Opportunities and Limits of Remote Timing Attacks

Scott A Crosby and Rudolf H. Riedi and Dan S. Wallach
Rice University

May 26, 2007

Abstract

Many algorithms can take a variable amount of time to complete depending on the data being processed. These timing differences can sometimes disclose confidential information. Indeed, researchers have been able to reconstruct an RSA private key purely by querying an SSL web server and timing the results. Our work analyzes the limits of attacks based on accurately measuring network response times and jitter over a local network and across the Internet. We present the design of filters to significantly reduce the effects of jitter, allowing an attacker to measure events with 15-100μs accuracy across the Internet, and as good as 100ns over a local network. Notably, security-related algorithms on web servers and other network servers need to be carefully engineered to avoid timing channel leaks at the accuracy demonstrated in this paper.

1 Introduction

Security researchers have studied a number of remote timing attacks, principally against cryptographic algorithms. If an attacker can precisely time cryptographic operations, the attacker may be able to solve for the cryptographic key. There has been signifigant interest in these attacks. Brumley and Boneh [8] showed that such attacks were practical, i.e., an attacker could measure the response-time variances of a secure web server with carefully chosen input and, after collecting enough samples, could derive that server's RSA private key. Likewise, it has been shown that an IMAP password could be extracted from a TLS/SSL channel across a network consisting of two switches and a firewall by measuring a 2ms difference in response time [9].

Brumley and Boneh based their attack on roughly 1.4 million queries which they found to be effective on a local area network yet ineffective across the Internet to derive the server's private 1024-bit RSA key. Clearly, additional network hops will increase the latency for packets to travel from one host to another. To an attacker trying to mount a timing attack, latency differences are irrelevant because the attacker is only interested in measuring the *differences* in latency across measured events. However, additional network hops may also add *jitter* (i.e., random noise) to the measured latency. An

attacker's goal is to make multiple timing measurements and hopefully smooth out the jitter to recover the actual time difference.

Timing attacks have broad relevance beyond protecting cryptographic keys. Consider algorithmic complexity attacks [14], where an attacker tries to induce algorithmic worst-case behavior in a program by sending carefully-chosen inputs that might, for example, cause every insert into a hash table to collide, causing expected $O(1)$ operations to consume their worst-case $O(N)$ running-time. One proposed solution to such attacks is to hide important details about the parameters used by internal algorithms. For example, several systems have replaced a deterministic hash function with a keyed but non-cryptographic hash function (e.g., universal hashing [10] or Jenkin's hash [17]). If an attacker can measure a server's response time with enough accuracy to determine if a collision occurred, then the attacker might be able to derive the key.

Timing attacks against some algorithms will require more precision than others. This paper aims to quantify the precision that an attacker might hope to achieve in discriminating between two events, on a remote computer, that take slightly different amounts of time to run. This paper will present the results of extensive measurements both on our local network and across the Internet.

Section 2 presents related work. Section 3 describes our model of the attackers goals. Section 4 describes our experimental program and how we collected measurements. Section 5 summarize our results. Section 6 describes out network model. Section 7 provides a statistical analysis of network jitter from the viewpoint of an attacker trying to perform a timing attack and identifies factors that impact jitter including CPU and network card dependencies, and correlation of network distance to jitter. Section 8 consists of our simulation-driven study of how well an attacker might be able to perform a remote timing attack using statistical hypothesis testing. We present our conclusions in Section 9.

2 Related work

Side channel attacks The timing attacks discussed in this paper are an example of side channel attacks, where a system leaks information due to its physical implementation. An early example of such an attack is the password authentication weakness discovered in the Tenex operating system [24]. Kocher was the first to observe that side-channels attacks could generally be applied against common cryptographic algorithms. His analyses using device response times [19] and power consumption [20] to derive cryptographic secrets were the basis for much of the recent work in this field. In the recent Advanced Encryption Standard (AES) competition, the ciphers were examined for the potential of side-channel attacks [15].

Side channels exist where a computer may leak its internal state through RF emissions [23]. Strikingly, methods as effortless as watching the diffuse reflections of CRT display against nearby walls, at a distance, may allow an observer to see the screen [22]. Side channels have been used to detect passwords over SSH, through the use of keystroke timing [38].

Timing attacks have been applied to cryptosystems. Kelsey et. al. [18] conjectured that cache miss behavior may also be used as a side channel. Page [30] later presented

an analysis of caching behavior and described an attack against DES. Bernstein [5] showed that AES's CPU cache-miss behavior leaks key bits. A more powerful attack on AES exploiting shared cache state has been done by Osvik et. al. [29]. Silverman and Whyte [37] summarize a timing attack on the NTRU cryptosystem that exploits a timing difference on the number of SHA-1 computations. Schindler described an attack against the chinese remainder theorem in RSA and Schindler [36] modeled and optimized attacks against RSA. More recently, Acıiçmez et. al [2] show a local attack on RSA that exploit branch mispredictions delays in order to determine the secret key. Acıiçmez et. al. [3] present an interprocess timing attack over the loopback interface on AES that exploits CPU cache timing differences using about 10^{11} encryptions and were able to distinguish the correct AES key among 2^{12} alternatives.

Timing attacks have been applied to the Internet. Kohno et. al. [21] showed that it is possible to fingerprint a host on the Internet by using TCP or ICMP timestamps to measure differences between machine clock skews as tiny as 1μs per second. Felton and Schneider [16] shows how servers can fingerprint anonymous web clients through detecting the timing difference between a client cache hits and misses. And, Bortz et. al. [7] showed that timing difference may leak secrets such as the existance of an account or shopping cart size in web applications.

Network measurement There have been many attempts to characterize the end-to-end behavior of the Internet. The most comprehensive work is Paxson's measurements and analysis of Internet end-to-end dynamics [32, 33]. Paxson characterized such issues as routing pathologies (e.g., routing loops), outages, flutter, and the stability and lifetime of routing information. He also examined Internet packet dynamics, including the effects of packet loss, corruption, and reordering on TCP.

The earliest studies of network latency and jitter focused on these attributes because of their effect on important parameters in the TCP protocol. An accurate value for the round trip time is needed to estimate the correct values for TCP retransmit timers and window size [13]. If jitter causes the round trip time to be incorrectly measured, the TCP protocol may incorrectly initialize its timers.

Internet path delay time was characterized as a shifted gamma distribution by Mukherjee [28]. His measurements used standard ICMP echo requests, and achieved millisecond precision. Many other researchers have performed end-to-end assessments of Internet packet behavior [1, 6, 35, 40]. Barford and Crovella [4] characterized causes of delay and jitter in a web server scenario. Casner et. al. [11] measured internet jitter to 20μs resolution on a wide area backbone network to study the feasibility of using backbone IP networks as virtual circuits.

Generally, these studies were concerned with millisecond-scale events, and did not consider the notion of an attacker willing to make thousands or even millions of repeated queries in order to gain increased timing accuracy of a remote machine's processing time.

Clock synchronization Clock synchronization and remote timing attacks must both handle Internet jitter and delay. Unlike clock synchronization algorithms, an attacker only needs to worry about the stability of one clock — their own, and over a timescale

of minutes. The attacker can also afford to collect many measurements.

The Network Time Protocol [25, 26] is designed to synchronize the system clocks of computers over the network. NTP must, by necessity, measure and compensate for network latency and jitter, but its goals are to achieve millisecond, not sub-microsecond accuracy.

Many types of network measurement depend more upon low clock skew variation across the measurements hosts than offsets from real time [34]. Protocols other than NTP have been designed to minimize clock skew [31, 39].

3 Attack model

We consider a simplified situation where the attacker can transmit two different requests to the target server that either take the same or different time to complete as a function of the server's secret. We assume that knowing whether or not they take the same time will divulge something about the secret. (We assume the attacker knows everything else about the target machine, including its hardware and software configuration. The attacker only lacks knowledge of the internal secret.) The attack then reduces to inferring such a difference in computing time with high reliability. Our model is powerful enough to represent the Brumely and Boneh OpenSSL attack [8].

To be more precise, a query is transmitted to the target machine, upon which the target performs some task which requires a *processing time* which the attacker would like to infer. However, the attacker can only measure the *response time*, i.e., the time from when the query is transmitted to when the reply is received.

Our goal is to *identify the smallest difference between two processing times* that can be reliably detected by an attacker given a reasonable number of measurements. Of course, the resolution with which an attacker can observe these differences will be a function of how many samples the attacker takes, how much random perturbation, called *jitter*, is introduced in the response time by the network and how effective the attacker can be at filtering that jitter.

4 Experimental setup

We ran tens of millions of timing experiments, both in our lab and over the Internet. Our system implemented a simple UDP ping-pong protocol where, for each measurement, a client sends a message to the server containing the specific amount of time the server should pause before replying. The server waits for the requested amount of time and then responds.

Upon receiving the response, the client logs the processing time requested of the server and the observed response time, then waits a random delay before sending the next request. This delay, averaging 20ms, avoids synchronization artifacts between the client and server. Furthermore, each client performed its trials in a random order. If no response is received within one second, the client assumes that a packet was dropped and repeats the measurement.

Dataset	#hosts starting	#hosts surviving	total #trials	# samples	Start Date	End Date
L	8	8	9.0M	27k	Sat 28 Feb 2004	Sun 29 Feb 2004
A	75	51	112.5M	40k	Tue 9 Mar 2004	Wed 17 Mar 2004
B	103	37	85.6M	30k	Fri 28 Jan 2005	Thu 3 Feb 2005
C	136	91	68.4M	13.5k	Wed 2 Mar 2005	Tue 8 Mar 2005
D	124	3	179.6M	68k	Wed 9 Mar 2005	Wed 23 Mar 2005

Table 1: Dataset statistics.

In all of our datasets the server machine was dedicated to the task, while we ran clients as background tasks on other machines. Clients would only make one request at a time, allowing them to run without disturbing the machines' users. In effect, we have busy and idle clients querying an idle server instead of mostly-idle clients querying a busy server, as we would expect when a server is under attack. We show measurements of loaded servers and discuss this issue further in Section 8.9.

4.1 Clock calibration

We accomplished nanosecond-precise timing by reading the CPU's cycle counter, available on all modern microprocessors. However, cycle counters count in cycles, which we must convert to nanoseconds, requiring our experimental harness to estimate the clock frequency of each machine. Rather than trying to tightly synchronize the clocks of these machines, perhaps with NTP [25, 26], we perform a one-second calibration of the cycle counter against the system clock to give an approximate solution, within 1% accuracy. Our network model includes a correction for clock skew. Each machine measures time independently, in nanoseconds, and we reconcile the differences in post-processing. We determine the clock skew by the slope of a least-squares linear fit of the delays requested with the delays measured. This process is described further in Section 7.4.

4.2 Collected datasets

We collected five primary datasets. One dataset was collected over a LAN and the other four datasets were collected over the Internet. In each dataset, we measured the same $M = 46$ distinct processing times on the server ranging from 100ns to 8ms. Table 1 summarizes these measurements, including the number of hosts that were involved at the start of the experiment, the number of hosts that were left when we finished the experiment, the total number of measurements in the dataset, and the maximum number of samples per host per processing time we collected.

Dataset L consists of 8 clients on our LAN. For each processing time, we collected 27,000 samples. Datasets A, B, C, and D were collected over the Internet, ranging from 13,000 to 68,000 measures per host per processing time.

To achieve a broad sampling of Internet hosts, we used PlanetLab, an open, globally distributed platform for developing, deploying and accessing planetary-scale network services [12]. Unfortunately, PlanetLab hosts are not very reliable and many hosts

failed during the experiment. We are unable to restart a failed host because the clock calibration would not match. These host failures were particularly apparent in Datasets B-D.

We would have preferred to use Dataset A because PlanetLab hosts were much more reliable and less overloaded when it was collected. Unfortunately, a concurrency bug in our earlier data collection system corrupted about 2100 measurements in Dataset A and about 30 in Dataset L. We fixed this problem for subsequent measurements, and we believe that our post-processing filters out these errors. Nonetheless, we will present results for each dataset.

