Unsupervised Clustering Under Temporal Feature Volatility in Network Stack Fingerprinting

Zain Shamsi and Dmitri Loguinov*
Department of Computer Science and Engineering
Texas A&M University, College Station, TX 77843 USA
{zain, dmitri}@cse.tamu.edu

ABSTRACT

Maintaining and updating signature databases is a tedious task that normally requires a large amount of user effort. The problem becomes harder when features can be distorted by observation noise, which we call *volatility*. To address this issue, we propose algorithms and models to automatically generate signatures in the presence of noise, with a focus on *stack fingerprinting*, which is a research area that aims to discover the operating system (OS) of remote hosts using TCP/IP packets. Armed with this framework, we construct a database with 420 network stacks, label the signatures, develop a robust classifier for this database, and fingerprint 66M visible webservers on the Internet.

Keywords

OS Fingerprinting; OS Classification; Internet Measurement

1. INTRODUCTION

With the immense growth of the Internet, classification of large networking datasets has become an important topic [1], [6], [9], [11], [14], [16], [19], [20], [26], [34], [35]. For classifiers to work, there must be a process that establishes signatures for known types of behavior and builds a database that contains all sufficiently different specimens found in the wild. To keep results up-to-date, new signatures must be periodically acquired and merged into the existing database. This is often a manual process that suffers from human error, poor repeatability, heuristic decisions, and database compositions incompatible across different classification methods.

To overcome these problems, we investigate algorithms and models for automated creation of clusters among the available samples, elimination of duplicates, and assignment of labels to the resulting signatures. We next explain the issues involved and our results.

*Supported by NSF grant CNS-1319984.

SIGMETRICS '16, June 14-18, 2016, Antibes Juan-Les-Pins, France
© 2016 ACM. ISBN 978-1-4503-4266-7/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2896377.2901449

1.1 Motivation and Contributions

Performance of each classifier depends on not only its internal algorithms, but also database $\mathcal{D}$ and types of volatility experienced during measurement. This makes comparison between different approaches (e.g., Nmap [21], Snacktime [3], Hershel [25], p0f [33]) fairly complicated, especially if they utilize incompatible sets of features, databases, or assumptions on feature determinism. For example, consider method $\mathcal{M}_1$ with n signatures and $\mathcal{M}_2$ with $m \ll n$. It may appear that $\mathcal{M}_1$ is more powerful because its $\mathcal{D}$ is bigger; however, its classification accuracy may be lower due to the larger number of options to choose from and/or less reliable decision-making. Additionally, the specific model of distortion $\mathcal{X}$ (i.e., noise in certain features) applied during the experiment may have a dramatic impact on the result. In such cases, it is possible that $\mathcal{M}_1$ resorts to random guessing and makes inferior choices to those of $\mathcal{M}_2$.

To capture these aspects, our first contribution is to propose that each classification method be characterized by the number of signatures $d(1-\epsilon, \mathcal{X})$, which we call the *dimension*, between which it can differentiate with probability at least $1-\epsilon$ under a given $\mathcal{X}$. We also argue that database $\mathcal{D}$ should be customized to each pair $(\epsilon, \mathcal{X})$ to contain exactly $d(1-\epsilon, \mathcal{X})$-separable signatures. To determine the dimension and the corresponding $\mathcal{D}$, our second contribution is to propose an algorithm we call Plata[1], which disturbs each candidate signature in $\mathcal{D}$ using $\mathcal{X}$ and verifies that it can be matched to itself with probability at least $1-\epsilon$. Samples that fail to meet this criterion are eliminated and classification decisions among other signatures are redistributed in an iterative procedure that stops when all remaining candidates are $(1-\epsilon, \mathcal{X})$-separable. Assuming availability of labels for a subset of initial candidates, we explain how Plata automatically assigns them to the d generated clusters.

We apply these concepts to Hershel [25], which is a classifier that, unlike all previous tools in stack fingerprinting, allows random OS behavior and provides probabilities, rather than heuristic weights, for the match across any pair of samples. We focus on its temporal network features (i.e., delay jitter) since they are highly volatile and fairly well-understood, but difficult to separate using manual analysis. This leads to our third contribution that consists of building a Plata database using 9.7K webservers discovered in our campus network and passing all HTTP headers through simhash [16] to label the elements of $\mathcal{D}$. Using only delay features, we show that Hershel achieves 80%-separation under

[1]The city of La Plata in Argentina pioneered fingerprint databases in 1892.

500-ms random distortion on 117 signatures. Adding deterministic header values, this number jumps to $d(0.8, \mathcal{X}) = 398$, which is 3.4 times larger than the biggest database in prior work [25].

While Plata works well, its Monte Carlo simulations require a large amount of CPU time to compute the Hershel probabilities (i.e., over 24 hours using 16 cores). Therefore, our fourth contribution is to build a closed-form model for the matrix produced by Plata. This leads to an interesting discovery that Hershel's iid (independent and identically distributed) jitter assumption [25] is violated in practice, making the model disagree with simulations. We therefore create a novel classifier for temporal features that relies on one-way delay instead of jitter. We call the resulting method Hershel+ and show that it is not only more accurate, but also faster than Hershel after an appropriate expansion of integrals. It also admits a closed-form representation of the entire Plata matrix, which reduces the separation time to just 12 minutes and boosts our database dimension to 420 separable signatures. All of this forms our fifth contribution.

We finish the paper by scanning the Internet on port 80 and applying Hershel+ to the result. Among Internet-wide studies, this is the largest population to be fingerprinted (i.e., 66M IPs), using the most extensive database (i.e., 420 signatures), and the first such attempt with an automatically generated $\mathcal{D}$. Compared to the scan six years ago [25], we find that the number of Linux and embedded devices has almost doubled, while that of Windows has remained stable. We compare some of our results with those of Nmap and discover a major flaw in the operation of the latter that surfaces in scenarios with non-ideal network conditions (e.g., firewalls). More importantly, however, we conclude that stochastic network effects do not impede the use of temporal features, but they require a more careful database construction process. Our proposed framework of Plata and Hershel+ is a step in the direction of automated, repeatable, and streamlined classification of massive datasets.

2. BACKGROUND

2.1 Remote OS Classification

Stack fingerprinting is often used in market-share analysis [19], [20], Internet characterization [11], [14], research measurements [4], [8], [15], and security, where administrators aim to discover vulnerable devices and/or stealth intruders in the network [1], [17], [26]. We split the work across two main categories in Figure 1. In the first tier are classifiers that rely only on deterministic features, usually selected from the headers of various protocols (i.e., TCP, IP, UDP, ICMP). Among these, Nmap [21] is the most prominent tool with rules to identify over 4K network stacks. With hundreds of transmitted probes, several protocols that must pass remote firewalls, and complaints from network administrators during fingerprinting of their networks, Nmap is not generally considered suitable for Internet-scale use. Additional classifiers in this category include p0f [33], Xprobe [32], and several others [2], [18], [28]. They have a smaller presence and significantly fewer signatures, but most of their ideas have been ported to Nmap.

The second direction in Figure 1 handles random features, usually in the form of delays produced by the OS. One option is use the clock drift in the kernel, which can be derived from observing the timestamp option in streams of reply

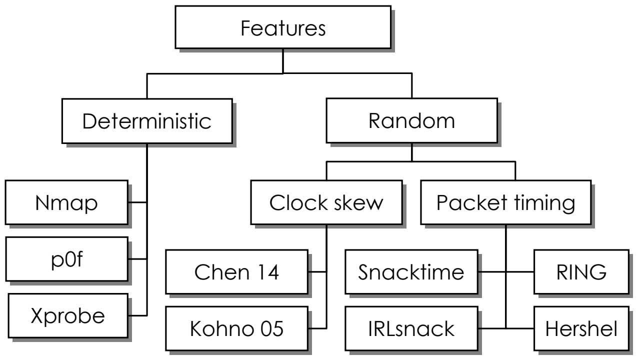


Figure 1: Taxonomy of previous work.

packets [13] or variation in timer frequency [6]. Another option is to monitor retransmission timeouts (RTOs) of SYN-ACKs for half-open connections, where the methods include Snacktime [3], RING [30], IRLsnack [14], and Hershel [25]. The last classifier uses a stochastic model that takes into account packet loss, delay jitter, and random header-field modifications by end-users, some of which we review below.

2.2 Database Creation

The majority of efforts in stack fingerprinting [2], [3], [6], [13], [18], [21], [25], [28], [30], [33] concentrate on introducing new features and designs to further distinguish between the OSes, thus improving the classification step; however, they universally rely on manual effort to construct databases. Since all of them rely only on deterministic features, database creation is fairly uncomplicated.

The closest related problem to ours is automatic discovery of features that can be used to differentiate one OS from another. For example, [5] proposes a set of rules built from sending out a large number of probes (i.e., 300K) to controlled hosts and randomly varying header fields to detect patterns that produce OS-specific responses. The authors show that this method can reliably differentiate between three stacks (i.e., Windows XP, Linux 2.6, and Solaris 9) in a LAN environment.

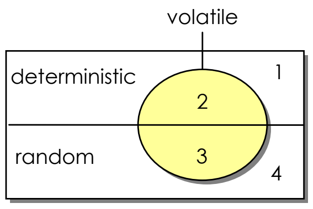
In [23], this idea is explored at a larger scale by increasing the number of network stacks and applying a wider range of machine-learning algorithms from the Weka tool [10]. However, their results from scaling this approach to more signatures are quite pessimistic – the authors conclude that over-fitting to non-deterministic header fields, training bias towards certain implementations, and lacking semantics lead to confusion for the learning algorithms.

