Faulds: A Non-Parametric Iterative Classifier for Internet-Wide OS Fingerprinting

Zain Shamsi, Daren B.H. Cline, and Dmitri Loguinov

***Abstract*—Recent work in OS fingerprinting [45], [46] has focused on overcoming random distortion in network and user features during Internet-scale SYN scans. These classification techniques work under an assumption that all parameters of the profiled network are known a-priori – the likelihood of packet loss, the popularity of each OS, the distribution of network delay, and the probability of user modification to each default TCP/IP header value. However, it is currently unclear how to obtain realistic versions of these parameters for the public Internet and/or customize them to a particular network being analyzed. To address this issue, we derive a non-parametric Expectation-Maximization (EM) estimator, which we call *Faulds*, for the unknown distributions involved in single-probe OS fingerprinting and demonstrate its significantly higher robustness to noise compared to methods in prior work. We apply Faulds to a new scan of 67M webservers and discuss its findings.**

I. Introduction

The Internet is a fascinating conglomerate of highly heterogeneous devices, which differ in hardware capability, security awareness, software features, and daily usage. Measuring the amount, type, and behavior of these devices, as well the networks they connect to, has become an important topic [14], [16], [18], [21], [27], [30], [36], [45], [46]. To categorize the makeup of today's networks, research in *active OS fingerprinting*, which is our topic in this paper, aims to determine the stack of remote hosts using their responses to external stimuli (i.e., TCP/IP probes) [4], [5], [7], [10], [19], [25], [26], [31], [33], [39], [44], [47], [51], [52], [55], [56], [57]. In addition to uncovering the operating system of computers, fingerprinting can expose household items (e.g., printers, cameras, TVs) and various cyber-physical systems (e.g., temperature monitors, lighting controllers), which are classes of devices that have enjoyed increased exploitation in recent years.

There are many uses for remote stack fingerprinting. First, it helps hackers in identification of vulnerable hosts and general network reconnaissance [50], especially during cyber-attacks that target only a specific OS implementation [22]. Second, OS fingerprinting is routinely deployed in security, e.g., by administrators of large networks seeking to find unpatched hosts and rogue entities [1], [32], [48]. Third, perimeter-defense systems (e.g., IDS, firewalls) may require the OS of the target host in order to detect certain types of exploits (e.g., those involving reassembly of IP fragments). In such cases, autonomous fingerprinting of the protected network allows these installations to function at maximum effectiveness [47], [51]. Finally, researchers/organizations use these techniques to understand usage trends [37], [38], discover the spread of new technologies [8], [17], [29], [41], and expose botnets [28].

Active stack fingerprinting can be partitioned into three categories – *banner-grabbing* via plain-text protocols (e.g., telnet, HTTP, FTP), *multi-probe* tools that elicit OS-specific responses from various non-standardized combinations of flags and/or unexpected usage of protocol fields (e.g., nmap [39], xprobe [55], p0f [57]), and *single-probe* methods that send a regular SYN to each host (e.g., Snacktime [6], RING [53], Hershel [46], Hershel+ [45]).

At large scale, banner-grabbing has several impediments – frequent removal of OS-identifying strings by administrators (e.g., for security purposes), high bandwidth overhead, and common interaction with non-platform-specific software (e.g., apache, nginx). Multi-probe tools have their own challenges – heavy load on the target, massive complaints about intrusive activity, and noisy results when the destination IP is load-balanced across a server farm (i.e., each packet hits a different machine). More importantly, the accuracy of multi-packet tools suffers a significant degradation when firewalls block auxiliary probes (e.g., a UDP to a closed port, rainbow flags in TCP headers, ICMP port unreachable) and the underlying classifier is not robust against unexpected feature removal/modification. As shown in [45], OS classification with nmap over the public Internet fails in almost 30% of the cases. Furthermore, nmap sometimes produces nonsensical results and worse accuracy than the alternatives utilizing a single probe.

Before modeling and improving multi-packet classifiers, which are still poorly understood, it is important to ask whether there exists a set of algorithms for maximizing performance of single-packet tools in Internet-wide scans. Such techniques provide a maximally stealthy option and may be able to bypass firewalls/IDS when packets loaded with "tricks" cannot. As it turns out, even the most advanced model in single-probe literature, i.e., Hershel+ [45], leaves room for improvement. It has many built-in assumptions that may be violated in practice, which in turn may affect its classification accuracy and overall performance on such basic metrics as the fraction of the Internet running a particular stack. Our motivation for this paper is to understand the limitations of existing single-probe techniques and offer novel avenues for increasing both the classification accuracy and amount of information recovered from responses to a SYN packet.

A. Overview of Results

Assume a database of known fingerprints $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and an observation $\mathbf{x}' = (\mathbf{x}'_1, \ldots, \mathbf{x}'_m)$ from a large number

A shorter version of this paper appeared in ACM CCS 2017.

All authors are with Texas A&M University, College Station, TX 77843 (zain@cse.tamu.edu, dcline@stat.tamu.edu, dmitri@cse.tamu.edu).

of Internet hosts. Suppose vector $\alpha = (\alpha_1, \ldots, \alpha_n)$ specifies the distribution of *popularity* among the known OSes, i.e., α_i is the fraction of hosts using fingerprint $\mathbf{x}_i$. Deciding which OS generated each $\mathbf{x}'_j$ is generally hindered by presence of *distortion* during observation, which adds random delays to packets, drops some of them, and modifies header fields.

Hershel+ relies on a-priori knowledge of not only α, but also additional parameters θ of distortion – the probability of change in each TCP/IP feature and distributions of network delay, packet loss, and server think time. While the underlying model in Hershel+ is more robust to distortion than those in prior approaches [6], [53], its performance does depend on how well α and θ can be estimated ahead of time. Unfortunately, extraction of these parameters from prior Internet scans and Hershel+ decisions is far from simple. In fact, using the fraction of previous classifications that went to OS i as a substitute for α_i may lead to inferior results compared to staying with the default parameters [46].

As the Internet is highly heterogeneous and constantly evolving, even if (α, θ) could be estimated by monitoring routers and/or using end-to-end measurement between strategically positioned hosts (e.g., PlanetLab), it is unclear whether conditions observed in the past or along certain paths can yield meaningful predictions about the specific network being fingerprinted (e.g., a corporate LAN is very different from the public Internet). Instead, we argue that (α, θ) should be the *output* of the classifier rather than the *input*. Doing so allows the unknown parameters to be customized to a specific observation $\mathbf{x}'$, i.e., reflect the OS composition of the network being analyzed and its distortion properties.

To accomplish this objective, we derive a non-parametric estimator for (α, θ) in Hershel+ under the theoretical framework of Expectation-Maximization (EM) [13], [20]. We call this approach Faulds and show that its iterative refinement of unknown distributions, followed by reclassification of $\mathbf{x}'$, can significantly improve the accuracy of Hershel+. Additionally, as the algorithm recovers both (α, θ), it provides important network characterization results for OS popularity, as well as distributions of delay, header-modification probabilities, and packet loss experienced by $\mathbf{x}'$.

We perform a fresh Internet scan and show new EM-guided classification decisions of Faulds. We not only update the OS-popularity vector α, which demonstrates non-trivial changes compared to Hershel+, but also expose statistical parameters of distortion θ observed by Faulds from 63M webservers.

II. Background

A. Nmap

Perhaps the most popular and exhaustive tool for OS fingerprinting is nmap [39]. To understand its infeasibility for wide-area usage, we briefly review its outgoing traffic and response requirements, as well as the matching algorithm. By default, nmap starts with a vertical scan of the target using $1{,}000$ well-known ports in an attempt to find two TCP ports, one of which is open and the other is closed, as well as a closed UDP port. It then sends 16 uniquely crafted probes – six regular TCP packets to an open port, one valid and one invalid ICMP ping, one UDP packet to a closed port, four malformed packets to an open TCP port, and three TCP packets to a closed port. It retransmits all probes multiple times to neutralize the impact of packet loss, which results in over 100 packets per host in addition to the initial port scan.