5 Results

We are interested in the *resolution* that events on a remote node can be timed. *Unfiltered jitter* is a measure of resolution based on statistical techniques and is suggestive of the timing difference that is measurable. We also measure resolution empirically by simulating an attack and identifying the minimum distinguishable timing difference. Most of our results, previewed here, concern how unfiltered jitter varies with CPU architecture, network distance, and other causes.

Does network latency follow a Gaussian distribution? No. The distribution of response time is a highly skewed distribution. (See Section 7.1.)

How is unfiltered jitter measured? Unfiltered jitter estimates the amount of residual noise which limits the resolution of a remote timing attack. Measurements are *filtered* to a single value that is supposed to be correlated with the remote processing time. As we know the actual processing time, we can check the strength of the correlation. Good filters should have high correlation. *Unfiltered jitter* measures the lack of correlation. (See equation 10 in Section 7.4.)

Isn't mean (or median) the best way to filter measurements? If the response times were distributed in a Gaussian fashion, then mean or median would be excellent filters. With the non-Gaussian distributions we see in practice, low-percentile filters tend to significantly outperform the mean or median. (See Sections 7.1 and 7.5 for details.)

Then surely the minimum response time should an excellent choice. Contrary to expectations, the minimum response time is *not* the least noisy signal. Low percentile filters exhibit significantly less noise than the minimum response times. (See Section 7.5.)

Is there a correlation between unfiltered jitter and network latency or hopcount? We observed no significant correlation between unfiltered jitter and either latency or the number of network hops. (See Section 7.6.)

Does the CPU of a machine affect unfiltered jitter? Yes, we found that our Pentium 4 measurement host introduced artifacts. For processing times between 100ns and 80μs, 40% of the measurements had a constant 180μs response time. (See Section 7.7.)

Does the network card affect unfiltered jitter? Yes, we found that our Intel Gigabit Ethernet card had 10 to 30 times more unfiltered jitter than our generic onboard Ethernet adapter, depending on whether interrupt coalescing was enabled. (See Section 7.8.)

At what empirical resolution can an attacker time a remote host? The resolution an attacker can time a remote host depends on how many measurements they can collect. Our simulated attacker using statistical hypothesis testing was able to reliably distinguish a processing time differences as low as 200ns and 30μs with 1000 measurements on the LAN and WAN respectively with. (See Section 8.)

How much extra noise is introduced by application load? With 1000 samples, the addition of application load from Apache introduces only 1μs of jitter, much less than the 20μs WAN jitter. (See Section 8.8.)

6 Network Model

It would be impossible to isolate and measure every possible contribution to network latency and jitter. Between an attacker's machine and the target machine, there may be any number of network bridges, hubs, switches, firewalls, and routers. Each of these may delay packets, drop packets, or suffer internal contention. Furthermore, if packets are arriving faster than they can be forwarded, a router will attempt to queue the packets and send them out later. As the load varies, so will the latency and jitter accumulated by packets as they pass through the network device.

In addition, the end-hosts will introduce their own jitter as a result of application load, virtual memory pressure, and network packet processing. Whereas the attacker may be able to dedicate a computer to the sole purpose of time measurement, and thus reduce the attacker's contribution to jitter, the target machine is likely to be running a general-purpose operating system and supporting a non-trivial workload.

We use an abstract model of the server and the network. The server is assumed to run at least two different *tasks* which have different processing times and all requests for the same task have the same processing time. We model the latency of a round-trip communication channel between one pair of hosts as

$$responseTime = a \cdot processingTime + propagationTime + jitter \tag{1}$$

with the following five assumptions:

1. *responseTime* is the measured round trip time on the network.
2. a accommodates clock skew and is constant over all requests for all tasks. This is estimated in our analysis independently for each host.
3. *processingTime* is *constant* for all requests for the *same* task. In our dataset, this corresponds to the time the remote hosts delays before sending a reply to a ping-pong.
4. *propagationTime* is constant over all requests, for *all* tasks. This is the 'average latency' and is estimated in our analysis independently for each host.

5. *jitter* is the jitter term and is identically and independently *distributed* for all requests and tasks.

Thus, the jitter term absorbs all randomness introduced by network conditions, load on the target machine and any other source.

More formally, the M different *known* processing times are denoted by $t[1],\ldots,t[M]$. The true, but unknown statistical distribution of response times to a query m with a processing time of $t[m]$ is denoted by $R[m]$. Thus, for each host pair and for each query m, $r_1[m]\ldots r_N[m]$ denote the N collected measurements which may also be thought of as samples taken from $R[m]$. Our channel model (1) can be restated more formally as:

$$\forall_{m\in\{1\ldots M\}} \quad r_n[m] = a\cdot t[m] + b + \varepsilon_n[m] \tag{2}$$

where b is the propagation time, a is the correction for clock skew and $\varepsilon_n[m]$ is the jitter in the measurement $r_n[m]$.

By our channel model, the random variables $\varepsilon_n[m]$ are all of the same distribution and are statistically independent (for all n and m). In other words, the distributions of $R[m]$ for different m are identical up to a shift term and the $r_n[m]$ constitute independent samples of these distributions.

Our model setup implicitly assumes that the application load is stationary and that the network will introduce the same random perturbations for all requests. This is reasonable if all response messages have the same payload and if routing is stable over the time of the measurements. Route changes would be easily detected by the displacement they would cause in the response time distribution or by traceroutes operating in parallel to an ongoing timing attack. An attacker may be limited to the number of measurements they can collect due to route changes. Combining measurements across route changes would require detecting when those changes occur and determining the propagation time difference between each set of samples. (Our own work assumes that the network routes are constant across the duration of our experiments.)

We repeat our analysis for each host pair in each dataset, treating each of them as a separate channel with unique jitter, propagation time, clock skew, and response time distribution.

The definition of jitter The splitting of the response latency into the sum of propagation and jitter is for convenience and does not affect methodology or performance of the attack. Indeed, since only *differences* in response times need to be inferred, and not actual estimates of the response times themselves, introducing the constant shift term $b = propagationTime$ is irrelevant. In particular, we do not require an accurate estimate of the propagation time as long as we use the same constant value throughout the attack. Although strictly speaking, jitter is always an additional positive latency, we consider *propagationTime* as an estimate of the propagation time plus the *average jitter*, allowing *jitter* to represent positive *and negative* deviations from that average.

The use of clock skew An attacker need not be concerned with clock skew on a remote host. They only need to reduce their own clock skew enough to avoid gross errors in their measurements. Ignoring this skew creates a relative error of $(a-1)\approx 1\%$ for an attacker, such as the one we simulate in Section 8, which is of no importance.

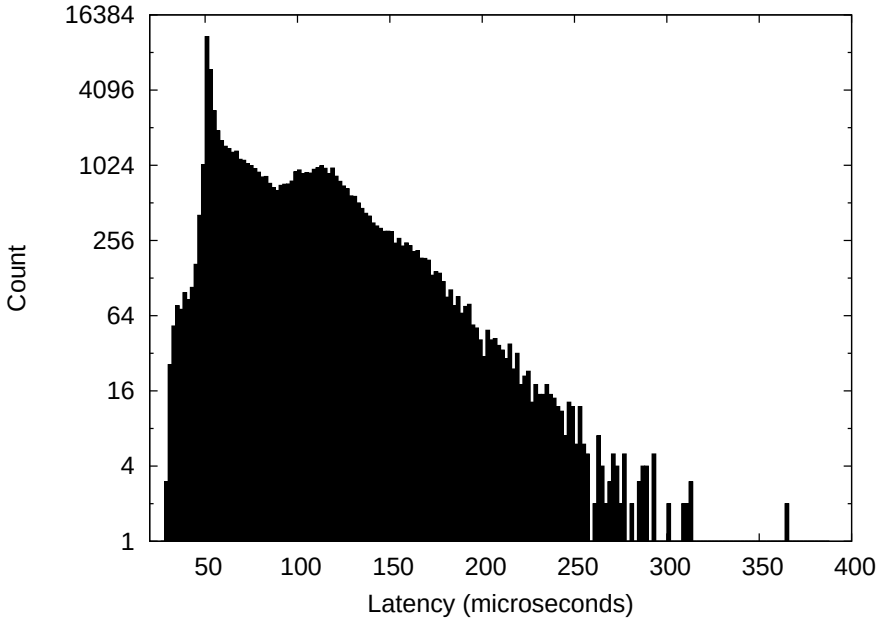

Figure 1: Histogram of response times for 66k measurements of a 1ms processing time from a host from Dataset D.

In contrast, an accurate assessment of clock skew between our client hosts and our server host is critical in our meta-analysis of Internet jitter. When comparing the jitter distribution for processing times between $t[1] = 100ns$ and $t[46] = 8ms$, a 1% inaccuracy in clock skew would cause us to misestimate the shift between $R[1]$ and $R[46]$ by $80\mu s$ which would completely dominate our results.

7 Statistics of the response time distribution

We first examine the response time distribution, examine techniques to filter it and we estimate the unfilterable jitter.

7.1 Response time distribution

The distribution of jitter is not Gaussian. It is highly skewed distribution with two modes. In Figures 1 we plot a histogram of the probability density function (PDF) of the response times for one Dataset D host. The response time is clearly asymmetric and non-Gaussian and includes two obvious modes and an exponentially distributed tail. In Figure 2 we plot the CDF corresponding to this host. The steep slope at about 52ms is the least varying part of the distribution and occurs in the 5th-15th percentile response time.

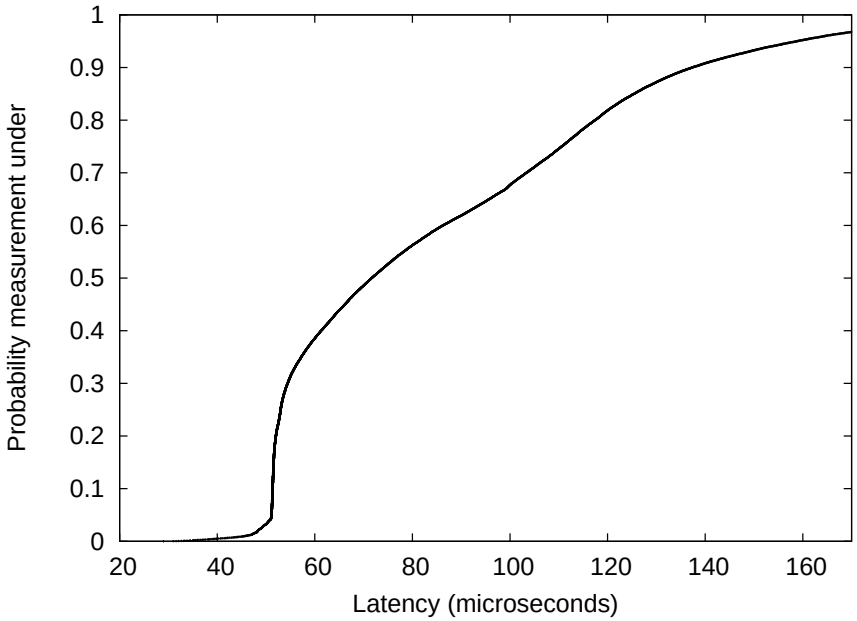

Figure 2: CDF of response times for 66k measurements of a 1ms processing time from a host from Dataset D.

7.2 Measurement quantiles

We define the i-th percentile (or quantile) q_i of a distribution R to be the smallest real number $q_i[m]$ such that $P[R \leq q_i] = i/100$. The 50-th percentile is the familiar *median* and the 0-th percentile is the minimum response time.

The true quantile q_i is unknown because it depends on the true distribution R. We can compute an *estimator* $\hat{q}_i$, which is an empirical estimate of q_i derived from our measurements by first ranking a set of measurements into

$$r_{(1)} \leq r_{(2)} \leq ... \leq r_{(N)} \tag{3}$$

and setting $\hat{q}_i = r_{(\lfloor iN/100 \rfloor)}$. The empirical quantiles $\hat{q}_i$ are well known to be weakly consistent under mild conditions, meaning that they converge to the true quantile q_i as the sample size increases.

