On the Reliability of Wireless Fingerprinting using Clock Skews [*][†]

Chrisil Arackaparambil Sergey Bratus Anna Shubina David Kotz

Dept. of Computer Science, Dartmouth College
{cja, sergey, ashubina, kotz}@cs.dartmouth.edu

Dartmouth Computer Science Technical Report TR2010-661

Abstract

Determining whether a client station should trust an access point is a known problem in wireless security. Traditional approaches to solving this problem resort to cryptography. But cryptographic exchange protocols are complex and therefore induce potential vulnerabilities in themselves. We show that measurement of clock skews of access points in an 802.11 network can be useful in this regard, since it provides fingerprints of the devices. Such fingerprints can be used to establish the first point of trust for client stations wishing to connect to an access point. Fingerprinting can also be used in the detection of fake access points.

We demonstrate deficiencies of previously studied methods that measure clock skews in 802.11 networks by means of an attack that spoofs clock skews. We then provide means to overcome those deficiencies, thereby improving the reliability of fingerprinting. Finally, we show how to perform the clock-skew arithmetic that enables network providers to publish clock skews of their access points for use by clients.

1 Introduction

Clock skews are the inherent tiny drifts in the clocks of hardware devices due to variations in the manufacturing process. The use of clock skews of devices on a network for the purpose of fingerprinting those devices was first studied by Kohno, Broido, and Claffy [15]. They showed that it was possible to remotely measure the microscopic skews of devices, and that their fingerprinting method could identify individual devices despite errors inherent in remote measurements. Such fingerprinting has innumerable applications. For instance it is useful from the point of view of network forensics for identification purposes. It is also useful in penetration testing to identify network systems to know their weaknesses (e.g., the method of Kohno et al. can be used to identify virtual hosts served by the same physical device).

The study of Kohno et al. focused on the measurement of skews in wide-area networks by observing timestamps in TCP and ICMP packets. On the wireless side, Jana and Kasera [14] studied the approach of Kohno et al. at the MAC layer of 802.11 networks. They observed that, due to the essentially zero latency and the availability of a high frequency stream of high precision beacon timestamps, the process of measuring clock skews became more accurate and effective in these networks. They also showed

[*]This technical report is an expanded version of the paper that is to appear in Proceedings of ACM Conference on Wireless Network Security (WiSec), 2010.

[†]This article results from a research program in the Institute for Security, Technology, and Society (ISTS), supported by the U.S. Department of Homeland Security under Grant Award Number 2006-CS-001-000001. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security.

that the clock skews of wireless devices remain consistent over time and changing external factors like temperature, and that the skews vary across devices.

In the past, research on security of wireless networks centered around securing access points (APs) from unauthorized malicious clients, since APs were deemed vulnerable, exposed entities. But advances in wireless security using authentication have mitigated the threat of unauthorized access. Because APs are managed by the network provider, their security could be managed centrally, and care can be taken to ensure known vulnerabilities do not remain. There has since been a shift in the focus of attention towards protecting clients in wireless networks. An important threat in this respect is from faked APs.

The main application considered by Jana and Kasera [14] was that of detecting fake APs. Today, tools like `rglueap` and `rfakeap` [2] are readily available that make it easy for an attacker to set-up an AP that fakes a real one. Identifying fields in 802.11 frames like MAC address, BSSID, and SSID can be easily set to values desired by the attacker. A client attempting to associate with the real AP can be diverted to the fake AP, thus becoming vulnerable to various kinds of attacks. As pointed out before [13, 14], the attacker may also attempt to avoid detection of the fake AP by either operating on a channel different from the real AP, or by providing a higher signal strength to the client.

Our contributions

In this work we show how previous methods for measuring clock skews are inadequate for fingerprinting and provide a means to overcome the problems that arise. Our work provides new insights into the implementation of the 802.11 standard in commodity hardware. In particular, we present

- a new method to measure clock skew, rather, a more precise clock to measure it against;
- an attack that spoofs the clock skew of a fake AP to mimic that of a real one, thereby rendering the two indistinguishable by the methods proposed previously;
- additional parameters to measure the authenticity of the skew, enabling the detection and mitigation of spoofing attempts;
- clock skew arithmetic, that enables a network provider to publish skews of APs in the network independent of client stations.

2 The Role of Fingerprinting in Securing Wireless Infrastructure

Initially, 802.11 link layer security measures concentrated on preventing access of unauthorized clients to the network's APs. The entire concept of 802.11 authentication, association, and in particular the design of the 802.11 client state machine, proceeded from the apparent assumption that the primary goal of the security mechanisms was to protect the infrastructure of the network from rogue clients that would seek to obtain access to the infrastructure. The APs were apparently thought of as the "perimeter" of the network, vested with the role of protecting it against rogue clients.

However, subsequent experience showed that the threat model underlying this design was inherently flawed. Clients (with their stored representations of trust relationships) turned to be a much more important piece of the holistic security puzzle than previously thought. In fact, they emerged as the weakest link in the so-called perimeter.

In ISO Layer 3, attacking clients of a network and through them gaining access to the presumably well-protected internal network resources (by exploiting existing trust relationships between these resources and the clients) has emerged as an efficient attack strategy. In fact, exploiting clients by tricking them into establishing connections to rogue services became a leading strategy for both exploitation and penetration testing as evidenced by an entire BlackHat 2009 track (e.g., [18]) devoted to client exploitation functionality in the popular Metasploit penetration testing tool [3].

It did not take long till the same attack approach was realized in 802.11 Layer 2: trusted clients were tricked into interaction with fake access points, pretending to be a part of the trusted infrastructure. The trend towards exploiting the clients was amplified by the complex nature of the 802.11 link establishment. Empirically, vulnerabilities are associated with complexity of processing diversely structured inputs. 802.11 link layer driver code is exemplary of just such complexity. In particular, even beacon and probe response frames—to be processed by clients before any trust in the sender can be established—contain many variable-length optional Information Element structures, some of which are also vendor-specific. It is hardly surprising that crafting malformed inputs in these fields quickly emerged as an extremely efficient attack methodology in [9]. This methodology yielded such achievements as "hijacking a Macbook in 60 seconds" [8] (by way of a crafted probe response leading to attack code execution within the ring zero driver kernel context) and the subsequent automation and refinement of this technique that revealed other 802.11 driver vulnerabilities— the so-called "Month of kernel bugs" (see, e.g., [7]). As we explained above, wireless clients became a prominent part of the network's attacked perimeter even before they attempted to establish association with a trusted infrastructure! We remark that potential vulnerabilities in processing of complex data structures required for cryptographic authentication of the access points by the client are still largely unexplored and might provide another efficient attack vector.