Besides overhead, running nmap against the entire Internet poses a number of additional challenges. First, there is a low likelihood that a port scan, combined with probes to closed ports, gets unnoticed by the IDS. Many software packages (e.g., snort) contain explicit rules to detect and block the rather esoteric nmap traffic. Certain networks take offense at being nmapped, which results in swift action to block the entire subnet/AS of the scanner and complaints about abusive behavior. Second, the firewall may allow select ports to reach the target host (e.g., port 80 to a webserver); however, there is little incentive to pass UDP or TCP packets to other ports that do not offer any services. Third, in similar fashion, the OS firewall can be configured (e.g., using domain group policy) to silently drop incoming packets to closed ports. In fact, Windows and Mac OS X suppress outgoing ICMP port unreachables *even when an explicit rule is created to allow such packets through the firewall* [3], [35].

Nmap expects responses to not deviate from those specified in the database (e.g., a RST to a TCP rainbow packet, ICMP port unreachable from a closed UDP port, ICMP echo reply to a ping). Because it considers absence of a response to be a feature, it can be misled into assigning large positive weights to firewall actions, which skews the result towards network stacks that inherently respond with fewer signals. This may occur despite a complete non-match in other features, meaning that the target may share nothing in the packet header (e.g., TCP window size, TTL, options, MSS) with the signature it is matched to [45]. Other issues include the database itself, which contains signatures that are subsets of others from completely unrelated stacks and allows special *null* header fields that can match any value in the observation. Unless the target responds to all 16 probes exactly as expected, an obscure device with the most null fields can trump the other alternatives, including the correct signature.

Additionally, certain TCP fields are quite *volatile*, i.e., change from user tweaks, underlying network MTU, and software setsockopt function calls. This does not inherently change the operating system, but creates an illusion of a different stack. For example, Server 2008 R2 accepts incoming connections with a kernel buffer (i.e., TCP window size) of $8{,}192$ bytes; however, an apache webserver can reconfigure this field to an arbitrary value before listening on the socket. Furthermore, this can be done on a per-socket basis and may vary over time depending on memory usage or other considerations. When faced with this type of uncertainty, nmap uses heuristic weights and thresholds that do not have rigorous theory/verification behind them. As a result, it exhibits highly unreliable identification in certain scenarios [45].

B. Single-Packet Tools

For accurate OS fingerprinting at Internet scale, low-overhead methods resilient to volatility are preferred. Our

TABLE I
FEATURE VECTORS $\mathbf{x}_i$ (TCP OPTIONS: M = MSS, N = NOP, S = SACK, T = TIMESTAMP, AND W = WINDOW SCALE)

OS name	Win	TTL	DF	OPT	MSS	RST	RTOs
Linux 3.2	5,792	64	1	MSTNW	1,460	0, 0, 0, 0	3, 6, 12, 24.2, 48.2
Windows 2003	16,384	128	0	MNWNNTNNS	1,380	0, 0, 0, 0	3, 6.5
Novell	6,144	128	1	MNWSNN	1,460	1, 1, 0, 1	1.4, 3.0

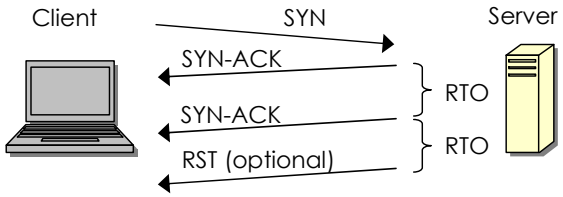


Fig. 1. Half-open connections in TCP.

focus in this paper is on single-probe techniques, which generally work by sending a TCP SYN to the target host and inducing a stream of SYN-ACK responses, possibly with a RST at the end. Since the connection is kept in the half-open state, the server continues retransmitting SYN-ACKs until its internal maximum-retry threshold is exceeded. Delays between the SYN-ACKs, known as *retransmission timeouts* (RTOs), as well as their count and presence of the last RST, reveal valuable information about the OS of the responding host. This is illustrated in Fig. 1. Coupling the RTOs with default TCP/IP header values makes stack classification possible.

The main difference between prior work [6], [27], [45], [46], [53] lies in the features they extract from TCP/IP headers and the assumed distortion model. As of this writing, Hershel+ [45] is both the most recent effort in this direction and most robust to observation noise. We review its operation and formulas later in the paper.

C. Other Techniques

Besides exploiting application-layer software (e.g., openSSL), cyber-attacks frequently target bugs in the OS kernel. In response to this, researchers have been developing methods to find and quantify unpatched systems [15], [41]. Services such as Shodan [30] and Censys [14] scan the Internet, parse the downloaded banners (e.g., HTTP "Server:" strings), and allow keyword search among the banners of responding hosts. Another direction of research has tried to identify industrial control devices by attempting to complete handshakes using various SCADA protocols [18], [36]. The effectiveness of identifying and attacking such systems was illustrated by the famous Stuxnet worm in 2010.

A related area to OS fingerprinting attempts to automatically discover features that can differentiate network stacks from each other [2], [9], [42] and build separable (i.e., maximal and non-redundant) databases from production systems [45]. Additional approaches, many of which are now part of nmap, include usage of TCP initial sequence numbers [33], [56], scraping of various fixed header fields [5], [7], [19], [52], [55], [57], tests for reassembly of IP fragments [47], [51], and reliance on clock-skew differences in kernels [10], [25]. Other protocols can be used in fingerprinting as well, e.g., HTTP [44], ICMP [4], [55], DNS [31], and DHCP [26].

III. LEARNING FROM OBSERVATION

A. General Problem

Suppose the OS database consists of $n \geq 1$ known stacks $(\omega_1, \ldots, \omega_n)$, each with some vector-valued *fingerprint* $\mathbf{x}_i = (x_{i1},\ x_{i2}, \ldots)$. As shown in Table I, fingerprints contain a combination of *features*, including default header values used for new connections and SYN-ACK retransmission timeouts (RTOs) of each OS. Further assume a set of observations $\mathbf{x}' = (\mathbf{x}'_1, \ldots, \mathbf{x}'_m)$ obtained by scanning the Internet and eliciting responses from m live servers, where $\mathbf{x}'_j = (x'_{j1}, x'_{j2}, \ldots)$ is a vector of sampled features from host j. For the type of OS fingerprinting considered here, i.e., single-probe, this is done by dispatching a SYN to every IP address in BGP and collecting SYN-ACKs/RSTs from the responding servers, as previously shown in Fig. 1.

The goal of the classifier is then to determine for each $\mathbf{x}'_j$ the most-likely fingerprint in the database. This task is complicated by the presence of distortion (also called *volatility* [45]) θ that randomly modifies the original features of the system before the observer gets them. This may involve a change in the temporal relationship between the packets (e.g., queuing delays), removal of some features (e.g., loss of RST packets), and rewriting of TCP headers in an effort to optimize or obscure the end-system.

Define $\alpha_i = p(\omega_i)$ to be the unknown fraction of hosts in $\mathbf{x}'$ with OS i and let $\alpha = (\alpha_1, \ldots, \alpha_n)$ be the corresponding vector. Now suppose $p(\mathbf{y}|\omega_i, \theta)$ is the probability that the fingerprint of signature i has been changed into $\mathbf{y}$ under θ. Similarly, assume that $p(\omega_i|\mathbf{y}, \theta, \alpha)$ is the probability that an observed vector $\mathbf{y}$ was produced by a host running OS i, conditioned on distortion model θ and popularity α. Then, application of Bayes' rule shows that the classifier must determine for each j the one database entry ω_i with the largest

$$p(\omega_i|\mathbf{x}'_j, \theta, \alpha) = \frac{\alpha_i p(\mathbf{x}'_j|\omega_i, \theta)}{p(\mathbf{x}'_j|\theta, \alpha)}, \tag{1}$$

where, for any vector of features $\mathbf{y}$, the denominator is

$$p(\mathbf{y}|\theta, \alpha) = \sum_{i=1}^{n} \alpha_i p(\mathbf{y}|\omega_i, \theta). \tag{2}$$

Analysis of (1) in existing work [45], [46] assumes that α is uniform (i.e., $\alpha_i = 1/n$) and θ is fixed by oracle input. Therefore, both α_i and denominator $p(\mathbf{x}'_j|\theta, \alpha)$ are independent of i and can be removed from the optimization, leaving only $p(\mathbf{x}'_j|\omega_i, \theta)$. In contrast, our goal here is to estimate both α and θ dynamically as the classifier is running, which

should both increase its accuracy and yield interesting Internet-characterization results as byproduct of classification. Before reaching this objective, a gradual build-up of formalization is needed. This section deals with estimating α, the next one covers network distortion, and the one after that focuses on modification to fixed header features.

