Versions of this paper, copyright the IEEE, appear in *IEEE Symposium on Security and Privacy 2005*, IEEE Computer Society Press, May 2005 and *IEEE Transactions on Dependable and Secure Computing*, 2(2), 2005. This is the full version.

Remote physical device fingerprinting

Tadayoshi Kohno* Andre Broido† K.C. Claffy‡

May 25, 2005

Abstract

We introduce the area of *remote physical device fingerprinting*, or fingerprinting a physical device, as opposed to an operating system or class of devices, remotely, and without the fingerprinted device's known cooperation. We accomplish this goal by exploiting small, microscopic deviations in device hardware: clock skews. Our techniques do not require any modification to the fingerprinted devices. Our techniques report consistent measurements when the measurer is thousands of miles, multiple hops, and tens of milliseconds away from the fingerprinted device, and when the fingerprinted device is connected to the Internet from different locations and via different access technologies. Further, one can apply our passive and semi-passive techniques when the fingerprinted device is behind a NAT or firewall, and also when the device's system time is maintained via NTP or SNTP. One can use our techniques to obtain information about whether two devices on the Internet, possibly shifted in time or IP addresses, are actually the same physical device. Example applications include: computer forensics; tracking, with some probability, a physical device as it connects to the Internet from different public access points; counting the number of devices behind a NAT even when the devices use constant or random IP IDs; remotely probing a block of addresses to determine if the addresses correspond to virtual hosts, e.g., as part of a virtual honeynet; and unanonymizing anonymized network traces.

Keywords: Remote physical device fingerprinting, forensics, privacy, NATs, virtual honeynets, network spectroscopy, clock skew.

*Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-mail: tkohno@cs.ucsd.edu. URL: http://www-cse.ucsd.edu/users/tkohno. Part of this work was performed while visiting the Cooperative Association for Internet Data Analysis. Supported by a National Defense Science and Engineering Graduate Fellowship, an IBM Ph.D. Fellowship, and the SciDAC program of the US Department of Energy (award # DE-FC02-01ER25466).

†Cooperative Association for Internet Data Analysis, San Diego Supercomputer Center, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-mail: broido@caida.org . Supported by the SciDAC program of the US Department of Energy (award # DE-FC02-01ER25466).

‡Cooperative Association for Internet Data Analysis, San Diego Supercomputer Center, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-mail: kc@caida.org . Supported by the SciDAC program of the US Department of Energy (award # DE-FC02-01ER25466).

1 Introduction

There are now a number of powerful techniques for *remote operating system fingerprinting*, i.e., techniques for remotely determining the operating systems of devices on the Internet [2, 3, 5, 37]. We push this idea further and introduce the notion of *remote physical device fingerprinting*, or remotely fingerprinting a physical device, as opposed to an operating system or class of devices, without the fingerprinted device's known cooperation. We accomplish this goal to varying degrees of precision by exploiting microscopic deviations in device hardware: clock skews. Our techniques do not require any modification to the fingerprinted devices. Our techniques report consistent measurements when the measurer is thousands of miles, multiple hops, and tens of milliseconds away from the fingerprinted device, and when the fingerprinted device is connected to the Internet from different locations and with different access technologies. One can apply our passive and semi-passive techniques when the fingerprinted device is behind a NAT or firewall, and when the device's system time is maintained via NTP [26] or SNTP [25].

One can use our techniques to obtain information about whether two devices on the Internet (possibly shifted in time or IP addresses) are actually the same physical device. One application is computer forensics, e.g., to argue whether a given machine was involved in a recorded attack, even if the attack machine was connected from a public wireless hotspot. In combination with existing operating system fingerprinting tools and other heuristics, one can use our techniques to track, with some probability, physical devices like laptops even as those devices move from one access location to another. One can use our techniques to count the number of hosts behind a NAT, even if not all of the hosts are active at the same time, even if all of the hosts run the same operating system, and even if the hosts use random or constant IP IDs to foil Bellovin's attack [7]. One can use our techniques to infer whether a set of IP addresses correspond to virtualized hosts, which could be useful in remotely determining whether a set of IP addresses correspond to a virtual honeynet. One can also use our techniques to catalyze the unanonymization of anonymized network traces.

Classes of fingerprinting techniques. We consider three main classes of remote physical device fingerprinting techniques: passive, active, and semi-passive. The first two have standard definitions — to *apply* a *passive* fingerprinting technique, the *fingerprinter* (*measurer*, *attacker*, *adversary*) must be able to *observe* traffic from the device (the *fingerprintee*) that the attacker wishes to fingerprint, whereas to apply an *active* fingerprinting technique, the fingerprinter must have the ability to *initiate* connections to the fingerprintee. Our third class of techniques, which we refer to as *semi-passive* fingerprinting techniques, assumes that *after* the fingerprintee *initiates* a connection, the fingerprinter has the ability to interact with the fingerprintee over that connection; e.g., the fingerprinter is a website with which the device is communicating, or is an ISP in the middle capable of modifying packets en route.

Each class of techniques has its own advantages and disadvantages. For example, passive techniques are completely undetectable to the fingerprinted device, passive and semi-passive techniques can be applied even if the fingerprinted device is behind a NAT or firewall, and semi-passive and active techniques can potentially be applied over longer periods of time; e.g., after a laptop connects to a website and the connection terminates, the website can still continue to run active measurements.

Methodology. For all our methods, we stress that the fingerprinter does not require any modification to or cooperation from the fingerprintee; e.g., we tested our techniques with default Red Hat 9.0, Debian 3.0, FreeBSD 5.2.1, OpenBSD 3.5, OS X 10.3.5, Windows XP SP2, and Windows for Pocket PC 2002 installations.[1] In Table 1 we summarize our preferred methods for fingerprinting

[1] One can apply our techniques to the default installs of other versions of these operating systems; here we just mention the most recent stable versions of the operating systems that we analyzed.

Technique and section	Class	NTP	Red Hat 9.0	OS X 10.3.5	Windows XP
TCP timestamps, §4	passive	Yes	Yes	Yes	No
TCP timestamps, §4	semi-passive	Yes	Yes	Yes	Yes
ICMP tstamp requests, §5	active	No	Yes	No	Yes (SP1 only)

Table 1. This table summarizes our main clock skew-based physical device fingerprinting techniques. A "Yes" in the NTP column means that one can use the attack regardless of whether the fingerprintee maintains its system time with NTP [26]. One can use passive and semi-passive techniques when the fingerprintee is behind a NAT or current generation firewall.

the most popular operating systems.

Our preferred passive and semi-passive techniques exploit the fact that most modern TCP stacks implement the *TCP timestamps option* from RFC 1323 [18] whereby, for performance purposes, each party in a TCP flow includes information about its perception of time in each outgoing packet. A fingerprinter can use the information contained within the TCP headers to estimate a device's clock skew and thereby fingerprint a physical device. We stress that one can use our TCP timestamps-based method even when the fingerprintee's system time is maintained via NTP [26]. While most modern operating systems enable the TCP timestamps option by default, Windows 2000 and XP machines do not. Therefore we developed a trick to convince Microsoft Windows 2000 and XP machines to use the TCP timestamps option in Windows-initiated flows; the trick involves an intentional violation of RFC 1323 on the part of a semi-passive or active adversary. In addition to our TCP timestamps-based approach, we consider passive fingerprinting techniques that exploit the difference in time between how often other periodic activities are supposed to occur and how often they actually occur, and we show how one might use a Fourier transform on packet arrival times to infer a device's clock skew. Since we believe that our TCP timestamps-based approach is currently our most general passive technique, we focus on the TCP timestamps approach in this paper.

An active adversary could also exploit the ICMP protocol to fingerprint a physical device. Namely, an active adversary could issue ICMP Timestamp Request messages to the fingerprintee and record a trace of the resulting ICMP Timestamp Reply messages. If the fingerprintee does not maintain its system time via NTP or does so only infrequently and if the fingerprintee replies to ICMP Timestamp Requests, then an adversary analyzing the resulting ICMP Timestamp Reply messages will be able to estimate the fingerprintee's system time clock skew. Default Red Hat 9.0, Debian 3.0, FreeBSD 5.2.1, OpenBSD 3.5, and Windows 2000 and XP and Pocket PC 2002 installations all satisfy the above preconditions.

Parameters of investigation. Toward developing the area of remote physical device fingerprinting via remote clock skew estimation, we must address the following set of interrelated questions:

(1) For what *operating systems* are our remote clock skew estimation techniques applicable?

(2) Can one expect two machines to have measurably different clock skews? Specifically, what is the *distribution* of clock skews across multiple fingerprintees and what is the *resolution* of our clock skew estimation techniques?

(3) For a single fingerprintee, can one expect the clock skew estimate of that fingerprintee to be relatively *constant* over long periods of time, and through reboots, power cycles, and periods of down time?

(4) What are the effects of a fingerprintee's *access technology* (e.g., wireless, wired, dialup, cable

modem) on the clock skew estimates for the device?

(5) How are the clock skew estimates affected by the *distance* between the fingerprinter and the fingerprintee?

(6) Are the clock skew estimates *independent of the fingerprinter*? I.e., when multiple fingerprinters are measuring a single fingerprintee at the same time, will they all output (approximately) the same skew estimates?

(7) How much *data* do we need to be able to remotely make accurate clock skew estimates?

Question (6) is pertinent because common fingerprinters will probably use NTP-based time synchronization when capturing packets, as opposed to more precise CDMA- or GPS-synchronized timestamps. Answers to the above questions will help determine the efficacy of our physical device fingerprinting techniques.

Experiments and high-level results. To understand and refine our techniques, we conducted experiments with machines that we controlled and that ran a variety of operating systems, including popular Linux, BSD, and Microsoft distributions. In all cases we found that we could use at least one of our techniques to estimate clock skews of the machines, and that we required only a small amount of data, though the exact data requirements depended on the operating system in question. For the most popular operating systems we observed that when the system did not use NTP- or SNTP-based time synchronization, then the TCP timestamps-based and the ICMP-based techniques yielded approximately the same skew estimates. Furthermore, for the most popular operating systems we observed that our TCP timestamps-based skew estimates were approximately the same regardless of whether or not a host used NTP-based time synchronization. These results, coupled with details that we describe in the body, motivated us to use the TCP timestamps-based method in most of our experiments. We survey some of our experiments here.

To understand the effects of topology and access technology on our skew estimates, we fixed the location of the fingerprinter and applied our TCP timestamps-based technique to a single laptop in multiple locations, on both North American coasts, from wired, wireless, and dialup locations, and from home, business, and campus environments (Table 5). All clock skew estimates for the laptop were close — the difference between the maximum and the minimum skew estimate was only 0.67 ppm, or 0.67 microseconds per second. We also simultaneously measured the clock skew of the laptop and another machine from multiple PlanetLab nodes throughout the world, as well as from a machine of our own with a CDMA-synchronized Dag card [1, 12, 15, 24] for taking network traces with precise timestamps (Table 6 and Table 7). With the exception of the measurements taken by a PlanetLab machine in India (over 300 ms round trip time away), for each experiment, all the fingerprinters (in North America, Europe, and Asia) reported skew estimates within only 0.56 ppm of each other. These experiments suggest that, except for extreme cases, the results of our clock skew estimation techniques are independent of access technology and topology.

Toward understanding the distribution of clock skews across machines, we applied the TCP timestamps technique to the devices in a trace collected on one of the U.S.'s Tier 1 OC-48 links (Figure 3). We also measured the clock skews of 69 (seemingly) identical Windows XP SP1 machines in one of our institution's undergraduate computing facilities (Figure 4). The latter experiment, which ran for 38 days, as well as other experiments, show that the clock skew estimates for any given machine are approximately constant over time, but that different machines can have detectably different clock skews. Lastly, we use the results of these and other experiments to argue that the amount of data (packets and duration of data) necessary to perform our skew estimation techniques is low, e.g., see Tables 3 and 5.

Applications and additional experiments. To test the applicability of our techniques, we

applied our techniques to a `honeyd` [33] virtual honeynet consisting of 100 virtual Linux 2.4.18 hosts and 100 virtual Windows XP SP1 hosts. Our experiments showed with overwhelming probability that the TCP flows and ICMP timestamp responses were all handled by a single machine as opposed to 200 different machines. We also applied our techniques to a network of five virtual machines running under VMware Workstation [4] on a single machine. In this case, the clock skew estimates of the virtual machines are significantly different from what one would expect from real machines (the skews were large and *not* constant over time; Figure 9). An application of our techniques, or natural extensions, might therefore be to remotely detect virtual honeynets.