3. OVERVIEW

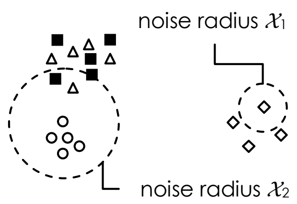
We start by defining the type of decisions we are facing and the inherent challenges. While later sections use examples from stack fingerprinting, the same concepts are applicable to broader families of problems.

3.1 Terminology

Classifiers rely on vectors of distinctive features that identify each specimen, either uniquely or with some reasonably high probability. The former case arises when the features are *deterministic*, meaning all inspections of a given system produce the same result (e.g., the order of TCP options).



(a) types



(b) separability

Figure 2: Classifier features.

The latter case occurs when the features are inherently *random* due to some non-deterministic processes running within the specimen (e.g., SYN-ACK retransmission delays). Features of either type may undergo additional modification due to influence of system owners or as byproduct of the measurement process, in which case we call them *volatile* (e.g., users tuning the TCP window size, queuing delays affecting packet spacing). All four types are illustrated in Figure 2(a).

Note that volatility and randomness are not the same – the former arises due to forces *external* to the object being classified, while the latter due to *internal*. This distinction is important when internal disturbances exhibit substantially larger variance than external, or produce patterns that cannot be accounted for in the volatility model alone. With this in mind, we call classifiers *simple* if they operate using only non-volatile deterministic features (i.e., type-1 in Figure 2(a)) and *complex* otherwise (i.e., types 2-4).

Consider an automaton that performs classification decisions for measurements x using some database $\mathcal{D}$. We call the matching process *membership* if it returns the probability that $x \in \mathcal{D}$, where determination of the most-likely match is not important. One example is intrusion detection that aims to decide whether payload x is malicious or benign against a database of known exploits. We call the process *identification* if the result must produce the one signature $y \in \mathcal{D}$ with the highest similarity to x. Stack fingerprinting falls into this category. In either case, the accuracy of the method is assessed by the percentage of correctly classified values under a particular model of volatility.

3.2 Challenges

We are now ready to describe the problem of creating $\mathcal{D}$. Assume a measurement of several, possibly duplicate, specimens. Membership classifiers are not overly concerned with high-precision duplicate elimination as these have no effect on accuracy, only on speed and memory consumption. Simple identification classifiers can construct $\mathcal{D}$ by retaining the observations with unique combinations of features, which makes the problem trivial. However, complex identification classifiers must instead ensure *separability* among the signatures, keeping only those that can be reliably distinguished from each other under various types of distortion $\mathcal{X}$. Inseparable specimens in $\mathcal{D}$ drop classification accuracy and increase overhead, while offering no tangible benefit.

To visualize this better, Figure 2(b) plots random features of four hypothetical systems – circles, squares, diamonds, and triangles – where each point is a random observation of the corresponding system. Assuming uniformly random noise centered at each sample, distortion $\mathcal{X}_1$ keeps circles and diamonds separable, but not necessarily triangles and squares. Dropping either of the last two leads to a separable 3-signature database. For larger radius of noise (e.g., $\mathcal{X}_2$ in the figure) the database may consist of only two separable signatures – diamonds and one of circles/squares/triangles.

Our goal in this paper is to study separation algorithms for volatile and/or random features, with application to inter-packet delays in wide-scale stack fingerprinting. This problem arises in single-packet techniques [3], [25], [30] whose classifier must heavily rely on temporal features. The general appeal of these methods includes low bandwidth consumption (i.e., no extra packets beyond those sent by the scanner), a reduced probability of tripping IDS, no requirement that the target respond on closed ports or multiple protocols, and good scalability in Internet-wide classification. However, unlike traditional tools [21] that rely on deterministic features, single-packet classifiers require prohibitively expensive manual effort to construct databases of non-trivial size. Since this problem has not been studied before, we address it below.

4. DATABASE CREATION USING PLATA

This section describes our technique for ensuring separability between observations with volatile/random features and building a database on top of such measurements.

4.1 Preliminaries

Traditional manual construction of $\mathcal{D}$ isolates each unique system and lets the classifier analyze it separately. In contrast, our framework assumes a one-step measurement process that remotely probes production systems $S_1, \ldots, S_n$ and builds the entire database without knowing which ones are duplicates of each other. We allow these specimens to exhibit feature randomness and aim to construct $\mathcal{D}$ that is $(1-\epsilon)$-separable under a known volatility model $\mathcal{X}$.

To capture random behavior, each specimen S_i must be observed several times to establish a distribution of its behavior. Let Δ_i be the corresponding random feature vector whose probability mass function (PMF)

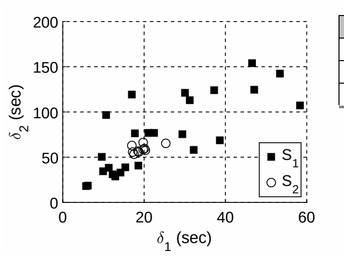
$$p_i(\delta) := P(\Delta_i = \delta) \tag{1}$$

is built from observation. Note that $\delta = (\delta_1, \delta_2, \ldots)$ is a deterministic feature vector that consists of multiple scalar values. Using a pair of initial RTOs (SYN-ACK retransmission timeouts), Figure 3(a) shows the distribution of Δ_i for two Xerox printers in our dataset. Depending on the target jitter model $\mathcal{X}$, these two hosts may very well be $(1-\epsilon)$-separable; however, doing so manually for hundreds of thousands of points is close to impossible. To compound the issue, the majority of systems use random vectors with at least 3 dimensions and some with over 20.

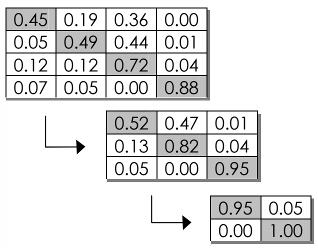
Classifiers that deal with random features must provide a function $p(\delta|\delta', \mathcal{X})$ that produces a *similarity score* for each pair of deterministic vectors (δ, δ') under a given volatility model $\mathcal{X}$. This metric estimates the likelihood that δ' has been distorted to δ during remote measurement. Then, similarity between two observed systems (S_i, S_j) is given by the following expectation

$$p(\Delta_i|\Delta_j, \mathcal{X}) = \sum_{\delta}\sum_{\delta'} p(\delta|\delta', \mathcal{X}) p_i(\delta) p_j(\delta'). \tag{2}$$

For a given i, classifiers are typically concerned with finding j that produces the largest value in (2). However, we are facing a different problem that requires normalization. Let $\pi_i(\mathcal{X}) := \sum_{j=1}^{n} p(\Delta_i|\Delta_j, \mathcal{X})$ be the total similarity weight



(a) random features



0.45	0.19	0.36	0.00
0.05	0.49	0.44	0.01
0.12	0.12	0.72	0.04
0.07	0.05	0.00	0.88

0.52	0.47	0.01
0.13	0.82	0.04
0.05	0.00	0.95

0.95	0.05
0.00	1.00

(b) matrix reduction

Figure 3: Randomness of RTO features and elimination of duplicates in Plata.

of system S_i across all available options j. Depending on the classifier, π_i may not always be 1. To handle such cases, define

$$q(\Delta_i|\Delta_j, \mathcal{X}) = \frac{p(\Delta_i|\Delta_j, \mathcal{X})}{\pi_i(\mathcal{X})} \qquad (3)$$

to be the probability that S_i gets classified as S_j. Now suppose systems $S_1, \ldots, S_n$ are deployed in a production environment (e.g., wide-area Internet) and measured using remote probing. Therefore, instead of seeing Δ_i, the observer now samples $\Delta_i + \theta$, where random vector θ is driven by the same distortion model $\mathcal{X}$. We are thus interested in identifying the largest subset of $S_1, \ldots, S_n$ in which each system can be matched back to itself with probability at least $1 - \epsilon$ under noise $\mathcal{X}$, i.e., $E[q(\Delta_i + \theta|\Delta_i, \mathcal{X})] \geq 1 - \epsilon$.

4.2 Matrix Construction

We next describe our database-construction framework, which we call Plata. It starts by building a confusion matrix $M = (M_{ij})$, where each cell $M_{ij} = E[q(\Delta_i + \theta|\Delta_j, \mathcal{X})]$ and the expectation is taken over θ. In general, classification decisions and vectors θ may be available only as output of some algorithm. For example, the former might be a C4.5 decision tree and the latter may require simulations of a specific queuing discipline. In such cases, the only solution is to run Monte-Carlo simulations that repeatedly distort Δ_i, classify the resulting observations, and average the result to obtain an approximation to M_{ij}.