B. Fingerprint Popularity

Observation vector $\mathbf{x}'$ gives rise to a number of equations in the form of (2), where the left side contains the empirical (known) probability of observing each unique vector $\mathbf{y} \in \mathbf{x}'$ and the right side is a model that embeds the unknown parameters. Extraction of α and θ from such systems of equations commonly involves the Expectation-Maximization (EM) method, which produces a solution using fixed-point iteration [13], [20]. At every step t, it maximizes the expected log-likelihood function conditioned on the parameters obtained during the previous iteration $t-1$. As long as the number of equations exceeds the number of unknown parameters, EM works well for many problems in practice.

For now, we treat $p(\mathbf{x}'_j|\omega_i, \theta)$ as a black-box classifier (e.g., Snacktime, Hershel, Hershel+), which does not attempt to estimate θ, and focus on determining α. This is the simplest (and only) case where (2) forms a linear system of equations, i.e., $p(\mathbf{y}_k|\theta, \alpha) = \sum_{i=1}^{n} \alpha_i c_{ik}(\theta)$, where all $c_{ik}(\theta)$ are constants. Throughout the paper, superscripts applied to parameters refer to the iteration number during which they are estimated, e.g., α_i^t approximates α_i during step t. Now notice that a sensible estimate of popularity for OS i is the average probability with which observations map to this fingerprint, conditioned on the previous estimate of popularity, i.e.,

$$\alpha_i^{t+1} = \frac{1}{m} \sum_{j=1}^{m} p(\omega_i|\mathbf{x}'_j, \theta, \alpha^t). \tag{3}$$

While the next result is fairly straightforward, its derivation methodology is needed for later parts of the paper.

Theorem 1: For a classifier with fixed θ, (3) represents the EM algorithm for recovering the popularity vector α.

Proof: For a given set of observations $\mathbf{x}' = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$, define the *likelihood function of* α *with respect to observation* $\mathbf{x}'$ as

$$p(\mathbf{x}'|\theta, \alpha) := \prod_{j=1}^{m} p(\mathbf{x}'_j|\theta, \alpha) = \prod_{j=1}^{m} \sum_{i=1}^{n} \alpha_i p(\mathbf{x}'_j|\omega_i, \theta). \tag{4}$$

Direct computation of the Maximum Likelihood Estimator (MLE) for $p(\mathbf{x}'|\theta, \alpha)$ is often impossible due to the complex shape of the function. Instead, EM introduces hidden variables, which help simplify (4), and applies maximization to the *expected* likelihood function, conditioned on the current estimate of unknown parameters. To this end, define hidden variables $z = (z_1, \ldots, z_m)$ to specify which OS produced each observation $\mathbf{x}'_j$. Note that the dataset of pairs $((\mathbf{x}'_1, z_1), \ldots, (\mathbf{x}'_m, z_m))$ is called *complete*, as opposed to just $\mathbf{x}'$, which is *incomplete*. Then, the *complete likelihood function* is given by

$$p(\mathbf{x}', z|\theta, \alpha) := \prod_{j=1}^{m} p(\mathbf{x}'_j, z_j|\theta, \alpha) = \prod_{j=1}^{m} p(\mathbf{x}'_j|z_j, \theta) p(z_j|\alpha)$$

$$= \prod_{j=1}^{m} \alpha_{z_j} p(\mathbf{x}'_j|\omega_{z_j}, \theta). \tag{5}$$

It is often more convenient to work with summations, in which case the above is replaced with

$$\begin{aligned} \log p(\mathbf{x}', z|\theta, \alpha) &= \sum_{j=1}^{m} (\log \alpha_{z_j} + \log p(\mathbf{x}'_j|\omega_{z_j}, \theta)) \\ &= \sum_{j=1}^{m} \sum_{i=1}^{n} (\log \alpha_i + c_{ij}) \mathbf{1}_{z_j=i}, \end{aligned} \tag{6}$$

where $c_{ij} := \log p(\mathbf{x}'_j|\omega_i, \theta)$ is a constant that can eventually be removed from optimization since it does not depend on α. Now, the E-step takes the expectation of (6) with respect to z, conditioned on the previous values α^t and the available observations, producing

$$\begin{aligned} Q(\alpha|\alpha^t) &:= E_z[\log p(\mathbf{x}', z|\theta, \alpha)|\mathbf{x}', \theta, \alpha^t] \\ &= \sum_{j=1}^{m} \sum_{i=1}^{n} (\log \alpha_i + c_{ij}) E[\mathbf{1}_{z_j=i}|\mathbf{x}', \theta, \alpha^t] \\ &= \sum_{j=1}^{m} \sum_{i=1}^{n} (\log \alpha_i + c_{ij}) \beta_{ij}^t, \end{aligned} \tag{7}$$

where

$$\beta_{ij}^t := p(\omega_i|\mathbf{x}'_j, \theta, \alpha^t) = \frac{\alpha_i^t p(\mathbf{x}'_j|\omega_i, \theta)}{\sum_{\ell=1}^{n} \alpha_\ell^t p(\mathbf{x}'_j|\omega_\ell, \theta)}. \tag{8}$$

The M-step maximizes (7) with respect to the unknown parameter α and entails solving

$$\frac{\partial Q(\alpha|\alpha^t)}{\partial \alpha_i} = 0. \tag{9}$$

Note that we can reduce the number of unknown variables using $\alpha_n = 1 - \sum_{i=1}^{n-1} \alpha_i$, which yields for $i = 1, 2, \ldots, n-1$

$$\sum_{j=1}^{m} \Big(\frac{\beta_{ij}^t}{\alpha_i} - \frac{\beta_{nj}^t}{\alpha_n} \Big) = 0. \tag{10}$$

Defining $c = \sum_{j=1}^{m} \beta_{nj}^t / \alpha_n$, we get

$$\alpha_i = \frac{1}{c} \sum_{j=1}^{m} \beta_{ij}^t. \tag{11}$$

From normalization $\sum_{i=1}^{n} \alpha_i = 1$, it follows that c must be m and that additionally (11) applies to $i = n$. We therefore get (3). ∎

Note that this is markedly different from deciding popularity using the fraction of classification decisions that go to each OS, which is known as *hard EM* and commonly used in clustering algorithms such as k-means [23]. In fact, all previous fingerprinting tools [6], [7], [39], [45], [46], [55], [57] can be viewed as performing one iteration of hard EM, i.e., outputting the fraction of classifications that belong to each OS ω_i as an estimate of its popularity α_i.

TABLE II
NETWORK DISTORTION IN SCENARIO S_1

Case	Forward latency (sec)		One-way delay (sec)		Loss
	Distribution	Mean	Distribution	Mean	
S_{11}	Erlang(2)	0.5	Exp	0.5	3.8%
S_{12}	Pareto	0.5	Pareto	0.5	50%
S_{13}	Reverse-exp	1.5	Erlang(2)	0.5	10%
S_{14}	Pareto	0.1	Uniform	0.1	50%

C. Discussion

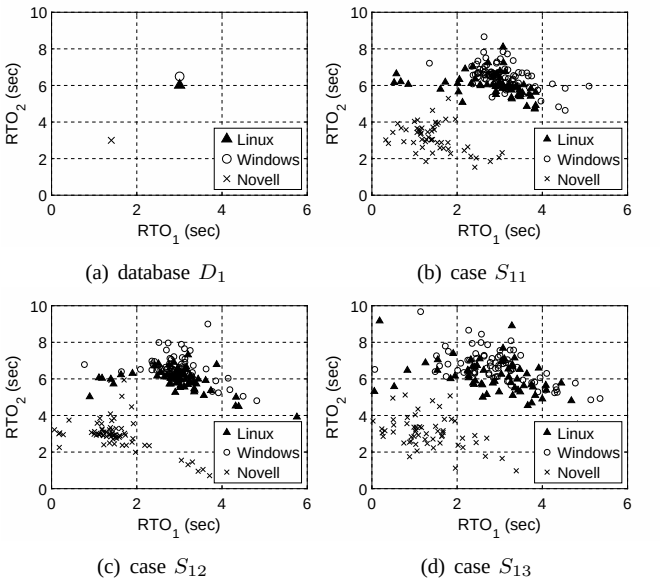
We now address the question of whether (3) is sufficient for achieving good classification on its own and how much of the accuracy depends on knowing the exact distortion model θ. If the majority of the benefit is already obtained from recovering α, the extra computational cost and modeling effort involved in estimating θ may be unnecessary. For discussion purposes, we use a set of toy databases that allow simple demonstration of the intended effects. Note that the same conclusions apply to larger datasets, but finding the corresponding scenarios may be more time-consuming.