In the light of the clients becoming the forefront of network exploitation, identifying the tools of such exploitation—fake access points— delivering crafted link layer inputs to the clients becomes very important. Rogue access points have long been seen as security threats; for example, non-security-minded employees may introduce unauthorized access points into organizations' networks for convenience and thus create a weak link in the network perimeter, or attackers may set up fake (or the so-called "evil-twin") access points to capture communications and conduct man-in-the-middle attacks between the unwitting client and the user. Popular exploitation tools, such as Karma [20], were developed to meet penetration testers' demand. Such early attacks were described by wireless security researchers in [13, 19]. However, all of these traditional fake AP scenarios assume successful establishment and maintenance of a layer 3 connection, whereas a new class of attacks is based on compromising the client at a much earlier point: either during scanning for available networks or during authentication or association attempts. As such, strong cryptographic schemes for authenticating access points, such as WPA2-Enterprise, cannot mitigate this threat. *Fake access points thus become a tool of delivering link layer exploits.*

Thus the problem of protecting 802.11 clients at their most vulnerable— in the early stages of establishing authentication/association —becomes paramount to the new client-centric view of network security. We note that at these stages the clients are most susceptible to deception, because they must make their decision to join a network based on easily fakeable data, such as the AP's MAC address, ESSID, and various Information Elements in the beacon and probe response frames, as well as physical layer characteristics, such as signal strength. (Creating superior signal strength is generally not a hard problem for an attacker.) Detecting such deceptions thus becomes important for both clients (where it needs to be easily and quickly accomplished as a pre-authentication step) and wireless intrusion detection systems (WIDS).

As we have seen, establishing trust for an AP can be a tricky issue for a client attempting to associate with it. Traditional approaches to such trust-relationship problems most often find solutions in cryptographic exchange protocols. With respect to wireless security, the 802.11i RSNA (Robust Security Network Association) provides such a functionality. Importantly however, such protocols that are dependent on cryptography are complex and therefore induce potential vulnerabilities in themselves. These protocols must be implemented with great care. Before involving in complex cryptographic exchange protocols with an untrusted entity, we propose using clock-skew fingerprinting as a means of providing a first point of trust for clients. Once the fingerprint of the AP is verified, the clients can proceed with such protocols to reaffirm their trust. As such, we propose our methods as a complement to the existing authentication methods.

Unlike Jana and Kasera's proposal [14], where the fake AP detection procedure was meant to be implemented in a WIDS node, in our work we even enable the measurements to be done on wireless clients themselves. To allow for this, we need to make use of clock skew arithmetic as described in Section 6.

In the following sections we describe our contributions in detail. The remainder of the paper is organized as follows. In Section 3 we outline the synchronization methods specified by the 802.11 protocol standard, and also some details on how these methods are implemented by commodity hardware and device drivers. We also present our methodology for measuring clock skews of APs in this section. Then, in Section 4 we present our attack that spoofs the clock skew of APs. We then present the two methods to detect attempts at clock skew spoofing in Section 5. In Section 6 we show how to do arithmetic with clock skews. Finally, we discuss related work in Section 7, and conclude in Section 8.

3 Measurement of Clock Skews

We first give an overview of the timing and synchronization processes in wireless networks as specified in the IEEE 802.11 standard [1]. These processes provide the timing information required to compute the clock skews of APs.

In an 802.11 network operating in infrastructure mode, every station maintains a timer. This timer is synchronized with the timer in the AP the station is associated with via a Timer Synchronization Function (TSF). The synchronization is achieved through the beacon frames transmitted by the AP at periodic intervals. The most common setting for the beacon interval is 100 milliseconds, so that beacons are commonly transmitted at the rate of 10 beacons/second. The beacon frames contain the TSF timer timestamp of the AP "at the time that the data symbol of the first bit of the timestamp is transmitted to the wireless medium," adjusting for hardware transmission delays. The timer is of microsecond resolution and is maintained as a 64-bit counter. Client stations set their local TSF timers to the values observed from beacon frames, again, adjusting for hardware delays. This means that the beacon timestamps provide a high-precision mechanism to measure the skew in the TSF timer of APs.

Clock skews

We now define the notion of clock skew as given by Moon, Skelly, and Towsley [17], and later used by Kohno et al. [15] and Jana and Kasera [14]. To measure the clock skew of an AP, we passively monitor the wireless interface of the measuring device for beacon frames from the AP. For beacon frame i we record the time t_i when it was received and the timestamp T_i in the beacon frame. In this manner we obtain a set of n measurements $(t_i, T_i), 1 \leq i \leq n$. The parameter n provides a tradeoff between the quality of measuring the skew and the time required to measure the skew. We found that sampling $n = 100$ beacons was sufficient in our experiments (as was also observed in [14]). This corresponds to an overhead of 10 seconds for measuring the clock skew with the common beacon interval of 100 milliseconds. We denote by x_i the elapsed time since the first beacon was observed, i.e., $x_i = t_i - t_1$. Similarly, if we let $w_i = T_i - T_1$, then the quantity $y_i = w_i - x_i$ is called the clock offset of the ith measurement. In this way we get a set of n points (x_i, y_i) representing the clock offsets. Ideally, there should be no relative skew between the measurer's clock and the beacon timestamps representing the AP's clock. In this case we would have that $w_i = x_i$ for all measurements so that all points would lie on the X-axis. In reality we observe that the clock offset points lie on an approximately linear pattern that has some non-zero slope. By approximating the slope of this linear pattern, we obtain the clock skew of the AP. Skews observed in practice are tiny, but consistent, and are reported in parts per million (ppm).