We use quantiles as a graphically powerful alternative to displaying a histogram for each processing time. We summarizing those histograms with *percentile contours*. The empirical percentiles $\hat{q}_i[m]$ are plotted as a function of the sampled server processing times, $t[1], ..., t[M]$, which are indicated with dots on the x-axis. These percentile contour summaries of the response time distribution are easier to compare than their full histograms.

Figures 3 and 4 summarizes the estimated percentiles of the response time distributions we observed for a host in Dataset L and D respectively. Figure 3 shows that that

despite intuition to the contrary, the minimum response time seems to be poorly correlated to the processing time. We can present all $M = 46$ response times on one plot on a log-scale if we first estimate and subtract off the linear fit. (This process is further described in Section 7.4.) Figures 5 and 6 show the response time distribution for a host in Dataset L and D respectively. The variation of the median and minimum response times across different processing times is visually more than the variation seen in measurements at small quantiles. The mean (not plotted) is similarly noisy. Because the median and mean of our measurements are extremely noisy, parametric inference techniques based on these very classical statistics are unlikely to work well.

Variability of the Empirical Percentile To study the error of this percentile estimation we need to consider the distribution of the estimator itself, i.e., the variability in $\hat{q}_i$ when the estimation is repeated. Fixing a percentile i and setting $k = k(i) = \lfloor iN/100 \rfloor$ we may write

$$\hat{q}_i[m] \quad = \quad r_{(k)} = a \cdot t[m] + b + \varepsilon_{(k)}[m]. \tag{4}$$

Here, we introduced the random variable $\varepsilon_{(k)}[m]$ which is the k-th ranking sample out of N independent and identically distributed samples from $\varepsilon[m]$. Since under our channel model all $\varepsilon[m]$ are identically distributed for different m, $\varepsilon_{(k)}[m]$ are identical in distribution as well and we conclude that the distribution of $\hat{q}_i[m]$ and $\hat{q}_i[m']$ are identical up to the shift term $a(t[m] - t[m'])$:

$$\hat{q}_i[m] \quad \stackrel{\text{distr.}}{=} \quad \hat{q}_i[m'] + a(t[m] - t[m']). \tag{5}$$

Precision of Percentile Estimation It follows immediately from (4) that the estimation errors $\hat{q}_i[m] - q_i[m]$ themselves have identical distributions, independently of m:

$$\hat{q}_i[m] - q_i[m] = (a \cdot t[m] + b + \varepsilon_{(k)}[m]) - (a \cdot t[m] + c) \tag{6}$$

$$= \varepsilon_{(k)}[m] \; + b - c \quad \stackrel{\text{distr.}}{=} \quad \varepsilon_{(k)}[m'] \; + b - c. \tag{7}$$

Two remarks are in order.

First, to study the variability of a percentile estimator $\hat{q}_i$ and the estimation error (6), we need a set of empirically obtained values. Typically, this is done by repeating the same estimation procedure several times. Here, we may exploit that the errors $\hat{q}_i[m] - q_i[m]$ obtained for the *same* percentile but for different values of the response time, i.e., for $m = 1 \ldots M$ are all of the same distribution (see (7)). In other words, we may consider these error values as samples of the same distribution $\varepsilon_{(k)} + b - c + a \cdot (t[m])$. In this context we recall that b is the (unknown) propagation time, a is the clock skew, and that c depends only on i and some arbitrary baseline response time $t[m^*]$, but not m.

Second, since we are interested in *differences* of processing times, and thus in (additive) differences of the response times we observed, the additive constant $(b - c)$ in (7) will cancel out in our inference schemes and be of no importance.

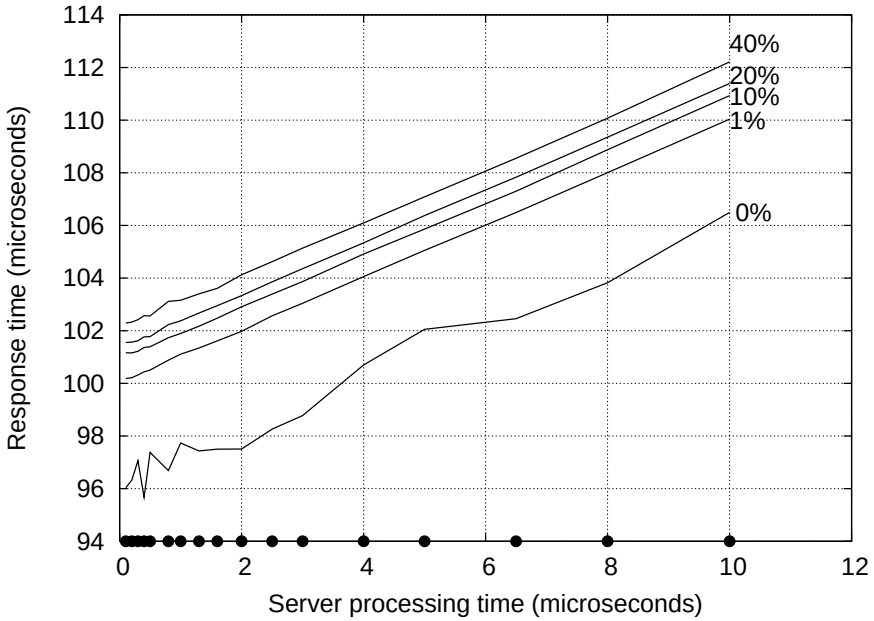

Figure 3: Response time percentiles as a function of processing time for a local host in Dataset L.

7.2.1 Filtering our data

Because of the noise in the jitter distribution, particularly at and above the median, we apply a "filter" Γ which reduces the set of measurements $r_1[m] \ldots r_N[m]$ to a single number that is hopefully closely correlated with the processing time $t[m]$ A filter, foremost, is a function of the measured response times. Filter design choices are driven mainly by the objective to minimize the variance as a measure of error which in turn impacts the reliability of a decision procedure based on the filter. In general, we may think of this procedure as some sort of de-noising.

While such a filter can not remove all noise, it will in help us understand the unfilterable jitter in our measurements and estimate the resolution an attacker may be able to measure. There are many filters we may choose. In Section 7.4 we describe how we evaluate filters and identify the 'best' such filter.

A first simple example of a filter would be the *i*-th empirical percentile:

$$\Gamma_{q_i}(r_1[m], \ldots r_N[m]) = \hat{q}_i[m] \tag{8}$$

Recall that the filters $\Gamma_{q_{50}}, \Gamma_{q_0}, \Gamma_{q_{100}}$ pick the *median*, *minimum* and *maximum* response times, respectively.

We tried three types of filtering strategies beyond simple percentile filtering but they performed no better than the best percentile filter. The best filter of each type was identified through brute force: We ran several hundred and recorded the best instance of each type. The four types of filtering we applied are:

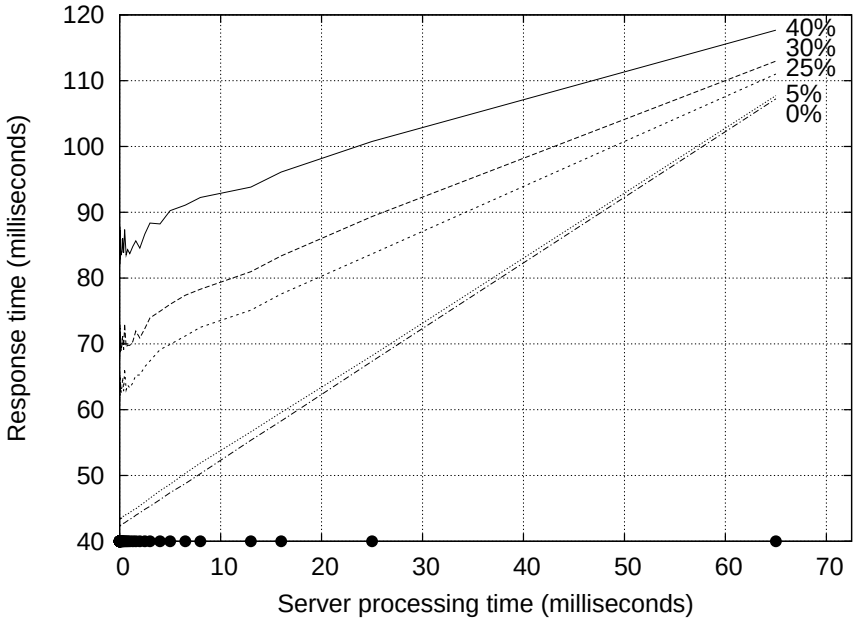

Figure 4: Response time percentiles as a function of processing time for a remote host in Dataset D.

1. **Percentile** This filter returns the xth quantile or percentile. We tried a hundred different percentile values, varying from 0% to 70%, with most of the percentile values less than 10%.

2. **Peak** This filter identifies modes in our measurements. Samples are first sorted, and then a window of a fixed width is moved across the sorted list. When the difference between the maximum and minimum sample within the window is minimized, that position of the window contains the highest density of samples. The median measurement within the window is reported as the result of the filter. We considered 25 different window widths, ranging from 1% to 70% of the collected samples. Even in the best case, our percentile filter outperformed the peak filter.

3. **Average Range** Inspired by the filter used in Brumley and Boneh's SSL attack, we computed a histogram over the samples and computed the average of the samples between the i-th and j-th percentiles. We tested this filter over 96 different percentile ranges. We used percentiles ranging from 0% to 40%; we mostly examined ranges below the 5th percentile. In general, the best instances of this filter performed within a few percent of the best percentile filter.

4. **Percentile Smoothing** This is an attempt to improve our estimate $\hat{q}_i$ of the real percentile q_i. Instead of computing the estimate from our measurements directly, we divide the measurements into k disjoint subsets, compute a separate estimate

for each subset, and then average those estimates. We tested this filter for $k \in \{4, 10, 20\}$ and for 21 different percentiles. This filter performed slightly worse than our best percentile filter.

7.3 Filtered measurements

Figures 3 and 4 show several percentile filters. For Dataset L, virtually all of the percentiles we plot show a smooth relationship to processing time, except for $\Gamma = q_0$, the minimum response time. For Dataset D, it is visually clear that the 25th percentile is much noisier than the 5th percentile. We note, for the particular client of Figure 3, and for plotted processing times ranging from 100ns and 8ms, 39 percent of the $27,000$ samples for each processing time occurred within a 2μs window, while the minimum response time was about 4μs faster. The minimum response time is clearly much noisier than the other percentiles.

In Figure 3 we also observe that indeed the first and tenth empirical percentiles follow a linear dependence on processing time with high accuracy. This is suggestive that the first and tenth percentile are better filters than higher percentiles.

In Figure 4, we show a similar plot for a typical host from Dataset D. First, note that the scale for this plot is in milliseconds and note that the 25th percentile looks noisier than the 40th percentile of Dataset L. This indicates more variability in the jitter in PlanetLab measurements than in the LAN measurements. For fine-grained measurements of response time, up to 75% or more of Internet measurements may be unacceptably noisy. This plot also appears to show non-parallel percentile contours. This is only an artifact caused by the non-uniform spacing of processing times where the 25ms and 65ms measurements are connected with a straight line.

7.4 Verifying channel model

We summarize the jitter distribution as a single filtered value and verify our channel model by how close the filtered measurements match the expected linear relationship. Let $\Gamma(r_1[m] \dots r_N[m])$ be a filter on the measurements $r_1[m] \dots r_N[m]$ for all $m \in 1 \dots M$, reducing them to a single value for each of the processing times $t[1] \dots t[M]$. We verify our channel model by using a least squares linear fit between the M different processing times and the M different filtered response times. Under the network model, the following relation should hold for $m = 1 \dots M$:

$$\Gamma[m] = \Gamma(r_1[m] \dots r_N[m]) = a \cdot t[m] + b \tag{9}$$

The least square fit will provide estimations of the clock skew a, the propagation time b and the variance σ^2 of the estimator, giving us a *measure of confidence* that the data follows a linear relationship. To this end, we compute the unique values $\hat{a}$ and $\hat{b}$ which minimize the average square deviation s_i^2 of $\Gamma(r_1[m] \dots r_N[m])$ from the linear relation. In other words:

$$s^2 = s_i^2 = \frac{1}{m} \sum_m (\delta[m])^2 \qquad \text{where} \qquad \delta[m] = \Gamma(r_1[m] \dots r_N[m]) - \hat{a} \cdot t[m] - \hat{b}. \tag{10}$$

Dataset	A	B	C	D	L	L′
Mean unfiltered jitter (μs)	11.7	14.1	19.0	20.9	2.1	.218
Median unfiltered jitter (μs)	6.5	7.2	6.7	7.4	.050	.046

Table 2: Mean and median, among all hosts within each dataset, of the unfiltered jitter.