Simulations below apply a forward latency to the SYN packet, pass each SYN-ACK through a FIFO queue, which adds random one-way delays along the return path, and drop packets using an iid (independent and identically distributed) loss model with some fixed probability. This is similar to the context in which prior methods [45], [46] have been tested. For the scenario we call S_1, there are four different cases for the distribution of foward/reverse delays and packet-loss probability. These are shown in Table II and discussed in more detail next.

The first row matches exactly the assumed parameters θ in Hershel+ [45]. The second row uses Pareto delays with mean 500 ms and 50% loss, emulating highly noisy network conditions. The next row uses a shifted *reverse-exponential* forward latency with CDF $e^{-\lambda(2-x)}$, defined for $-\infty < x \leq 2$, which tests contrary-to-intuition examples where larger delays are more likely than smaller. We employ $\lambda = 2$ and truncate this distribution at zero, obtaining the average forward SYN delay of 1.5 sec. The last case in the table examines smaller average delays than the assumed model θ in Hershel+, but couples it with substantial loss.

Our first database D_1 contains truncated signatures of Linux 3.2 (ω_1), Windows Server 2003 (ω_2), and Novell Netware (ω_3) from Table I. We retain the first two retransmission timeouts (RTOs), remove all fixed header features, and obtain the fingerprints in Fig. 2(a). Note that these Linux and Windows signatures are pretty close to each other, albeit not identical; however, they are quite different from Novell. The first three distortions $S_{11} - S_{13}$ applied to this database are illustrated in the remaining subfigures, where we show the first 200 samples and remove observations with lost packets.

Define ρ^t to be the fraction of correct classifications for a given method during iteration t, where $t = \infty$ represents the convergence point of the underlying estimator (usually $20-40$ iterations). If the method does not perform iteration, only ρ^1 is meaningful. We consider three techniques – Hershel+, hard EM with multiple iterations, and soft EM in (3), all using the same function $p(\mathbf{x}'_j|\omega_i, \theta)$ and starting from uniform



(a) database D_1 (b) case S_{11}

(c) case S_{12} (d) case S_{13}

Fig. 2. Database and distorted observations.

TABLE III
CLASSIFICATION RESULTS IN D_1

Case	α	Hershel+		Hard EM		EM in (3)	
		ρ^1	α^1	ρ^∞	α^∞	ρ^∞	α^∞
S_{11}	0.90	0.67	0.59	0.95	0.95	0.95	0.89
	0.05		0.35		0.00		0.06
	0.05		0.06		0.05		0.05
S_{12}	0.05	0.48	0.45	0.06	0.98	0.89	0.11
	0.90		0.41		0.00		0.82
	0.05		0.12		0.02		0.07
S_{13}	0.90	0.45	0.37	0.09	0.01	0.10	0.11
	0.05		0.51		0.88		0.79
	0.05		0.12		0.11		0.10
S_{14}	0.3	0.60	0.65	0.33	0.97	0.34	0.81
	0.6		0.23		0.00		0.13
	0.1		0.12		0.03		0.05

popularity $\alpha_i^0 = 1/n$.

Results of this process with $m = 2^{18}$ observations are shown in Table III. In the first row, Hershel+ performs quite well, achieving $\rho^1 = 67\%$. Since Novell Netware is an easy-to-separate signature from the other two, Hershel+ recovers α_3 pretty accurately; however, it is utterly confused about the frequency of the other two stacks. Applying hard EM increases accuracy, but full reconstruction of α still proves difficult. Application of (3) solves this issue.

Swapping (α_1, α_2), the second simulation in Table III shows that Hershel+ is essentially guessing between Linux and Windows, while hard EM is misled into divergence, where it drops accuracy from 48% to 6%. While (3) is immune to divergence in this case, its estimate of α suffers from non-negligible errors. The next two cases in the table are even more difficult. They show that EM can be driven into inferior states when the assumed θ greatly deviates from that of the underlying network. In fact, application of (3) not only fails to obtain vectors α that resemble the true distribution, but also harms performance of the system, i.e., $\rho^\infty \ll \rho^1$.

It is interesting that hard-EM techniques, universally used in prior work [6], [7], [39], [45], [46], [55], [57], may generally

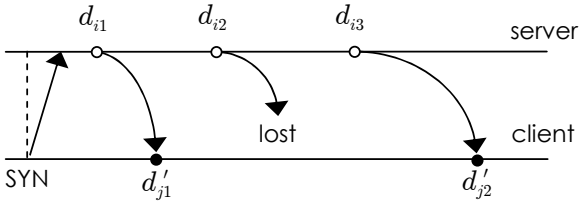


Fig. 3. Delay features (stack ω_i produces observation x'_j).

be unsuitable for characterizing the fraction of hosts running each OS, especially if α is highly skewed. Additionally, EM iteration is meaningful only when θ is either known a-priori, or can be accurately extracted from the collected observations. We investigate the latter direction next.

IV. Network Features

A. Distortion Model

Our goal in this section is to estimate unknown distortion parameters θ inside $p(\mathbf{x}'_j|\omega_i,\theta)$. Let features $\mathbf{x}_i = (\mathbf{d}_i, \mathbf{u}_i)$ consist of network components (i.e., delays $\mathbf{d}_i$) and user-modified header values (i.e., $\mathbf{u}_i$). Since classification [45], [46] usually assumes that distortion is applied to each subvector independently, it follows that

$$p(\mathbf{x}'_j|\omega_i,\theta) = p(\mathbf{d}'_j|\omega_i,\theta_d)p(\mathbf{u}'_j|\omega_i,\theta_u), \tag{12}$$

where θ_d, θ_u are the network/user distortion models, respectively. Each of them contains multiple PMFs (probability mass functions) that we elaborate on below. Since in this section we consider only the network component, we assume that $p(\mathbf{u}'_j|\omega_i,\theta_u) = 1$ for all i, j, i.e., all observed user features are the same and thus perfectly match all fingerprints.

To understand the notation involved in expanding the first factor in (12), examine Fig. 3 where a host with network signature $\mathbf{d}_i$ generates an observation $\mathbf{d}'_j$. Measurement begins with a SYN packet, which takes some time to get to the target, followed by the server "think" delay before it generates the first SYN-ACK response. Database feature vectors $\mathbf{d}_i$ consist of departure timestamps from the server, where $d_{i1} = 0$ for all i. Note that $d_{i,r+1} - d_{ir}$ is the r-th retransmission timeout (RTO) of the stack, which was commonly used in early estimators [6], [53], [46]. Recently, however, usage of absolute timestamps d_{ir} was identified [45] as having certain modeling advantages, which is our approach as well.

On the client side, arrival timestamps d'_{jr} are measured relative to the transmission time of the SYN. Assume T_j represents the sum of the forward delay, server think time, and propagation/transmission delays of the reverse path, where T_j has some unknown distribution $f_T(\tau) = P(T_j = \tau)$. Furthermore, let $\Delta_{j1}, \Delta_{j2}, \ldots$ be iid queuing delays of the return path, with another unknown distribution $f_\Delta(\delta) = P(\Delta_{jr} = \delta)$. Then, assuming no loss, $d'_{jr} = T_j + d_{ir} + \Delta_{jr}$. In practice, T_j and Δ_{jr} are continuous variables, but it is convenient to discretize them into small bins and directly work with PMFs.