To this end, suppose we generate r vectors $\theta_1, \ldots, \theta_r$ by simulating $\mathcal{X}$. Using the PMF in (1), we obtain the same number of instances from random variable Δ_i, which we call $\delta_i^1, \ldots, \delta_i^r$. Then, the approximate matrix is given by

$$\tilde{M}_{ij} = \frac{1}{r} \sum_{m=1}^{r} q(\delta_i^m + \theta_m|\Delta_j, \mathcal{X}). \qquad (4)$$

Since this expands to

$$\tilde{M}_{ij} = \frac{1}{r} \sum_{m=1}^{r} \sum_{\delta'} q(\delta_i^m + \theta_m|\delta', \mathcal{X}) p_j(\delta'), \qquad (5)$$

the overhead of constructing $\tilde{M}$ is determined by the product of r, matrix size n^2, the number of unique values δ', and complexity of computing $p(\delta|\delta', \mathcal{X})$, which typically is a linear function of the combined vector length $|\delta| + |\delta'|$.

4.3 Separation

Once complete, the diagonal of $\tilde{M}$ contains the probability of self-classification under $\mathcal{X}$. The next task is to iteratively eliminate specimens that disperse a significant fraction of classification decisions to non-diagonal cells until the target $(1 - \epsilon)$-separability is achieved, i.e., all $\tilde{M}_{ii} \geq 1 - \epsilon$. At each step, Plata removes row k with the smallest diagonal value and redistributes its probability weights to the remaining systems. The naive approach is to re-run Monte-Carlo simulations and build a new matrix with dimension $(n-1) \times (n-1)$; however, this is extremely expensive, especially when r is orders of magnitude larger than n.

The second option is to infer the new weights using a model and build a sequence of approximations that produce a final matrix similar to that in the naive method. Consider row i that needs to partition $\tilde{M}_{ik}$, i.e., the probability to classify i as k, among the other columns. If we assume that in the absence of system k, classification decisions follow the remaining probabilities in row i, the likelihood to classify $\delta_i^m + \theta_m$ as $j \neq k$ now becomes $\tilde{M}_{ij}/(1 - \tilde{M}_{ik})$. Multiplying this by the weight being removed and adding to the current $\tilde{M}_{ij}$, we get the following transformation that keeps row sums invariant

$$\tilde{M}_{ij} = \tilde{M}_{ij} + \frac{\tilde{M}_{ij}}{1 - \tilde{M}_{ik}} \tilde{M}_{ik}. \qquad (6)$$

Note that if none of i's classifications went to system k, i.e., $\tilde{M}_{ik} = 0$, row i does not change. This process continues until all diagonal values are above $1 - \epsilon$. The remaining systems at that stage are added to the database and their number establishes the $(1 - \epsilon, \mathcal{X})$-dimension of the classifier. An example of this reduction process is shown in Figure 3(b), where the rows are sorted in ascending order of $\tilde{M}_{ii}$ for convenience of presentation. Setting $\epsilon = 0.2$, there are three rows that violate separability constraints. Since (S_1, S_2) are both similar to S_3, but none of them resembles S_4, intuition suggests the initial measurement may contain only two separable specimens. After removal of the first row, all diagonals receive a boost, but (S_2, S_3) are still inseparable. Another iteration produces the expected two vectors that match themselves with probability 0.95 or better.

Note that $1 - \epsilon$ can be used as a tuning parameter – larger values reduce the number of eventual vectors in the database, while smaller values preserve more, but at the risk of having more duplicates and poor classification accuracy. Although only $\tilde{M}_{ii}$ is compared against $1 - \epsilon$, the entire matrix needs to be recomputed after each iteration. This is necessary in order to properly distribute the weights of eliminated systems using (6). Thus, the complexity of each step is n^2, repeated $n - d$ times, where $d := |\mathcal{D}|$ is the size of the final database.

4.4 Labeling

Once database $\mathcal{D}$ is created, Plata needs to assign system-identifying labels to the available signatures. Assume a process that collects mappings from each S_i to the corresponding label l_i using some type of download (e.g., port-80 HTTP requests), oracle input, or other means, but possibly for a subset of the known specimens. Incomplete labeling may occur due to bandwidth constraints, obfuscation of certain systems by their administrators, and generic software names (e.g., apache) that fail to identify the underlying system. Since labels might be available for hosts that have been discarded during the matrix-separation step, we must again consider the entire set $S_1, \ldots, S_n$. To this end, we classify each known specimen using $\mathcal{D}$ and produce a set of clus-

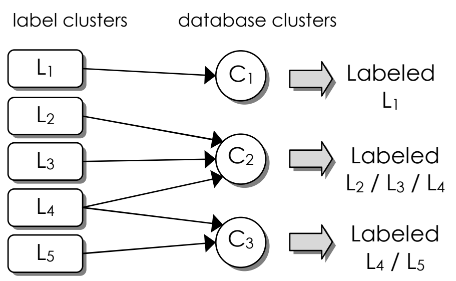


Figure 4: Applying labels to database clusters.

ters $C_1, \dots, C_d$, where d is the $(1-\epsilon, \mathcal{X})$-dimension of the database/classifier obtained earlier by Plata.

To eliminate duplicate labels, a separate procedure clusters them into multiple categories $L_1, L_2, \dots$ using some type of string-similarity matching. As shown in Figure 4, there is a directed edge between clusters L_k and C_j if there exists a system $S_i \in C_j$ such that its label $l_i \in L_k$. Note that this forms a bipartite graph in which L_k may point to multiple clusters C_j. Plata leaves the specifics of choosing the right label for each C_j to the application. One option is to combine the labels of all in-neighbors, as done in Figure 4. Another option is to assign weights to edges (e.g., equal to the number of corresponding S_i's) and enforce some minimum frequency before a label is considered valid. This can be further extended to allow for majority voting. For example, 100 hosts with label "Linux 2.4" and two with "Windows 7" mapping to C_j probably indicate the former is more appropriate than the latter.

5. OS FINGERPRINTING DATABASE

Plata is quite general and does not assume much beyond existence of similarity function p, algorithms to produce distortion θ, and ability to observe remote systems. We now apply this framework to one specific problem – OS stack fingerprinting under random/volatile features.

5.1 Classifier

As discussed earlier, the majority of stack fingerprinting tools treat all features as deterministic, in which case database construction is rather straightforward. The only exceptions are clock-skew methods [6], [13] and single-packet classifiers [3], [14], [25], [30]. The former direction has serious bandwidth overhead and suffers from inability to scale the database beyond a handful of hosts. The latter category, which is our focus, sends a single SYN packet to the target and observes a stream of SYN-ACKs, as illustrated in Figure 5. The vector of retransmission timeouts (RTOs) observed by the client forms the network features of the classifier. Combining unique RTO patterns with various fixed TCP/IP header fields, these algorithms can produce pretty robust OS identification.

The database for single-packet tools has evolved from 25 signatures in [3] to 98 in [14], eventually reaching 116 in [25], but the corresponding $(1-\epsilon, \mathcal{X})$-dimensions of the underlying classifiers remain unknown. So far, manual construction of $\mathcal{D}$ in these tools has relied on separation only across deterministic features (e.g., window size, TTL, RST bit) and never examined how to determine whether two hosts with the same fixed header values have sufficiently distinct

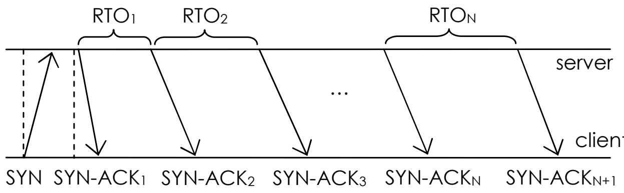


Figure 5: RTOs of half-open connections.

RTO vectors. To address this issue, we next apply Plata to temporal features of single-packet classifiers and build the first OS-fingerprinting database that is separable across random/volatile features.

5.2 Data Collection

We scan our campus network (three /16 blocks) on port 80 to obtain observations $\Delta_1, \dots, \Delta_n$ from responsive hosts $S_1, \dots, S_n$. Since each Δ_i may be random due to kernel-scheduling peculiarities, as in Figure 3(a), we persist in gathering $w = 50$ RTO vectors from each host, which is typically enough to capture whatever variation Δ_i may exhibit. Additionally, to exclude lossy vectors from being included in the database, the scanner continues until it receives w samples of the maximum length seen so far. Since packet loss in our network is low, quick convergence follows – the average number of SYN probes per responsive IP was 50.14.

As each S_i is a public server, care needs to be exercised to not overload the target with w back-to-back requests and cause unnecessary side-effects (e.g., rejected connections, CPU overload). However, as it turned out, even conservative 1-second inter-SYNs delays were too small. One such problem surfaced with certain printers, whose SYN-backlog queue [35] was smaller than w. When the queue was full, the printers terminated the oldest ongoing sequence of SYN-ACKs and started a new one. This caused the corresponding Δ_i to exhibit random truncation and presented difficulties in obtaining w loss-free observations. We eventually settled on delaying SYN probes by 240 seconds, i.e., double the TCP MSL (maximum segment life), which solved the problem.

The final caveat relates to OS kernel timing of RTOs. As speculated in [25], some hosts use a global timer that is independent of the SYN arrival time to generate SYN-ACKs for half-open connections. This causes the first RTO (and sometimes the remaining ones) to be randomized in some default range. In such cases, it is important to capture these effects in the database. We thus add random variable U to 240 seconds to avoid SYNs synchronization with any global clocks. Our U is uniform in $[0, 3]$ seconds, but other options are possible as well.