Explicit formulas for $\hat{a}$ and $\hat{b}$ are quickly derived using standard calculus.[1]

For our best filters, s^2 is very small compared to the processing time differences which leads us to accept the linear relation between processing times and percentiles up to a small random error and *validates* our channel model. Furthermore, $\sqrt{s^2}$ which we call *unfiltered jitter* measures the effectiveness of a filter. The filter with the smallest unfiltered jitter is the most effective filter — the one with the best linear fit to the channel model.

Unfiltered jitter is also suggestive of the the resolution that an attacker can distinguish. Using the rule of thumb that a measurement has less than a 5% probability of being more than 1.7 standard deviations from its mean, a timing difference must be at least $3.4 \cdot s$ to be distinguishable with a 5% false positive and 5% false negative rate. In Section 8 we empirically measure an attacker's resolving power based on our network trace data.

In addition to this validation of our channel model, the least square fit (10) provides *simultaneous* estimates of the clock skew via $\hat{a}$, and propagation time via $\hat{b}$. An estimated value $\hat{a} \approx 1$ provides further confirmation of the channel model (1).

Some care has to taken in the interpretation of $\hat{b}$. The least squares fit minimizes the sum of the squares of $\delta[m]$, some of which will be positive and some negative. Therefore, we define *define b* to be the *average propagation*, rather than the minimal propagation; jitter may be negative and then the average of $\varepsilon_{(k)}$ is zero by definition and $\hat{b} \approx b$.

We summarize the unfiltered jitter across each dataset in Table 2 by giving the average and median unfiltered jitter. Across all of the datasets, the mean unfiltered jitter is noticeably larger than the median unfiltered jitter, indicating that although many hosts have low unfiltered jitter, some are much worse. For example, we created Dataset L' based on Dataset L, with one outlier removed. The outlier is a host on the other side of a "traffic shaper" which introduced a significant amount of jitter.

In summary, we accept a small value of s^2 as a validation of the channel model and accept the estimated clock skew $\hat{a}$ to be the true skew a. In statistical terms, this procedure is called a model fitting. We do not overfit as we estimate at most two parameters from $M = 46$ different response times.

We also determine if the estimator error remains the same across all processing times. In Figure 5, we plot the difference from the best-matching percentile filter and the actual measurement percentiles for various processing times on a logarithmic scale. We refer to these plots as *deviation contours* as they emphasize the residual noise in

[1] Setting $\overline{t} = (1/M)\sum_m t[m]$, $\overline{q} = (1/M)\sum_m \Gamma[m]$, $\overline{tq} = (1/M)\sum_m t[m] \cdot \Gamma[m]$ and $\overline{t^2} = (1/M)\sum_m t^2[m]$ we have

$$\hat{a} = \frac{\overline{tq} - \overline{t} \cdot \overline{q}}{\overline{t^2} - (\overline{t})^2} \qquad \text{and} \qquad \hat{b} = \overline{q} - \hat{a}\overline{t}$$

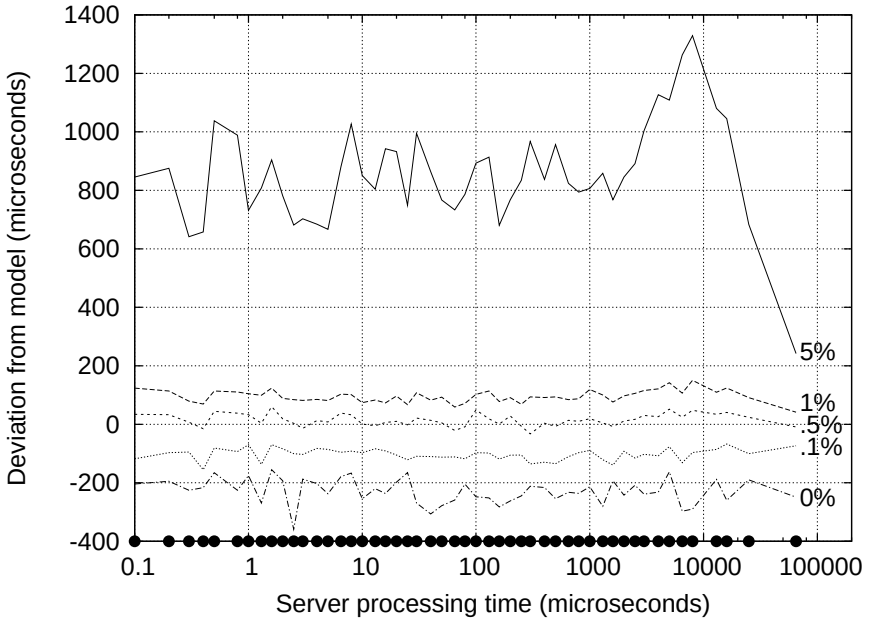

Figure 5: Difference from ideal estimator as a function of processing time for a host in Dataset D. Dots on the X-axis denote measured processing times.

that could not be filtered. Observe the lack of a trend for small percentile values, where the absolute estimator error remains about the same across all processing times. In contrast, Figure 6 presents larger quantiles from the same host showing several times the random variation but *less* variation in the measurements for the longer processing times. The jitter distribution is *not* the same across all processing times for this host.

Some hosts follow the channel model and have the same jitter distribution across all processing times, as we would expect. Other hosts violate the channel model, including those using an Intel Gigabit Ethernet NIC or an Intel Pentium 4 CPU (See Sections 7.7 and7.8). For those, our analysis still finds the best filter, although the resolution of the filtering process may be limited.

7.5 Effectiveness of filtering among quantiles

Intuition says that minimum response time should be the ideal filter because network devices can only introduce additional variable latency. Therefore, the minimum response time should have the least noise introduced. In our experiments to identify the filter with the lowest unfiltered jitter we found filters significantly more effective than using the minimum response time, contradicting this intuition.

To better understand this, we examined the relationship between percentiles and the resultant unfiltered jitter. In Figure 7 we plot the unfiltered jitter for 150 different percentile filters ranging from .001th percentile to 70th percentile, averaged across all

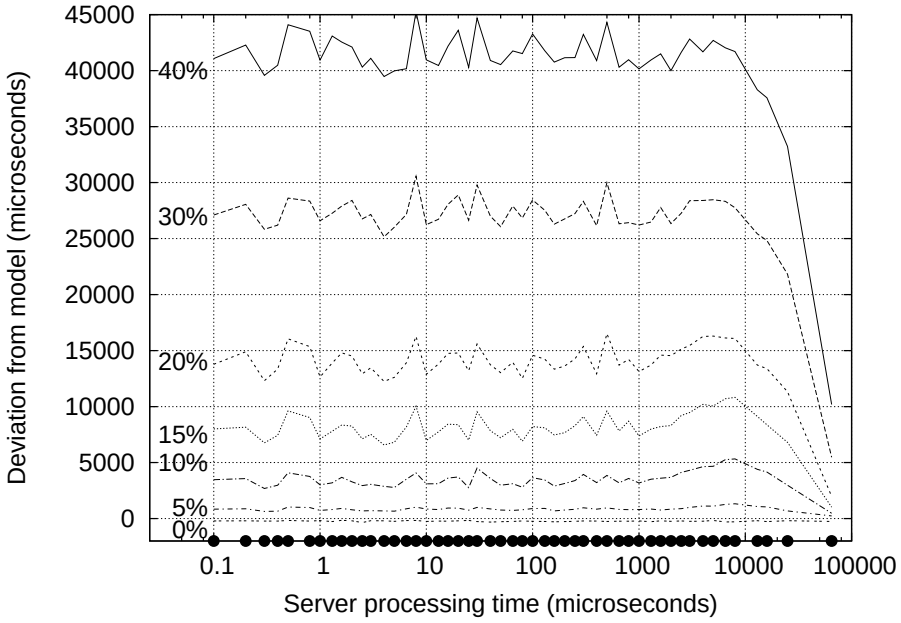

Figure 6: Difference from ideal estimator as a function of processing time for the same Dataset D host as Figure 5 but using larger-quantile filters. Dots on the X-axis denote measured processing times.

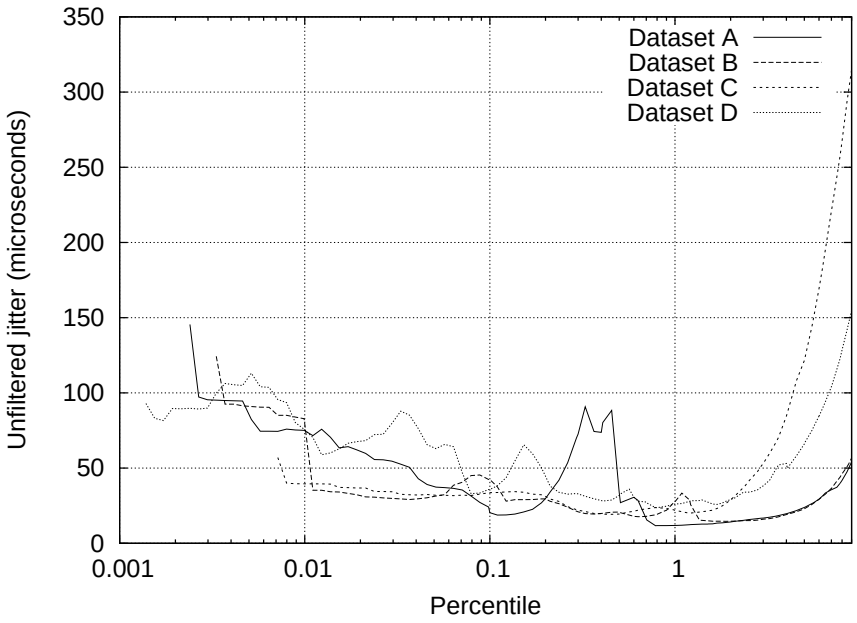

Figure 7: Unfiltered jitter as a function of filter percentile over PlanetLab.

hosts in their respective datasets. As expected, percentiles over 10% were very noisy. *Every* dataset shows a trend indicating a higher noise as the percentile declines from the 1st percentile toward the 0th percentile or minimum response time. These curves show several local minima and demonstrate that using the minimum response time leads to several times the error of using the empirically best filter.

To examine this further, we performed separate experiments using a an laptop and desktop computer connected with either a crossover cable or a network switch. All hosts and the switch were idle during the experiment. Figure 8 plots the unfiltered jitter as the percentile changes for Dataset L as well as the dataset using the switch and crossover cable. We note all three curves have at least two local minima that are not at the 0th percentile. For a switch and a crossover cable, the lowest unfiltered jitter is the 0.9th percentile and 12th percentile, respectively. These results confirm that minimum response time is not an accurate filter and that the ideal percentile filter can be difficult to predict a priori.

7.6 Unfiltered jitter versus network distance

In Figures 9 and 10, we show scatter plots of network distance, as measured in the propagation delay of the best linear fit, versus the lowest unfiltered jitter for that host. Each dot on these plots corresponds to the results of doing a least-squares linear fit on the filtered measurements for one host.