To handle packet loss, assume that γ_j is a random vector that maps the received packets in observation j to their order on the server, i.e., $\gamma_j(r) = k$ means that the r-th received packet was originally in position k. In Fig. 3, for example, we have $\gamma_j = (1, 3)$. Then, if the j-th observation comes from system ω_i, it follows that

$$d'_{jr} = T_j + d_{i,\gamma_j(r)} + \Delta_{jr}, \quad r = 1, 2, \ldots, |\mathbf{d}'_j|. \tag{13}$$

As in prior work [6], [45], [46], we assume no reordering due to the large spacing between the packets (often several seconds), which implies $\gamma_j(r+1) > \gamma_j(r)$. Let $\Gamma(i, j)$ be the set of all monotonic loss vectors that start with $|\mathbf{d}_i|$ packets and finish with $|\mathbf{d}'_j|$. Then, the Hershel+ network classifier uses $p(\mathbf{d}'_j|\omega_i,\theta_d)$ equal to [45]

$$\sum_\tau f_T(\tau) \sum_{\gamma\in\Gamma(i,j)} p_i(\gamma) \prod_{r=1}^{|\mathbf{d}'_j|} f_\Delta(d'_{jr} - \tau - d_{i,\gamma(r)}), \tag{14}$$

where $p_i(\gamma)$ is the probability to observe loss pattern γ under $|\mathbf{d}_i|$ transmitted packets. To avoid clutter, we omit the formulas for handling random signatures $\mathbf{d}_i$ in Hershel+, which require an extra summation over all possible sub-OSes and normalization by the corresponding weights, but keep this functionality in the code. For lack of a better assumption, Hershel+ uses binomial $p_i(\gamma)$, Erlang(2) $f_T(\tau)$, and exponential $f_\Delta(\delta)$, all with some fixed parameters. Since θ_d encapsulates the set of these distributions, our next goal is to recover them using EM iteration.

B. Intuition

We start with a heuristic explanation of the proposed update formulas, which is followed by a more rigorous treatment. Recall that $f_T^t(\tau)$ is an estimate of $P(T_j = \tau)$ during iteration t. Then, one obvious approach is to set this value to the average probability that each observation j has experienced a forward latency τ, conditioned on the previous estimates of unknown parameters, i.e.,

$$f_T^{t+1}(\tau) = \frac{1}{m}\sum_{j=1}^{m} P(T_j = \tau|\mathbf{d}'_j, \theta_d^t, \alpha^t). \tag{15}$$

Next, each database signature with k original packets admits $2^k - 1$ unique loss patterns γ, where k goes as high as $k_{max} = 21$ in the most recent effort in the field [45]. Estimating the probability $p_i(\gamma)$ for each possible option γ is likely to produce too many unknown variables and lead to poor convergence of EM. Instead, suppose that all $\binom{k}{\ell}$ patterns of losing ℓ packets out of k are equally likely and define the probability of this event to be $q_k(\ell)$, where $k = 1, 2, \ldots, k_{max}$. The resulting reduction in the number of unknown variables is significant – from roughly $2^{k_{max}+1} = 4$M to just $k_{max}(k_{max}-1)/2 = 210$. Despite its simplicity, the framework of using $q_k(\ell)$ allows more general scenarios than the traditional iid Bernoulli model used in previous literature [45], [46].

To update distribution $q_k(\ell)$, our approach involves computing the probability that observations experienced loss of ℓ packets out of k transmitted, normalized by the probability that the original host sent k packets in the first place. To express this mathematically, define Y_j to be the number of

SYN-ACKs originated by the host in observation j. Letting $\mathbf{1}_X$ be an indicator of event X, we get

$$q_k^{t+1}(\ell) = \frac{\sum_{j=1}^{m} P(Y_j = k|\theta_d^t, \alpha^t)\mathbf{1}_{|\mathbf{d}'_j|=k-\ell}}{\sum_{j=1}^{m} P(Y_j = k|\theta_d^t, \alpha^t)\mathbf{1}_{|\mathbf{d}'_j|\leq k}}, \tag{16}$$

from which the estimated probability of pattern γ is given by

$$p_i^t(\gamma) = \frac{q_{|\mathbf{d}_i|}^t(|\mathbf{d}_i| - |\gamma|)}{\binom{|\mathbf{d}_i|}{|\gamma|}}. \tag{17}$$

Finally, updates to PMF $f_\Delta^t(\delta)$ involve computing the probability that one-way delay of each received packet equals δ, normalized by the total number of packets collected during the scan, i.e.,

$$f_\Delta^{t+1}(\delta) = \frac{\sum_{j=1}^{m}\sum_{s=1}^{|\mathbf{d}'_j|} P(\Delta_{js} = \delta|\mathbf{d}'_j, \theta_d^t, \alpha^t)}{\sum_{j=1}^{m}|\mathbf{d}'_j|}. \tag{18}$$

C. Analysis

To make the framework outlined above usable, our next task is to express the probability of events that cannot be directly observed (e.g., $Y_j = k$, $\Delta_{jr} = \delta$) using a recurrence that depends on only the distributions contained in θ_d^t, i.e., $(f_T^t, f_\Delta^t, q_k^t)$. Let

$$\delta_{ij\tau\gamma r} = d'_{jr} - \tau - d_{i,\gamma(r)} \tag{19}$$

be the one-way delay Δ_{jr} conditioned on $T_j = \tau$, loss pattern γ, signature ω_i, and observation j. For brevity of notation, suppose $\sum_{ij\tau\gamma s}$ refers to five nested summations, where i goes from 1 to n, j rolls from 1 to m, τ moves over all bins of the PMF $f_T(\tau)$, γ iterates over all monotonic loss vectors in $\Gamma(i,j)$, and s travels from 1 to $|\mathbf{d}'_j|$. If some of the variables are absent from the subscript, the corresponding sums are omitted from the result. With this in mind, define

$$p_{ij\tau\gamma}^t := \alpha_i^t f_T^t(\tau) p_i^t(\gamma) \prod_{r=1}^{|\mathbf{d}'_j|} f_\Delta^t(\delta_{ij\tau\gamma r}), \tag{20}$$

$$\beta_{ij\tau\gamma}^t := p(\omega_i, \tau, \gamma|\mathbf{d}'_j, \theta_d^t, \alpha^t) = \frac{p_{ij\tau\gamma}^t}{\sum_{i\tau\gamma} p_{ij\tau\gamma}^t} \tag{21}$$

and consider the next result.

Theorem 2: Under network distortion, estimators (3), (15), (16), and (18) can be written as

$$\alpha_i^{t+1} = \frac{1}{m}\sum_{j\tau\gamma} \beta_{ij\tau\gamma}^t, \tag{22}$$

$$f_T^{t+1}(\tau) = \frac{1}{m}\sum_{ij\gamma} \beta_{ij\tau\gamma}^t, \tag{23}$$

$$q_k^t(\ell) = \frac{\sum_{ij\tau\gamma} \beta_{ij\tau\gamma}^t \mathbf{1}_{|\mathbf{d}'_j|=k-\ell,|\mathbf{d}_i|=k}}{\sum_{ij\tau\gamma} \beta_{ij\tau\gamma}^t \mathbf{1}_{|\mathbf{d}'_j|\leq|\mathbf{d}_i|=k}}, \tag{24}$$

$$f_\Delta^t(\delta) = \frac{\sum_{ij\tau\gamma s} \beta_{ij\tau\gamma}^t \mathbf{1}_{\delta_{ij\tau\gamma s}=\delta}}{\sum_j |\mathbf{d}'_j|}. \tag{25}$$

Proof: We start with the recurrence on α. Keeping distortion limited to network features, (3) becomes

$$\alpha_i^{t+1} = \frac{1}{m}\sum_{j=1}^{m} \frac{\alpha_i^t p(\mathbf{d}'_j|\omega_i, \theta_d^t)}{p(\mathbf{d}'_j|\theta_d^t, \alpha^t)}.$$

With the help of (14), we get

$$p(\mathbf{d}'_j|\omega_i, \theta_d^t) = \sum_{\tau\gamma} f_T^t(\tau) p_i^t(\gamma) \prod_{r=1}^{|\mathbf{d}'_j|} f_\Delta^t(\delta_{ij\tau\gamma r}), \tag{26}$$

which leads to

$$\alpha_i^t p(\mathbf{d}'_j|\omega_i, \theta_d^t) = \sum_{\tau\gamma} p_{ij\tau\gamma} \tag{27}$$

and, leveraging (2) for the denominator of (26),

$$\alpha_i^{t+1} = \frac{1}{m}\sum_{j=1}^{m} \frac{\sum_{\tau\gamma} p_{ij\tau\gamma}^t}{\sum_{i\tau\gamma} p_{ij\tau\gamma}^t} = \frac{1}{m}\sum_{j\tau\gamma} \beta_{ij\tau\gamma}^t. \tag{28}$$