There are two methods used in practice to approximate the slope of the clock offset points, and both involve fitting a line $y = s \cdot x + c$ to the points and reporting the resulting slope s as the clock skew. The first one is the Linear Programming Method (LPM) [17, 15, 14]. This method produces a line that upper-bounds the set of clock offset points, while minimizing the sum of the distances of the points from the line. In other words, we have the optimization problem with constraints

$$s \cdot x_i + c \geq y_i, \ \ 1 \leq i \leq n,$$

and objective function

$$\sum_{i=1}^{n} (s \cdot x_i + c - y_i)$$

that needs to be minimized. This method chooses an upper bound that captures the effects of outliers due to network delays. This linear programming problem can be solved by standard LP-solvers, but the special 2-variable version we consider has a linear time solution [10, 16].

LPM is useful when the set of clock offset points contains many outliers, as is the case when the available measurements are not very accurate (e.g., when network delays play a role in the measurements). However, it was shown by Jana and Kasera [14] that since measurements of clock skews from beacon timestamps are very precise and do not suffer from these effects, a simple linear least square fit (LSF) suffices. LSF is a simple statistical regression method that fits a line $y = s \cdot x + c$ to the set of clock offset points (y_i, x_i) by minimizing the least square error

$$\sum_{i=1}^{n} (y_i - (s \cdot x_i + c))^2.$$

LSF was successfully applied in measuring clock skews from beacon timestamps [14]. The researchers also point out that for the specific application of measuring clock skews, the use of LPM may actually be undesirable because the attacker may be able to affect the outcome of the measurements, and spoof the clock skew of the real AP, by injecting a small number of beacons with carefully chosen timestamps that would appear as outliers. LSF also has the advantage that it can be implemented with low overhead—all that is required to fit the line are the sums $\sum x_i$, $\sum x_i^2$, $\sum y_i$, and $\sum x_i y_i$, all of which can be computed in an online fashion with about 7 arithmetic operations per beacon. So it may even be feasible to implement clock-skew fingerprinting with LSF in hardware.

Monitor mode synchronization

The timestamps in the beacon frames form one half of the required information for estimating clock skews. The measurement of the arrival time of the beacon frame is an important problem, since its accuracy impacts the accuracy of estimation. There were several clocks considered in [14] to report the beacon arrival time. It was observed that the timestamps reported by the `pcap` packet capture library were not accurate enough for measurement. Another clock considered was the `jiffies` counter maintained in the Linux kernel and reported in the Prism header of the frames. But this counter had a low (a few millisecond) resolution that was not sufficient. The researchers considered using the timestamp reported in the Radiotap header in the `pcap` field `radiotap.mactime`. This timestamp is reported by the driver from the TSF timer maintained by the wireless hardware. But the approach was abandoned since the timer values were updated from the incoming beacon timestamps and hence did not serve as a stable clock. The approach finally found to work was to use the time reported by the Linux kernel via the function `do_gettimeofday()`.

However, this method too suffers from some drawbacks. The `do_gettimeofday` function is implemented using timer interrupts, and is adjusted in the kernel for anticipated delays. So it could be expected to shift in accuracy, and is only as accurate as the underlying interrupt mechanism. Also, since

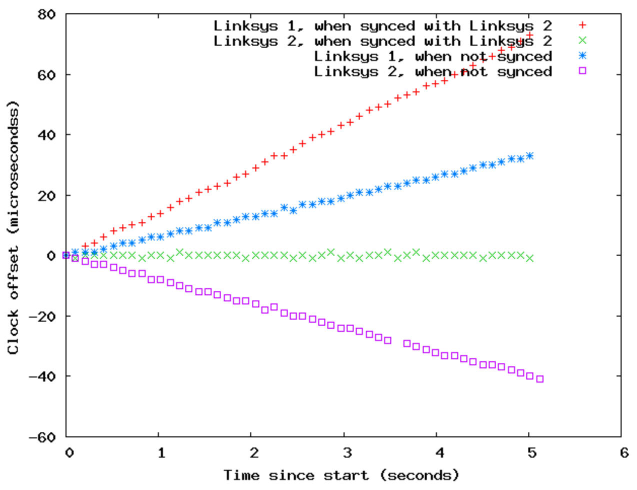


Figure 1: Clock offsets with and without the measuring station syncing its TSF timer with Linksys 2.

the skew of the clock represented by the function depends on the implementation of the function and underlying routines, we expect this skew to vary with the updates to the system. This would require clients to recalibrate their skew measurements before fingerprinting APs again. We have observed significant changes in the implementation of the `do_gettimeofday` function between kernel releases.

We present a new method of measuring the arrival time of beacons that is more accurate than using `do_gettimeofday`. Since this measurement is critical to estimating the clock skew, our method leads to more accurate measurements. Further the clock used in our method is implemented in the wireless hardware, and hence its skew does not change with software updates. Our method depends on the synchronization behavior of the Atheros chipset based cards that we use. We explain this behavior now. In the course of our discussion we state some observations and verify them empirically. These observations turn out to be crucial to our techniques described in later sections.

For the experiments in the rest of the paper we use two laptops as measurement stations. These laptops run the Ubuntu 9.04 GNU/Linux distribution (with kernel 2.6.28-15) and are each equipped with wireless card based on the Atheros 5212 chipset. Our choice of this chipset is dictated by the availability of the open-source Madwifi driver. The Madwifi driver `ath_pci` is used with these cards for the measurements. In our experiments we also use two Linksys APs, henceforth referenced as Linksys 1 and Linksys 2 respectively.

The monitoring typically performed to capture the beacon frames is done in the so-called monitor-mode of the wireless interface. The TSF timer maintained in the wireless hardware is a high-accuracy microsecond resolution timer, and it would serve best for our measurements of beacon arrival time, since its value is provided directly to the driver by the hardware and is not affected by other processes in the system. However, this timer was deemed as unusable in [14] because the timer was kept synchronized to the incoming beacon timestamps even in monitor mode. We now give a method to use this timer. It should be noted that in monitor mode, it is not necessary to synchronize the TSF timer with the incoming beacon timestamps, since the card is completely passive in this mode. However, cards with the Atheros chipset continue to synchronize with the beacon frames observed from the AP that the card was *last* associated with.