Another applications of our techniques is to count the number of hosts behind a NAT, even if those hosts use random or constant IP IDs to counter Bellovin's attack [7], even if all the hosts run the same operating system, and even if not all of the hosts are up at the same time. Furthermore, when both our techniques and Bellovin's techniques are applicable, we expect our approach to provide a much higher degree of resolution. One could also use our techniques for forensics purposes, e.g., to argue whether or not a given laptop was connected to the Internet from a given access location. One could also use our techniques to help track laptops as they move, perhaps as part of a Carnivore-like project (here we envision our skew estimates as one important component of the tracking; other components could be information gleaned from existing operating system fingerprinting techniques, usage characteristics, and other heuristics). One can also use our techniques to catalyze the unanonymization of prefix-preserving anonymized network traces [38, 39].

Protecting against our current attacks and future directions. Although the physical device fingerprinting techniques that we introduce in this paper will likely remain applicable to current generation systems, we suspect that future generation security systems might try to resist some of our fingerprinting techniques. To aid the developers of future systems, we explore some possible mechanisms for protecting against our current fingerprinting techniques. As a simple solution, a device might simply ignore ICMP Timestamp Requests and not enable the TCP timestamps option in outgoing TCP packets. A device might also choose to maintain its system time via NTP and somehow reduce the skew in its TCP timestamps clock. A device might also randomize or mask the TCP timestamps that it includes in each outgoing TCP packet. We then propose several possible research directions for fingerprinting physical devices that implement some of our protection mechanisms.

2 Background and related work

It has long been known that seemingly identical computers can have disparate clock skews. The NTP [26] specification describes a method for reducing the clock skews of devices' system clocks, though over short periods of time an NTP-synchronized machine may still have slight clock skew. In 1998 Paxson [31] initiated a line of research geared toward eliminating clock skew from network measurements, and one of the algorithms we use is based on a descendent of the Paxson paper by Moon, Skelly, and Towsley [27, 28]. Further afield, though still related to clock skews, Pásztor and Veitch [30] have created a software clock on a commodity PC with high accuracy and small clock skew. One fundamental difference between our work and previous work is our goal: whereas all previous works focus on creating methods for eliminating the effects of clock skews, our work exploits and capitalizes on the effects of clock skews.

Well-known operating system fingerprinting tools include `nmap` [2], `xprobe2` [5], `p0f` [3], and RING [37]. The `nmap` and `p0f` tools use TCP timestamps to remotely obtain the uptimes of some systems, per [22]. Anagnostakis *et al.* [6] use ICMP Timestamp Requests to study router queuing delays. With respect to tracking physical devices, it is well known that a network card's MAC

address is supposed to be unique and therefore could serve as a fingerprint of a device assuming that the adversary can observe the device's MAC address and that the owner of the card has not changed the MAC address. The main advantage of our techniques over a MAC address-based approach is that our techniques are mountable by adversaries thousands of miles and multiple hops away. One could also use cookies or any other persistent data to track a physical device, but such persistent data may not always be available to an adversary, perhaps because the user is privacy-conscious and tries to minimize storage and transmission of such data, or because the user never communicates that data unencrypted. The amateur radio community has independently developed a tool for fingerprinting radios that exploits a radio's frequency characteristics [34].

Our work in Section 11 builds on previous network spectroscopy research in detecting link-layer technologies (ATM, DSL, cable modems) by delay quantization of IP packet traffic [10], and on operating system fingerprinting by DNS update traffic [11]. In [9] we use traceroute delays to fingerprint routers. Partridge *et al.* [29] use Lomb periodograms to detect the presence of traffic that is otherwise not observed on a network via the spikes in the spectral density that correspond to intervals occupied by missing packets. Hussain *et al.* [17] use the power distribution in the Fourier spectrum of autocorrelation functions of binned packet traffic to distinguish between multiple-source and single-source denial-of-service attacks. In Section 11 we propose using a full precision (without binning) Fourier transform of packet arrival times to extract clock skews from periodic processes.

An extended abstract of this paper appears at the 2005 IEEE Symposium on Security and Privacy [20] and the journal version of this paper appears in the IEEE Transactions on Dependable and Secure Computing [21].

3 Clocks and clock skews

When discussing clocks and clock skews, we build on the nomenclature from the NTP specification [26] and from Paxson [31]. A *clock* C is designed to represent the amount of time that has passed since some initial time $\mathsf{i}[\mathsf{C}]$. Clock C's *resolution*, $\mathsf{r}[\mathsf{C}]$, is the smallest unit by which the clock can be incremented, and we refer to each such increment as a *tick*. A resolution of 10 ms means that the clock is designed to have 10 ms granularity, not that the clock is always incremented *exactly* every 10 ms. Clock C's *intended frequency*, $\mathsf{Hz}[\mathsf{C}]$, is the inverse of its resolution; e.g., a clock with 10 ms granularity is designed to run at 100 Hz. For all $t \geq \mathsf{i}[\mathsf{C}]$, let $\mathsf{R}[\mathsf{C}](t)$ denote the time reported by clock C at time t, where t denotes the true time as defined by national standards. The *offset* of clock C, $\mathsf{off}[\mathsf{C}]$, is the difference between the time reported by C and the true time, i.e., $\mathsf{off}[\mathsf{C}](t) = \mathsf{R}[\mathsf{C}](t) - t$ for all $t \geq \mathsf{i}[\mathsf{C}]$. A clock's *skew*, $\mathsf{s}[\mathsf{C}]$, is the first derivative of its offset with respect to time, where we assume for simplicity that $\mathsf{R}[\mathsf{C}]$ is a differentiable function in t. We report skew estimates in microseconds per second ($\mu s/s$) or, equivalently, parts per million (ppm). As we shall show, and as others have also concluded [31, 28, 36], it is often reasonable to assume that a clock's skew is constant. When the clock in question is clear from context, we shall remove the parameter C from our notation; e.g., $\mathsf{s}[\mathsf{C}]$ becomes s.

A given device can have multiple, possibly independent, clocks. For remote physical device fingerprinting, we exploit two different clocks: the clock corresponding to a device's *system time*, and a clock internal to a device's TCP network stack, which we call the device's *TCP timestamps option clock* or *TSopt clock*. We do not consider the hardware bases for these clocks here since our focus is not on understanding why these clocks have skews, but on exploiting the fact these clocks can have measurable skews on popular current generation systems.

The system clock. To most users of a computer system, the most visible clock is the device's

Operating system	$\mathsf{Hz}[\mathsf{C}_{\mathsf{tcp}}]$
Debian 3.0 (Linux kernel 2.2.20-idepci)	100 Hz
FreeBSD 5.2.1	100 Hz
OpenBSD 3.5	2 Hz
OS X 10.3.5	2 Hz
Red Hat 9.0 (Linux kernel 2.4.20-8)	100 Hz
Windows XP SP2	10 Hz
Windows Pocket PC 2002	10 Hz

Table 2. $\mathsf{Hz}[\mathsf{C}_{\mathsf{tcp}}]$ values for the TCP timestamps option clock on several popular operating systems. All the entries, except for OS X 10.3.5 and Windows Pocket PC 2002, are for the operating systems on a 32-bit Intel Pentium processor. The OS X 10.3.5 entry is for the operating system on an Apple G4. The Windows Pocket PC 2002 entry is for the operating system on an HP iPAQ h5450.

system clock, $\mathsf{C}_{\mathsf{sys}}$, which is designed to represent the real time as defined by national standards. Although the system clocks on professionally administered machines are often approximately synchronized with true time via NTP [26] or, less accurately, via SNTP [25], we stress that it is much less likely for the system clocks on non-professionally managed machines to be externally synchronized. This lack of synchronization is because the default installations of most of the popular operating systems that we tested *do not* synchronize the hosts' system clocks with true time or, if they do, they do so only infrequently. For example, default Windows XP Professional installations only synchronize their system times with Microsoft's NTP server when they boot and once a week thereafter. Default Red Hat 9.0 Linux installations do not use NTP by default, though they do present the user with the option of entering an NTP server. Default Debian 3.0, FreeBSD 5.2.1, and OpenBSD 3.5 systems, under the configurations that we selected (e.g., "typical user"), do not even present the user with the option of enabling `ntpd`. For such a non-professionally-administered machine, if an adversary can learn the values of the machine's system clock at multiple points in time, the adversary will be able to infer information about the device's *system clock skew*, $\mathsf{s}[\mathsf{C}_{\mathsf{sys}}]$.

The TCP timestamps option clock. RFC 1323 [18] specifies the *TCP timestamps option* to the TCP protocol. A TCP flow will use the TCP timestamps option if the network stacks on both ends of the flow implement the option and if the initiator of the flow includes the option in the initial SYN packet. All modern operating systems that we tested implement the TCP timestamps option. Of the systems we tested, Microsoft Windows 2000 and XP are the only ones that do not include the TCP timestamps option in the initial SYN packet (Microsoft Windows Pocket PC 2002 does include the option when initiating TCP flows). In Section 4 we introduce a trick for making Windows 2000- and XP-initiated flows use the TCP timestamps option.

For physical device fingerprinting, the most important property of the TCP timestamps option is that if a flow uses the option, then a portion of the header of each TCP packet in that flow will contain a 32-bit timestamp generated by the creator of that packet. The RFC does not dictate what values the timestamps should take, but does say that the timestamps should be taken from a "virtual clock" that is "at least approximately proportional to real time [18];" the RFC 1323 PAWS algorithm does stipulate (Section 4.2.2) that the resolution of this virtual clock be between 1 ms and 1 second. We refer to this "virtual clock" as the device's *TCP timestamps option clock*, or its *TSopt clock* $\mathsf{C}_{\mathsf{tcp}}$. There is no requirement that a device's TSopt clock and its system clock be correlated. Moreover, for popular operating systems like Windows XP, Linux, and FreeBSD, a device's TSopt clock may be unaffected by adjustments to the device's system clock via NTP. Table 2 lists the

intended frequencies, $\mathsf{Hz}[\mathsf{C}_{\mathsf{tcp}}]$, for several popular operating systems.[2] Most systems reset their TSopt clock to zero upon reboot; on these systems $\mathsf{i}[\mathsf{C}_{\mathsf{tcp}}]$ is the time at which the system booted. If an adversary can learn the values of a device's TSopt clock at multiple points in time, then the adversary may be able to infer information about the device's *TSopt clock skew*, $\mathsf{s}[\mathsf{C}_{\mathsf{tcp}}]$.

4 Exploiting the TCP Timestamps Option

In this section we consider (1) how an adversary might obtain samples of a device's TSopt clock at multiple points in time and (2) how an adversary could use those samples to fingerprint a physical device. We consider the efficacy of and the data requirements for our approach in later sections. We assume for now that there is a one-to-one correspondence between physical devices and IP addresses, and defer to Section 9 a discussion of how to deal with multiple active hosts behind a NAT; in this section we do consider NATs with a single active device behind them.

THE MEASURER. The measurer can be any entity capable of observing TCP packets from the fingerprintee, assuming that those packets have the TCP timestamps option enabled. The measurer could therefore be the fingerprintee's ISP, or any tap in the middle of the network over which packets from the device travel; e.g., we apply our techniques to a trace taken on a major Tier 1 ISP's backbone OC-48 links. The measurer could also be any system with which the fingerprintee frequently communicates; prime examples of such systems include a search engine like Google, a news website, and a click-through ads service that displays content on a large number of websites. If the measurer is active, then the measurer could also be the one to initiate a TCP flow with the fingerprintee, assuming that the device is reachable and has an open port. If the measurer is semi-passive or active, then it could make the flows that it observes last abnormally long, thereby giving the measurer samples of the fingerprintee's clock over extended periods of time.

A TRICK FOR MEASURING WINDOWS 2000 AND XP MACHINES. We seek the ability to measure TSopt clock skews of Windows 2000 and XP machines even if those machines are behind NATs and firewalls. More generally, we are interested in measuring the TSopt clock skews of Windows machines when we are limited to analyzing flows initiated by the Windows machines. Unfortunately, because Windows 2000 and XP machines do not include the TCP timestamps option in their initial SYN packets, the TCP timestamps RFC [18] mandates that *none* of the subsequent packets in Windows-initiated flows can include the TCP timestamps option. Thus, assuming that all parties correctly implement the TCP RFCs, a passive adversary will not be able to exploit the TCP timestamps option with Windows 2000/XP-initiated flows.