Moving on to the forward latency, notice that (15) becomes

$$\begin{aligned} f_T^{t+1}(\tau) &= \frac{1}{m}\sum_{j=1}^{m} \frac{p(\mathbf{d}'_j|\tau, \theta_d^t, \alpha^t) p(\tau|\theta_d^t)}{p(\mathbf{d}'_j|\theta_d^t, \alpha^t)} \\ &= \frac{1}{m}\sum_{j=1}^{m} \frac{\sum_i \alpha_i^t p(\mathbf{d}'_j|\omega_i, \tau, \theta_d^t) f_T^t(\tau)}{p(\mathbf{d}'_j|\theta_d^t, \alpha^t)} \\ &= \frac{1}{m}\sum_{j=1}^{m} \frac{\sum_{i\gamma} p_{ij\tau\gamma}^t}{\sum_{i\tau\gamma} p_{ij\tau\gamma}^t} = \frac{1}{m}\sum_{ij\gamma} \beta_{ij\tau\gamma}^t. \end{aligned} \tag{29}$$

Next, the probability that the host in observation j sent k packets is

$$\begin{aligned} P(Y_j = k|\theta_d^t, \alpha^t) &= \sum_{i=1}^{n} p(\omega_i|\mathbf{d}'_j, \theta_d^t, \alpha^t)\mathbf{1}_{|\mathbf{d}_i|=k} \\ &= \sum_{i=1}^{n} \frac{\alpha_i^t p(\mathbf{d}'_j|\omega_i, \theta_d^t, \alpha^t)\mathbf{1}_{|\mathbf{d}_i|=k}}{p(\mathbf{d}'_j|\theta_d^t, \alpha^t)}. \end{aligned} \tag{30}$$

Using this, the numerator of (16) expands to

$$\begin{aligned} &\sum_{j=1}^{m} \frac{\sum_i \alpha_i^t p(\mathbf{d}'_j|\omega_i, \theta_d^t, \alpha^t)\mathbf{1}_{|\mathbf{d}'_j|=k-\ell,|\mathbf{d}_i|=k}}{p(\mathbf{d}'_j|\theta_d^t, \alpha^t)} \\ &\qquad = \sum_{ij\tau\gamma} \beta_{ij\tau\gamma}^t \mathbf{1}_{|\mathbf{d}'_j|=k-\ell,|\mathbf{d}_i|=k}. \end{aligned} \tag{31}$$

Applying the same logic to the denominator of (16), we get (24). Finally, updates to one-way delay admit the following interpretation

$$\begin{aligned} P(\Delta_{js} = \delta|\mathbf{d}'_j, \theta_d^t, \alpha^t) &= \frac{\sum_{i\tau\gamma} p_{ij\tau\gamma}^t \mathbf{1}_{\delta_{ij\tau\gamma s}=\delta}}{p(\mathbf{d}'_j|\theta_d^t, \alpha^t)} \\ &= \sum_{i\tau\gamma} \beta_{ij\tau\gamma} \mathbf{1}_{\delta_{ij\tau\gamma s}=\delta}, \end{aligned} \tag{32}$$

which is a sum of match probabilities over all signatures, forward latencies, and loss patterns that result in one-way delay δ in the s-th received packet. Adding the two summations over j, s and dividing by the total number of observed packets, we get (25). ∎

While the result of Theorem 2 may appear daunting due to the number of nested summations, its implementation in practice can be done with little extra cost compared to Hershel+. Specifically, usage of (14) in (1) for all i, j already requires five nested loops. In the inner-most loop of that algorithm, (25) adds one increment to a hash table that maintains the PMF of one-way delay. Updates in (22)-(24) are performed

TABLE IV
CLASSIFICATION RESULTS OF NETWORK EM IN D_1

Case	ρ^1	ρ^∞	α^∞
S_{11}	0.67	0.95	0.90, 0.05, 0.05
S_{12}	0.48	0.91	0.05, 0.90, 0.05
S_{13}	0.45	0.95	0.90, 0.05, 0.05
S_{14}	0.60	0.85	0.30, 0.60, 0.10

less frequently and, in comparison, consume negligible time. The only caveat is that Hershel+ can be optimized [45] to remove the outer summation in (14) when f_T is Erlang(2) and f_Δ is exponential. Our approach, on the other hand, requires a hash-table lookup for both distributions. This makes its single iteration similar in speed to unoptimized Hershel+.

Theorem 3: Iteration (22)-(25) is the EM algorithm for (θ_d, α).

Proof: Assume $H_j = (z_j, T_j, \gamma_j)$ are the hidden variables that specify for observation j its true OS, forward latency, and loss pattern, respectively. Further suppose $H = (H_1, \ldots, H_m)$ is the collection of hidden variables for the entire measurement. Then, the complete likelihood function is given by

$$p(\mathbf{d}', H|\theta_d, \alpha) := \prod_{j=1}^{m} p(\mathbf{d}'_j, H|\theta_d, \alpha) = \prod_{j=1}^{m} p(\mathbf{d}'_j|H_j, \theta_d, \alpha) p(H_j|\theta_d, \alpha), \quad (33)$$

where

$$p(\mathbf{d}'_j|H_j, \theta_d, \alpha) = \prod_{r=1}^{|\mathbf{d}'_j|} f_\Delta(d'_{jr} - \tau_j - d_{z_j,\gamma_j(r)}) \quad (34)$$

$$p(H_j|\theta_d, \alpha) = \alpha_{z_j} f_T(T_j) p_{z_j}(\gamma_j). \quad (35)$$

Define

$$p_{ij\tau\gamma} = \alpha_i f_T(\tau) p_i(\gamma) \prod_{r=1}^{|\mathbf{d}'_j|} f_\Delta(d'_{jr} - \tau - d_{i,\gamma(r)}). \quad (36)$$

Following the proof of Theorem 1, the log-likelihood expands to

$$\log p(\mathbf{d}', H|\theta_d, \alpha) := \sum_{j=1}^{m} \log(p_{z_j,j,T_j,\gamma_j}) = \sum_{j=1}^{m} \sum_{i\tau\gamma} \log(p_{ij\tau\gamma}) \mathbf{1}_{z_j=i, T_j=\tau, \gamma_j=\gamma}. \quad (37)$$

The expected log-likelihood function is then given by

$$Q(\theta_d, \alpha|\theta_d^t, \alpha^t) = \sum_{ij\tau\gamma} \log(p_{ij\tau\gamma}) p(\omega_i, \tau, \gamma|\mathbf{d}'_j, \theta_d^t, \alpha^t) = \sum_{ij\tau\gamma} \log(p_{ij\tau\gamma}) \beta^t_{ij\tau\gamma}. \quad (38)$$

Taking partial derivatives with respect to α_i and $f_T(\tau)$, we get a set of equations similar to (9)-(10). Their solution is trivially given by (22)-(23). A more interesting case is the loss PMF. Using substitution

$$q_k(k-1) = 1 - \sum_{\ell=0}^{k-2} q_k(\ell), \quad (39)$$

in (38), we get for $\ell = 0, 1, \ldots, k-2$ that

$$\frac{\partial Q(\theta_d, \alpha|\theta_d^t, \alpha^t)}{\partial q_k(\ell)} = \sum_{ij\tau\gamma} \frac{\mathbf{1}_{|\mathbf{d}'_j|=k-\ell, |\mathbf{d}_i|=k}}{q_k(\ell)/\binom{k}{\ell}} \beta^t_{ij\tau\gamma} - \sum_{ij\tau\gamma} \frac{\mathbf{1}_{|\mathbf{d}'_j|=1, |\mathbf{d}_i|=k}}{q_k(k-1)} \beta^t_{ij\tau\gamma}. \quad (40)$$

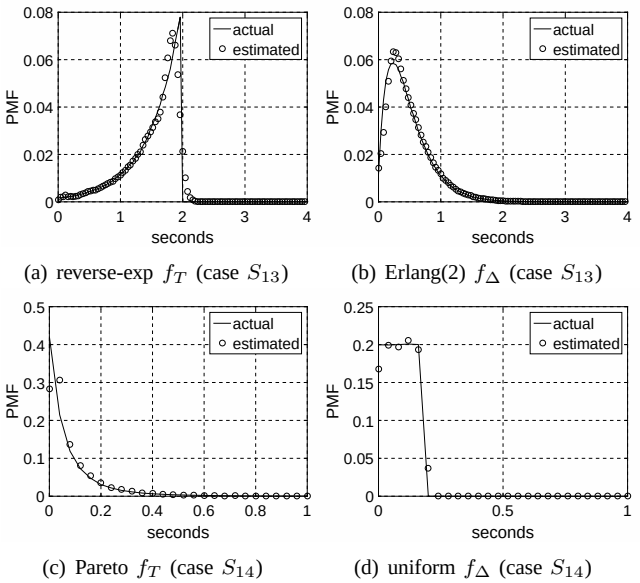
Setting c to be the second summation in (40) and equating the derivative to zero, we get

$$q_k(\ell) = \frac{1}{c} \sum_{ij\tau\gamma} \mathbf{1}_{|\mathbf{d}'_j|=k-\ell, |\mathbf{d}_i|=k} \binom{k}{\ell} \beta^t_{ij\tau\gamma}. \quad (41)$$