Along with the scan, a separate process opens a connection to each responsive host and attempts to download its root page over HTTP. This is known as *banner grabbing* – a general technique for discovering host type using some text-based protocol (e.g., telnet, finger, HTTP, FTP, and SMTP). This was used for OS fingerprinting in the 1990s, when Unix-based servers would readily volunteer their OS name and version. It has since fallen out of favor because these identification strings may be replaced by OS-oblivious names (e.g., Apache) or altogether removed, which makes the technique less reliable. However, banner grabbing works for our purpose since admins have no incentive to obfuscate

Grouping	RING	Snacktime	IRLsnack	Hershel
Deterministic only	28	209	209	344
Random only	23	52	50	117
Both	39	260	257	398

Table 1: Database dimensions.

OS names behind our campus firewall, and Plata only needs a subset of $S_1, \ldots, S_n$ to be labeled. This provides a fast, repeatable process that requires no manual intervention.

We receive SYN-ACKs from 9,879 IPs, assemble w loss-free RTO vectors from $n = 9{,}701$ hosts, and successfully complete a banner download from 9,594 of them.

5.3 Separating Features

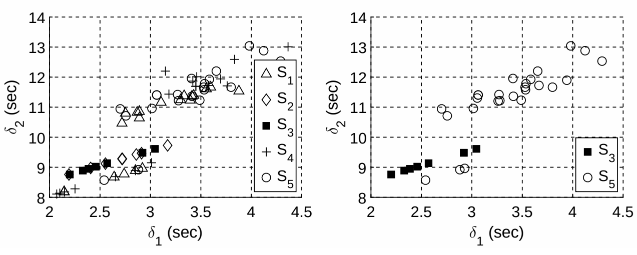
Single-packet OS-fingerprinting tools use both deterministic and random features. For each S_i, we move the former into vector u_i and the latter into Δ_i. In general, Hershel treats u_i as volatile, which means it allows users to change TCP/IP header values without making the OS fundamentally different. However, there is no even remotely accurate model for distortion $\mathcal{X}$ applied by users to these features. We therefore limit our efforts to the better-understood network delay jitter and its volatility. If a realistic noise model $\mathcal{X}$ becomes available for u_i, Plata can be used to compact duplicate hosts even further.

The simplest way to achieve separation on the deterministic features is to combine u_i with the size of RTO vector Δ_i. Splitting the available hosts $S_1, \ldots, S_n$ into clusters based on the deterministic pair $(u_i, |\Delta_i|)$ produces the first row of Table 1, with 28 signatures for RING [30], 209 for Snacktime [3] and IRLsnack [14], and 344 for Hershel [25]. Note that hosts within each cluster have same-length RTO vectors and our next goal is to further subdivide them into smaller groups that are $(1 - \epsilon, \mathcal{X})$-separable.

To decide on $\mathcal{X}$, assume the objective is to achieve sufficient accuracy during Internet-wide scanning, where each Δ_i is disturbed by random queueing delays along the path from the server back to the scanner. Due to constant SYN-ACK packet size, fixed transmission/propagation delays cancel out during RTO computation [25]. It is thus sufficient to use a FIFO-queue simulator that adds random delay jitter θ to each measurement, ensuring that no packets are reordered. As Hershel is fairly insensitive to the assumed model of jitter [25], we use exponentially distributed queueing delays with mean 500 ms, which results in θ being zero-mean Laplace. If better knowledge of network conditions is acquired, θ can be modified accordingly.

We generate r = 1K random noise vectors $\theta_1, ..., \theta_r$ and add them to each observation of Δ_i, resulting in wr = 50K disturbed samples per host S_i. We run Plata for each candidate classifier using their similarity function p and compute (5), in which $p_j(\delta') = 1/w$. This creates one matrix $\tilde{M}$ for each unique combination of deterministic features, which is fed to Plata's separation algorithm with $1 - \epsilon = 0.8$. After all matrices are compacted, we combine the surviving specimens into the final database $\mathcal{D}$.

Going back to Table 1, the second row shows that RTO features alone allow single-packet tools to differentiate between $23 - 117$ stacks under this combination $(\epsilon, \mathcal{X})$. Hershel more than doubles the dimension of its nearest competitor, which stems from its more sophisticated model for $p(\delta|\delta')$. Combining both deterministic and random features, Hershel



(a) candidate hosts (b) $(1 - \epsilon)$-separable hosts

Figure 6: Plata example.

ends up with 398 signatures, which is quite significant given the limited scope of the initial scan. Due to its higher accuracy and better separation ability, the rest of this paper stays with Hershel as the underlying classifier for Plata.

To demonstrate how matrix reduction works in practice, consider five actual Windows hosts in Figure 6(a) with $|\Delta_i| = 2$. While all of these OS kernels produce noisy RTOs, there are two distinct patterns. Figure 6(b) shows the result of Plata separation, which successfully extracts both patterns (Windows Server 2003 with two different service packs) out of the group and represents them using hosts (S_3, S_5).

5.4 Label Clustering

Note that Plata does not specify how to assign labels to clusters $\{L_j\}$. Besides ground-truth obtained from device owners, which may be infeasible for large decentralized networks, some of this information can be collected automatically. Our approach is to proceed along this route. Recall that HTTP headers contain the "Server:" string that sometimes identifies the version of the web server and uniquely ties it to a particular OS (e.g., Windows IIS). However, in other cases, the operating system can be inferred only from the HTML content of the page, as is the case with certain embedded devices (e.g., printers, cameras). We thus combine the "Server:" field with the entire HTML page and perform clustering using simhash [16], which is a well-known technique for detecting similar webpages. This creates 515 clusters $L_1, L_2, \ldots$, which we match to $d = 398$ Hershel signatures $C_1, \ldots, C_d$ using the procedure in Figure 4.

The final step is to perform manual verification of label sanity, determine which tags in the HTML to use (e.g., head, title), and convert low-level software versions to the corresponding OS name (e.g., IIS 7.5 to Windows Server 2008 R2). With enough coding effort to account for the various formats, most of this can be automated [29], but we found it easier to just show each page to a human and let them decide which of the found labels is appropriate. Plata does this by sequentially rendering one page from each L_k and recording the user's response. Even for $n \to \infty$, the number of unique clusters should remain reasonably small.

Results reveal that our label clustering works quite well – 326 out of 398 signatures (82%) receive a meaningful description. They are responsible for 98% of $n = 9{,}701$ measured hosts. Table 2 shows the top five most-popular signatures on our campus, where Plata successfully shrinks the most common Windows RTO pattern from 3,803 hosts down to 1. Heavy usage of Windows (43% of all servers) and Linux (12%) is no surprise, but we also find a large amount of HP LaserJet printers in fifth place. The $398 - 326 = 72$ unla-

Banner	Hosts	Deterministic features						Mean RTOs
		Win	TTL	DF	TCP options	MSS	RST	
Windows Vista / 7 / 8 / 2008 / 2012	3,803	8,192	128	1	MNWST	1,460	1, 0, 0, 1	3, 6, 12
Ubuntu / Debian / CentOS / Sci. Linux	822	5,792	64	1	MSTNW	1,460	0, 0 ,0, 0	4.3, 6, 12, 24.1, 48.2
Windows 2008 R2 / 2012	394	8,192	128	1	MNWST	1,460	0, 0, 0, 0	3, 6
Ubuntu / Redhat / CentOS / SUSE	366	14,480	64	1	MSTNW	1,460	0, 0, 0, 0	1.1, 2, 4, 8, 16
HP LaserJet Series	310	11,680	64	1	MNWNNT	1,460	0, 0, 0, 0	3, 6, 12

Table 2: Top 5 database signatures gathered from our campus scan (Win = window size, TTL = time to live, DF = do not fragment, MSS = max segment size, RST = reset packet features).

beled cases belong to network elements that either fail to provide a banner or supply one that contains no clue about the underlying OS. The latter case often happens with extremely rare devices for which we have only one banner to analyze. If Plata is exposed to additional data collection and user input (i.e., outside of our network), these gaps can be eliminated. The main benefit of our framework is that only a small fraction of n (i.e., 72/9701 = 0.7%) requires further attention.

Note that using automated banners for labeling does limit our ability to distinguish between OS versions. For example, the two Linux signatures in Table 2 are likely from different kernel versions. However, if the application requires more fine-granular labeling, additional effort – installing each OS in a test environment or contacting the owner – is needed in conjunction with Plata.

6. OPTIMIZING PLATA

While Plata works well, it bottlenecks on generating θ_m and recomputing $p(\delta_i^m + \theta_m|\delta', \mathcal{X})$ for each of the r random noise samples. This becomes especially noticeable in large groups, such as Windows with 3.8K hosts. Using 16 AMD Opteron cores @ 2.8 GHz and 64 GB of RAM, a parallelized C++ implementation requires over 24 hours to compute $\tilde{M}$. Although database creation is a one-time process, it is still desirable to have faster and more scalable algorithms that can tackle larger input. We address this next.

Analyzing (5), there are two obvious ways to reduce complexity – lowering r and making function $p(.)$ faster. However, for Hershel, we can attempt to do even better – replace Monte-Carlo simulations with a directly evaluated model that produces the expected probability that S_i gets classified as S_j under random noise θ. The rest of the section treats $\theta = (\theta_1, \theta_2, \ldots)$ as a vector consisting of scalar random variables, with respect to which all expectations are taken. Since $M_{ij} := E[p(\Delta_i + \theta|\Delta_j, \mathcal{X})]$ can be written as

$$\sum_{\delta}\sum_{\delta'} E[p(\delta+\theta|\delta', \mathcal{X})]p_i(\delta)p_j(\delta'), \tag{7}$$

construction of M in Plata requires only knowing $E[p(\delta + \theta|\delta', \mathcal{X})]$ for two deterministic, same-length vectors δ, δ'.