These plots show how well network round trip time correlates with measurement

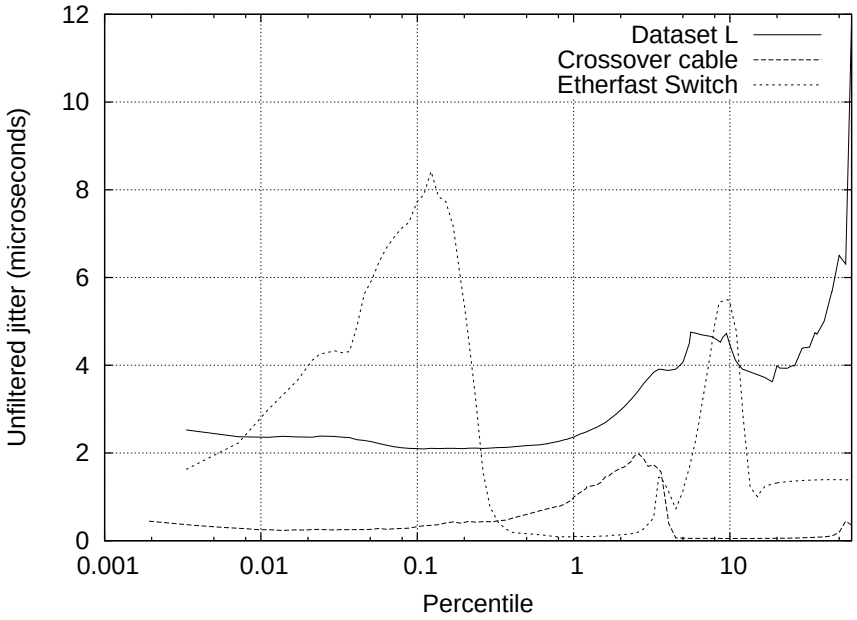

Figure 8: Unfiltered jitter as a function of filter percentile over the LAN.

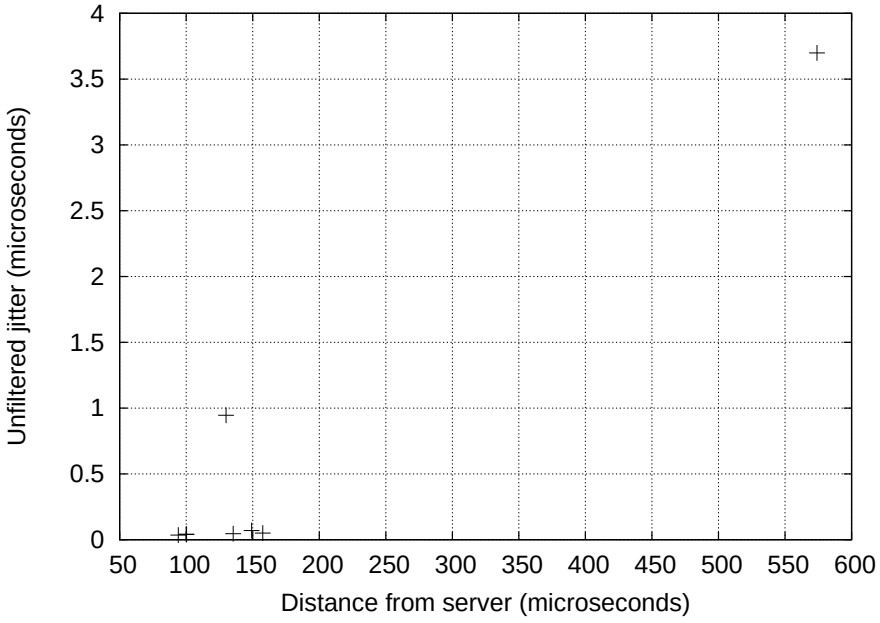

Figure 9: Scatter plot of unfiltered jitter versus network distance for each host in Dataset L.

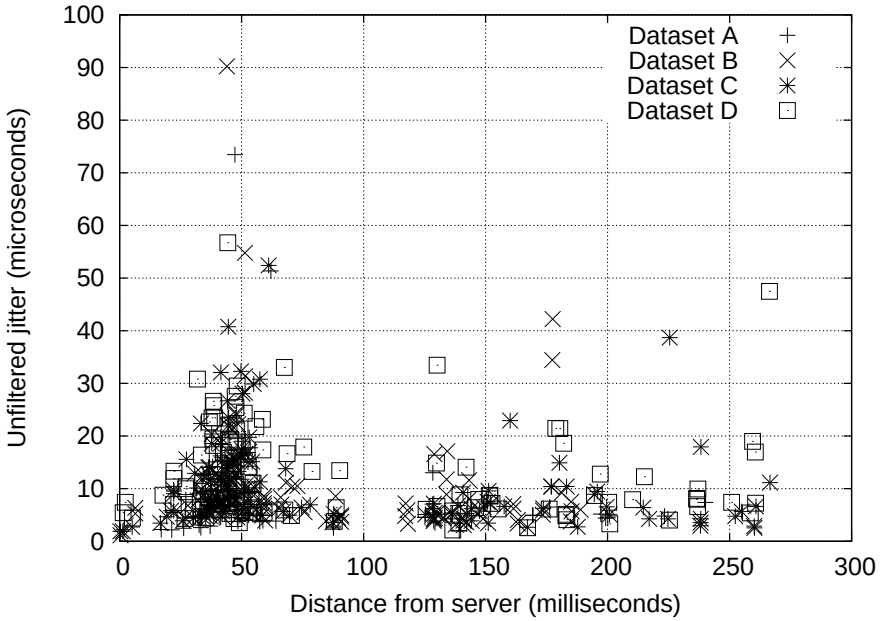

Figure 10: Scatter plot of unfiltered jitter versus network distance for each host in Datasets A,B,C and D.

accuracy. For Dataset L, our LAN measurements, Figure 9 shows most hosts clustered in the bottom left. These hosts were only connected by switches with no router hops and have low jitter. The outlier in the upper right is still on campus, but is behind a traffic shaping box. Figure 10 shows our measurements of the PlanetLab hosts. The *x*-axis scale is now much wider, reflecting the greater distance of these hosts. The largest cluster of hosts in the bottom left reflects the large number of PlanetLab hosts within the United States. Interestingly, the international hosts, with significantly longer network latencies, do not have noticeably higher unfiltered jitter. The host with the least unfiltered jitter in Dataset A, 2μs, was physically located on another continent. As such, physical distance does not necessarily imply much about unfiltered jitter. The hosts with higher unfiltered jitter, notably the outliers in the upper left, may reflect the lack of data we were able to collect from some PlanetLab hosts before they failed (see Section 4.2).

For Dataset D, we additionally measured the number of hops from the local system to each PlanetLab host used in the experiment. This allows us to determine whether the number of hops, rather than the latency, influences the unfiltered jitter. Figure 11 plots the unfiltered jitter versus the network hop count for hosts that had at least 10k measurements and a successful traceroute. One PlanetLab host located 2 hops away on the LAN had twice the unfiltered jitter of the best host, located 24 hops away. This plot also shows no correlation between hopcount and unfiltered jitter.

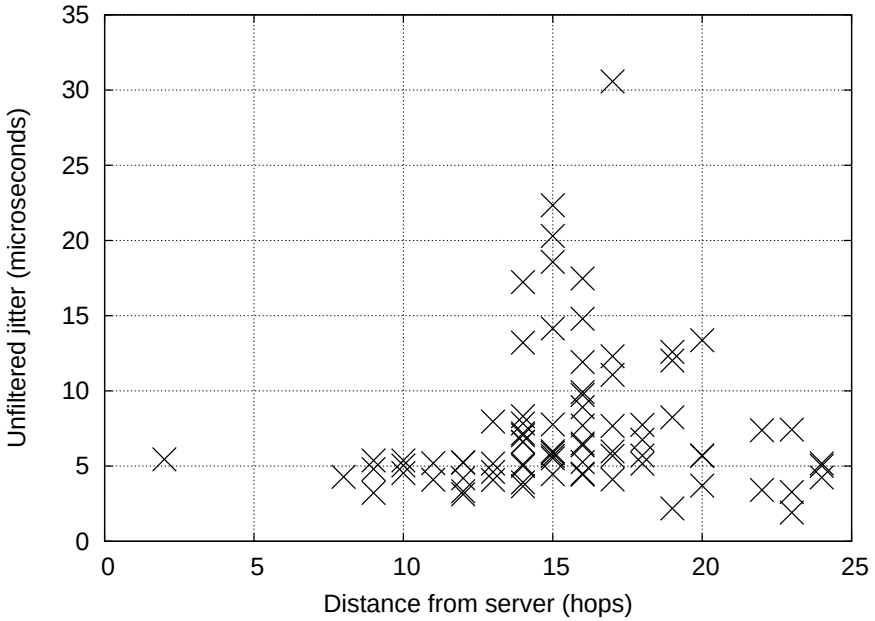

Figure 11: Scatter plot of unfiltered jitter versus network hop count for 80 hosts in Dataset D that were replied to at least 10k measurements.

7.7 CPU dependencies

During the course of our experiments we observed unusual behavior when the delay being measured was less than 100μs. In that configuration, we happened to use a Intel Pentium 4 laptop as the measuring host. Figure 12 demonstrates this problem. 35% of the samples with a processing time between 100ns and 80μs take a constant time to return, regardless of the actual processing time. We initially suspected this was an artifact of a router or switch. In fact, the culprit was the CPU itself.

To confirm that, indeed, the Intel Pentium 4 CPU was the source of the problem, we performed 12 timing measurements between four different computers (two models of Intel Pentium 3, a Intel Pentium 4 desktop CPU and an AMD Athlon 1GHz). In these measurements, the artifact occurred only when the Pentium 4 desktop CPU was used in the client host and occurred nowhere else.

Our original Intel Pentium 4 laptop (a 1.8GHz Pentium 4-M) and the new Intel Pentium 4 desktop (a 3.06GHz Pentium 4) have entirely different motherboards, Ethernet devices, and so forth. We conjecture that the artifact may be a consequence of the Pentium 4's power management, putting the computer to sleep at inopportune moments. While a more detailed study of this effect is beyond the scope of this paper, prospective attackers will certainly select and profile their timing host to ensure these artifacts do not occur.

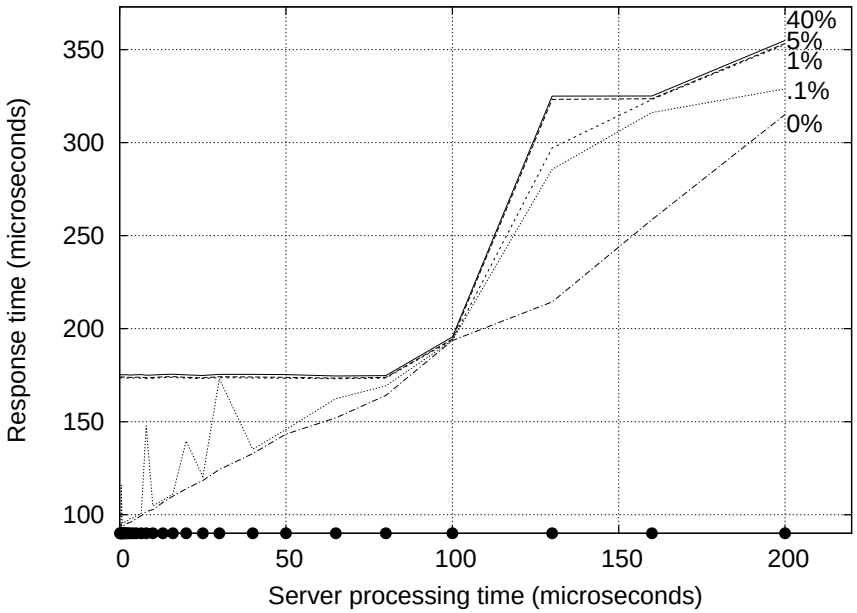

Figure 12: Response time percentiles as a function of processing time for a Intel Pentium 4 laptop measurement host and AMD Athlon target on a crossover cable.

7.8 Network card dependencies

We performed a further experiment to characterize how the choice of networking card (or *NIC*) might impact unfiltered jitter. We ran a new experiment between two machines that each had two networking cards. Both machines had a 2.1GHz AMD Athlon CPU and an Intel 82540EM gigabit ethernet controller. Machine A acted as the sender and has an onboard Via VT6102 100baseT ethernet controller. Machine B acted as the receiver and has an onboard 3c905C-TX/TX-M 100baseT ethernet controller. We connected these machines with either a crossover cable or a generic SpeedStream SS2108 10/100 baseT switch.