Since the PMF q_k must add up to 1, it follows that

$$c = \sum_{\ell=0}^{k-1} \sum_{ij\tau\gamma} \mathbf{1}_{|\mathbf{d}'_j|=k-\ell, |\mathbf{d}_i|=k} \binom{k}{\ell} \beta^t_{ij\tau\gamma} = \sum_{ij\tau\gamma} \mathbf{1}_{|\mathbf{d}'_j|\le k, |\mathbf{d}_i|=k} \binom{k}{\ell} \beta^t_{ij\tau\gamma}. \quad (42)$$

Using this in (41) and canceling $\binom{k}{\ell}$ yields (24). Note that derivation of (25) is very similar. We omit it for brevity. ■



(a) reverse-exp f_T (case S_{13}) (b) Erlang(2) f_Δ (case S_{13})

(c) Pareto f_T (case S_{14}) (d) uniform f_Δ (case S_{14})

Fig. 4. Recovery of delay parameters in D_1.

D. Discussion

We revisit earlier simulations on dataset D_1, run (22)-(25) over the same input, and show the result in Table IV. Compared to Table III, the derived EM estimator significantly improves the accuracy of both classification and vector α. Note that S_{12} contain 43% of the observations with just one packet, i.e., zero RTOs. In methods that rely on RTO [6], [46], [53], these samples would be either discarded as impossible

to classify or assigned to a uniformly random signature. In contrast, estimator (22)-(25) manages to do much better as it learns distributions (f_T, f_Δ, α) and makes the best decision possible under these conditions. The accuracy of estimated delay distributions is shown in Fig. 4. With the exception of noise at the points of discontinuity of each density, functions $f_T^\infty, f_\Delta^\infty$ match the true parameters quite well.

Recalling (13), where $T_j + \Delta_{jr}$ are always measured together, it may not be obvious how T_j can be separated from Δ_{jr} and why the result in Fig. 4 is possible. Indeed, this is reminiscent of the classical deconvolution problem: given observations $\{X_i + Y_i\}_{i=1}^m$, where $X_i \sim F_X(x)$ and $Y_i \sim F_Y(x)$ are iid, determine the individual distributions F_X, F_Y. Deconvolution is generally unsolvable unless either F_X or F_Y is known ahead of time. While our problem is similar, there is a crucial difference – EM can see the same value T_j coupled with *multiple* instances of Δ_{jr}, for $r = 1, 2, \ldots, |\mathbf{d}'_j|$. As long as $q_k(k-1) < 1$ (i.e., packet loss leaves at least two packets in enough observations) and $m \to \infty$, deconvolution is possible in our setting, but up to a location shift, i.e., one of the estimated distributions may be shifted left by a constant and the other right by the same amount. If we know that one of them starts at zero, it is possible to determine the shift after the fact. Furthermore, if both estimated densities $f_T^\infty, f_\Delta^\infty$ already begin at zero, no correction is needed. This is the case in Fig. 4 and later in our Internet scan.

Since all signatures in D_1 had three packets, it was easy to figure out the number of them lost in each $\mathbf{d}'_j$, which led to q_k^∞ being perfectly accurate, regardless of whether (24) was used or not. In a more interesting database, which we call D_2, Linux is augmented with a fourth packet that follows after a 3-second RTO. To experiment with different loss patterns, define BinT(k, p) to be a binomial distribution with parameters (k, p) truncated to the range $[0, k-1]$. Since the loss of all k packets cannot be observed, we avoid generating this case in the simulator. Additionally, suppose RevBin(k, p) is the *reverse binomial distribution* such that $X \sim$ BinT(k, p) and $Y = k - 1 - X$ implies $Y \sim$ RevBin(k, p). With this in mind, consider scenario S_2 in Table V, which shows q_k and the average observed loss rate among the signatures with k packets.

Table VI shows classification results for three methods – Hershel+, the partial EM framework without loss updates (24), and the full algorithm from Theorem 2. Not surprisingly, Hershel+ again struggles to recover α, even when its classification accuracy is pretty high. Omission of (24) does create challenges for partial EM, where in all four cases it produces worse results than Hershel+. On the other hand, the full algorithm improves accuracy and delivers the exact α despite complex underlying network conditions. The corresponding distributions q_k^∞ are shown in Table VII. They all match ground-truth q_k with high precision.

TABLE V
NETWORK PARAMETERS OF SCENARIO S_2

Case	Delay	q_3	Loss	q_4	Loss
S_{21}	As in S_{12}	BinT$(3, 0.3)$	28%	BinT$(4, 0.3)$	30%
S_{22}	As in S_{12}	BinT$(3, 0.1)$	10%	BinT$(4, 0.8)$	66%
S_{23}	As in S_{12}	RevBin$(3, 0.1)$	57%	RevBin$(4, 0.1)$	65%
S_{24}	As in S_{13}	BinT$(3, 0.7)$	54%	BinT$(4, 0.7)$	61%

TABLE VI
CLASSIFICATION RESULTS IN D_2

Case	α	Hershel+		EM α, f_T, f_Δ		Full EM	
		ρ^1	α^1	ρ^∞	α^∞	ρ^∞	α^∞
S_{21}	0.90	0.76	0.68	0.70	0.63	0.91	0.90
	0.05		0.25		0.31		0.05
	0.05		0.07		0.05		0.05
S_{22}	0.90	0.45	0.34	0.13	0.06	0.97	0.90
	0.05		0.47		0.84		0.05
	0.05		0.19		0.10		0.05
S_{23}	0.90	0.45	0.36	0.10	0.06	0.90	0.90
	0.05		0.46		0.90		0.05
	0.05		0.18		0.04		0.05
S_{24}	0.90	0.42	0.33	0.14	0.10	0.92	0.90
	0.05		0.38		0.88		0.05
	0.05		0.29		0.02		0.05

V. USER FEATURES

A. Distortion Model

Our goal in this section is to expand the second factor in (12) and develop an estimator for its distortion model. This is done in isolation from the network features, i.e., using $p(\mathbf{d}'_j|\omega_i, \theta_d) = 1$ for all i, j. Assume $b \geq 1$ user features, where each observation j provides a constant-length vector $\mathbf{u}'_j = (u'_{j1}, \ldots, u'_{jb})$. These include the TCP window size, IP TTL (Time to Live), IP DF (Do Not Fragment flag), TCP MSS (Maximum Segment Size), and TCP options, for a total of $b = 5$ integer-valued fields. Since RST features depend on network loss, we delay their discussion until the next section. Note that each field may be allocated a different number of bits in the TCP/IP header and the number of available options a_v for u'_{jv} may depend on v (e.g., two for DF and 64K for Win).

Modification to user features at the target host, which we model with a set of distributions θ_u, can be accomplished by manually changing default OS parameters (e.g., editing the registry), using specialized performance-tuning software, requesting larger/smaller receiver kernel buffers while waiting on sockets (i.e., using setsockopt), and deploying network/host scrubbers [12], [40], [43], [49], [54] whose purpose is to obfuscate the OS of protected machines. Besides intentional feature modification, distortion θ_u may also accommodate unknown network stacks that build upon a documented OS, but change some of its features (e.g., new versions of embedded Linux customized to a particular device).