This leads to an interesting possibility: what happens when the AP that the card was last associated with becomes inactive and stops broadcasting beacon frames? In this case the timer on the card, not being able to synchronize with the beacon timestamps, should begin to drift with its *own* skew. And indeed, this is confirmed empirically with our experiments. We measured the skew of Linksys 1 and Linksys 2 in monitor mode, by first associating the measuring laptop with Linksys 2 and then switching

	Clock Skew
Linksys 1, when synced with Linksys 2	14.37
Linksys 2, when synced with Linksys 2	-0.01
Linksys 1, when not synced	6.68
Linksys 2, when not synced	-7.85

Table 1: Clock skew measurements with sample sizes of 100 beacon timestamps, with and without synchronization with Linksys 2. Figures rounded to the nearest hundredth.

the laptop to monitor mode. We then turned Linksys 2 off and again measured the skew of Linksys 1. Figure 1 shows the clock offsets points measured in the different cases, and Table 1 reports the measured clock skews. Observe that the estimated skew of Linksys 1 varied significantly before and after Linksys 2 was turned off. Also, the estimated skew of Linksys 2 was negligible. Note that to similarly measure the skew of Linksys 2 when the associated AP was turned off, we had to first associate with Linksys 1 and repeat the process. This leads us to the following observations.

Observation 1. *Given a wireless card in Station mode and associated with an AP A, when the card is switched to Monitor mode, it continues to update its TSF timer register with the beacon timestamps from AP A.*

Observation 2. *Given a wireless card in Station mode and associated with an AP A, when the card is switched to Monitor mode, if AP A ceases to transmit beacons, then the TSF timer maintained in the wireless card begins to drift with its own, actual skew.*

The next two observations follow as a consequence of Observation 1.

Observation 3. *Given a wireless card in Station mode and associated with an AP A, when the card is switched to Monitor mode, the clock skew of AP A as measured by the card is zero (imperceptible).*

Observation 3 suggests that the skew of the measuring card becomes equal to the skew of the AP it is synchronized with. From Table 1 it may further be observed that the skew of Linksys 1 when measured by the card synchronized with Linksys 2 is approximately equal to the difference of the skews of Linksys 1 and Linksys 2 when there is no synchronization. We have observed this behavior consistently with different APs; we omit the data for the sake of brevity. This indicates that it is possible to compute the skew of a wireless device as measured by another, by passively measuring the skews of the two devices. This brings us to our next observation.

Observation 4. *Given a wireless card in Station mode and associated with an AP A, when the card is switched to Monitor mode, the clock skew of another AP B as measured by the card is equal the skew of AP B as would be measured by AP A.*

The issue of performing arithmetic to determine the skew between a pair of wireless devices is covered in more detail in Section 6.

Our measurement technique

The observations listed earlier give us a new method of measuring beacon arrival times—using the TSF timer to do it. For the experiments in the rest of the paper that use the TSF timer, we use the timer by first associating with an AP and then switching off power to the AP. On a client station the same effect can be achieved by either removing and re-inserting the wireless card, or even entirely through software by removing and reloading the driver modules. It may even be possible to power-cycle the card and flush the state through the driver interface, but we have not verified this.

	Mean	Variance
TSF Timer	6.7011	0.0001245
`do_gettimeofday`	-28.1347	0.0659

Table 2: Mean and variance of 10 clock skew measurements (ppm) using LSF with the two clocks. Beacon timestamp sample size is 100.

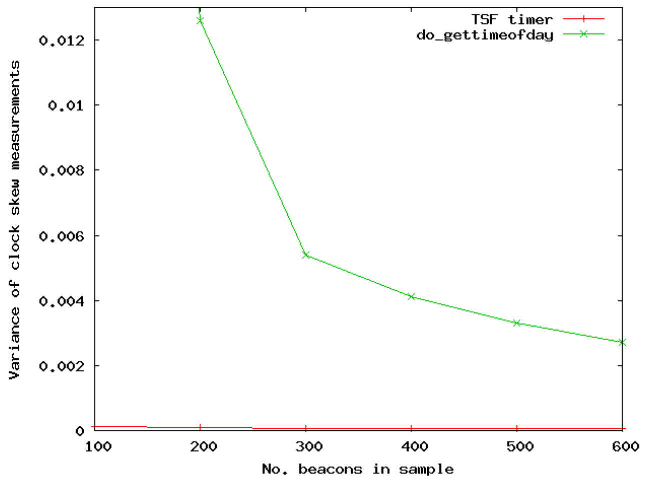


Figure 2: Variance in clock skew measurements using LSF as a function of the beacon timestamp sample size.

Our next experiments show that using the TSF timer yields much higher accuracy than using `do_gettimeofday`. To compare the two methods we collected 10 sets of beacon traces with each method. As in [14] we disable NTP to avoid its effect on the `do_gettimeofday` method. For each set of traces we measured the clock skew of Linksys 1 using sample sizes ranging from 100 beacons to 600 beacons. Note that at the standard beacon interval of 100ms, it takes 1 minute to observe 600 beacons. Then for each set of traces, and each sample size we computed the mean clock skew and the variance of the clock skew. Our observations are presented in Table 2 and Figure 2 (using LSF), and in Table 3 (using LPM). Observe that the variance of the clock skew when using the TSF timer is consistently several orders of magnitude smaller that the variance when using the `do_gettimeofday` function to report the beacon arrival times. This points to the superior stability and accuracy of the TSF timer method.

#beacons	tsf-Mean	tsf-Var	gtod-Mean	gtod-Var
100	6.70	4.62e-04	-36.70	1000
200	6.70	1.92e-04	-27.22	249.41
300	6.70	7.92e-05	-28.07	6.24
400	6.70	7.04e-05	-26.01	3.18
500	6.70	7.10e-05	-27.88	0.35
600	6.70	5.74e-05	-28.05	0.19

Table 3: Mean and variance of 10 clock skew measurements using LPM with the two clocks, for different timestamp sample sizes. `do_gettimeofday` is abbreviated as gtod.

Real skew	Real intercept	Fake skew	Fake intercept
16.79	0.53	16.78	-2.13
16.82	0.51	16.69	1.43
16.80	-0.02	16.74	-1.34
16.81	0.17	16.78	-1.10

Table 4: Clock skews and line intercepts from real and fake AP, rounded to the nearest hundredth.

4 Vulnerability of previous measurement methods

In this section we present a spoofing attack that is able to fool the method of [14] that relied only on the clock skew measurement to detect spoofing. Our technique finds its basis in two key points:

1. Observations 3 and 4 show that a measurement device measures different clock skews depending on whether its TSF timer is synchronized with the beacon timestamps from an AP, because that timer, being synchronized, *acquires* the skew of the AP.