Table 3 summarizes our results on four new datasets where we collected up to 150k measurements for 60 processing times ranging from 1ns to 6.5ms on a 100baseT ethernet network. We found that the higher performance gigabit card, with interrupt coalescing enabled, had 30 times the unfiltered jitter than the generic onboard ethernet card.

There is a caveat to these results. Datasets A' and B' violate our network model because their jitter distribution is very different across different processing times. Figure 13 shows the deviation contours for the host in Dataset A' and demonstrates the variability in the jitter distribution. These results indicate that some modern high performance servers, with high performance NICs, may be safer from remote timing attacks as a direct consequence of their performance features.

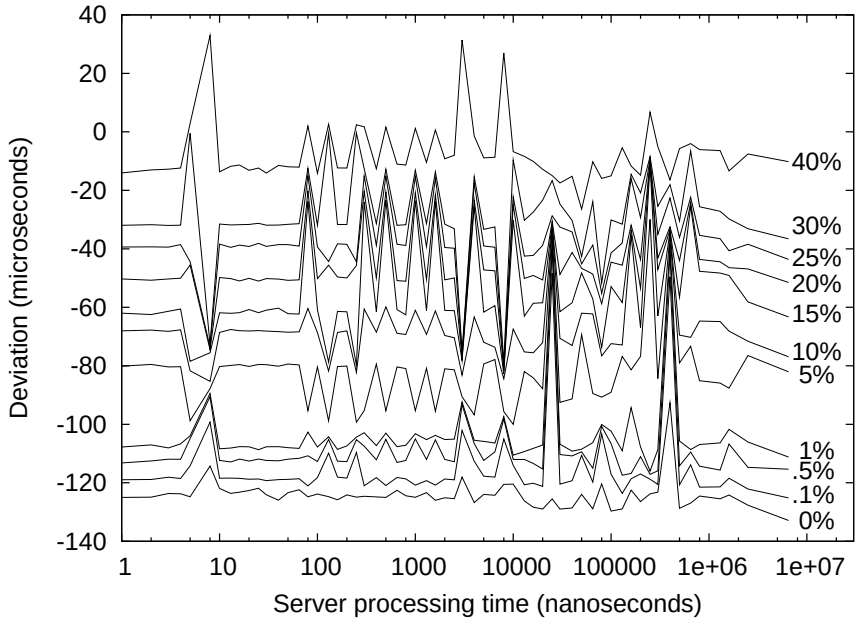

Figure 13: Deviation contours from ideal straight line for Dataset A', using an Intel gigabit NIC.

Experiment	Sender NIC	Receiver NIC	IRQ coalescing	Connection	Measurements	Unfiltered jitter
A'	Intel	Intel	default	switch	150k	1419ns
B'	Intel	Intel	disabled	switch	100k	676ns
C'	Onboard	Onboard	n/a	crossover	100k	56ns
D'	Onboard	Onboard	n/a	switch	100k	49ns

Table 3: Unfiltered jitter differences between networking cards.

8 Simulating Attacks

A statistical analysis of unfiltered jitter is useful in that it identifies the lower bound of noise in the system and indicates what resolution is theoretically distinguishable. In this section, we simulate an attacker performing a timing attack and evaluate the empirical resolution that a real attacker could identify. In statistical terminology this amounts to performing a *hypothesis test*.

In hypothesis testing, the statistician tries to show that their observations are statistically significant enough to exclude the default or *null hypothesis*. As our measurement dataset contains millions of measurements of 'ground truth', where we know the actual processing time, we may evaluate the empirical effectiveness of different hypothesis testing approaches an attacker may attempt. Our choices of which approaches we try are guided by the results in the last section, where we determined that the mean, median, and measurements above the 10-20th percentile are extremely noisy.

We show and test a framework of statistical hypothesis testing to determine how fine a time difference Δ might be detectable within a reasonable number of measurements, both for LAN and Internet attackers. We find that, with as few as 2000 measurements, it is possible for an attacker to distinguish a 100ns processing-time difference on a LAN with false negative and false positive rates under 5%.

8.1 Classic hypothesis testing

Many hypothesis testing approaches that distinguish whether two measurements sets U, V are from the same distribution or different distribution, such as the Student's t-test, use a *test statistic* ϕ, computed from the summary statistics of U, V such as the mean, standard deviation of U,V and the sample count. If ϕ exceeds a statistically significant threshold, then the null hypothesis, that U and V are from the same distribution, is rejected.

Of course, any such analysis of hypothesis testing with a single threshold incurs a chance of two types of errors: false positives, when we improperly reject the null hypothesis, and false negatives, when we fail to reject the null hypothesis when we should.

8.2 Empirical hypothesis testing

In our scenario, we are not limited to computing the theoretical effectiveness of a hypothesis test because our datasets contain ground truth. We may simulate an attacker performing empirical hypothesis tests and we rate the attacker's effectiveness assuming it has an oracle that helps it choose optimal parameters. Rather than collect a new dataset for hypothesis testing, we reuse our previously-collected raw measurements, as analyzed in the previous section.

Our simulated attacks follow this procedure: the hypothesis test H is given two sets of N samples of response times $X = r_1[i], ...r_N[i]$ and $Y = r_1[j], ...r_N[j]$ corresponding to processing times $t[i]$ and $t[j]$ which may or may not differ by Δ. For any given host pair and Δ we wish to discriminate, we can randomly choose data from our network measurements $r_n[m]$ to populate X, Y. We let the null hypothesis be that $\Delta = 0$, i.e., that

X, Y are from the same distribution. We then perform the hypothesis test $H(X, Y)$. To *accept the null* means that H believes that X, Y are from the same distribution and to *reject the null* means that H believes it has sufficient evidence that the X, Y are in fact different distributions.

We use the same testing procedure for each hypothesis test to compute the empirical false positive (FP) and false negative (FN) rate for each host pair and Δ. To compute the false negative rate, we perform 200 trials where we populate the queries X, Y from different distributions ($t[j] = t[i] + \Delta$) and count how many times the hypothesis test mistakenly accepts the null. To compute the false positive rate, we perform 200 trials populating the queries X, Y from the same distribution ($t[i] = t[j]$) and count how many times the test mistakenly rejects the null. We summarize these measurements by computing, for each host in question, the smallest Δ with a FN and FP rate below 5% and denote this as *empirical resolution*. If the hypothesis test H_p includes parameters p, we find the ideal parameters by brute force. We try H with all choices and keep the best-performing instance. This simulates a "best-case" attacker.

8.3 Hypothesis testing approaches

We considered four different statistical hypothesis testing approaches:

1. **Students t-test or other parametric approaches** We considered attempting the t-test on our raw measurements. We rejected it and other parametric approaches because our sample mean, sample variance and upper percentiles have high estimation error. We describe our reasons for rejecting these approaches in Section 8.4.

2. **The Wilcoxian Rank-sum** The Wilcoxian rank sum test is a standard test for identifying if two sets are from different distributions. It performed poorly; see Section 8.5 for details.

3. **Modified Students t-test** We model an attacker that instead of running the t-test on the raw measurements, runs the t-test upon filtered measurements. The empirical quantile is distributed about the true quantile in a Gaussian distribution. By measuring the variance of the empirical quantile, we can run the t-test. Our results are described in Section 8.6.

4. **The 'Box' Test** This is our best-performing test. This test exploits our observation that smaller quantiles in our measurements have less noise. Given a measurement set U and two quantiles i, j, we define an interval $[\hat{q}_i(U), \hat{q}_j(U)]$. The test rejects the null if the intervals induced by those quantiles are non-overlapping and in the proper order. Our results are described in Section 8.7.

8.4 Parametric approaches

A parametric statistical approach assume the distribution of the jitter follows a parameterized distribution (e.g. Gaussian, exponential, etc.) and we need only compute parameters of the fit such as the mean and variance. Doing so allows us to design

stronger estimation procedures which exploit the assumed structure. However, it also requires us to perform a goodness-of-fit test, to see whether the model is appropriate.

Our data clearly shows the medium and upper quantile estimates, including the sample mean and median, suffer from extremely large variability, precluding most parametric models as they would be rejected in a goodness-of-fit test. An inference made on the basis of a poor model would be unlikely to be accurate.

8.5 Wilcoxon rank-sum test

The Wilcoxian rank-sum test (also called the Mann-Whitney rank sum test) is a non-parametric test to determine if two sets of observation come from the same distribution. We start with two sets of samples of response times X, Y. The two sample sets are combined and ranked (i.e., sorted together). The rank of an element is its index in the sorted joined list. The rank sum statistic is then computed as the sum of the ranks of *one* of the sets:

$$\phi_{\text{rank}}(r_1[m], \ldots r_N[m], r_1[m'], \ldots r_L[m']) = \sum_{j=1}^{N} \text{rank}(r_j[m]) \tag{11}$$

If the sample sets represent the same distribution, one would find the mean and variance of ϕ_{rank} to be

$$\begin{aligned} E[\phi_{\text{rank}}] &= \frac{N(N+L+1)}{2} \\ \text{var}(\phi_{\text{rank}}) &= \frac{NL(N+L+1)}{12} \end{aligned} \tag{12}$$

If the obtained rank sum ϕ_{rank} is close to this mean, then we conclude that the processing times were identical. If instead the distributions were different, i.e., $t[m'] > t[m]$, then the responses $r_n[m]$ will tend to be smaller and earn smaller ranks, thus reducing the sum of their ranks. This process works well, even if the distributions are non-Gaussian, so long as they represented shifted copies of each other.

Following our hypothesis test procedure to determine the false positive and false negative rates, we found this methodology significantly under-performed the "box test" described in Section 8.7.

8.6 Modified t-test

The t-test is a well-known test for distinguishing if two sets of measurements come from the same or different Gaussian distributions. Given the mean, sample count, and variance of two sets of samples, the t-test computes the probability that the two sets are drawn from the same distribution. Unfortunately, our raw measurements have a very skewed, non-Gaussian distribution. Instead, we apply the t-test to our percentiles and other filtered measurements which can be expected to be distributed with a Gaussian distribution.

The modified t-test we apply is parameterized by a filter Γ and a threshold T. We test each of 19 well-performing filters and 12 thresholds. As before, we start with two

sets of N samples of response times X, Y. For each of 200 random subsets of size $N/10$ from the N measurements in X, Y, we apply the filter Γ. We estimate the variance and means of $\Gamma(X), \Gamma(Y)$ from these 200 sampled random subsets and apply the standard t-test with a threshold of T on the t-statistic.

Following our testing procedure for hypothesis tests to determine the false positive and false negative rates, we found that this test had half of the resolution of the 'box test', described next.

8.7 The box test

The final test we tried is a custom test designed to exploit our observation that small percentiles have the least noise. We also argue for this filter design based on its simplicity[2]. The 'box test' is parameterized by two quantiles i, j and, as before, we start with two random subsets X, Y of N measurements.

To perform this test, X, Y, are sorted and two intervals are formed: $[\hat{q}_i(X), \hat{q}_j(X)]$ and $[\hat{q}_i(Y), \hat{q}_j(Y)]$. The test accepts if the intervals $[\hat{q}_i(X), \hat{q}_j(X)]$ $[\hat{q}_i(Y), \hat{q}_j(Y)]$ do not overlap and $[\hat{q}_i(X), \hat{q}_j(X)]$ is before $[\hat{q}_i(Y), \hat{q}_j(Y)]$. We calculated the false positive and false negative rates by following our testing procedure for hypothesis tests.

Since we cannot a priori know which values for the parameters i, j create the most effective statistical test, we perform this experiment for 12,000 i, j pairs, keeping whichever pair i, j has the lowest FN rate while also having a FP rate below 5%. From our measurements, the lower quantiles are the statistically most reliable part of the distribution. Our exhaustive search for optimum parameters chose $i, j < 6\%$ for over half of the hosts. Although a real attacker would not have such an oracle against an unknown target, an attacker could always perform measurements similar to ours against a known machine, hopefully near the target. Also, we prefer to state an upper bound on the capabilities of an attacker.

Figures 14 and 15 show our ability to measure Δ for two hosts chosen from datasets L and five hosts chosen from D, respectively. Data for other hosts and other datasets reflects a similar pattern. These plots modeled an attacker with 500 measurements and plot the FN rate with the FP rate held to at most 5%. The FN rate remains until the time difference increases past a critical threshold, at which time the FN rate drops quickly to zero and the discriminator is exceptionally accurate.

