Image Underst.*, 57(2):131–154, 1993.