2. The Madwifi driver allows the multiple creation of virtual interfaces (VAPs) for a single physical device. These virtual interfaces may be in different modes—station, master, or monitor—and in particular, one station VAP is allowed to exist along with several AP VAPs. These virtual interfaces can then be brought “up” to begin operation.

These points suggest that we might be able to have an AP VAP and a station VAP, with the station VAP associated with the real AP, and the AP VAP configured as the fake AP. Since the two interfaces would share the same hardware TSF timer, the timer would acquire the skew of the real AP due to the station VAP associating with it. This skew would be reflected in the timestamps in the beacons emitted by the AP VAP, thereby spoofing the clock skew of the real AP.

However, carrying out the above attack required some modification of the Madwifi driver. The hardware device can be put into only one of the operating modes at a time—either station or AP—and in the case when a station VAP and an AP VAP are created, the driver puts the card into AP mode and simulates the operation of the station VAP in software. Since the updating of the hardware TSF timer upon receipt of beacons is done by the hardware logic, the station VAP does not update that timer when the card is in AP operating mode.

The interface provided to the driver does not allow for setting of the TSF timer directly. However, it does allow for getting the timer value. The code for getting the timer reveals the address of the hardware register maintaining the TSF timer. We modify the driver to write the beacon timestamp to that register whenever the station VAP receives a beacon frame from the real AP. This leads to another problem: the timers in the beacon scheduling queue for the AP VAP are disrupted by our register updates and the AP VAP stops broadcasting beacons. To continue to transmit beacons we use the `ath_send_beacon()` function in the driver that is used to send out beacon frames. We force beacon transmission each time we update the hardware timer register by calling this function. We then find that beacons are transmitted as required, and we compare the clock skews of the real AP and the fake AP. Table 4 shows the skews from four measurements. It can be seen that it is not possible to detect the fake AP by comparing the clock skew alone, with a reasonable degree of certainty. Figure 3 shows the clock offset points from the two APs. The synchronization behavior produces periodic dips in the plot. In Section 5 we show how to capture this behavior to measure of the reliability of the clock skew.

The authors of [14] present several arguments showing why the skews of APs cannot be fabricated. We briefly discuss the reasons why their arguments do not hold in our case. The failing assumption made in their arguments is that the attacker, on knowing the clock skew of the the real AP, would need to perform arithmetic with his local timer values to compensate for his own skew. In our attack this

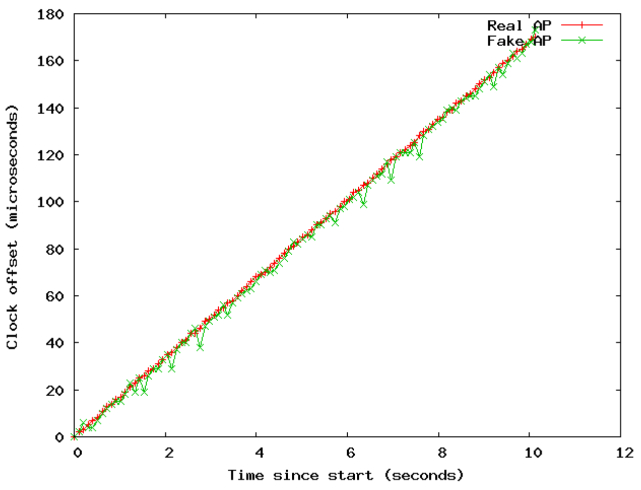


Figure 3: Clock offsets with 100 beacon timestamps from the real AP and the fake AP.

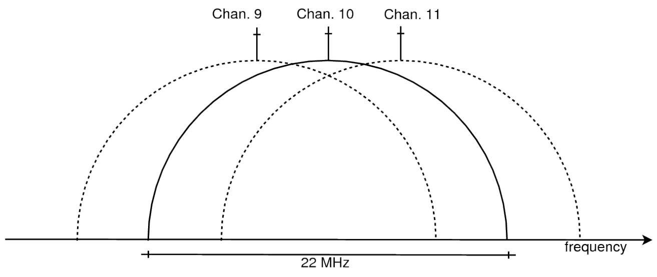


Figure 4: Overlapping channels in 802.11bg networks.

is not the case. We do not measure the skew of the real AP in advance and try to compensate for it. Rather we *continuously* use the timestamps in the beacon frames from the real AP to update the TSF timer. The first argument in [14] shows the failure of trying to compensate for the attacker's local skew when injecting beacons frames using the raw packet injection mode allowed by the driver. This is not the approach we take so we do not suffer from the transmission delays that affect the ability to spoof the clock skew. The other approach considered was that of setting the wireless hardware timer directly through software. The authors argue that such an attempt at spoofing might be detected by measuring the medium contention time for the AP. Since the attacker would need to perform floating point operations in the wireless hardware to select the right offset and mimic the clock skew of the real AP, the overhead would have an observable effect on the medium contention time. As we note above, we use the beacons timestamps from the real AP continuously and do not need to do any arithmetic, so we avoid this problem altogether.

Extending the scope of the attack

We now show how the scope of the attack can be extended by using a "bridge" AP. The function of the bridge AP is to allow the attacker to move the fake AP to cover a wider range (perhaps in order to be out of range of the real AP), or to operate in a different channel, all while still spoofing its clock skew. The bridge AP synchronizes its TSF timer with that of the real AP as described earlier, and the fake AP synchronizes its timer with that of the bridge AP. To operate on a different channel we take advantage of the fact that frequency ranges of adjacent channels as prescribed by the 802.11 standards overlap (see Figure 4).

Real skew	Real intercept	Fake skew	Fake intercept
17.35	-0.78	16.29	71.49
17.29	0.70	17.58	-10.63
17.26	-0.46	17.54	-10.49
17.25	-0.19	16.49	-19.53

Table 5: Clock skews and line intercepts from the real and *bridged* fake AP, rounded to the nearest hundredth.