If the adversary is semi-passive, we observe the following trick. Assume for simplicity that the adversary is the device to whom the Windows machine is connecting. After receiving the initial SYN packet from the Windows machine, the adversary will reply with a SYN/ACK, but the adversary will *break the RFC 1323 specification and include the TCP timestamps option in its reply.* After receiving such a reply, our Windows 2000 and XP machines ignored the fact that they did not include the TCP timestamps option in their initial SYN packets, and included the

[2] We do not generalize the Debian and Red Hat columns to all Linux distributions since Knoppix 3.6 with the 2.6.7 experimental kernel has 1 ms resolution. It is worth elaborating on our claim that Pocket PC 2002 systems have TSopt clocks with 100 ms resolution, or intended frequencies of 10 Hz. In experiments with five HP iPAQ h5450 PDAs running Windows Pocket PC 2002, using our techniques from Section 4 we measured TSopt clock frequencies between 8.4 and 9.6 Hz. Despite these measurements, we believe that the intended frequency for Pocket PC 2002 devices is 10 Hz since (1) the intended frequency for Windows 2000 and XP machines is 10 Hz and (2) we assume that the large difference between 10 Hz and the frequencies that we measured is due to the fact that the PDAs likely have cheaper clocks than our laptop and desktop systems.

TCP timestamps option in all of their subsequent packets. An adversary can therefore apply our techniques to estimate the TSopt clock skews of Windows machines even if they are behind NATs or firewalls.

As an extension, we note that the adversary does not have to be the device to whom the Windows machine is connecting. Rather, the adversary simply needs to be able to mount a "device-in-the-middle" attack and modify packets such that the Windows machine receives one with the TCP timestamps option turned on. If the adversary is the device's ISP, then the adversary could rewrite the Windows machine's initial SYN packets so that they include the TCP timestamps option. The SYN/ACKs from the legitimate recipients will therefore have the TCP timestamps option enabled and from that point forward the Windows machine will include the TCP timestamps option in all subsequent packets in the flows.

We applied this technique to Windows XP machines on a residential cable system with a LinkSys Wireless Access Point and a NAT, as well as to Windows XP SP2 machines using the default XP SP2 firewall, and to Windows XP SP1 machines with the Windows ZoneAlarm firewall. While current firewalls do not detect this trick, future firewalls might.

Estimating the TSopt clock skew. Let us now assume that an adversary has obtained a trace $\mathcal{T}$ of TCP packets from the fingerprintee, and let us assume for simplicity that all $|\mathcal{T}|$ packets in the trace have the TCP timestamps option enabled. Toward estimating a device's TSopt clock skew $\mathsf{s}[\mathsf{C_{tcp}}]$ we adopt the following additional notation. Let t_i be the time in seconds at which the measurer observed the i-th packet in $\mathcal{T}$ and let T_i be the $\mathsf{C_{tcp}}$ timestamp contained within the i-th packet. Define

$$\begin{aligned} x_i &= t_i - t_1 \\ v_i &= T_i - T_1 \\ w_i &= v_i/\mathsf{Hz} \\ y_i &= w_i - x_i \\ \mathcal{O}_{\mathcal{T}} &= \{\, (x_i, y_i) \,:\, i \in \{1, \ldots, |\mathcal{T}|\} \,\} \,. \end{aligned}$$

The unit for w_i is seconds; y_i is the *observed offset* of the i-th packet; $\mathcal{O}_{\mathcal{T}}$ is the the *offset-set* corresponding to the trace $\mathcal{T}$. We discuss below how to compute Hz if it is not known to the measurer in advance. As an example, Figure 1 shows the offset-sets for two devices in a two-hour trace of traffic from an Internet backbone OC-48 link on 2004-04-28 (we omit IP addresses for privacy reasons). Respectively shifting the clocks by t_1 and T_1 for x_i and v_i is not necessary for our analysis but makes plots like in Figure 1 easier to read.

If we could assume that the measurer's clock is accurate and that the t values represent true time, and if we could assume that there is no delay between when the fingerprintee generates the i-th packet and when the measurer records the i-th packet, then $y_i = \mathsf{off}(x_i + t_1)$. Under these assumptions, and if we make the additional assumption that R is differentiable, then the first derivative of y, which is the slope of the points in $\mathcal{O}_{\mathcal{T}}$, is the skew s of $\mathsf{C_{tcp}}$. Since we cannot generally make these assumptions, we are left to approximate s from the data.

Let us consider plots like those in Figure 1 more closely. We first observe that the large band corresponds to a device where the TSopt clock has low resolution ($\mathsf{r} = 100$ ms) and that the narrow band corresponds to a device with a higher resolution ($\mathsf{r} = 10$ ms). The width of these bands, and in particular the wide band, means that if the duration of our trace is short, we cannot always approximate the slope of the points in $\mathcal{O}_{\mathcal{T}}$ by computing the slope between any two points in the set. Moreover, as Paxson and others have noted in similar contexts [31, 28], variable network delay renders simple least-squares linear regression insufficient. Figure 2 shows the offset-sets for two

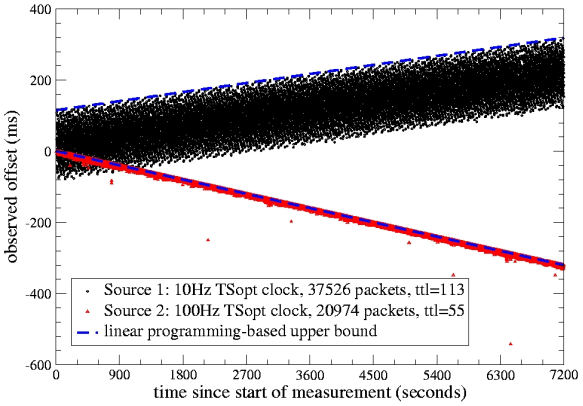

Figure 1. TSopt clock offset-sets for two sources in $\mathsf{BB_N}$. Trace recorded on an OC-48 link of a U.S. Tier 1 ISP, 2004-04-28 19:30–21:30PDT. The source with the wide band has a 10 Hz TSopt clock, the source with the narrow band has a 100 Hz TSopt clock. A source with no clock skew would have a horizontal band.

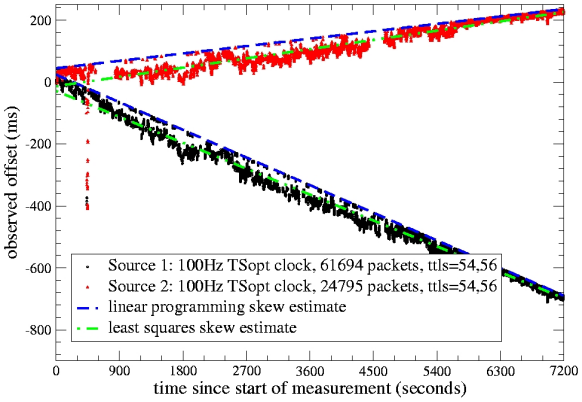

Figure 2. TSopt clock offset-sets for two sources in $\mathsf{BB_N}$. Trace recorded on an OC-48 link of a U.S. Tier 1 ISP, 2004-04-28 19:30–21:30PDT. These offset-sets show how least squares-based skew estimation can be adversely affected by variable network delay, and why we use linear programming-based skew estimation.

hosts in the same /24 that have variable delay. Consequently, to approximate the skew s from $\mathcal{O}_\mathcal{T}$, we borrow a linear programming solution from Moon, Skelly, and Towsley [27, 28], which has as its core Graham's convex hull algorithm on sorted data [16]; see also [13, 23].

The linear programming solution outputs the equation of a line $\alpha x + \beta$ that upper-bounds the set of points $\mathcal{O}_\mathcal{T}$. We use an upper bound because network and host delays are all positive. The slope of the line, α, is our estimate of the clock skew of $\mathsf{C_{tcp}}$. In detail, the linear programming constraints for this line are that, for all $i \in \{1, \ldots, |\mathcal{T}|\}$,

$$\alpha \cdot x_i + \beta \;\geq\; y_i \;,$$

which means that the solution must upper-bound all the points in $\mathcal{O}_\mathcal{T}$. The linear programming solution then minimizes the average vertical distance of all the points in $\mathcal{O}_\mathcal{T}$ from the line; i.e., the linear programming solution is one that minimizes the objective function

$$\frac{1}{|\mathcal{T}|} \cdot \sum_{i=1}^{|\mathcal{T}|} \left(\alpha \cdot x_i + \beta - y_i\right) .$$

Although one can solve the above using standard linear programming techniques, as Moon, Skelly, and Towsley [27, 28] note, there exist techniques to solve linear programming problems in two variables in linear time [13, 23]. We use a linear time algorithm in all our computations.

It remains to discuss how to infer Hz if the measurer does not know it in advance. One solution involves computing the slope of the points

$$\mathcal{I} \;=\; \{\, (x_i, v_i) \;:\; i \in \{1, \ldots, |\mathcal{T}|\,\} \}$$

and rounding to the nearest integer. One can compute the slope of this set by adapting the above linear programming problem to this set.

An equivalent view. If A is the slope of the points in the above set $\mathcal{I}$, derived using the linear programming algorithm, then one could also approximate the skew of $\mathsf{C_{tcp}}$ as $A/\mathsf{Hz} - 1$. We can prove that, when using the linear programming method for slope estimation, both approaches produce the same skew estimate. We use the offset-set approach since these sets naturally yield

figures where the skews are clearly visible; e.g., Figure 1.

5 Exploiting ICMP Timestamp Requests

We now consider how an adversary might obtain samples of a device's system clock and how an adversary could use those samples to fingerprint a physical device.

The measurer. To exploit a device's system time clock skew, the measurer could be any website with which the fingerprintee communicates, or any other device on the Internet provided that the measurer is capable of issuing ICMP Timestamp Requests (ICMP message type 13) to the fingerprintee. The measurer must also be capable of recording the fingerprintee's subsequent ICMP Timestamp Reply messages (ICMP message type 14). The primary limitation is that the device must not be behind a NAT or firewall that filters ICMP.

Estimating the system clock skew. Let us now assume that an adversary has obtained a trace $\mathcal{T}$ of ICMP Timestamp Reply messages from the fingerprintee. The ICMP Timestamp Reply messages will contain two 32-bit values generated by the fingerprintee. The first value is the time at which the corresponding ICMP Timestamp Request packet was received, and the second value is the time at which the ICMP Timestamp Reply was generated; here time is according to the fingerprintee's system clock, $\mathsf{C_{sys}}$, and is reported in milliseconds since midnight UTC. Windows machines report the timestamp in little endian format, whereas all the other machines that we tested report the timestamp in big endian notation. The remaining notation and the method for skew estimation is now identical to what we presented in Section 4, with two minor exceptions. First, the adversary does not have to compute Hz since RFC 792 [32] requires that Hz be 1000 (or, if not, that a special bit be set to indicate non-compliance). Second, since the time reported in the ICMP Timestamp Reply is in milliseconds since midnight UTC, we expect the timestamps reported in the ICMP Timestamp Reply messages to reset approximately once a day; we adjust the v values accordingly. The only thing special that our attack exploits about ICMP is the fact that ICMP has a message type that will reveal a device's system time; our techniques would work equally well with any other protocol that leaks information about a device's system or other clock.

Brief comparison with TCP timestamps. For much of the rest of this paper, we focus on our TCP timestamps-based approach for physical device fingerprinting rather than our ICMP-based approach. We do so not because we consider the ICMP-based approach to be inferior. Rather, we focus on the TCP timestamps-based approach because most systems have TSopt clocks that operate at lower frequencies than the 1000 Hz clocks included in the ICMP timestamp reply messages, which means that it should require less data for an active adversary to mount our ICMP fingerprinting technique than to mount our TCP timestamps technique. Our positive results for the TCP timestamps-based fingerprinting techniques imply that the ICMP-based fingerprinting technique will be effective and will have low data requirements. Focusing on our TCP timestamps based approach also allows us to experiment with machines behind NATs and firewalls. Lastly, for popular operating systems, if a system does not externally synchronize its system time, then the system's TSopt and system clocks will be highly correlated (Section 8), which means that the distribution of system clock skews for machines not using NTP will be similar to the distribution of TSopt clocks skews.

min pkts/hour (minpkts)	min duration/hour (mindur, minutes)	total sources ($\|\mathcal{S}\|$)	sources with stable skews ($\|\mathcal{S}'\|$)	entropy (bits)
0	10	18335	8225	4.87
0	30	13517	6859	5.39
0	50	7246	4120	5.87
500	10	4356	2583	5.99
500	30	4016	2446	6.11
500	50	3368	2104	6.18
2000	10	1730	1116	6.22
2000	30	1629	1077	6.32
2000	50	1489	1009	6.41

Table 3. Entropy estimates from $\mathsf{BB_N}$ when ppmvar = 1 ppm. Trace recorded on an OC-48 link of a U.S. Tier 1 ISP, 2004-04-28 19:30–21:30PDT.