6.1 Closed-Form Plata-Hershel Matrix

To understand the results that follow, we briefly review how Hershel deals with delay jitter. Assuming $f(x)$ is the distribution (density or PMF) of one-way jitter and $e_m = \delta_m - \delta'_m$ is the error term in the m-th RTO, the similarity between two deterministic vectors is [25]

$$p(\delta|\delta', \mathcal{X}) = \prod_{m=1}^{|\delta|} f(e_m). \tag{8}$$

Note that (8) treats error values $(e_1, e_2, \ldots)$ as iid random observations. For the default model of $\mathcal{X}$, Hershel uses exponential one-way delay [25]. This produces Laplace jitter with density $f(x) = (\lambda/2)e^{-\lambda|x|}$, where parameter λ should conservatively reflect the amount of jitter anticipated in the network during actual measurement (i.e., $1/\lambda$ should upper-bound the real mean). With this in mind, our goal is to derive the following expectation

$$E[p(\delta+\theta|\delta', \mathcal{X})] = E\Big[\prod_{m=1}^{|\delta|} f(e_m + \theta_m)\Big], \tag{9}$$

where each θ_m is a random variable.

Given vectors δ and δ', we are interested in how similar Hershel considers them after the former undergoes random modification by the network. Suppose variables $(\theta_1, \theta_2, \ldots)$ are iid Laplace with rate μ. Note that μ may not equal λ if separation is performed for purposes other than future scanning of the Internet. In that case, μ may be set to match the environment in which $S_1, \ldots, S_n$ are probed (e.g., 5-ms average jitter for a campus network). Define $b_m = e^{-|e_m|}$ and consider the next result.

THEOREM 1. *For the Hershel classifier, the expected similarity between $\delta + \theta$ and δ' is*

$$E[p(\delta+\theta|\delta', \mathcal{X})] = \Big(\frac{\lambda\mu}{4}\Big)^{|\delta|} \prod_{m=1}^{|\delta|} \begin{cases} g_m & \lambda \neq \mu \\ h_m & \lambda = \mu \end{cases}, \tag{10}$$

where

$$g_m = \frac{2(\lambda b_m^{\mu} - \mu b_m^{\lambda})}{\lambda^2 - \mu^2}, \quad h_m = b_m^{\lambda}\Big(|e_m| + \frac{1}{\lambda}\Big). \tag{11}$$

The next logical step is to investigate whether matrix M built using (10) matches the Monte-Carlo version $\tilde{M}$. We consider a simple scenario with $|\delta| = 2$, $\delta = \delta'$, and $\lambda = \mu = 10$. This represents some diagonal cell M_{ii}, i.e., similarity score of S_i to itself, for a deterministic Δ_i. Setting $e_m = 0$ for all m, (10) produces 6.25, while Monte-Carlo simulations yield $\tilde{M}_{ii} = 6.7$. The error increases with RTO vector length and is more difficult to predict for off-diagonal values M_{ij}.

Further analysis uncovers that the source of this bias lies in Hershel's assumption on delay jitter. To illustrate this point, consider distorting a two-RTO vector δ using $\theta = (\theta_1, \theta_2)$. From the queuing model of [25], consecutive Laplace jitter values can be expressed using three iid exponential one-way delays X, Y, Z, i.e., $\theta_1 = Y - X$ and $\theta_2 = Z - Y$. While [25] is reasonable in arguing that X, Y, Z are independent due to the large gaps between SYN-ACKs, the same logic unfortunately does not apply to jitter because θ_1 and θ_2 share a common variable Y. For $e_m = 0$ and $|\delta| = 2$, the correct expectation of (9) is $E[f(\theta_1)f(\theta_2)]$. On the other hand, Theorem 1 uses Hershel to deduce the result

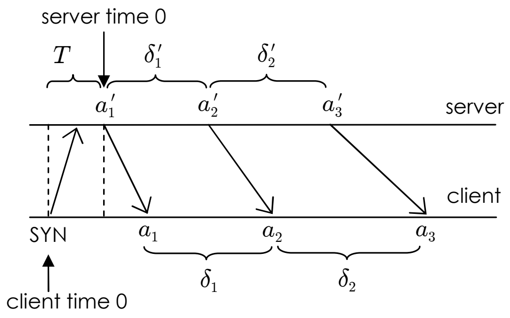


Figure 7: Features in Hershel (δ) and Hershel+ (a).

as $E[f(\theta_1)]E[f(\theta_2)] = \lambda^2/16$. We next expand the former term and show that it deviates from the latter for all λ.

THEOREM 2. *For $\mu = \lambda$ and $e_m = 0$, the expected Hershel similarity under dependent two-RTO jitter (θ_1, θ_2) is*

$$E[f(\theta_1)f(\theta_2)] = \frac{29\lambda^2}{432}. \tag{12}$$

Using $\lambda = 10$ in (12) produces 6.7 observed in simulations. While we succeeded in correctly modeling M_{ii} for two RTOs, doing the same for $i \neq j$ and longer vectors δ is very tedious.

6.2 Hershel+

We now show how the classification problem can be solved using only one-way delay (OWD). This requires a new model for $p(\delta|\delta', \mathcal{X})$ and additional constraints during creation of $\mathcal{D}$. For host S_i, define A_i to be a random vector of SYN-ACK transmission timestamps relative to the departure time of the first reply. Then, assuming that network delays are negligible, the distribution of elements inside A_i can be accurately obtained at the measurement client by subtracting the RTT of the first SYN-ACK from all observed values.

Now suppose that the scanner finds a remote host on the Internet and obtains a vector of SYN-ACK arrival instances as A, which are relative to the transmission time of the SYN. The main caveat here is that the forward SYN delay and server think time, which we collectively call T, are not just unknown in the public Internet, but also likely non-negligible. Consequently, the classifier must consider all options for T in its decision whether the observed A could have been produced by some known vector A_i. Delay randomness is handled similar to (7), which means that it is again sufficient to consider only deterministic pairs of delay vectors, i.e., by conditioning on $A = a = (a_1, a_2, \ldots)$ and $A_i = a' = (a_1', a_2', \ldots)$. This is illustrated in Figure 7. Supposing that Q_m is the m-th OWD from the server to the client, we have $a_m = T + a_m' + Q_m$.

With the new model, redefine the error as $e_m = a_m - a_m'$ and let $s = \min_m\{e_m\}$ be the largest possible value of T when a system equipped with a' is responsible for observation a. Then, the similarity function becomes

$$p(a|a', \mathcal{X}) = E\Big[\prod_{m=1}^{|a|} f_Q(e_m - T)\Big], \tag{13}$$

Distribution of OWD	Features used	Hershel	Hershel+
Pareto (mean = 0.5)	RTO only	22.1%	24.2%
Pareto (mean = 0.1)	RTO only	33.1%	33.3%
Uniform (mean = 0.5)	RTO only	21.7%	22.1%
Uniform (mean = 0.5)	All	99.9%	99.9%

Table 3: Accuracy on the Hershel database.

Distribution of T	Loss	Hershel	Hershel+
Exponential (mean = 0.1)	–	96.9%	97.6%
Pareto (mean = 0.1)	–	96.9%	97.4%
Pareto (mean = 0.1)	3.8%	95.2%	95.8%
Pareto (mean = 0.1)	10%	92.3%	92.9%

Table 4: Accuracy on the Plata database.

where $f_Q(x)$ is the density of OWD from model $\mathcal{X}$. Assuming $f_T(x)$ is the PDF of T, this leads to

$$p(a|a', \mathcal{X}) = \int_0^s \Big[\prod_{m=1}^{|a|} f_Q(e_m - x)\Big] f_T(x)dx. \tag{14}$$

We apply Hershel's exponential OWD with $f_Q(x) = \lambda e^{-\lambda x}$ and additionally represent T as a sum of two exponential variables (i.e., forward SYN delay and server think time), which leads to $f_T(x) = \nu^2 x e^{-\nu x}$, i.e., Erlang(2) distribution with some rate ν and mean $2/\nu$. The OWD classifier (14) is more complex than Hershel's as it requires numerical integration of a computationally expensive product of shifted density functions. Our next result shows that this can be avoided through additional derivations.

THEOREM 3. *The closed form for* (14) *is*

$$p(a|a', \mathcal{X}) = \mathbf{1}_{s\geq 0}\nu^2\lambda^{|a|}\psi \prod_{m=1}^{|a|} e^{-\lambda e_m}, \tag{15}$$

where $\mathbf{1}$ *is an indicator variable and*

$$\psi = \begin{cases} \frac{1-e^{-(\nu-\lambda|a|)s}(1+(\nu-\lambda|a|)s)}{(\nu-\lambda|a|)^2} & |a| \neq \frac{\nu}{\lambda} \\ \frac{s^2}{2} & |a| = \frac{\nu}{\lambda} \end{cases}. \tag{16}$$

Replacing Hershel's $p(\delta|\delta', \mathcal{X})$ with (15) and keeping the rest of the method unchanged gives rise to a technique we call Hershel+. Our next step is to verify that its accuracy is no worse than that of Hershel even when the assumed Erlang model for T, which uses $\nu = 4$ in all computation below, does not match the true distribution. To this end, we use the simulation setup from [25], where the only new parameter is T. In the first scenario, we keep T uniform in $[0, 1]$ seconds, maintain zero packet loss, and run both methods over Hershel's original database with 116 stacks. The result is shown in Table 3. As the new model only changes the RTO classifier, the most important comparison involves the first three rows of the table, which confirm superiority of Hershel+. In the second scenario, we fix the OWD to be uniform in $[0, 1]$ and use the larger Plata database. Table 4 shows that Hershel+ again edges out Hershel, despite its higher uncertainty related to T.