Rather than showing similar scatter plots for every host that we measured, we summarize our measurements in Figure 16 by computing, for each host in question, the smallest Δ which we can discriminate with a FN rate below 5%. We call this the *empirical resolution*. We see that many LAN hosts can accurately resolve a Δ in the hundreds of nanoseconds, and that PlanetLab hosts can resolve Δ's around 100μs, with the best hosts resolving $\Delta \approx 30\mu$s.

So far, we have modeled the empirical resolution of an attacker performing 500 measurements. To see how additional measurements would increase the attacker's abilities, we varied the number of simulated measurements from 10 to 10,000. We summarize the histogram over each of the 124 hosts in Dataset D by its 5 quartiles

[2] "Complicated computations do not guarantee a valid statistical analysis. Always start ... with a careful examination of the data." [27]

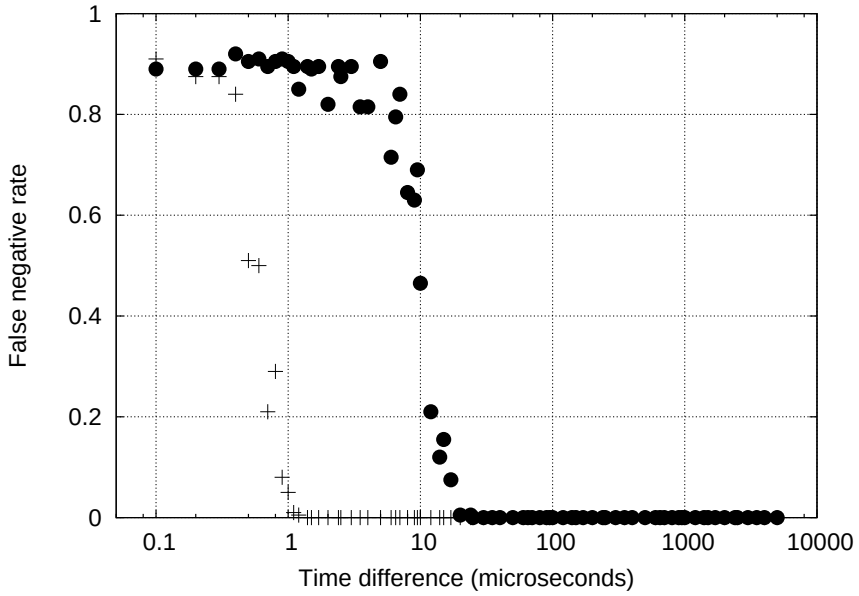

Figure 14: Scatter plot of Δ versus the FN rate for two Dataset L hosts. Each dot represents the average success rate over 200 trials, each of which simulates 500 network measurements.

(0th, 25th, 50th, 75th and 100th percentile), and plot those with respect to the simulated measurement count in Figure 17.

As expected, increasing the number of queries radically improves the empirical resolution. Where the best Dataset D host, with 500 measurements, could resolve $\Delta = 50\mu$s with $< 5\%$ FP and FN rate, with 10,000 queries 31 hosts could resolve $\Delta < 35\mu$s with a $< 5\%$ FP and FN rate. Empirical resolution improves for all other hosts as well. We can also see that the network location of an attacker can make a significant difference. The top 25% hosts can resolve a difference 3 times smaller than the bottom 25% hosts, regardless of the number of samples. This implies that an Internet attacker will do much better if many machines are available, allowing the best positioned machine to be used for attack measurements.

Figure 18 shows the same simulation performed over our 8 LAN hosts. Unfortunately, we did not sample enough discrete processing times below 100ns to directly compute the empirical resolution for these small time steps. However, we can observe a similar improvement in the empirical resolution of LAN hosts as they measure more timing samples. It is reasonable to expect that a LAN attacker making 10,000 queries might accurately resolve $\Delta < 50$ns.

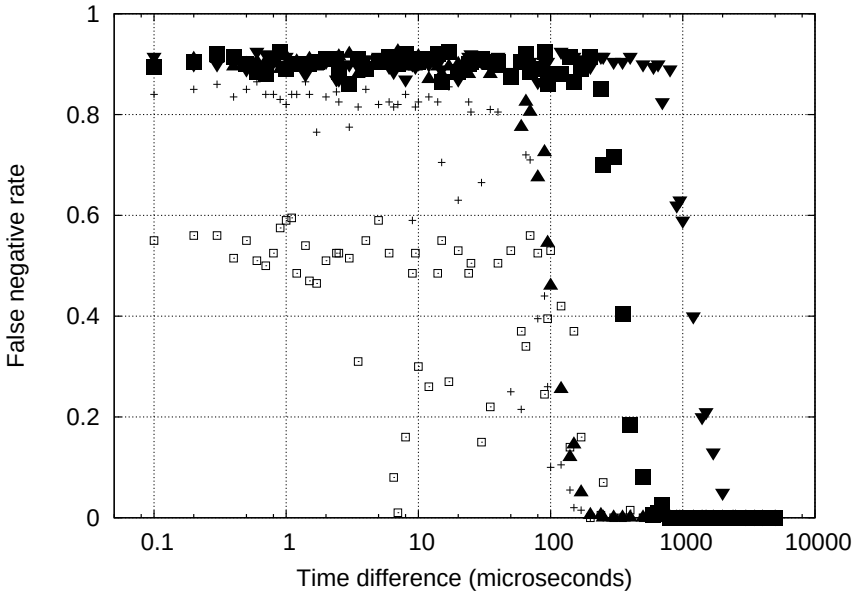

Figure 15: Scatter plot of Δ versus the FN rate for five Dataset D hosts. Each dot represents the average success rate over 200 trials, each of which simulates 500 network measurements.

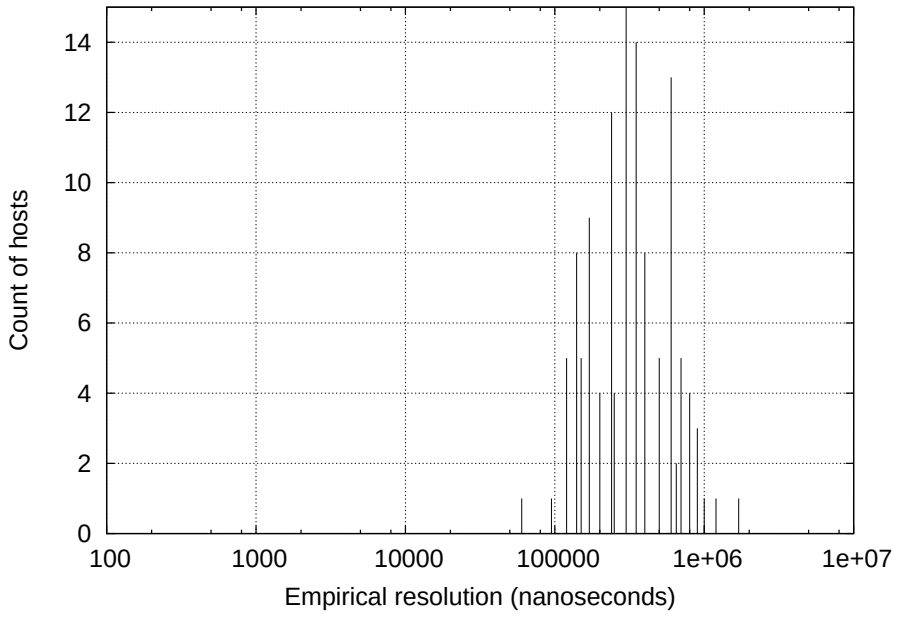

Figure 16: Empirical resolution histogram for Dataset D.

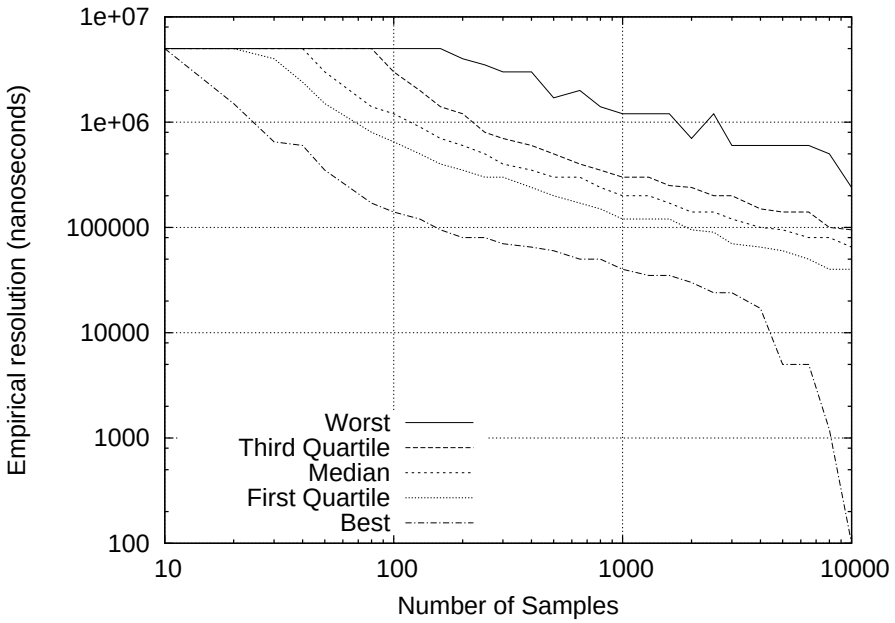

Figure 17: Empirical resolution on Dataset D as a function of the sample count.

8.8 Attack applications

While most of this paper has focused abstractly on the empirical resolution of an attacker to discriminate remote processing times, it is important to consider the processing times of common operations that an attacker might wish to attack.

Brumley and Boneh's attack on SSL exploits a timing difference that is a function of the RSA private key and the message being exponentiated [8]. Their attack creates two sets of inputs that *on average* have a difference. Depending on the particular input taken from those sets, the actual difference will vary. Many samples may need to be averaged together. As a result, our techniques do not directly apply to their attack.

We tested their attack and found that on an 1.8GHz Athlon, it depends on timing differences of about 30,000 clock cycles (16μs). From Figure 18, we can estimate that the attack should succeed on any host within our LAN with a few hundred measurements. From a histogram similar to Figure 16, we estimate that 4 hosts in Dataset D could perform the attack with fewer than 10,000 measurements for each bit of RSA key, while others would require significantly more. Brumley and Boneh found their attack did not work over the Internet, but our results suggest that an attacker with access to enough machines, across the Internet, might well have one that works.

8.9 Application jitter

Our analysis of network jitter used a custom application on an unloaded server. A realistic application load might introduce its own jitter into the response time because of

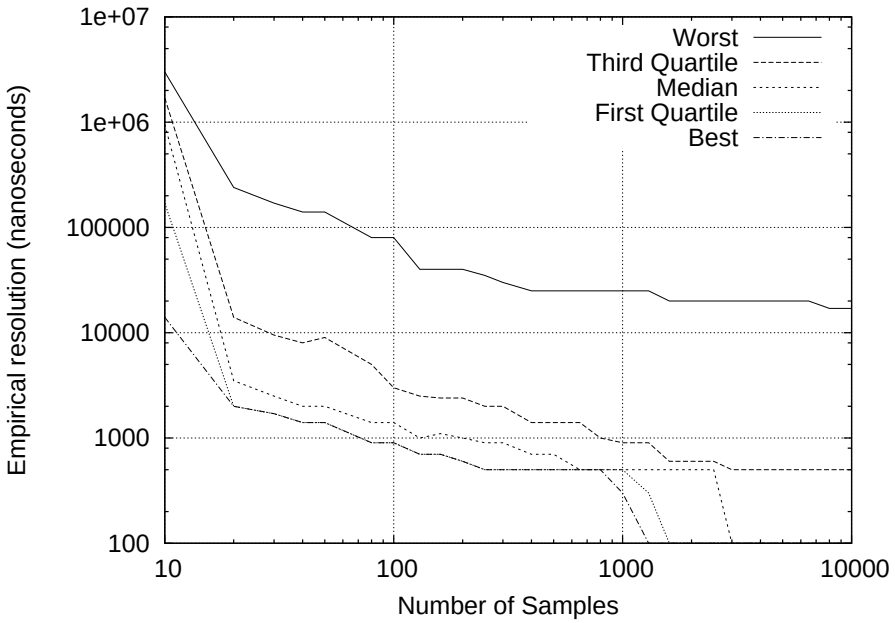

Figure 18: Empirical resolution on the local network (Dataset L) as a function of the sample count.

cache misses, page faults, TLB faults, lock contention and other sources. To characterize such jitter, we analyzed the Apache web server, version 1.3.33, both under no load and under two trace-driven loads with a 1.1GB working set (larger than system RAM, guaranteeing pressure on the disk system). For simplicity, the load generation clients ran on the same computer as the HTTP server. One run was performed with sequential requests and another run was performed with 30 concurrent requests. The CPU was saturated during both runs.