6 Distribution and stability of TSopt clock skew measurements

We now address two fundamental properties that must hold in order for remote clock skew estimation to be an effective physical device fingerprinting technique. First, we show that there is variability in different devices' clock skews, meaning that it is reasonable to expect different devices on the Internet to have measurably different clock skews. Second, we give evidence to suggest that clock skews, as measured by our techniques, are relatively constant over time. These two facts provide the basis for our use of remote clock skew estimation as a physical device fingerprinting technique since they imply that an adversary can gain (sometimes significant) information by applying our techniques to measure a device's or set of devices' clock skews.

The novelty here is not in claiming that these properties are true. Indeed, it is well known that different computer systems can have different clock skews, and others [31, 28, 30, 36] have argued that a given device generally has a constant clock skew. Rather, the contribution here is showing that these properties survive our remote clock skew estimation techniques and, in the case of our analyses of the distribution of clock skews, measuring the bits of information (entropy) a passive adversary might learn by passively measuring the TSopt clock skews of fingerprintees.

Distribution of clock skews: Analysis of passive traces. Our first experiment in this section focuses on understanding the distribution of clock skews across devices as reported by our TCP timestamps-based passive fingerprinting technique. For this experiment we analyzed a passive trace of traffic in both directions of a major OC-48 link; CAIDA collected the trace between 19:30 and 21:30 PDT on 2004-04-28. Since the OC-48 link runs North-South, let $\mathsf{BB_N}$ denote the Northbound trace, and let $\mathsf{BB_S}$ denote the Southbound trace (BB stands for backbone). CAIDA obtained the traces using different Dag [15] cards in each direction; these cards' clocks were synchronized with each other, but not with true time. This latter property does not affect the following discussion because (1) the clock skews of the Dag cards appear to be constant and therefore only shift our skew estimates by a constant amount and (2) here we are only interested in the general distribution of the clock skews of the sources in the traces.

Let minpkts and mindur be positive integers. For simplicity, fix $\mathsf{BB} = \mathsf{BB_N}$ or $\mathsf{BB_S}$. Also assume for simplicity that BB only contains TCP packets with the TCP timestamps option turned on. Recall that the trace BB lasts for two hours. At a high-level, our analysis considers the set $\mathcal{S}$ of sources in BB that have $\geq$ minpkts packets in both the first and the second hours, and where the differences in time between the source's first and last packets in each hour are $\geq$ mindur minutes.

For each source in $\mathcal{S}$, we apply our clock skew estimation technique from Section 4 to the full

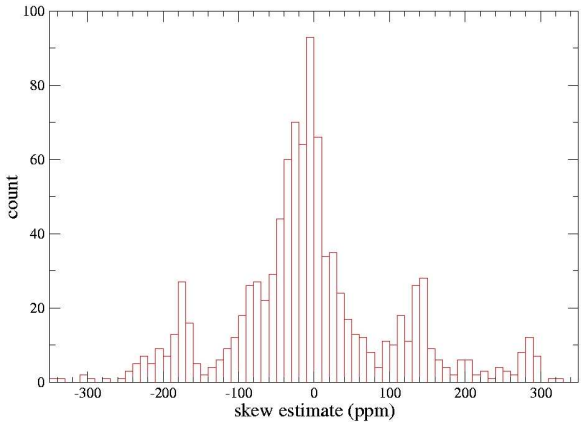

Figure 3. Histogram of TSopt clock skew estimates for sources in $\mathsf{BB_N}$. Trace recorded on an OC-48 link of a U.S. Tier 1 ISP, 2004-04-28 19:30–21:30PDT. Here minpkts = 2000 packets, mindur = 50 minutes, and ppmvar = 1 ppm.

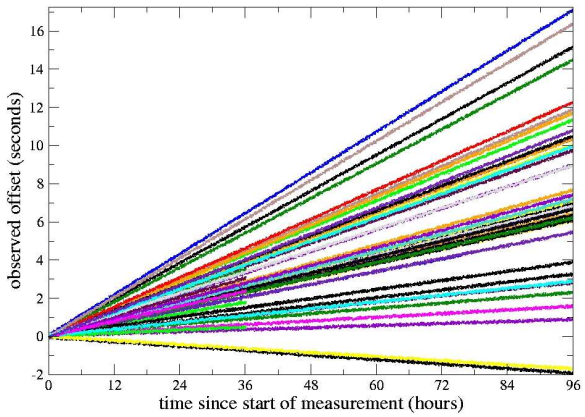

Figure 4. TSopt clock offset-sets for 69 Micron 448MHz Pentium II machines running Windows XP Professional SP1. Trace recorded on `host2`, three hops away, 2004-09-10 08:30PDT to 2004-09-14 08:30PDT.

trace, the first hour only, and the second hour only. Let ppmvar be a positive number, and let $\mathcal{S}'$ be the subset of $\mathcal{S}$ corresponding to the sources whose skew estimates for the full trace, the first hour, and the second hour are all within ppmvar ppm of each other, and whose intended frequency Hz is one of the standard values (1, 2, 10, 100, 512, 1000). If ppmvar is small, then we are inclined to believe that the skew estimates for the sources in $\mathcal{S}'$ closely approximate the true skews of the respective sources. Table 3 shows values of $|\mathcal{S}|$ and $|\mathcal{S}'|$ for different values of minpkts and mindur and $\mathsf{BB} = \mathsf{BB_N}$ and when we arbitrarily choose ppmvar = 1 ppm.

The value $|\mathcal{S}'|/|\mathcal{S}|$ gives an indication of the ratio of sources for which we can accurately (within ppmvar ppm) measure the clock skew. For example, more than 50% of the sources in $\mathcal{S}$ are also in $\mathcal{S}'$ when we consider sources that are active for at least 30 minutes in each hour (mindur = 30). When we add the constraint that each source in $\mathcal{S}$ send at least 500 packets per hour, the percentage of sources in $\mathcal{S}'$ increases to 60%. While useful, this ratio provides little information about the actual distribution of the clock skew estimates. Much more (visually) telling are images such as Figure 3, which shows a histogram of the skew estimates (for the full two hour trace) for all the sources in $\mathcal{S}'$ when minpkts = 2000, mindur = 50 minutes, and ppmvar = 1 ppm. (The true histogram may be shifted horizontally based on the clock skew of the Dag cards, but a horizontal shift does not affect the general shape of the distribution.) Empirically, for any given values for minpkts, mindur, and ppmvar, we can compute the entropy of the distribution of clock skews. Doing so serves as a means of gauging how many bits of information an adversary might learn by passively monitoring a device's clock skew, assuming that devices' clock skews are constant over time, which is something we address later. To compute the entropy, we consider bins of width ppmvar, and for each source s in S', we increment the count of the bin corresponding to devices with clock skews similar to the skew of s (here we use the skew estimate computed over full two hours). We then allocate another bin of size $|S| - |S'|$; this bin counts the number of sources that do not have consistent clock skew measurements. We apply the standard entropy formula [35] to compute the entropy of this distribution of bins, the results of which appear in the last column of Table 3. As one might expect, the amount of information available to an adversary increases as minpkts and mindur increase.

Assuming that clock skews are constant over time, our results suggest that a passive adversary

measured frequency (Hz)	total sources	max skew difference (in ppm) for lower 50%	75%	90%
2	93	2.49	19.81	43.50
10	328	2.76	11.12	94.85
100	860	0.10	0.24	0.76
512	41	0.10	0.30	2.39
1000	74	0.08	0.13	0.35

Table 4. Stability of clock skews in $\mathsf{BB_N}$ for common values of $\mathsf{Hz}[\mathsf{C_{tcp}}]$. Trace recorded on an OC-48 link of a U.S. Tier 1 ISP, 2004-04-28 19:30–21:30PDT. Here $\mathsf{minpkts} = 2000$ packets and $\mathsf{mindur} = 50$ minutes.

could learn at least six bits of information about a physical device by applying our techniques from Section 4. We anticipate that more bits of information will be available to an active adversary since an active adversary might be able to force the fingerprintee to send packets more frequently or over longer periods of time. Additionally, these entropy estimates will be higher for devices with high intended TSopt clock frequencies $\mathsf{Hz}[\mathsf{C_{tcp}}]$ (see Table 4 and the discussion below). The latter observation suggests that high $\mathsf{Hz}[\mathsf{C_{tcp}}]$ values for the purposes of RTT estimation and optimizing TCP performance may imply a slight trade-off in privacy. Similarly, since the TSopt clock frequencies on some systems are derived from the kernels' `HZ` variables, high `HZ` values for the purpose of increasing the performance of some applications [14] may also imply a slight trade-off in privacy.

We can use our two-hour OC-48 traces to evaluate the stability and accuracy of our TSopt clock skew measurements over the course of the two hours. Table 4 considers hosts that transmit more than $\mathsf{minpkts} = 2000$ packets in both the first and the second hours of $\mathsf{BB_N}$ and which also transmit for at least $\mathsf{mindur} = 50$ minutes in both hours. The rows in Table 4 are broken down into (our estimates of the) intended frequencies for this subset of devices. For each device, we use our technique from Section 4 to estimate the clock skew of the device over the whole trace, just the first hour, and just the second hour, and then we compute the maximum difference among these three estimates. The last three columns in Table 4 show the maximum such difference for the lower n-th percentile; e.g., for 90% of the hosts with $\mathsf{Hz}[\mathsf{C_{tcp}}] = 100$ Hz, $\mathsf{minpkts} = 2000$ packets, and $\mathsf{mindur} = 50$ minutes, our skew estimation technique reported clock skews that differed by at most 0.76 ppm between the first hour, the second hour, and the whole trace. Table 4 suggests that our clock skew estimates are generally more accurate for devices with higher intended frequencies; this result is as one would expect since higher frequency clocks have finer granularity than low frequency clocks. (The low number of sources with $\mathsf{Hz}[\mathsf{C_{tcp}}] = 10$ in Table 4 is consistent with our observation from Section 4 that Windows machines do not typically initiate flows with the TCP timestamps option enabled.)

Distribution of clock skews: Experiments with a homogeneous lab. One observation on the above analysis is that we applied it to a wide variety of machines running a wide variety of operating systems. Here we investigate whether the distribution shown in Figure 3 is due to operating system differences or to actual physical differences on the devices. We conducted an experiment with 69 (apparently) homogeneous machines in one of UCSD's undergraduate computing laboratories. All the machines were Micron PCs with 448MHz Pentium II processors running Microsoft Windows XP Professional Service Pack 1. Our measurer, `host2`, was a Dell Precision 410 with a 448MHz Pentium III processor and running Debian 3.0 with a recompiled 2.4.18 kernel; `host2` is located within the University's computer science department and is 3 hops and a half a millisecond away from the machines in the undergraduate laboratory.

To create the requisite trace of TCP packets from these machines, we repeatedly opened and

then closed connections from `host2` to each of these machines. Each open-then-close resulted in the Windows machines sending two packets to `host2` with the TCP timestamps option turned on (the Windows machine sent three packets for each flow, but the TCP timestamp was always zero in the first of these three packets). Because of our agreement with the administrators of these machines, we were only able to open and close connections with these Windows machines at random intervals between zero and five minutes long. Thus, on average we would expect to see each machine send `host2` 48 TCP packets with the TCP timestamps option turned on per hour. The experiment lasted for 38 days, beginning at 19:00PDT 2004-09-07 and ending at approximately 20:30PDT 2004-10-15.