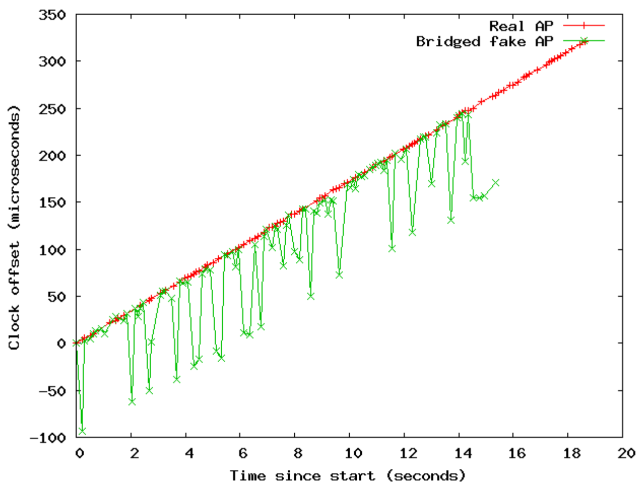


Figure 5: Clock offsets with 100 beacon timestamps from the real AP and the *bridged* fake AP.

In our experiment we have the real AP operating on channel 11, the bridge AP on channel 10, and the fake AP on channel 9. Our results from four traces are shown in Table 5 and Figure 5. As it may be expected, the quality of spoofing degrades due to the bridging, but the clock skew of the fake AP is still fairly close to that of the real AP.

Beacon interval	c-Real	c-Fake	% change	γ-Real	γ-Fake	% change
25	0.20	2.13	965	0.50	3.1	520
50	0.58	1.23	112	0.84	3.34	298
100	0.31	1.50	384	1.71	3.11	82
200	0.40	1.68	320	3.43	3.88	13

Table 6: Variation in parameters c and γ with different values of the beacon interval.

5 Improving the Reliability of Fingerprinting

We now present techniques to mitigate the risks of attacks like those presented in the last section, by gauging the reliability of the measured clock skews.

5.1 Line-fitting error

The most straightforward approach is to measure the error in line fitting. We observed that the spoofing attack in the previous section introduced an artifact—the dips in the plots of clock offset points in Figures 3 and 5. There are several ways to measure these fluctuations. We could measure the (least square) error of line fitting, i.e., the sum of the distance of the clock offset points from the line fitted to them. However, this approach requires first fitting the line using LSF and then using the clock offset points again to compute the fitting error. Since that would involve more computational overhead, and also require storing the clock offset points, we avoid the approach. Instead we use two metrics that do not require this overhead.

First, we consider the y-intercept c of the fitted line $y = s \cdot x + c$. Since we assume that, in the ideal case, the line passes through the origin, the absolute value of the intercept serves as one parameter to measure the fitting error. Tables 4–5 show the values of the parameter c with our proposed attacks. The absolute value of c is higher for the fake AP when compared to that of the real AP.

We also consider the jitter of the beacon timestamps as a means to measure reliability of the clock skew. Given a set of clock offset points, the jitter γ is computed as

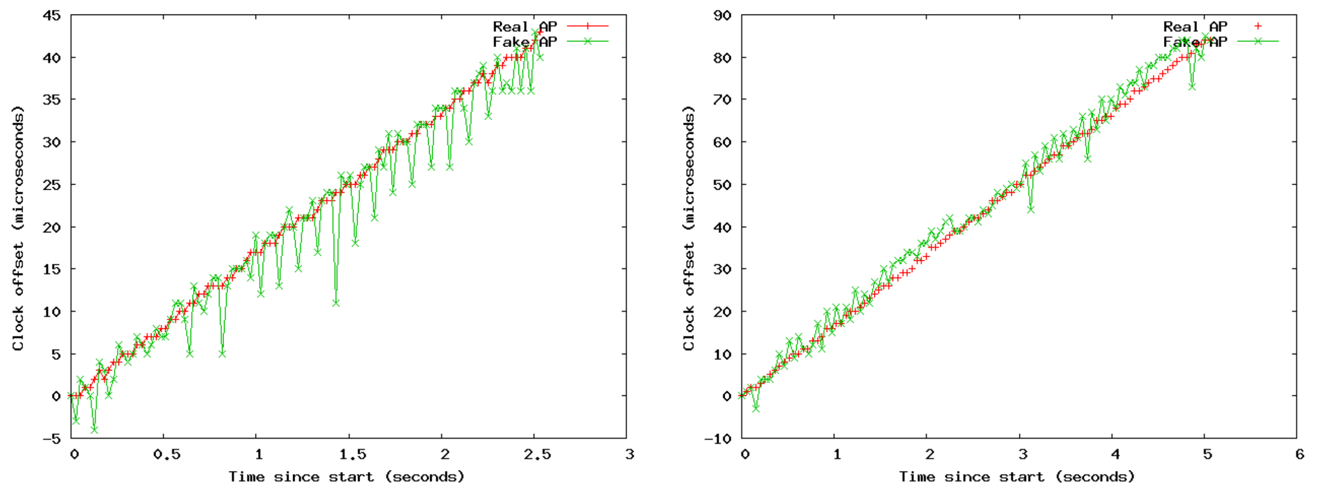
$$\gamma = \frac{1}{n-1} \sum_{i=1}^{n-1} |y_{i+1} - y_i|$$

and provides a measure of the temporal variations in the beacon timestamps. We defer the measurements of jitter in our attacks to the next section, where we analyze the effect of the beacon interval on our attacks.