6.3 Closed-Form Plata-Hershel+ Matrix

Armed with the new classifier, we revisit the issue of obtaining a Plata matrix without Monte-Carlo simulations. To model $\mathcal{X}$, we disturb each A_i using a random OWD vector

$V = (V_1, V_2, \ldots)$, where all V_i are iid exponential with rate λ. We additionally apply noise to the forward SYN delay and server think time, which are collectively given by an Erlang(2) random variable W with rate ν. Note that we use λ and ν from Hershel+, although other options are possible.

Define matrix $H = (H_{ij})$ to consist of all pairwise Hershel+ similarities between the signatures in the database under distortion $V + W$. This requires computing

$$\zeta(a, a') := E[p(a + V + W|a', \mathcal{X})] \tag{17}$$

and setting $H_{ij} = E[\zeta(A_i, A_j)]$, where the second expectation is taken over random variables (A_i, A_j).

THEOREM 4. *Define* $v = (\lambda/2)^{|a|}\nu/4$. *Then,*

$$\zeta(a, a') = v \int_{-\infty}^{\infty} e^{-\lambda \sum_m |e_m + z|}(1 + \nu|z|)e^{-\nu|z|}dz. \tag{18}$$

Note that (18) can be computed by splitting the integral into $|a|+2$ regions such that $|z|$ and $|e_m + z|$ are conclusively resolved as being either positive or negative. Each of these smaller integrals expands in closed-form; however, due to the large number of terms involved and lacking structure, this result is difficult to represent symbolically. Algorithmically, however, this is simple to code using a bit-vector of size $|a|+1$ that keeps track of which of the terms $(z, e_1 + z, e_2 + z, \ldots)$ is positive. Moving from one interval to the next flips one bit from 0 to 1 and switches to the corresponding integral.

After verifying that (18) and its $|a| + 2$ sub-integrals produce correct results, we run Plata separation over H instead of $\tilde{M}$ and obtain 420 signatures, out of which 79 come out unlabeled. Recalling Table 1, notice that Hershel+ increases the dimension of its predecessor by 22 entries, indicating a more powerful classifier. Performance improvement is remarkable as well – the runtime reduces from over 24 hours to just 12 minutes. Added benefits include higher accuracy of Hershel+ decisions and alleviation of uncertainty if r is large enough to keep Monte-Carlo results convergent.

7. INTERNET SCAN

We now use Hershel+ to classify every visible webserver on the Internet against the previously constructed Plata database.

7.1 Classification Results

In July 2015, we sent 2.7B SYN probes on port 80 to every IP address advertised in BGP and obtained SYN-ACK responses from 66.4M hosts. This is almost double the 37M IPs used in the Hershel study [25]. The scan lasted 6 hours and operated at 125K packets per second.

Table 5 shows the Hershel+ output on the Internet data. We break down the result by OS category, showing the 5 most-popular signatures in each. Not surprisingly, Linux still dominates the webserver market. Although its top-5 signatures are separable at the feature level, limitations of our banner-based labeling do not allow identification of the specific version of these OSes. In second place, there is a large number of embedded devices, mostly routers and printers. This finding agrees with those in previous measurements at this scale [11], [25]. In third place, we combine hosts that map to a signature without a useful banner and those with a zero probability of matching to anything in $\mathcal{D}$. The former category is responsible for 94% of these cases, where 79 "mystery" signatures in $\mathcal{D}$ catch almost 12% of all Internet classification, despite being rare on our campus. Future work will attempt to uncover their OS.

Category	OS / Device	Hosts
Linux	Ubuntu / Redhat / CentOS	14,551,706
	Ubuntu / Redhat / SUSE	2,620,566
	Ubuntu / Debian / Redhat	2,381,733
	Ubuntu / CentOS / SUSE	1,831,519
	Ubuntu / Redhat / Sci. Linux	1,413,660
	Total in category	25,679,480
Embedded	3Com Routers	2,661,835
	Dell Laser / Xerox Printers	1,985,840
	Embedded Linux	1,869,053
	Cisco Embedded	1,699,418
	Citrix Netscaler	1,118,748
	Total in category	24,447,390
Unknown	No label in database	7,936,268
	Zero probability of match	474,585
	Total in category	8,410,853
Windows	Windows 7 / 8 / 2008 / 2012	2,186,229
	Windows XP / 2003	822,130
	Windows XP / 2000 / 2003	791,298
	Windows 2008 R2 / 2012	701,204
	Windows 2008 R2 / 2012	427,401
	Total in category	7,124,444
Other	FreeBSD	480,789
	FreeBSD	107,635
	Novell Netware	37,981
	Mac OSX Server	35,613
	Solaris 9 / Solaris 10	35,375
	Total in category	752,602

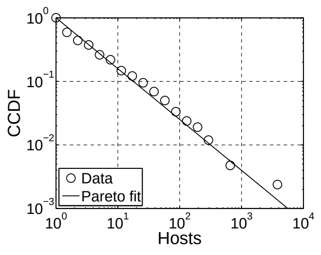
Table 5: OS classification of the Internet dataset.

Next, there is Windows in fourth place with 7M hosts. Unlike the previous categories, we can identify the specific type of Windows from its IIS version in the HTTP header. While it is by far the most popular desktop OS [19], its penetration of the webserver domain has been lagging behind Linux. This is in contrast to our campus scan, which was dominated by Windows. One explanation for Unix prevalence is migration of online services to enterprise clouds, which have traditionally favored Linux installations. Another is the possibility that Linux distributions more commonly enable a webserver in their default configurations or alias more IPs to the same physical server. And yet another is a higher percentage of Unix computers not being protected by a firewall (either corporate or host-level).

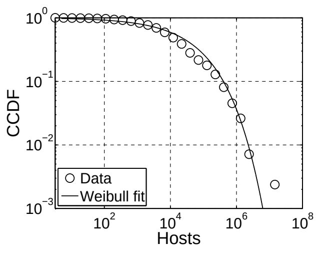
The table ends with 752K devices (1.1% of the total) in the "other" category that includes BSD, Mac, Novell, and Solaris. Compared to the previous large-scale fingerprinting effort [25] that used scans from July 2010, the table shows that Linux and embedded have doubled their numbers (i.e., from 13-14M to 25-26M), Windows remained pretty much unchanged (i.e., a slight drop from 7.5M to 7.1M), and the remaining group lost 68% of its membership (i.e., from 2.3M to 752K). In summary, 99.3% of all IPs are successfully classified and 87.3% have a label.

7.2 OS Popularity and Confidence

To better understand device deployment at different scales, we next examine the distribution of cluster size W for each of the 420 signatures in our database. Figure 8(a) shows the CCDF $P(W > x)$ using the initial campus scan. This plot is a close match to Pareto tail $(x/\beta)^{-\alpha}$, where $\alpha = 0.8$ and $\beta = 1$. Interestingly, the bottom 40% of the signatures map to a single host each. In contrast to the well-known stacks

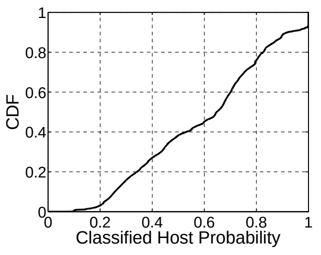


(a) campus

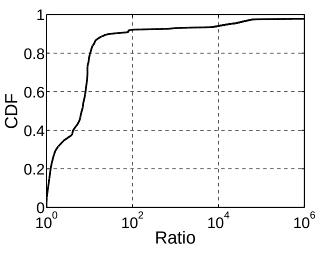


(b) Internet

Figure 8: OS popularity distributions.



(a) CDF of top $p(a|A_j, \mathcal{X})$



(b) ratio top two $p(a|A_j, \mathcal{X})$

Figure 9: Hershel+ classification confidence.

in Table 2, these clusters enjoy more esoteric items such as security cameras, room-temperature controllers, UPS (uninterruptable power supplies), tape backup, humidity sensors, and even discontinued oscilloscopes. Figure 8(b) plots the same tail for the Internet scan, which is a good match to the Weibull distribution $\exp(-(x/\lambda)^k)$, where $k = 0.4$ and $\lambda = 45$K. Each of the top-14 signatures accounts for at least 1M hosts and the top-17 are responsible for 60% of IPs. The bottom 204 signatures match a combined 1% of the servers (i.e., 664K).