Figure 4 shows a plot, similar to Figure 1, for the 69 Micron machines as measured by `host2` but sub-sampled to one out of every two packets. Note that the plot uses different colors for the observed offsets for different machines (colors are overloaded). Since the slopes of the sets of points for a machine corresponds to the machine's skew, this figure clearly shows that some machines in the lab have measurably different clock skews. Thus, we can easily distinguish some devices by their clock skews (for other devices, we cannot). Because Windows XP machines reset their TSopt clocks to zero when they reboot, some of the diagonal lines seem to disappear several days into the figure. Our algorithms handle reboots by recalibrating the initial observed offset, though this recalibration is not visible in Figure 4. The time in Figure 4 begins on 8:30PDT 2004-09-10 (Friday) specifically because the administrators of the lab tend to reboot machines around 8:00PDT, and beginning the plot on Friday morning means that there are fewer reboots in the figure. We consider this experiment in more detail below, where our focus is on the stability of our clock skew estimates.

Stability of clock skews. We now consider the stability of the TSopt clock skews for the devices in the above-mentioned undergraduate laboratory. Recall that our experiment began at 19:00PDT 2004-09-07 and that the experiment ran for 38 days; see also Figure 4. Consider a single machine in the laboratory. We divide the trace for this machine into 12- and 24-hour periods, discarding 12-hour periods with less than 528 packets from the device, and discarding 24-hour periods with less than 1104 packets from the device (doing so corresponds to discarding 12-hour periods when the device is not up for at least approximately 11 hours, and discarding 24-hour periods that the device is not up for at least 23 hours). We compute the device's clock skew for each non-discarded period, and then compute the difference between the maximum and minimum estimates for the non-discarded periods. This value gives us an indication of the stability of the device's clock skew.

For 12-hour periods, the maximum difference for a single device in the lab ranged between 1.29 ppm and 7.33 ppm, with a mean of 2.28 ppm. For 24-hour periods, the maximum difference for a single device ranged between 0.01 ppm and 5.32 ppm, with a mean of 0.71 ppm. There seems to have been some administrator function at 8:00PDT on 2004-09-10 that slightly adjusted the TSopt clock skews of some of the machines. If we conduct the same analysis for the trace beginning at 8:30PDT 2004-09-10 and ending on 2004-10-15, for 24-hour periods, the range for maximum difference for each device in the lab dropped to between 0.00 ppm and 4.05 ppm. Over 24-hour periods beginning 8:30PDT 2004-09-10, and over all 69 hosts, our minimum skew estimate was -5.94 ppm and our maximum skew estimate was 49.28 ppm. See the table in Appendix A for a complete summary.

The current results strongly support our claim that modern processors have relatively stable clock skews. Moreover we believe that if the administrators of the lab allowed us to exchange more packets with the 69 fingerprintees, we would have found the clock skews to be even more stable. In Section 7 we apply our clock skew estimates to a single computer at multiple locations and on multiple dates, and the skew estimates again are close (Table 5); our results below further support our claim of the stability of clock skews over time.

Laptop location	Start time (PDT)	Duration	Packets	Wireless	NAT	Skew est.
San Diego, CA, home cable	2004-07-09, 22:00	3 hours	181	Yes, WEP	Yes	−58.17 ppm
SD Supercomputer Center	2004-07-10, 10:00	3 hours	182	Yes	No	−58.00 ppm
CSE Dept, UCSD	2004-07-12, 12:00	3 hours	180	Yes	No	−58.24 ppm
San Diego, CA, home cable	2004-07-12, 21:00	3 hours	180	Yes	Yes	−58.21 ppm
Clinton, CT, home cable	2004-07-26, 06:00	3 hours	182	No	Yes	−58.19 ppm
San Diego, CA, home cable	2004-09-14, 21:00	30 min	1795	Yes	Yes	−58.22 ppm
SD Supercomputer Center	2004-09-22, 12:00	30 min	1765	Yes	Yes	−58.13 ppm
San Diego dialup, 33.6kbps	2004-10-18, 10:00	30 min	1749	No	No	−57.57 ppm
SD Public Library	2004-10-18, 14:45	30 min	946	Yes	Yes	−57.63 ppm

Table 5. TCP timestamps-based skew estimates of `laptop` running Red Hat Linux 9.0 when connected to `host1` from multiple locations and when not running `ntpd`. The traces were recorded at `host1`.

7 Access technology-, topology-, and measurer-independent measurements

Here we consider our experiments which suggest that clock skew estimates are relatively independent of the fingerprintee's access technology, the topology between the fingerprintee and the measurer, and the measurer's machine.

Laptops in multiple locations. Our first set of experiments along these lines measures `laptop` connected to the Internet via multiple access technologies and locations (Table 5). For all these experiments, `laptop` is a Dell Latitude C810 notebook with a 1.133GHz Pentium III Mobile processor and running a default installation of Red Hat 9.0 (Linux kernel 2.4.20-8). The measurer in all these experiments, `host1`, is a Dell Precision 340 with a 2GHz Intel Pentium 4 processor located within the UCSD Computer Science and Engineering department and running Debian 3.0 with a recompiled 2.4.18 Linux kernel; `host1` is also configured to synchronize its system time with true time via NTP.

For all experiments, we establish a TCP connection between `laptop` and `host1`, and then exchange TCP packets over that connection. On `host1`, we record a trace of the connection using `tcpdump`. We then use our techniques from Section 4 to estimate the skew of `laptop`'s TSopt clock. As the horizontal line in Table 5 indicates, we divide our experiments into two sets. In the first set, our experiments last for three hours and exchange one TCP packet every minute (we do this by performing a `sleep(60)` on `host1`). For the second set of experiments, the connections last for 30 minutes, and a packet is exchanged at random intervals between 0 and 2 seconds, as determined by a `usleep` on `host1`. With few exceptions, the packets from `laptop` are all ACKs with no data.

We conduct experiments when the laptop is connected to the Internet via residential cable networks on both coasts (Table 5). For our residential experiments, we use a 802.11b wireless connection with 128-bit WEP encryption, a standard (unencrypted) 802.11b wireless connection, and a standard 10Mbps 10baseT wired connection. We also conducted experiments with our laptop connected to the San Diego Supercomputer Center's 802.11b wireless network, from the UCSD Computer Science and Engineering wireless network, and from the San Diego Public Library's wireless network. As the final column in the table shows, the skew estimates are all within a fraction of a ppm of each other. (If we subsample the first set of experiments to one packet every 3 minutes, then the difference between the skew estimates for any two measurements in the first set is at most 0.45 ppm.)

PlanetLab and topology questions. Although the above results strongly suggest that skew

Measurer	`laptop`, 2004-09-17, 08:00–10:00 PDT Skew estimate	Dist. from measurer		`laptop`, 2004-10-08, 08:00–10:00 PDT Skew estimate	Dist. from measurer	
`host1`	−58.23 ppm	7 hops	2.77 ms	−58.03 ppm	8 hops	1.16 ms
San Diego, CA	−58.07 ppm	7 hops	1.21 ms	−58.03 ppm	8 hops	1.15 ms
Berkeley, CA	−58.17 ppm	10 hops	4.02 ms	−58.02 ppm	12 hops	5.06 ms
Seattle, WA	−58.15 ppm	8 hops	14.74 ms	−58.01 ppm	9 hops	15.12 ms
Toronto, Canada	−58.31 ppm	16 hops	44.43 ms			
Princeton, NJ	−57.97 ppm	13 hops	37.59 ms	−57.91 ppm	14 hops	36.97 ms
Boston, MA	−57.93 ppm	12 hops	35.82 ms	−58.10 ppm	13 hops	41.09 ms
Cambridge, UK	−58.06 ppm	20 hops	84.19 ms	−58.18 ppm	21 hops	86.45 ms
ETH, Switzerland	−58.38 ppm	20 hops	90.51 ms	−58.40 ppm	21 hops	84.07 ms
IIT, India				−59.60 ppm	16 hops	199.27 ms
Equinix, Singapore	−58.10 ppm	18 hops	99.50 ms	−58.05 ppm	15 hops	93.55 ms
CAIDA test lab				−57.98 ppm	5 hops	0.24 ms

Table 6. Skew estimates of `laptop`, running Red Hat 9.0 with `ntpd`, for traces taken simultaneously at multiple locations. On 2004-09-17 the laptop was connected to the SDSC wireless network, and on 2004-10-08 the laptop was connected to the CAIDA wired network. The Toronto and India lines have empty cells because the PlanetLab machines at those locations were down during the experiment. The Boston machine on 2004-10-08 was a different PlanetLab machine than the one on 2004-09-17. The empty cell for the CAIDA test lab is because the lab is only reachable from CAIDA's wired network.

Measurer	`host1`, 2004-09-16, 16:00–18:00 PDT Skew estimate	Dist. from measurer		`host1`, 2004-10-19, 08:00–10:00 PDT Skew estimate	Dist. from measurer	
`host1`	−12.98 ppm	localhost		−13.15 ppm	localhost	
San Diego, CA	−12.78 ppm	1 hop	5.45 ms	−13.21 ppm	1 hop	1.48 ms
Berkeley, CA	−12.93 ppm	12 hops	6.57 ms	−12.72 ppm	12 hops	3.81 ms
Seattle, WA	−12.65 ppm	10 hops	18.91 ms	−13.25 ppm	10 hops	15.74 ms
Toronto, Canada	−12.85 ppm	18 hops	47.94 ms			
Princeton, NJ	−13.07 ppm	15 hops	41.73 ms	−13.21 ppm	15 hops	34.48 ms
Boston, MA	−12.94 ppm	14 hops	44.32 ms	−12.82 ppm	14 hops	39.68 ms
Cambridge, UK	−13.02 ppm	22 hops	84.28 ms	−13.25 ppm	22 hops	88.19 ms
ETH, Switzerland	−12.83 ppm	22 hops	88.33 ms	−12.99 ppm	22 hops	85.71 ms
IIT, India	−13.62 ppm	19 hops	181.02 ms	−12.25 ppm	17 hops	198.73 ms
Equinix, Singapore	−12.98 ppm	20 hops	104.24 ms	−12.69 ppm	17 hops	158.82 ms

Table 7. Skew estimates of `host1` for traces taken simultaneously at multiple locations. If we expand the measurement window of `host1` to 16:00–20:00 PDT, then the skew estimate from India becomes −12.98 ppm. The Toronto line has an empty cell because the PlanetLab machines at that location were down during the experiment. The Boston machine on 2004-10-19 was a different PlanetLab machine than the one on 2004-09-16.

estimates are independent of access technology, the above experiments do not stress-test the topology between the fingerprinter and the fingerprintee. Therefore, we conducted the following set of experiments. We selected a set of PlanetLab nodes from around the world that reported, via `ntptrace`, approximately accurate system times. We chose PlanetLab machines located at UC San Diego, UC Berkeley, U. of Washington, U. of Toronto (Canada), Princeton, MIT, U. of Cambridge (UK), ETH (Switzerland), IIT (India), and Equinix (Singapore). These PlanetLab machines, along with `host1` and (in one case) CAIDA's test machine with a CDMA-synchronized Dag card, served as our fingerprinters. Our fingerprintees were `laptop` and `host1`, where `laptop` was connected both to the SDSC wireless and to the CAIDA wired networks.

For each of our experiments, and for each of our chosen PlanetLab nodes, we created a flow between the node and the fingerprintee. Over each flow our fingerprintee sent one packet at random intervals between 0 and 2 seconds; here the fingerprintee executed `usleep` with appropriate parameters. We then recorded the flows on the PlanetLab machines using `plabdump`, the `tcpdump` equivalent for PlanetLab machines. On `host1` we recorded the corresponding flow using `tcpdump`. And on the machine with the Dag card we used `Coral` [19] (that machine was only reachable when `laptop` was connected directly to CAIDA's wired network). We then computed the skew using the techniques from Section 4. The results are shown in Table 6 and Table 7. Notice that the skew estimates are in general within a fraction of a ppm of each other, suggesting that our skew estimates are independent of topology.

For distance measurements for Table 6 and Table 7, we used traceroute to determine hop count, and then used *mean* time between when `tcpdump` recorded a packet on the measured device and the time between when `plabdump` recorded the packet on the measurer. This distance estimate also includes the time spent in the application layers on the machines, but should give a rough estimate of the time it takes packets to go from the fingerprintee to the measurer.

The results of these experiments suggest that our TSopt clock skew estimation technique is generally independent of the topology and distance between the fingerprinter and the fingerprintee. Furthermore, these results suggest that our skew estimation technique is independent of the actual fingerprinter, assuming that the fingerprinter synchronizes its system time with NTP [26] or something better [36].