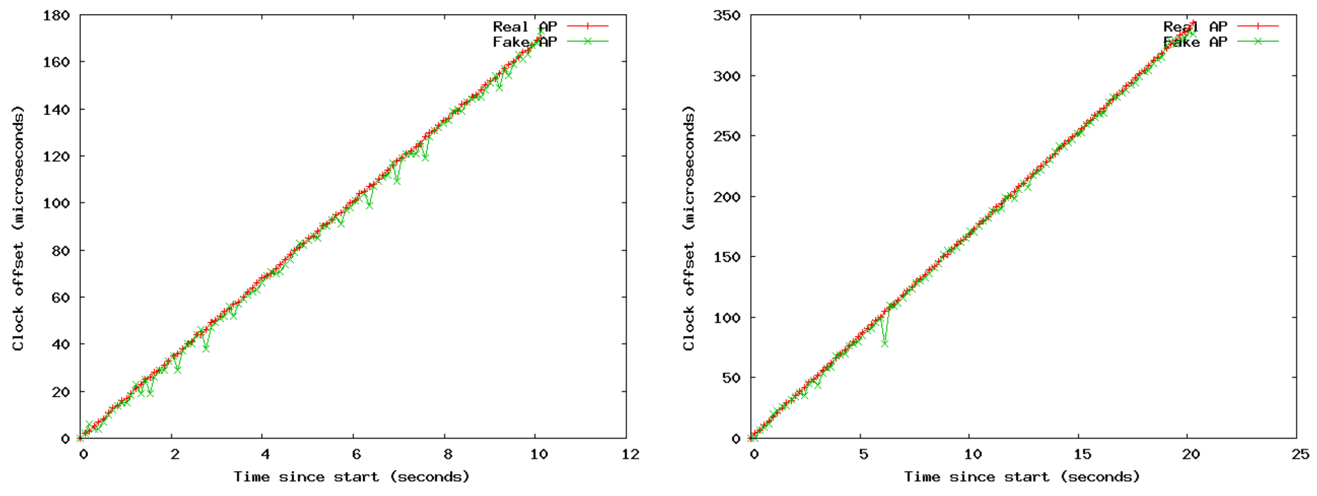
5.2 Analysis of beacon-interval on skew measurements

The value of the beacon interval of the AP affects the ability of the attacker to spoof its clock skew with our attack. When the beacon interval is set to smaller values, the attacker needs to present a finer-grained clock via the beacon timestamps. At lower beacon intervals the fluctuations in the synchronized clock of the attacker become more prominent since the various processing delays play a relatively larger role. As the beacon interval is increased, the behavior of the fake AP tends towards presenting a beacon timestamp from the real AP in the beacon timestamp of the fake AP with minimal effect of those delays.

To validate this hypothesis we perform our attack with different settings of the beacon interval parameter, and measure the parameters c and γ described earlier for testing the reliability of clock skew measurements. The clock offset points for this experiment are shown in Figure 6 (note the out-of-order



(a) Beacon interval = 25ms (b) Beacon interval = 50ms



(c) Beacon interval = 100ms (d) Beacon interval = 200ms

Figure 6: Clock offsets of the real AP and fake AP at different beacon intervals

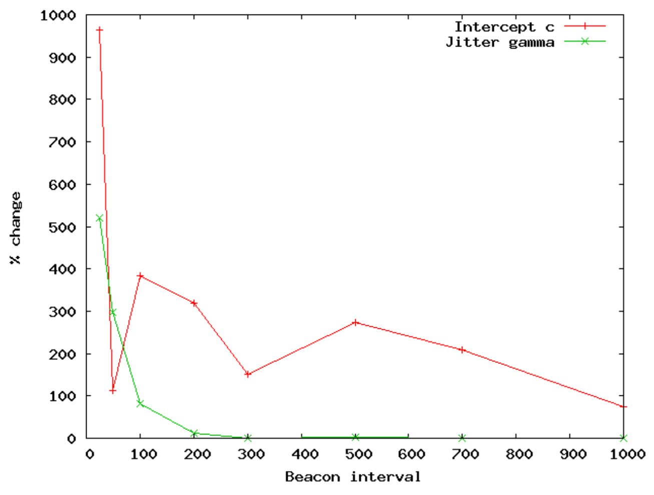


Figure 7: Variation in parameters c and γ with different values of the beacon interval.

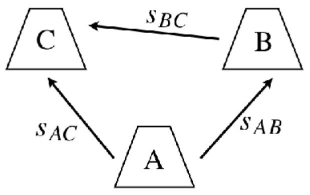


Figure 8: Measuring skews between a pair of access points using skew arithmetic.

numbering). We observe that the dips in the plots increase in magnitude as the beacon interval is reduced. To be sure that this is not an artifact of the different scale in the plots, in Table 6 and Figure 7 we show the variation of the parameters c and γ in terms of the *relative* change of their average values over four measurements. We observe that the changing magnitude of the dips seen in Figure 6 is captured very well by the jitter parameter γ, and to a good extent also by the intercept c.

Thus, to avoid clock skew spoofing attempts it is advisable to use a beacon interval that is as small as permissible, and use the parameters c and γ to gauge the reliability of clock skew measurements.

6 Skew arithmetic

In this section we show how to perform arithmetic with clock skews. For example, if we know the clock skew s_{AB} of AP B as would be measured by AP A, then we can compute the skew s_{BA}, i.e., the clock skew of AP A as would be measured by AP B. While measuring clock skews, we assume that the clock offsets lie on a line passing through the origin. If Δ^A and Δ^B denote the time elapsed since the start of an experiment as reported by the clocks of AP A and AP B respectively, then the clock offset (x, y) is given by $x = \Delta^A, y = \Delta^B - \Delta^A$. By our assumption we have that $y = s_{AB}x$, so that $\Delta^B = (1 + s_{AB})\Delta^A$. By a symmetric argument we get that $\Delta^A = (1 + s_{BA})\Delta^B$. Solving these two equations for s_{BA} we find that

$$s_{BA} = -s_{AB}/(1 + s_{AB}). \tag{1}$$

We can also compute the clock skew s_{BC} of AP C as measured by AP B, when given the clock skews s_{AB} and s_{AC} (see Figure 8).

Using notation as before, we have,

$$\begin{aligned} \Delta^B &= (1 + s_{AB}) \cdot \Delta^A, \\ \Delta^C &= (1 + s_{AC}) \cdot \Delta^A, \text{ and} \\ \Delta^C &= (1 + s_{BC}) \cdot \Delta^B. \end{aligned}$$

Solving these for s_{BC}, we get

$$s_{BC} = (s_{AC} - s_{AB})/(1 + s_{AB}). \tag{2}$$

In Equations 1 and 2, the term in the denominator is of the form $(1 + s)$, where s is a clock skew. Since $s \ll 1$ (of the order of parts per million), we can safely ignore the denominator so long as we are not performing several such arithmetic operations that affect each other. Then we get that,

$$\begin{aligned} s_{BA} &= -s_{AB}, \text{ and} \qquad (3) \\ s_{BC} &= s_{AC} - s_{AB}. \qquad (4) \end{aligned}$$

We present empirical results to validate the above equations. In our experiment we use Linksys 1 as AP C, and the two laptops as AP A and AP B. Switching AP B into monitor mode allows us to estimate the clock skew s_{BC}. We take four sets of measurements to determine each skew and take the mean. The mean skews s_{BC}, s_{AC}, and s_{AB} are observed as 6.3767, 16.7719, and 10.4801. The skew s_{BC} when estimated using Equation 2 is 6.29173 resulting in an error of 1.332% when compared with the value estimated directly. When using Equation 4 the skew is estimated as 6.2918 resulting in 1.331% error. Thus we see that the approximation in Equation 4 does not affect the result of the arithmetic.