Another interesting issue is the amount of confidence with which Hershel+ selects the best OS during classification. Assuming a is a measurement from some IP, (14)-(15) can be used to obtain similarity score $p(a|A_j, \mathcal{X})$ for each OS j, the highest of which is selected as the match after normalization. Figure 9(a) plots the distribution of this probability across all 66.4M IPs. Observe that almost no classifications occur with less than 20% likelihood and over half the hosts fit some signature with probability at least 65%. The far end of the CDF shows 7% of the IPs with a 100% match, which are devices with truly unique combinations of features. In the same vein, to determine if the second-best match follows closely the top signature and how often the classifier might be "guessing," Figure 9(b) shows the CDF for the ratio of the two highest probabilities. In 17% of the cases, the second-best match is pretty close, i.e., within a factor of 1.2. Afterwards, the curve sharply rises and yields over 68% of IPs with a decisive winner (i.e., ratio 2:1 or better).

8. COMPARISON WITH NMAP

Since ground-truth for millions of Internet hosts is difficult to obtain, we next perform comparison against Nmap v6.49 [21]. During the scan, we randomly selected 1% of responsive hosts and invoked Nmap to fingerprint them as soon as the first SYN-ACK was received. Real-time execution was needed to minimize the possibility they left the network and other hosts appeared in their place (e.g., due to DHCP). We used Nmap's least-verbose mode to limit the traffic and complaints from target networks; however, this still resulted in 80 sent and 60 received packets per IP, as well as several notifications to campus network administrators about intrusive activity coming from our subnet. The complaints identified Nmap by name, which raises questions how often IDS tools not just detect, but drop its traffic.

Category	Category match	String match	Total
Linux	301 (98.6%)	25 (8.1%)	305
Embedded	158 (75.5%)	34 (16.2%)	209
Windows	82 (95.3%)	82 (95.3%)	86
Other	3 (100%)	3 (100%)	3
Total	544 (90.2%)	144 (23.8%)	603

Table 6: Internet subsample classification.

Out of 664K IPs, Nmap was successful for only 481K (i.e., 72%). To rule out host departure, we verified that an overwhelming majority (99.8%) of the attempted IPs returned at least one reply to Nmap probes. The failed cases include responses unknown to the database and firewall obstruction of non-SYN packets. We uniformly subsampled these 481K IPs, excluded roughly 12% for which Hershel+ returned "unknown," and ended up with 603 cross-labeled samples for further manual analysis.

8.1 Agreement

We first investigate how well Nmap and Hershel+ agree on the classification of the selected subset of hosts. Comparison with Nmap is far from straightforward since its stack names are human-created and rather fine-granular. The most detailed category in our Plata database is Windows, while the majority of other hosts are marked with just the name of the OS and/or device. Thus, it makes sense to separately consider whether Hershel+ matches the exact signature string of Nmap or just the category.

Table 6 shows the result of this process, where we group hosts based on Hershel+ classification. In the category match, we achieve over 98% agreement in Linux, 95% in Windows, and 100% in "other." With embedded systems, Nmap often claims the host is running Linux, whereas we have a specific (non-Linux) device name from the banner. Without tedious manual effort, it is difficult to know if Nmap has been exposed to these devices and whether it can reliably identify them. With that said, we still mark these cases as a mismatch, which drops the agreement rate to 75%.

As for OS strings, lower numbers were expected due to the difference in how the databases are labeled. The biggest drop occurs in Linux, where our $\mathcal{D}$ consists of just distribution names (e.g., Ubuntu, Redhat, SUSE), while Nmap provides both major and minor kernel versions (e.g., Linux 2.6.18-22). Nevertheless, there are 25 matching signatures for which both methods can identify only the Linux family. For embedded systems, Nmap produces a large variety of device names, many absent from our campus. Finally, the Windows group keeps the same 95% consensus rate since all 82 agreed-upon cases are exact string matches.

8.2 Disagreement

We now analyze the peculiar case of the four Windows hosts from Table 6 where Nmap and Hershel+ disagree. We

Vector	Win	TTL	DF	TCP options	MSS	RST	SYN-ACK arrival (sec)	Label
D_1	8,192	128	1	MNWST	1,464	1,0,0,1	0.00, 2.99, 9.00, 21.00	Windows 7 / 2008 R2
S_1	8,192	128	1	MNWST	1,464	1,0,0,1	0.22, 3.22, 9.22, 21.22	
S_2	8,192	64	1	MNWST	1,460	0,0,0,0	0.18, 3.17, 9.17	
D_2	16,384	128	0	MNWNNTNNS	1,380	0,0,0,0	0.00, 2.65, 9.21	Windows 2000 / 2003
S_3	16,384	128	0	MNWNNTNNS	1,460	0,0,0,0	0.21, 2.67, 9.22	
S_4	16,384	128	0	MNWNNTNNS	1,370	0,0,0,0	0.21, 3.07, 9.63	

Table 7: Hershel+ classification and features.

Vector	R_1	F_{11} = Win	F_{12} = TTL	F_{13} = DF	F_{14} = TCP options	F_{15} = MSS	$(R_2, \ldots, R_{10})$	Label
D_1	1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	0000 111 00	iPXE 1.0.0+
D_2	1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	0000 111 01	Tomato 1.28
S_1	1	8,192	128	1	MNWST	1,464	0000 000 00	
D_3	1	$\emptyset$	64	1	MNNSNWNNT	1,460	1000 100 11	OpenBSD 4.3
S_2	1	8,192	64	1	MNWST	1,460	1000 100 11	
D_4	1	$\emptyset$	64	0	$\emptyset$	$\emptyset$	0000 111 01	D-Link DWL-624
S_3	1	16,384	128	0	MNWNNTNNS	1,460	0000 000 01	
S_4	1	16,384	128	0	MNWNNTNNS	1,370	0000 111 01	

Table 8: Nmap classification and features.

call these observations $S_1, \ldots, S_4$. Table 7 shows their features and the corresponding database signatures $D_1 - D_2$ for the Hershel+ classification. Notice that S_1 is an easy classification decision because the RTT is small (i.e., $\approx$ 220 ms) and D_1 matches all of its features. For S_2, Hershel+ prefers the same OS, overcoming a change in TTL/MSS and a loss of the RST packet at 21 sec. For the other two hosts, both matching to D_2, the only discrepancy is the MSS, which is a highly volatile field that depends on the MTU [25]. Judging from the OPT and RTO features, the accuracy of these decisions is not in doubt.

To explain the Nmap outcome for these IPs, we need to review its classification technique. Suppose vector $R = (R_1, \ldots, R_l)$ consists of indicator variables such that $R_i = 1$ iff probe i elicits a response from the network stack. We split R into several groups – a regular SYN to an open port (R_1), four TCP packets with extra flags (i.e., ECN, null, rainbow, ACK) to an open port ($R_2 - R_5$), three TCP packets to closed ports ($R_6 - R_8$), and UDP/ICMP probes ($R_9 - R_{10}$). For cases with $R_i = 1$, suppose F_{ij} records the j-th feature of that packet, where $F_{ij} = \emptyset$ indicates a missing optional header field. A combination of vector R and corresponding matrix F constitutes a signature $\Phi = (R, F)$.

Suppose a match in R_i carries weight w_i and that in feature F_{ij} some other weight w_{ij}. Then, Nmap computes similarity between an observation Φ and a signature $\Phi' = (R', F')$ from the database using the following

$$\frac{\sum_{i=1}^{l}(Y_i \mathbf{1}_{R_i=R'_i} w_i + R_i R'_i \sum_j Z_{ij} \mathbf{1}_{F_{ij}=F'_{ij}} w_{ij})}{\sum_{i=1}^{l}(Y_i w_i + R_i R'_i \sum_j Z_{ij} w_{ij})}, \qquad (19)$$

where variable $Z_{ij} = 1$ iff field j in packet i is non-empty in both the observation and database (i.e., $F_{ij} \neq \emptyset, F'_{ij} \neq \emptyset$) and $Y_i = R_i$ for $i \in [6, 8]$ and 1 otherwise. The last rule ignores closed-port tests unless Φ contains a response to them. All signatures Φ' with at least 0.85 similarity are reported as likely matches.

This algorithm has no provisions for packet loss, which makes it increasingly unreliable as more probes are blocked. The issue is compounded by the usage of large weights $w_i \gg w_{ij}$, which ensure that a mismatch in a feature carries little impact compared to that in the receipt/non-receipt of a packet. As a result, presence of firewalls skews the score towards signatures Φ' that originally had fewer responses, regardless of their F_{ij}. Empty features cause $Z_{ij} = 0$ to remove the corresponding weight w_{ij} from consideration, gravitating the classifier towards results with more frequent occurrence of $\emptyset$. Finally, if the target does not respond to a given closed-port test, i.e., $Y_i = 0$, the denominator no longer contains the associated weight w_i. This allows Nmap to match $R_i = 0$ and $R'_i = 1$ with no penalty for $6 \leq i \leq 8$.

Armed with this insight, consider in Table 8 the Nmap features of $S_1 - S_4$, as well as their best matches – a network boot card, modem jail-break firmware, a decade-old OpenBSD 4.3, and an ancient D-link switch – where S_1 scores over 85% with both D_1 and D_2. From the table, notice that Nmap sampled the same SYN features as Hershel+, meaning they contacted similar network stacks. For inexplicable reasons, the database allows $\emptyset$ for mandatory values (e.g., Win, TTL, DF), where all four entries $D_1 - D_4$ contain at least one such case.