8 Effects of operating system, NTP, and special cases

Operating systems and NTP on fingerprintee. In Table 8 we show skew estimates for the same physical device, `laptop`, running both Red Hat 9.0 and Windows XP SP2, and both with and without NTP-based system clock synchronization. (For this experiment, `laptop` sent one packet to the measurer, `host1`, at random intervals between 0 and 2 seconds; `laptop` was connected to the SDSC wireless network, and was 7 hops away from `host1`; `host1` also sent a ICMP Timestamp Request to `laptop` at random intervals between 0 and 60 seconds.) The table shows that, for the listed operating systems, the system clock and the TSopt clock effectively have the same clock skew when the device's system time is not synchronized with NTP, and that the TSopt clock skew is independent of whether the device's system clock is maintained via NTP. Although not shown in the figure, our experiments with OpenBSD 3.5 on another machine suggest that the TSopt clock and system clock on default OpenBSD 3.5 installations have the same skew (approximately 68 ppm). On the other hand, at least with this test machine, the TSopt clock and system clock on a default FreeBSD 5.2.1 system have different skews (the TSopt clock skew estimate is about the same as with OpenBSD, but the system clock skew estimate is approximately 80 ppm). When we turn on `ntpd` under FreeBSD 5.2.1, the TSopt clock skew remained unchanged.

Start time	Operating system	NTP	skew estimate (TCP tstamps)	skew estimate (ICMP tstamps)
2004-09-22, 12:00 PDT	Red Hat 9.0	No	−58.20 ppm	−58.16 ppm
2004-09-17, 08:00 PDT	Red Hat 9.0	Yes	−58.16 ppm	−0.14 ppm
2004-09-22, 21:00 PDT	Windows XP SP2	No	−85.20 ppm	−85.42 ppm
2004-09-23, 21:00 PDT	Windows XP SP2	Yes	−84.54 ppm	1.69 ppm

Table 8. Experiments for the same physical device, `laptop`, running different operating systems and with NTP synchronization both on and off. For all experiments, `laptop` was located on the SDSC wireless network. All traces last for six hours. Additionally, `laptop` was up for an hour before the Windows measurements.

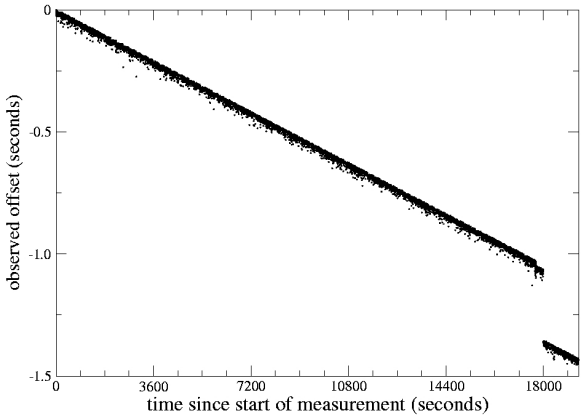

Figure 5. TSopt clock offset-sets for `laptop` running Red Hat 9.0, starting shortly after `laptop` boots; power removed toward end of trace. Measurer is `host1` and `laptop` is located on a residential cable system. Beginning 19:00PST 20041119.

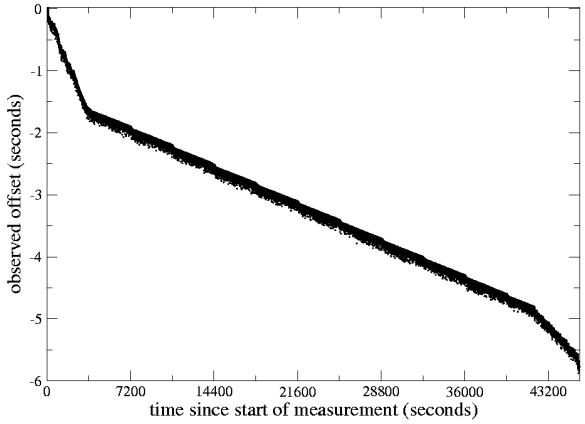

Figure 6. TSopt clock offset-sets for `laptop` running Windows XP SP2, starting shortly after `laptop` boots; power removed toward end of trace. Measurer is `host1` and `laptop` is located on a residential cable system. Beginning 17:30PST 20041117.

Power options for laptops. We also consider how the clock skews of devices are affected by the power options of laptops. In the case of Red Hat 9.0, when `laptop` is running with the power connected, if we switch to battery power, there is a brief jump in the TSopt clock offset-set for the device, and then the device continues to have the same (within a fraction of a ppm) clock skew; see Figure 5. For `laptop` running Windows XP SP2, if the laptop is idle from user input but continues to maintain a TCP flow that we can monitor, then the TSopt clock skew changes after we switch to battery power; see Figure 6. If we repeat this experiment several times, and if we boot with only battery power, we find that the clock skews with battery power are in all cases similar. When booting with outlet power, the clock skew on `laptop` running Windows XP initially begins with a large magnitude, and then stabilizes to a skew like that in Table 8 until we disconnect the power; the initially large skew may be due to the laptop recharging its batteries. We have not sampled a large enough set of laptops to determine whether the clock skews with battery power are a simple function of the clock skews with outlet power, though the skews with battery power seem to be consistent for a single laptop.

Temperature. Although our experiments do not suggest a significant (beyond a fraction of a ppm) variation in `laptop`'s skew when the surrounding temperature varies (from a temperature-controlled machine room to an un-airconditioned room during the summer), we did not rigorously investigate the effects of temperature on our clock skew estimates. We acknowledge that such a study would help provide greater insights into the efficacy of our techniques. If a rigorous study finds that temperature variations do cause remotely-detectable changes in modern devices' TSopt clock skews, then the information leakage about the environment surrounding a device might be useful to certain adversaries.

9 Applications

We now consider some applications of our techniques, though we emphasize that we consider our most important results to be the foundations we introduced in the previous sections that make the following applications possible.

Virtualization and virtual honeynets. We created a `honeyd` [33] version 0.8b virtual honeynet consisting of 100 Linux 2.4.18 virtual hosts and 100 Windows XP SP1 virtual hosts. Our server in this experiment, `host3`, is identical to `host1`, has 1GB of RAM, and maintains its system time via NTP. We ran `honeyd` with standard `nmap` and `xprobe2` configuration files as input; `honeyd` used the information in these files to mimic real Linux and Windows machines. We ran `nmap` and `xprobe2` against the virtual hosts to verify that `nmap` and `xprobe2` could not distinguish the virtual hosts from real machines.

We applied our TCP timestamps- and ICMP-based skew estimation techniques to all 200 virtual hosts. Our fingerprinter in this experiment was on the same local network. We observed several methods for easily distinguishing between `honeyd` virtual hosts and real machines. First, we noticed that unlike real Linux and Windows machines, the virtual hosts always returned ICMP Timestamp Replies with zero in the transmit timestamp field. Additionally, we observed that the `honeyd` Windows XP virtual hosts had TSopt clocks $\mathsf{C}_{\mathsf{tcp}}$ with $\mathsf{Hz}[\mathsf{C}_{\mathsf{tcp}}] = 2$, whereas all of the real Windows XP machines that we tested had $\mathsf{Hz}[\mathsf{C}_{\mathsf{tcp}}] = 10$. The lesson here is that although the `nmap` and `xprobe2` configuration files provide enough information for the respective programs to effectively fingerprint real operating systems, the configuration files do not provide enough information for `honeyd` to be able to correctly mimic all aspects of the Linux and Windows protocol stacks.

Even if `honeyd` completely mimicked the network stacks of real Linux 2.4.18 and Windows XP SP1 machines, we could still use our remote physical device fingerprinting techniques to distinguish

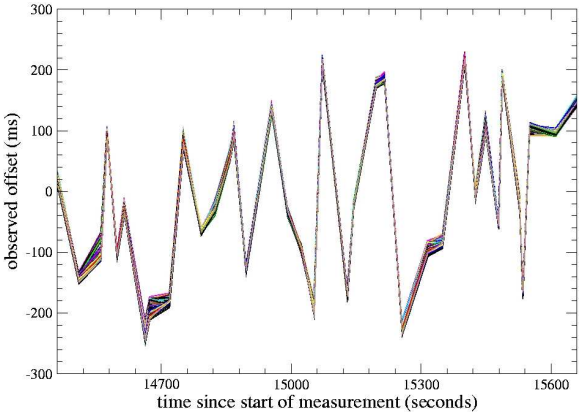

Figure 7. TSopt clock offset-sets for 100 `honeyd` 0.8b Windows XP SP1 virtual hosts. Start time: 2004-09-19, 23:00PDT; `honeyd` running on `host3`. Points are connected in this figure to highlight the correlation between the virtual hosts.

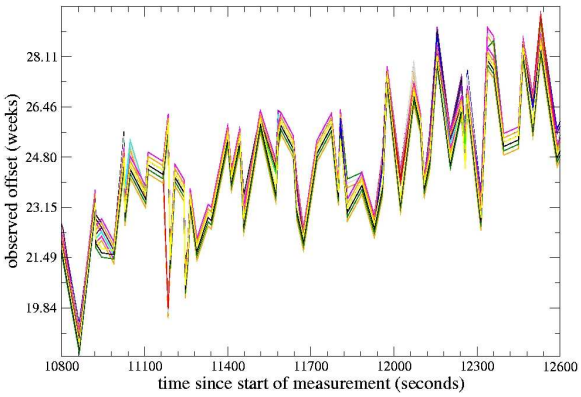

Figure 8. ICMP-based system clock offset-sets for 100 `honeyd` 1.0 Linux 2.4.18 virtual hosts. Start time: 2005-01-21, 19:00PDT; `honeyd` running on `host1`. Points are connected in this figure to highlight the correlation between the virtual hosts.

between our 200 virtual hosts and 200 real machines. Our TSopt clock skew estimates for all 200 virtual hosts were approximately zero and the system clock skew estimates for all 200 virtual hosts were approximately the same positive value. Given our discussion in Section 6 of the distribution of clock skews, this lack of variability in clock skews between virtual hosts is not what one would expect from real machines. Furthermore, the TSopt and system clocks between all the virtual hosts of the same operating system were highly correlated; e.g., Figure 7 shows the TSopt offset-sets for all 100 Windows XP SP1 virtual hosts 241 minutes into our experiment. In Figure 7 we connect the points in the offset-sets for each virtual host to highlight the correlation between the hosts. Recall that we observed no such correlation in our experiment with 69 real Windows XP machines (Figure 4). We communicated our results to the author of `honeyd` and, in response, version 1.0 of `honeyd` randomly assigns TSopt clock skews to each virtual host using a Gaussian distribution around the server's system time. This decision may affect other components of the system, e.g., if the server runs `ntpd`, changes to the server's system time may appear as global changes to the distribution of the virtual hosts' clocks. Version 1.0 of `honeyd` still issues ICMP Timestamp Replies with zero transmit timestamps. Furthermore, the system clocks on version 1.0 `honeyd` virtual hosts are still highly synchronized and are too fast by several orders of magnitude (Figure 8; the vertical axis is correctly labeled in weeks).

To experiment with real virtualization technologies, we installed VMware Workstation 4.5.2 on `host3`, but this time `host3` ran Red Hat 9.0. We then installed five default copies of Red Hat 9.0 under VMware. We applied our skew estimation techniques to these five virtual machines, as well as to `host3`. The results show that the five virtual machines do not have constant (or near constant) clock skews, shown by the non-linearity of the points in Figure 9. Furthermore, the overall magnitude of the clock skews on these virtual machines (greater than 400 ppm) is larger than we would expect for physical desktop machines. We feel confident that these observations and natural extensions could prove useful in distinguishing virtual honeynets from real networks.

Counting the number of devices behind a NAT. Another natural application of our techniques is to count the number of devices behind a NAT. To briefly recall previous work in this area, Bellovin [7] showed that an adversary can exploit the IP ID field to count the number of devices

behind a NAT, but his approach is limited in three ways: (1) the IP ID field is only 16-bits long; (2) recent operating systems now use constant or random IP ID fields; and (3) his technique cannot count the total number of devices behind a NAT if not all of them are active at the same time. Our suggested approach to this problem has two phases. First, partition the trace into (candidate) sets corresponding to different sequences of time-dependent TCP timestamps; creating such a partition is relatively easy to do unless two machines have approximately the same TSopt clock values at some point in time, perhaps because the machines booted at approximately the same time. Then apply our clock skew estimation techniques to each partition, counting hosts as unique if they have measurably different clock skews. If two devices have approximately the same TSopt clock values at some point in time but have measurably different clock skews, then one can detect and correct this situation in the analysis of the partition's offset-set.