Measurements were taken by a custom HTTP client, on a separate system, requesting a small file over an unloaded network. We collected 100,000 measurements over the three different workloads. As an attacker would do, our client avoids connection startup jitter by first sending only part of the HTTP request, then deliberately delaying to ensure that it is received and processed by the server before it sends the remaining part of the request.

In Figure 19, we graph the unfiltered application jitter as a function of the number of request samples. We estimate the application jitter as the standard deviation of the lowest variance quantile filter. We choose 10,000 random subsets of n measurements, filter them using a quantile filter q_i, and compute the variance of the empirical quantile $\hat{q}_i$ across all 10k subsets.

We also plot quartiles of empirical resolution from Figures 17 and 18 from Dataset L and D trials for comparison. We can see that a realistic application like Apache introduces only a microsecond of jitter at 1000 samples even when highly loaded, and

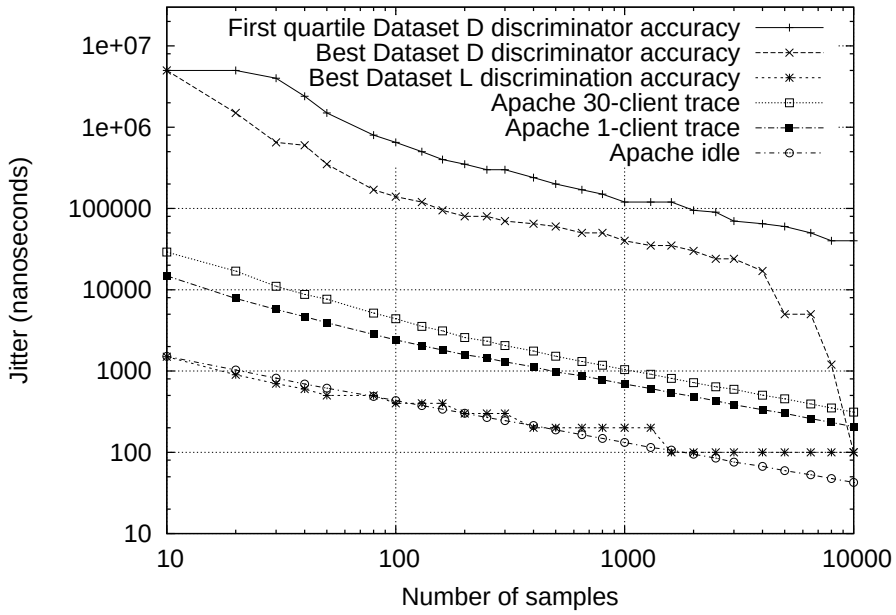

Figure 19: Unfiltered application jitter induced by a loaded Apache web server compared to empirical resolution from our local (L) and PlanetLab (D) measurements.

less jitter than the local network when unloaded.

9 Conclusion

This paper studied the scope and potential of performing a remote timing attack, both on a LAN and across the Internet. Any such attack will require making multiple timing measurements of an an event on a remote server and filtering those measurements to eliminate noise that might be induced by the network or by the end hosts. We have shown that, even though the Internet induces significant timing jitter, we can reliably distinguish remote timing differences as low as 20μs. A LAN environment has lower timing jitter, allowing us to reliably distinguish remote timing differences as small as 100ns (possibly even smaller). These precise timing differences can be distinguished with only hundreds or possibly thousands of measurements.

Good filtering of those measurements is fundamental to mounting a successful attack. Contrary to conventional wisdom, using either the median response time or the minimum response time observed as a filter significantly under-performs filters that sort the data and look at values early in the range (e.g., 1% into the sorted list). Based on filters that use these low percentiles, we can construct a "box test" that reliably distinguish small timing differences, when they are present, with low false positive and low false negative rates.

We also observed, generally, that the round trip time or network hop count did not significantly contribute to the network jitter, and thus network distance may not confer immunity to remote timing attacks. We found that the choice CPU or networking card may introduce more jitter than a local area networking. Prospective attackers can work around this by benchmarking their measurement machines, in advance.

If an attacker can accurately perform timing measurements, then a number of cryptographic or algorithmic weaknesses in a server might leak critical information to the attacker. As a consequence, we recommend that the algorithms used inside web and other Internet servers that process important secrets be carefully audited and, where necessary, be modified to limit observable differences in execution times to at most a few microseconds.

References

[1] A. Acharya and J. Saltz. A study of Internet round-trip delay. Technical Report CS-TR-3736, Department of Computer Science, University of Maryland, Dec. 1996.

[2] O. Acıiçmez, Çetin Kaya Koç, and J.-P. Seifert. Predicting secret keys via branch prediction. In *The Cryptographers' Track at the RSA Conf. (CT-RSA)*, pages 225–242, San Francisco, Feb. 2007.

[3] O. Acıiçmez, W. Schindler, and Çetin Kaya Koç. Cache based remote timing attack on the AES. In *The Cryptographers' Track at the RSA Conf. (CT-RSA)*, pages 271–286, San Francisco, Feb. 2007.

[4] P. Barford and M. Crovella. Critical path analysis of TCP transactions. *IEEE/ACM Trans. Netw.*, 9(3):238–248, 2001.

[5] D. J. Bernstein. Cache-timing attacks on AES, Apr. 2005. `http://cr.yp.to/papers.html#cachetiming`.

[6] J.-C. Bolot. End-to-end packet delay and loss behavior in the Internet. In *ACM SIGCOMM: Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 289 – 298, San Francisco, CA, 1993.

[7] A. Bortz, D. Boneh, and P. Nandy. Exposing private information by timing web applications. In *16th Int. World Wide Web Conf.*, Banff, Alberta, Canada, May 2007.

[8] D. Brumley and D. Boneh. Remote timing attacks are practical. In *Proc. of the 12th USENIX Security Symposium*, Washington, DC, Aug. 2004.

[9] B. Canvel, A. Hiltgen, M. Vuagnoux, and S. Vaudenay. Password interception in a TLS/SSL channel. In *Advances in Cryptology — Crypto Proc.*, volume 2729 of *LNCS*, pages 583–599. Springer-Verlag, Aug. 2003.

[10] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, Apr. 1979.

[11] S. Casner, C. Alaettinoglu, and C.-C. Kuan. A fine-grained view of high-performance networking. NANOG22 meeting, May 2001. `http://www.nanog.org/mtg-0105/casner.html`.

[12] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An overlay testbed for broad-coverage services. *ACM Computer Communications Review*, 33(3), July 2003.

[13] K. C. Claffy, G. C. Polyzos, and H.-W. Braun. Measurement considerations for assessing unidirectional latencies. *Internetworking: Research and Experience*, 4(3):121–132, Sept. 1993.

[14] S. Crosby and D. S. Wallach. Denial of service via algorithmic complexity attacks. In *Proc. of the 12th USENIX Security Symposium*, Aug. 2003.

[15] J. Daemen and V. Rijmen. Resistance against implementation attacks: A comparative study of the AES proposals. In *2nd AES Candidate Conf.*, Rome, Italy, Mar. 1999.

[16] E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In *Proc. of 7th ACM Conf. on Computer and Communications Security*, Athens, Greece, Nov. 2000.

[17] R. J. Jenkins. Hash functions for hash table lookup, 1995. `http://burtleburtle.net/bob/hash/evahash.html`.

[18] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Side channel cryptanalysis of product ciphers. *Journal of Computer Security*, 8(2-3):141–158, 2000.

[19] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology — Crypto Proc.*, volume 1109 of *LNCS*, Santa Barbara, california, Aug. 1996. Springer-Verlag.

[20] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology — Crypto Proc.*, volume 1666 of *LNCS*, Santa Barbara, CA, Aug. 1999. Springer-Verlag.

[21] T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. In *Proc. of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2005.

[22] M. G. Kuhn. Optical time-domain eavesdropping risks of CRT displays. In *Proc. of the IEEE Symp. on Security and Privacy*, Oakland, CA, May 2002.

[23] M. G. Kuhn and R. J. Anderson. Soft Tempest: Hidden data transmission using electromagnetic emanations. In *2nd Workshop on Information Hiding*, Portland, OR, Apr. 1998.

[24] B. W. Lampson. Hints for computer system design. *ACM Operating Systems Review*, 15(5):33–48, Oct. 1983. Reprinted in *IEEE Software* 1(1):11-28 (Jan. 1984).

[25] D. L. Mills. Internet time synchronization: The Network Time Protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, Oct. 1991.

[26] D. L. Mills. Network time protocol (version 3). Technical Report RFC-1305, Internet Engineering Task Force, Mar. 1992. `ftp://ftp.rfc-editor.org/in-notes/rfc1305.txt`.

[27] D. Moore and G. McCabe. *Introduction to the Practice of Statistics*. Freeman, New York, 5th edition, 2004.

[28] A. Mukherjee. On the dynamics and significance of low frequency components of Internet load. *Internetworking: Research and Experience*, 5(4):163–205, 1994.

[29] D. A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: The case of AES. In *The Cryptographers' Track at the RSA Conf. (CT-RSA)*, volume 3860 of *LNCS*, pages 1–20, San Jose, CA, Feb. 2006. Springer.

[30] D. Page. Theoretical use of cache memory as a cryptanalytic side-channel. Technical Report CSTR-02-003, Department of Computer Science, University of Bristol, June 2002.

[31] A. Pasztor and D. Veitch. PC based precision timing without GPS. In *Proc. of the ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Computer Systems*, pages 1–10, Marina Del Rey, CA, June 2002.

[32] V. Paxson. End-to-end Internet packet dynamics. In *ACM SIGCOMM: Applications, Technologies, Architectures and Protocols for Computer Communication*, Cannes, France, Sept. 1997.

[33] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of CA at Berkeley, Berkeley, CA, Apr. 1997.

[34] V. Paxson. On calibrating measurements of packet transit times. In *SIGMETRICS/PERFORMANCE: Joint Int. Conf. on Measurement and Modeling of Computer Systems*, pages 11–21, Madison, WI, June 1998.

[35] D. Sanghi, A. K. Agrawala, O. Gudmundsson, and B. N. Jain. Experimental assessment of end-to-end behavior on Internet. In *Twelfth Annual Joint Conf. of the IEEE Computer and Communications Societies*, pages 867–874, San Francisco, CA, Mar. 1993.

[36] W. Schindler. Optimized timing attacks against public key cryptosystems. *Statistics and Decisions*, 20:191–210, 2002.

[37] J. H. Silverman and W. Whyte. Timing attacks on NTRUEncrypt based on variation in number of hash calls. In *The Cryptographers' Track at the RSA Conf. (CT-RSA)*, San Francisco, Feb. 2007.

[38] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Proc. of the Tenth USENIX Security Symposium*, Washington, DC, Aug. 2001.

[39] D. Veitch, S. Babu, and A. Pasztor. Robust synchronization of software clocks across the Internet. In *Proc. of the 4th ACM SIGCOMM Conf. on Internet Measurement*, pages 219–232, Taormina, Sicily, Italy, Oct. 2004.

[40] Z. Wang, A. Zeitoun, and S. Jamin. Challenges and lessons learned in measuring path RTT for proximity-based applications. In *Passive and Active Measurement Workshop*, San Diego, CA, Apr. 2003.