The ability to perform this kind of skew arithmetic is essential to our goal of allowing clients to fingerprint APs. To determine whether to trust an AP by measuring its clock skew, the client must know the skew of the real AP beforehand. Since the clock used by the client has a skew of its own, it would be necessary for the client to have measured the skew of the real AP using its own clock beforehand. However, the ability to perform skew arithmetic eliminates this requirement. Network providers can publish the skews of the APs in their network as measured against a high-precision clock of negligible skew. Then to measure the skew of an AP using skew arithmetic, the client only needs to know the skew of its own clock against a similar high-precision clock. The process is simplified further if network card vendors measure and publish the skews of the cards they produce at the time of testing.

7 Related work

Existing methods of *passive* L2 fingerprinting of 802.11 client stations aimed to improve client identification for defensive or forensic purposes by verifying facts about the client. In particular, Franklin et al. [12] fingerprinted clients based on the clients' driver-specific probing behavior, and in the tour de force [11] Ellch fingerprinted clients based solely on statistical distributions of the 2-byte NAV field in established client connections (e.g., by watching several minutes worth of web traffic).

A passive method that a client could use for detecting the presence of fake APs was presented by Bahl et al. [4]. In particular, they used the anomaly in successive sequence numbers seen in beacon frames from the real and fake APs to detect the fake AP. Because of mixing of frames from the two sources, the sequence numbers do not form an increasing sequence. Their method is effective only when both APs are active at the same time. For the case when this does not hold, location-based detection was suggested. Still, it was observed [14] that even such methods are not very reliable, and fail to work if the attacker is able to position his AP carefully.

Bratus et al. pointed out [5] the importance of *protecting clients from access points* in the early stages of connection before cryptography-based trust in the AP could be established, and proposed an *active* fingerprinting scheme that tested certain properties of the AP before accepting any complex data from it. The related BlackHat 2008 talk [6] also mentioned results in fingerprinting access points by the skew of their timestamps transmitted in their beacon frames, and pointed out that such fingerprinting might serve the same purpose of client protection.

8 Summary

In this work we consider the reliability of previously proposed approaches to measure clock skews of wireless devices. By means of a spoofing attack that mimics the clock skew of an AP, we demonstrate the fallibility of those methods, when used in isolation. We provide a method that uses a more accurate clock to measure clock skews. Further, we propose new parameters—the fitted line intercept, c, and the jitter γ—to gauge the reliability of measured skews. We show that spoofing attempts might be detected by checking the values of these parameters. Our work also provides new insights into the implementation of wireless standards by commodity hardware.

Fingerprinting using clock skews is useful in establishing trust for an AP by a client. We provide methods to perform clock-skew arithmetic that allow a client to verify the clock skew of an AP without itself having to measure the real clock skew in advance.

References

[1] Wireless LAN medium access control (MAC) and physical layer (PHY) specification. IEEE Std 802.11, 2007.

[2] Raw wireless tools homepage. http://rfakeap.tuxfamily.org/.

[3] The Metasploit project. http://www.metasploit.com.

[4] Paramvir Bahl, Ranveer Chandra, Jitendra Padhye, Lenin Ravindranath, Manpreet Singh, Alec Wolman, and Brian Zill. Enhancing the security of corporate wi-fi networks using dair. In *MobiSys '06: Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 1–14, New York, NY, USA, 2006. ACM.

[5] Sergey Bratus, Cory Cornelius, David Kotz, and Daniel Peebles. Active behavioral fingerprinting of wireless devices. In *WiSec '08: Proceedings of the first ACM conference on Wireless network security*, pages 56–61. ACM, 2008.

[6] Sergey Bratus, Cory Cornelius, Daniel Peebles, and Axel Hansen. Active 802.11 fingerpinting: a "secret handshake" to know your APs. Black Hat USA 2008, http://baffle.cs.dartmouth.edu/.

[7] Laurent Butti. Wi-Fi advanced fuzzing. Black Hat Europe, February 2007.

[8] Johnny Cache and David Maynor. Hijacking a MacBook in 60 seconds. Black Hat, August 2006.

[9] Johnny Cache, H D Moore, and skape. Exploiting 802.11 wireless driver vulnerabilities on Windows. *Uninformed.org*, 6, January 2007.

[10] M. E. Dyer. Linear time algorithms for two- and three-variable linear programs. *SIAM Journal on Computing*, 13(1):31–45, 1984.

[11] J.P. Ellch. Fingerprinting 802.11 devices. Master's thesis, U.S. Naval Postgraduate School, September 2006.

[12] Jason Franklin, Damon McCoy, Parisa Tabriz, Vicentiu Neagoe, Jamie Van Randwyk, and Douglas Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In *Proceedings of 15th USENIX Security Symposium*, pages 167–178. USENIX, August 2006.

[13] The Shmoo group. 802.11 bait, the tackle for wireless phishing. Toorcon, October 2005.

[14] Suman Jana and Sneha Kumar Kasera. On fast and accurate detection of unauthorized wireless access points using clock skews. In *MobiCom '08: Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 104–115. ACM, 2008.

[15] Tadayoshi Kohno, Andre Broido, and K. C. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.

[16] Nimrod Megiddo. Linear-time algorithms for linear programming in r^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

[17] S.B. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. In *IEEE INFOCOM '99*, volume 1, pages 227–234, March 1999.

[18] H D Moore. Mastering the Metasploit framework. http://www.blackhat.com/html/bh-usa-09/train-bh-usa-09-hdm-meta.html, July 2009.

[19] Simple Nomad. Hacking the friendly skies. Shmoocon, January 2006.

[20] Dino A. Dai Zovi and Shane ”K2” Macaulay. Karma. http://trailofbits.wordpress.com/karma/.