Forensics and tracking individual devices. The utility of our techniques for forensics purposes follows closely from our claims (1) that there is variability in the clock skews between different physical devices (Section 6), (2) that the clock skew for a single device is approximately constant over time (Section 6), and (3) that our clock skew estimates are independent of access technology, topology, and the measurer (Section 7). For forensics, we anticipate that our techniques will be most useful when arguing that a given device was not involved in a recorded event. With respect to tracking individual devices, we stress that our techniques do not provide unique serial numbers for devices, but that our skew estimates do provide valuable bits of information that, when combined with other sources of information such as operating system fingerprinting results, can help track individual devices on the Internet.

DHCP. The use of DHCP can cause significant problems for many forms of network mapping since the measurer may not be able to uniquely identify a node by its IP address and therefore has to deal with a mix of measurements coming from different hosts, e.g., [8]. The use of DHCP thus renders conclusions about any kind of network statistics tenuous because one is unable to say whether the measured phenomena represents a certain fraction of hosts; the measurement is influenced by the dynamics of hosts joining and leaving the network. Any technique that can help disambiguate hosts behind a DHCP server therefore has both network mapping and security applications. One might be able to use our techniques to help remotely track (with some probability) the assignment of IP addresses within an address block to physical machines via DHCP.

Unanonymizing anonymized data sets. It is common for organizations that provide network traces containing payload data to anonymize the IP addresses in the traces using some prefix-preserving anonymization method [38, 39]. If an organization makes available both anonymized and unanonymized traces from the same link, one can use our techniques to catalyze the unanonymization of the anonymized traces. Such a situation is not hypothetical: in addition to the 2004-04-28 trace that we used in Section 6, CAIDA took another trace from the same link on 2004-04-21, but the 2004-04-21 trace included payload data and was therefore anonymized.

To study how one might use our clock skew estimation techniques to help unanonymize anonymized traces, on 2005-01-13 and 2005-01-21 CAIDA took two two-hour traces from a major OC-48 link (the same link from which CAIDA captured the 2004-04-28 trace). We anonymized the 2005-01-13 trace and experimented with our ability to subsequently unanonymize it. Given the value of a device's TSopt clock and knowledge of that clock's intended frequency Hz, we can compute the approximate uptime of the device. (Prior to our work, one method for inferring Hz from a passive trace would be to use a program like `p0f` [3].) As a first attempt at unanonymizing the 2005-01-13 trace, we paired anonymized IP addresses from 2005-01-13 with IP addresses from 2005-01-21 when our uptime estimate of a host in 2005-01-21 is eight days higher (plus or minus five minutes)

than the uptime of a host in 2005-01-13 and when both hosts have the same TTLs and intended frequencies. Our program produced 4613 pairs of candidate anonymous to real mappings, of which 2660 (57.66%) were correct. To reduce the number of false matches, especially for small uptimes, we modified our program to filter out pairs that have TSopt clock skews different by more than 3 ppm. We also incorporated our clock skew estimates into our uptime estimates. These changes reduced the number of candidate mappings to 2170, of which 1902 (87.65%) were correct; the fraction of false positives was reduced by over 3.4 (from 42.34% to 12.35%). There are a total of 11862 IP addresses in both the 2005-01-13 and 2005-01-21 traces that have the TCP timestamps option enabled. Since the anonymization is prefix-preserving, given the candidate mappings one can begin to unanonymize address blocks. We are unaware of any previous discussion of the problems to prefix-preserving anonymization caused by leaking information about a source via the TCP timestamps option.

Intentional clock skew abnormalities. Most users would likely not notice if their devices had large or non-constant TSopt clock skews. A hardware vendor wishing to track physical devices with few false positives could therefore design their devices to have large or otherwise abnormal clock skews. A user could also use his or her device's TSopt clock as a covert channel by intentionally varying the device's TSopt clock skew in a controlled manner. This observation confirms McDanel's conjecture [22] that it may be possible to embed covert information in the TCP timestamps option field of a device's outgoing packets. We consider the applications in this subsection to be mostly of academic interest.

10 Potential countermeasures

The primary focus of this paper is on developing techniques to fingerprint current generation physical devices when running current generation operating systems. Although (by definition) the techniques we describe above will remain applicable to current generation systems, we suspect that future generation security systems might incorporate countermeasures to some of the fingerprinting techniques that we uncover. We explore some possible protection mechanisms in this section.

The surest way for a device to protect itself against our ICMP-based fingerprinting technique (Section 5) would be for the device to not reply to ICMP Timestamp Requests. Another solution might be for the device to synchronize its system time via NTP. Although synchronizing a device's system time with NTP would address our current ICMP-based fingerprinting technique, we caution that this solution might still be susceptible to more sophisticated fingerprinting techniques that exploit detectable clock skews between NTP adjustments (or the NTP server could be the adversary wishing to fingerprint different devices).

Similar to an observation above, the surest way for a device to protect itself against our TCP timestamps-based fingerprinting technique (Section 4) would be for the device to not enable the TCP timestamps option in outgoing packets. If there are circumstances that make this solution undesirable (e.g., because the use of TCP timestamps improves RTT estimation and TCP performance), another approach for protecting against our TCP timestamps-based fingerprinting technique would be to reduce a device's clock skew. An operating system might reduce its clock skew by, at boot, making a more precise estimation of the oscillator frequencies supplying the hardware basis for its clocks. An operating system might also incorporate the techniques for precise software clocks from Pásztor and Veitch [30] and Veitch, Babu, and Pásztor [36]. An operating system might also pick a random multiplication factor at boot and multiply its TCP timestamps by that factor in order to mask its clock skew.

Although the above suggestions may protect against our current TCP timestamps-based finger-

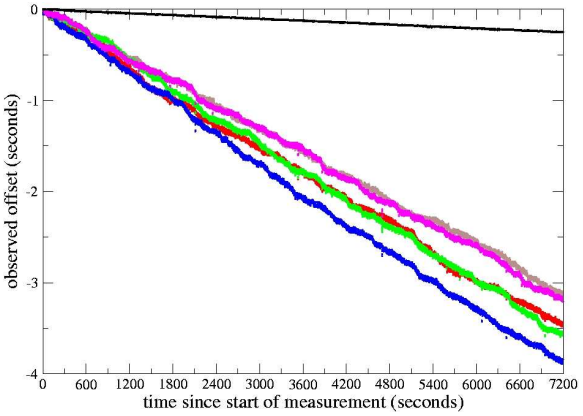

Figure 9. TSopt clock offset-sets for five VMware Workstation virtual machines running Red Hat 9.0, and for the host, `host3`, also running Red Hat 9.0. 2004-10-27 17:00–19:00PDT. The top set of points corresponds to the TSopt clock offset set for `host3`.

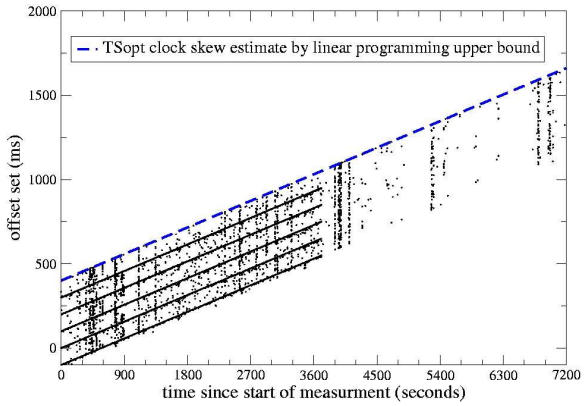

Figure 10. TSopt clock skew estimate for a source in BB_5. Trace recorded on an OC-48 link of a U.S. Tier 1 ISP, 2004-04-28 19:30–21:30PDT. TSopt clock skew estimate via linear programming: 175.2 ppm. Clock skew estimate via the Fourier transform: 175.6 ppm.

printing technique, the above techniques may still leak some information about a device in the TCP timestamps option field of each outgoing packet. To address this concern, from RFC 1323 [18] we conclude that two parties in a TCP flow do not actually need to know each other's TCP timestamps. Consequently, a device could encrypt its timestamps using a secret key, assuming an appropriate encryption mechanism for 32-bit blocks. The device could also maintain a table mapping the (possibly random) 32-bit values that it includes in the TCP timestamps fields of outgoing packets to its internal representation of real timestamps; the table should only need to be as large as the device's TCP retransmission window. We have not evaluated the performance of these recommendations. We do remark that the recommendations in this paragraph break strict compliance with RFC 1323 since the RFC says: "The timestamp value to be sent in TSval is to be obtained from a (virtual) clock that we call the 'timestamp clock'. *Its values must be at least approximately proportional to real time*, in order to measure actual RTT [18]."

11 Other measurement techniques

In Section 10 we argue that although the techniques we explore in this paper will likely remain applicable to current generation systems, future generation security systems might try to resist some of our techniques; e.g., future generation systems might incorporate some of the protection mechanisms from Section 10. In anticipation of these future systems, we consider possible avenues for clock-based physical device fingerprinting when information about a system's TSopt clock or system clock is not readily available to an adversary; we do not consider here but recognize the possibility of fingerprinting techniques that profile other aspects of a device's hardware, e.g., processor speed or memory. These directions assume that new operating systems mask or do not include the TSopt clock values in the TCP headers and do not reply to ICMP Timestamp Requests, but that the systems' underlying clocks still have non-negligible skews. The techniques we propose in this section are less refined than the techniques elsewhere in this paper; we envision the techniques here as starting points for more sophisticated techniques.

FOURIER TRANSFORM. Some systems send packet at 10 or 100 ms intervals, perhaps due to interrupt processing or other internal operating system feature on one side of a flow. When this condition holds, we can use the Fourier transform to extract information about the system's clock skew. Figure 10 plots the TSopt clock offset-sets for a device in $\mathsf{BB_S}$ with a 2 Hz TSopt clock. The five diagonal bands suggests that the machine clusters packet transmissions at approximately 100 ms intervals, and we can use the Fourier transform on packet arrival times to estimate the frequency at which the device actually transmits packets (here packet arrival times refers to the times at which the monitor records the packets). For the source shown in Figure 10, after computing the Fourier transform, the frequency with the highest amplitude was 25.00439, which implies a skew of $25.00439/25 - 1$, or 175.6 ppm. Moreover the top 19 frequencies output by the Fourier transform all imply skews between 171.0 ppm and 179.3 ppm. These values are all close to the 175.2 ppm output by our TCP timestamps-based approach but do not make any use of the TCP timestamps contained with the packets.

Although our Fourier-based technique does not require knowledge of a device's TSopt or system clocks, our Fourier-based solution is currently not automated. This lack of automation, coupled with the fact that current generation systems readily relinquish information about their TSopt and system clocks, means that our Fourier-based solution is currently less attractive than the techniques we described in Sections 4 and 5. If in the future operating system designers decide to address the information leakage concerns we raise with respect to the TCP timestamps option and ICMP Timestamp Requests, then the technique we mention here may become more relevant.

PERIODIC USER-LEVEL ACTIVITIES. Toward estimating the system clock skew of devices that do not synchronize their system times with NTP, we note that many applications perform certain operations at semi-regular intervals. For example, one can configure most mail clients to poll for new mail every n minutes. As another example, Broido, Nemeth, and claffy show that some Microsoft Windows 2000 and XP systems access DNS servers at regular intervals [11]. It may be possible to infer information about a device's system clock skew by comparing differences between actual intervals of time between these periodic activities and what the application intends for those intervals of time to be.

12 Conclusions

In this study we verified the ability and developed techniques for remote physical device fingerprinting that exploit the fact that modern computer chips have small yet non-trivial and remotely detectable clock skews. We showed how our techniques apply to a number of different goals, ranging from remotely distinguishing between virtual honeynets and real networks to counting the number of hosts behind a NAT. Although the techniques we described will likely remain applicable to current generation systems, we suspect that future generation security systems might offer countermeasures to resist some of the fingerprinting techniques that we uncover. In anticipation of such developments, we discussed possible avenues for physical device fingerprinting when information about a system's TSopt clock or system clock are not readily available to the adversary. Our results compellingly illustrate a fundamental reason why securing real-world systems is so genuinely difficult: it is possible to extract security-relevant signals from data canonically considered to be noise. This aspect renders perfect security elusive, and even more ominously suggests that there remain fundamental properties of networks that we have yet to integrate into our security models.