Based on Table 8, it is pretty clear that Nmap decisions are heavily influenced by the R vector and empty fields. Indeed, iPXE/Tomato have no features F_{ij} in common with S_1, OpenBSD 4.3 matches S_2 only in three fairly generic fields TTL/DF/MSS, and D-Link agrees with S_3/S_4 in just the DF bit. We thus find no evidence to suggest that Nmap signatures $D_1 - D_4$ are statistically probable, let alone better than the Hershel+ result in Table 7. In fact, D_3 and S_2 are conclusively different stacks judging from their ordering of non-NOP TCP options (i.e., MSWT vs MWST).

From a broader perspective, Table 9 shows the number of hosts for which Nmap decides that $D_1 - D_4$ exceed the 85% threshold. Remarkably, Tomato appears in 21% of the cases and OpenBSD in 13%. These results raise questions about Nmap's ability to provide meaningful classification, not just in the four cases we dissected, but generally in wide-area networks, where R is easily distorted by IDS, host-level packet filters, and network firewalls.

9. DISCUSSION AND CONCLUSION

Network stack fingerprinting has well-known pitfalls (e.g., scrubbers [7], [24], [22], [27], [31], traffic intercepts by middleboxes [12], load-balancers, RST injection by IDS), but nevertheless it is fascinating that a single SYN packet can

Signature	Subsample	Total
Tomato 1.28	132 (21.8%)	105,525 (21.9%)
OpenBSD 4.3	91 (15.0%)	64,050 (13.3%)
D-Link DWL-624	12 (1.9%)	6,454 (1.3%)
iPXE 1.0.0+	6 (0.9%)	5,723 (1.1%)

Table 9: Popularity of Nmap signatures.

elicit so much information about the target. With our algorithm for automated construction of databases and robust classification (i.e., Plata, Hershel+), single-packet tools may eventually become a legitimate competitor to Nmap for use over the public Internet. However, despite the recent developments in this field, there are still many open problems and avenues for improvement, which we discuss next.

One question is whether distortion $\mathcal{X}$ can include packet loss. This seems like a viable direction, which Plata can handle transparently in the Monte-Carlo version; however, deriving a Hershel+ matrix in closed-form requires additional research. Another avenue that is worth investigating is whether Hershel+ can increase accuracy by abandoning the single-packet assumption and sending multiple SYNs to each target IP. In these cases, it should be compared to other tools with retransmission, including their $(1-\epsilon, \mathcal{X})$-dimension under a common model of distortion. For Nmap, $\mathcal{X}$ may include blocking of ICMP/UDP packets to match the firewall assumptions of Hershel+, loss of SYN-ACKs, censorship of certain invalid flag combinations known to IDS, emulation of load-balancers, and presence of fingerprint scrubbers [24]. All of this requires a kernel-level driver that can intercept Nmap packets and reproduce the desired conditions, which is yet another direction for future work.

10. REFERENCES

[1] H. J. Abdelnur, R. State, and O. Festor, “Advanced Network Fingerprinting,” in *Proc. RAID*, Sep. 2008, pp. 372–389.

[2] P. Auffret, “SinFP, Unification of Active and Passive Operating System Fingerprinting,” *Journal in Computer Virology*, vol. 6, no. 3, pp. 197–205, Nov. 2010.

[3] T. Beardsley, “Snacktime: A Perl Solution for Remote OS Fingerprinting,” Jun. 2003. [Online]. Available: http://www.planb-security.net/wp/snacktime.html.

[4] R. Beverly and A. Berger, “Server Siblings: Identifying Shared IPv4/IPv6 Infrastructure via Active Fingerprinting,” in *Proc. PAM*, Mar. 2015, pp. 149–161.

[5] J. Caballero, S. Venkataraman, P. Poosankam, M. G. Kang, D. Song, and A. Blum, “FiG: Automatic Fingerprint Generation,” in *Proc. NDSS*, Feb. 2007, pp. 27–42.

[6] Y.-C. Chen, Y. Liao, M. Baldi, S.-J. Lee, and L. Qiu, “OS Fingerprinting and Tethering Detection in Mobile Networks,” in *Proc. ACM IMC*, 2014, pp. 173–180.

[7] A. Crenshaw, “OSfuscate,” 2008. [Online]. Available: http://www.irongeek.com/i.php?page=security/code.

[8] R. Ensafi, D. Fifield, P. Winter, N. Feamster, N. Weaver, and V. Paxson, “Examining How the Great Firewall Discovers Hidden Circumvention Servers,” in *Proc. ACM IMC*, Oct. 2015, pp. 445–458.

[9] J. Erman, M. Arlitt, and A. Mahanti, “Traffic Classification Using Clustering Algorithms,” in *Proc. ACM SIGCOMM MineNet*, Sep. 2006, pp. 281–286.

[10] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software: An update,” *SIGKDD Explorations*, vol. 11, pp. 10–18, Jul. 2009.

[11] J. Heidemann, Y. Pradkin, R. Govindan, C. Papadopoulos, G. Bartlett, and J. Bannister, “Census and Survey of the Visible Internet,” in *Proc. ACM IMC*, Oct. 2008, pp. 169–182.

[12] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, “Is It Still Possible to Extend TCP?” in *Proc. ACM IMC*, Nov. 2011, pp. 181–194.

[13] T. Kohno, A. Broido, and K. C. Claffy, “Remote physical device fingerprinting,” *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 93–108, May 2005.

[14] D. Leonard and D. Loguinov, “Demystifying Service Discovery: Implementing an Internet-Wide Scanner,” in *Proc. ACM IMC*, Nov. 2010, pp. 109–122.

[15] M. Luckie, R. Beverly, T. Wu, and M. Allman, “Resilience of Deployed TCP to Blind Attacks,” in *Proc. ACM IMC*, Oct. 2015, pp. 13–26.

[16] G. S. Manku, A. Jain, and A. D. Sarma, “Detecting Near Duplicates for Web Crawling,” in *Proc. WWW*, May 2007, pp. 141–149.

[17] C. McNab, *Network Security Assessment: Know Your Network*. O’Reilly Media, Inc., 2007.

[18] J. P. Medeiros, A. Brito, and P. M. Pires, “An Effective TCP/IP Fingerprinting Technique Based on Strange Attractors Classification,” in *Proc. Data Privacy Management and Autonomous Spontaneus Security*, Sep. 2009, pp. 208–221.

[19] NetApplications, “Market Share Statistics for Internet Technologies.” [Online]. Available: http://netmarketshare.com/.

[20] Netcraft Web Server Survey. [Online]. Available: http://news.netcraft.com/.

[21] Nmap. [Online]. Available: http://nmap.org/.

[22] G. Prigent, F. Vichot, and F. Harrouet, “IpMorph: fingerprinting spoofing unification,” *Journal in Computer Virology*, vol. 6, no. 4, pp. 329–342, Nov. 2010.

[23] D. Richardson, S. Gribble, and T. Kohno, “The Limits of Automatic OS Fingerprint Generation,” in *Proc. ACM AISec*, Oct 2010, pp. 24–34.

[24] G. Roualland and J.-M. Saffroy, “IP Personality.” [Online]. Available: http://ippersonality.sourceforge.net/.

[25] Z. Shamsi, A. Nandwani, D. Leonard, and D. Loguinov, “Hershel: Single-Packet OS Fingerprinting,” in *Proc. ACM SIGMETRICS*, Jun. 2014, pp. 195–206.

[26] B. Skaggs, B. Blackburn, G. Manes, and S. Shenoi, “Network Vulnerability Analysis,” in *Proc. IEEE MWSCAS*, Aug. 2002, pp. 493–495.

[27] M. Smart, G. R. Malan, and F. Jahanian, “Defeating TCP/IP Stack Fingerprinting,” in *Proc. USENIX Security*, Jun. 2000, pp. 229–240.

[28] G. Taleck, “SYNSCAN: Towards Complete TCP/IP Fingerprinting,” *CanSecWest*, Apr. 2004.

[29] THC-RUT Fingerprint Database. [Online]. Available: https://www.thc.org/thc-rut/thcrut-os-fingerprints.

[30] F. Veysset, O. Courtay, O. Heen, and I. R. Team, “New Tool and Technique for Remote Operating System Fingerprinting,” Apr. 2002. [Online]. Available: http://www.ouah.org/ring-full-paper.pdf.

[31] K. Wang, “Frustrating OS Fingerprinting with Morph,” 2004. [Online]. Available: http://hackerpoetry.com/images/defcon-12/dc-12-presentations/Wang/dc-12-wang.pdf.

[32] F. V. Yarochkin, O. Arkin, M. Kydyraliev, S.-Y. Dai, Y. Huang, and S.-Y. Kuo, “Xprobe2++: Low Volume Remote Network Information Gathering Tool,” in *Proc. IEEE/IFIP DSN*, Jun. 2009, pp. 205–210.

[33] M. Zalewski, “p0f v3: Passive Fingerprinter,” 2012. [Online]. Available: http://lcamtuf.coredump.cx/p0f3.

[34] S. Zander, T. Nguyen, and G. Armitage, “Automated Traffic Classification and Application Identification Using Machine Learning,” in *Proc. IEEE LCN*, Nov. 2005, pp. 250–257.

[35] X. Zhang, J. Knockel, and J. Crandall, “Original SYN: Finding Machines Hidden Behind Firewalls,” in *Proc. IEEE INFOCOM*, Apr. 2015, pp. 720–728.