Acknowledgments

We thank Young Hyun, David Moore, Bruce Potter, Stefan Savage, Tsutomu Shimomura, and Darryl Veitch for helpful discussions, Emile Aben, Dan Andersen, Colleen Shannon, and Brendan White for collecting some of the traces that we analyzed, Pat Wilson for allowing us to experiment with the machines in one of UCSD's undergraduate computing labs, and William Griswold for loaning us PDAs from the HP Mobile Technology Solutions gift to the ActiveCampus project. A. Broido thanks UCLA IPAM for a visit in Spring 2002 when he started working on network spectroscopy.

References

[1] Endace measurement systems, 2004. URL: `http://www.endace.com/`.

[2] Nmap free security scanner, 2004. URL: `http://www.insecure.org/nmap/`.

[3] Project details for p0f, 2004. URL: `http://freshmeat.net/projects/p0f/`.

[4] VMware virtual infrastructure, 2004. URL: `http://www.vmware.com/`.

[5] Xprobe official home, 2004. URL: `http://www.sys-security.com/index.php?page=xprobe`.

[6] K. G. Anagnostakis, M. Greenwald, and R. S. Ryger. cing: Measuring network-internal delays using only existing infrastructure. In *INFOCOM*, 2003.

[7] S. M. Bellovin. A technique for counting NATted hosts. In *IMW*, 2002.

[8] R. Bhagwan, S. Savage, and G. M. Voelker. Understanding availability. In *The 2nd International Workshop on Peer-to-peer systems*, 2003.

[9] A. Broido, Y. Hyun, and k. claffy. Spectroscopy of traceroute delays. In *PAM*, 2005.

[10] A. Broido, R. King, E. Nemeth, and k. claffy. Radon spectroscopy of inter-packet delay. In *Proceedings of the IEEE High-Speed Networking Workshop*, 2003.

[11] A. Broido, E. Nemeth, and k. claffy. Spectroscopy of DNS update traffic. In *SIGMETRICS*, 2003.

[12] S. Donnelly. *High precision timing in passive measurements of data networks*. Ph.D. thesis, University of Waikato, Hamilton, New Zealand, 2002.

[13] M. E. Dyer. Linear time algorithms for two- and three-variable linear programs. *SIAM J. Comput.*, 13, 1984.

[14] Y. Etsion, D. Tsafrir, and D. G. Feitelson. Effects of clock resolution on the scheduling of interactive and soft real-time processes. In *SIGMETRICS*, 2003.

[15] I. D. Graham, M. Pearson, J. Martens, and S. Donnelly. Dag - A cell capture board for ATM measurement systems. URL: `http://dag.cs.waikato.ac.nz/dag/papers/dag1997.html`.

[16] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1, 1972.

[17] A. Hussain, J. Heidemann, and C. Papadopoulos. A framework for classifying denial of service attacks. In *SIGCOMM*, 2003.

[18] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. RFC 1323, May 1992.

[19] K. Keys, D. Moore, R. Koga, E. Lagache, M. Tesch, and k. claffy. The architecture of the CoralReef Internet traffic monitoring software suite. In *PAM*, 2001.

[20] T. Kohno, A. Broido, and k. claffy. Remote physical device fingerprinting. In *IEEE Symposium on Security and Privacy*, 2005. Extended abstract of this paper.

[21] T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2), 2005.

[22] B. McDanel. TCP timestamping and remotely gathering uptime information. Email to `bugtraq@securityfocus.com`, 2001.

[23] N. Megiddo. Linear-time algorithms for linear programming in r^3 and related problems. *SIAM J. Comput.*, 12, 1983.

[24] J. Micheel, S. Donnelly, and I. Graham. Precision timestamping of network packets. In *IMW*, 2001.

[25] D. Mills. Simple network time protocol (SNTP) version 4 for IPv4, IPv6 and OSI. RFC 2030, 1996.

[26] D. L. Mills. Network time protocol (version 3): Specification, implementation and analysis. RFC 1305, 1992.

[27] S. B. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. Department of Computer Science University of Massachusetts at Amherst Technical Report, 98-43, October 1998.

[28] S. B. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. In *INFOCOM*, 1999.

[29] C. Partridge, D. Cousins, A. W. Jackson, R. Krishnan, T. Saxena, and W. T. Strayer. Using signal processing to analyze wireless data traffic. In *WiSe*, 2002.

[30] A. Pásztor and D. Veitch. PC based precision timing without GPS. In *SIGMETRICS*, 2002.

[31] V. Paxson. On calibrating measurements of packet transit times. In *SIGMETRICS*, 1998.

[32] J. Postel. Internet control message protocol. RFC 792, 1981.

[33] N. Provos. A virtual honeypot framework. In *Usenix Security 2004*, 2004.

[34] R. Rager. XMIT_ID version 2.61, 2005. URL: `http://xmit.penguinman.com/xmit_id.html`.

[35] C. Shannon. *The mathematical theory of communication.* 1949. Urbana, University of Illinois Press.

[36] D. Veitch, S. Babu, and A. Pásztor. Robust synchronization of software clocks across the Internet. In *IMC*, 2004.

[37] F. Veysset, O. Courtay, and O. Heen. New tool and technique for remote operating system fingerprinting, 2002. URL: `http://www.intranode.com/fr/doc/ring-short-paper.pdf`.

[38] J. Xu, J. Fan, M. Ammar, and S. B. Moon. On the design and performance of prefix-preserving IP traffic trace anonymization. In *IMW*, 2001.

[39] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon. Prefix-preserving IP address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In *ICNP*, 2002.

A Experiment with undergraduate computing facility

This table accompanies the discussion in Section 6 and summarizes information about the clock skew estimates for the 69 machines in the undergraduate computing facility from 8:30PDT 2004-09-10 to 2004-10-15.

host IP	12 hour periods			24 hour periods		
(last byte)	# periods	min skew est	Δ max	# periods	min skew est	Δ max
191	39	−9.37 ppm	5.49 ppm	10	−5.22 ppm	0.44 ppm
167	26	−6.47 ppm	1.85 ppm	6	−5.94 ppm	0.78 ppm
189	34	−1.81 ppm	6.07 ppm	8	0.91 ppm	2.62 ppm
156	34	1.09 ppm	2.65 ppm	10	2.02 ppm	0.56 ppm
158	30	3.41 ppm	5.17 ppm	7	4.17 ppm	3.20 ppm
152	25	3.47 ppm	2.07 ppm	4	4.76 ppm	0.35 ppm
163	39	3.51 ppm	4.13 ppm	10	6.02 ppm	0.65 ppm
161	30	6.83 ppm	2.77 ppm	7	7.40 ppm	0.66 ppm
195	30	7.35 ppm	1.59 ppm	8	7.96 ppm	0.55 ppm
142	34	8.46 ppm	3.13 ppm	6	10.47 ppm	0.25 ppm
187	35	8.67 ppm	5.65 ppm	9	9.04 ppm	4.05 ppm
173	20	8.71 ppm	1.50 ppm	3	9.33 ppm	0.27 ppm
183	23	9.86 ppm	2.10 ppm	6	10.74 ppm	0.71 ppm
165	24	10.20 ppm	1.56 ppm	6	10.82 ppm	0.73 ppm
170	21	11.41 ppm	3.51 ppm	3	13.31 ppm	0.54 ppm
184	25	13.73 ppm	4.19 ppm	6	14.88 ppm	1.52 ppm
179	25	14.21 ppm	1.90 ppm	6	15.35 ppm	0.47 ppm
177	17	14.36 ppm	1.60 ppm	1	15.07 ppm	0.00 ppm
139	32	14.41 ppm	4.83 ppm	8	17.71 ppm	0.63 ppm
147	30	14.59 ppm	4.19 ppm	7	17.89 ppm	1.02 ppm
171	21	14.62 ppm	1.74 ppm	4	14.83 ppm	0.54 ppm
148	27	15.51 ppm	5.71 ppm	7	20.03 ppm	0.70 ppm
168	39	15.77 ppm	2.77 ppm	11	17.05 ppm	0.63 ppm
136	19	16.35 ppm	2.61 ppm	3	16.84 ppm	0.79 ppm
137	23	16.61 ppm	3.17 ppm	4	18.99 ppm	0.32 ppm
182	22	16.61 ppm	2.09 ppm	4	17.59 ppm	0.64 ppm
166	30	16.87 ppm	1.93 ppm	7	17.37 ppm	0.61 ppm
196	26	17.28 ppm	1.79 ppm	3	18.09 ppm	0.06 ppm
134	29	17.59 ppm	1.92 ppm	6	18.20 ppm	0.45 ppm
155	33	17.73 ppm	3.20 ppm	9	18.52 ppm	1.45 ppm
160	35	17.73 ppm	2.34 ppm	9	18.69 ppm	0.84 ppm
174	13	18.51 ppm	2.29 ppm	1	19.40 ppm	0.00 ppm
149	34	18.57 ppm	2.64 ppm	8	19.88 ppm	0.31 ppm
164	32	18.75 ppm	1.79 ppm	8	18.82 ppm	0.79 ppm
172	24	19.05 ppm	2.62 ppm	6	20.14 ppm	0.42 ppm

host IP	12 hour periods			24 hour periods		
(last byte)	# periods	min skew est	Δ max	# periods	min skew est	Δ max
141	32	19.17 ppm	4.35 ppm	7	21.69 ppm	0.62 ppm
175	14	19.28 ppm	1.72 ppm	2	20.15 ppm	0.27 ppm
181	28	19.92 ppm	2.08 ppm	7	20.51 ppm	0.46 ppm
180	23	20.23 ppm	3.75 ppm	3	22.69 ppm	0.52 ppm
194	37	20.29 ppm	1.93 ppm	9	20.98 ppm	0.69 ppm
151	28	21.60 ppm	3.71 ppm	5	23.94 ppm	0.61 ppm
140	27	25.04 ppm	3.77 ppm	5	27.61 ppm	0.42 ppm
193	38	25.24 ppm	4.54 ppm	11	25.34 ppm	3.37 ppm
145	24	25.40 ppm	1.85 ppm	4	26.26 ppm	0.66 ppm
144	26	25.46 ppm	1.48 ppm	3	26.32 ppm	0.14 ppm
138	26	25.61 ppm	3.05 ppm	5	25.92 ppm	0.37 ppm
153	29	26.34 ppm	3.41 ppm	6	28.42 ppm	0.43 ppm
197	32	26.61 ppm	2.94 ppm	7	27.26 ppm	1.63 ppm
143	25	27.27 ppm	1.86 ppm	4	27.99 ppm	0.13 ppm
162	26	27.49 ppm	1.89 ppm	4	28.64 ppm	0.32 ppm
150	31	27.83 ppm	2.91 ppm	7	28.57 ppm	0.45 ppm
199	25	28.27 ppm	2.88 ppm	4	29.62 ppm	0.85 ppm
178	17	28.42 ppm	2.09 ppm	6	29.62 ppm	0.44 ppm
185	38	28.53 ppm	2.54 ppm	10	29.23 ppm	1.12 ppm
132	24	28.89 ppm	1.58 ppm	7	29.47 ppm	0.74 ppm
135	31	29.97 ppm	3.07 ppm	5	30.34 ppm	1.52 ppm
198	39	30.26 ppm	3.76 ppm	12	30.96 ppm	2.82 ppm
131	16	30.61 ppm	3.32 ppm	2	33.00 ppm	0.83 ppm
169	27	31.43 ppm	1.73 ppm	4	31.86 ppm	0.24 ppm
159	31	31.51 ppm	3.35 ppm	7	33.66 ppm	0.56 ppm
188	41	31.58 ppm	4.35 ppm	11	34.62 ppm	0.95 ppm
154	33	31.75 ppm	3.77 ppm	8	33.80 ppm	0.50 ppm
133	24	31.85 ppm	1.59 ppm	8	32.33 ppm	0.47 ppm
146	29	36.91 ppm	7.78 ppm	7	43.15 ppm	0.79 ppm
186	33	37.96 ppm	5.08 ppm	8	41.33 ppm	0.75 ppm
157	32	37.98 ppm	2.25 ppm	6	38.90 ppm	0.15 ppm
176	16	38.00 ppm	2.30 ppm	2	39.77 ppm	0.02 ppm
192	20	46.38 ppm	1.66 ppm	7	47.04 ppm	0.45 ppm
190	28	47.81 ppm	2.19 ppm	6	49.03 ppm	0.25 ppm