Prior work either omits formally modeling user volatility [6], [7], [39], [55], [57], or assumes that u_{iv} stays the same with some probability π_v and changes to another integer with probability $1 - \pi_v$ [45], [46]. While the latter approach works well in certain cases, it has limitations. Besides the fact that π_v is generally unknown, binary decision-making fails to create a distribution over the available choices. For example, $\pi_v = 0.9$ assumes that *each* of the $65{,}534$ non-default window sizes occurs with probability 0.1. Instead, a more balanced approach would be to assume a uniform distribution over the distortion possibilities and assign them probability $(1 - \pi_v)/(a_v - 1)$. Second, it is likely that certain devices are modified less

TABLE VII
RECOVERY OF LOSS PMFS IN D_2

Case	Vector	$k=3$	$k=4$
S_{21}	q_k	$(0.35, 0.45, 0.19)$	$(0.24, 0.41, 0.27, 0.08)$
	q_k^∞	$(0.35, 0.45, 0.19)$	$(0.24, 0.41, 0.27, 0.08)$
S_{22}	q_k	$(0.73, 0.24, 0.03)$	$(0.00, 0.04, 0.26, 0.69)$
	q_k^∞	$(0.73, 0.24, 0.03)$	$(0.00, 0.04, 0.26, 0.69)$
S_{23}	q_k	$(0.03, 0.24, 0.73)$	$(0.00, 0.05, 0.29, 0.66)$
	q_k^∞	$(0.03, 0.24, 0.73)$	$(0.00, 0.05, 0.29, 0.66)$
S_{24}	q_k	$(0.04, 0.29, 0.67)$	$(0.01, 0.10, 0.35, 0.54)$
	q_k^∞	$(0.04, 0.29, 0.67)$	$(0.01, 0.10, 0.35, 0.54)$

frequently than others (e.g., due to firmware restrictions) and individual distortions are OS-specific, which implies that π_v should depend on i. Finally, the existing methods have no way of tracking the location and probability mass of distortion, which does not have to be uniform in practice (e.g., a non-default window size 257 bytes is less likely than 64K).

To overcome these problems, assume that $\pi_{iv}(y)$ is the probability that feature v of OS i is modified to become y, which gives rise to a set of nb distributions that comprise our user-distortion model θ_u. Then, the proposed classifier can be summarized by

$$p(\mathbf{u}'_j|\omega_i, \theta_u) = \prod_{v=1}^{b} \pi_{iv}(u'_{jv}), \tag{43}$$

where modification to features is assumed to be independent. Note that doing otherwise does not appear tractable at this point (i.e., estimation of covariance matrices produces too many variables for EM to handle).

B. Iteration

We begin by discussing under what conditions the problem is identifiable, despite having a large number of unknown distributions. Assume $\phi_{iv} := \pi_{iv}(u_{iv})$ is the probability with which feature v stays the same for OS i. Because we do not know ahead of time the reasoning of the user for changing the features or the new values of modified fields, the estimation problem for π_{iv} is unsolvable unless enough of the probability mass remains at the original location, i.e., ϕ_{iv} is above some threshold. From common sense, it is likely that $\phi_{iv} \geq 0.5$ holds among the general population of Internet hosts; however, EM converges under even weaker conditions (e.g., when ϕ_{iv} is the largest value in each PMF π_{iv}). Coupling this with an initial state that satisfies the same constraint allows EM to discover a unique solution.

We define the estimator for user distortion as the probability to observe y in feature v across all matches against OS i, i.e.,

$$\pi_{iv}^{t+1}(y) = \frac{\sum_{j=1}^{m} p(\omega_i|\mathbf{u}'_j, \theta_u^t, \alpha^t)\mathbf{1}_{u'_{jv}=y}}{\sum_{j=1}^{m} p(\omega_i|\mathbf{u}'_j, \theta_u^t, \alpha^t)}. \tag{44}$$

To allow simplification of this expression below, define

$$p_{ij}^t := \alpha_i^t p(\mathbf{u}'_j|\omega_i, \theta_u^t, \alpha^t) = \alpha_i^t \prod_{v=1}^{b} \pi_{iv}^t(u'_{jv}), \tag{45}$$

$$\beta_{ij}^t := p(\omega_i|\mathbf{u}'_j, \theta_u^t, \alpha^t) = \frac{p_{ij}^t}{\sum_{i=1}^{n} p_{ij}^t}. \tag{46}$$

TABLE VIII
USER FEATURES OF DATABASE D_3

OS	Win	TTL	DF	OPT	MSS
Linux	5,792	64	1	MSTNW	1,460
Windows	16,384	128	0	MNWNNTNNS	1,380
Novell	6,144	128	1	MNWSNN	1,460

The next result follows from substitution of (45)-(46) into (3) and (44), as well as earlier proofs of Theorems 1 and 2.

Theorem 4: Under user distortion, estimators (3) and (44) can be written as

$$\alpha_i^{t+1} = \frac{1}{m} \sum_{j=1}^{m} \beta_{ij}^t, \tag{47}$$

$$\pi_{iv}^{t+1}(y) = \frac{\sum_{j=1}^{m} \beta_{ij}^t \mathbf{1}_{u'_{jv}=y}}{m\alpha_i^{t+1}}. \tag{48}$$

Furthermore, this is the EM algorithm for (θ_u, α).

C. Discussion

To evaluate the result of Theorem 4, we construct a new database D_3, shown in Table VIII, by switching from RTOs to user features. Note that this Linux signature ties Novell in DF and MSS, while Windows does the same in TTL. For simplicity of presentation, we use simulation scenarios with $\phi_{iv} = \phi_v$ for all i, where ϕ_v is the probability with which feature v stays at the default value. This replaces matrix ϕ_{iv} with a vector ϕ_v, which is easier to follow across the different tables.

The initial PMFs π_{iv}^0 of EM are set up to include 90% of the mass on the default value and split the remainder uniformly across the viable alternatives. Since it is believed [46] that the order of non-NOP options cannot be changed without rewriting the TCP/IP stack of the OS, we initialize π_{i4}^0 to allow only candidates compatible with the original u_{i4}. For example, MST is feasible for Linux, but not the other two signatures in Table VIII. Note that any single option (M, S, W) and the empty set are valid for all three OSes.

We use two models for generating noisy observations. The first one, which we call RAND, picks uniformly from the space of possible values observed in our Internet scan, except OPT is limited to compatible subsets/supersets of the original. We have $5{,}695$ candidates for Win, four for TTL, two for DF, 266 for OPT, and $1{,}082$ for MSS. Decisions are made independently for each feature v and each observation j, which models users "tweaking" their OS without coordinating with each other or sharing a common objective. Even though RAND can generate 13.1 billion unique combinations $\mathbf{u}'_j$, only a small subset is encountered by the classifier in our simulations below.

The second model, which we call PATCH, selects an alternative vector of features $\mathbf{u}''_i$ for each OS ω_i and switches the default value u_{iv} to u''_{iv} with probability $1 - \phi_v$, again independently for each v. This represents deployment of software patches that change one of the features to an updated value. The probability for a host to use multiple patches is the product of corresponding $(1-\phi_v)$'s. For example, modification

TABLE IX
PATCHED USER FEATURES

Vector	Win	TTL	DF	OPT	MSS
$\mathbf{u}_1''$	5,793	128	0	M	1,461
$\mathbf{u}_2''$	16,386	32	1	M	1,382
$\mathbf{u}_3''$	6,147	64	0	M	1,463

TABLE X
PARAMETERS OF SCENARIO S_3

Case	Model	Feature stay prob ϕ_v	Popularity α
S_{31}	RAND	$(0.3, 0.2, 0.5, 0.4, 0.4)$	$(0.90, 0.05, 0.05)$
S_{32}	RAND	$(0.0, 0.0, 0.1, 0.2, 0.0)$	$(0.90, 0.05, 0.05)$
S_{33}	PATCH	$(0.2, 0.2, 0.2, 0.2, 0.2)$	$(0.7, 0.2, 0.1)$

to both Win and OPT affects $(1-\phi_1)(1-\phi_4)$ fraction of hosts. Vectors $\mathbf{u}_i''$ are non-adversarial and do not attempt to confuse the classifier. We construct them by flipping the DF flag, setting OPT to M, and adding i to all remaining fields (modulo the max field value). The result is given in Table IX.

To estimate vector ϕ_v^t in the classifier, we use a weighted average of feature non-modification across all OSes, i.e., $\phi_v^t = \sum_{i=1}^n \alpha_i^t \phi_{iv}^t$. Our next scenario S_3 is detailed in Table X and the corresponding outcome is given by Table XI. We omit vector α^∞ since it matches ground-truth α very accurately. Due to the new treatment of non-default features in (43), the first iteration of EM in Table XI is superior to Hershel+. However, both are much worse than the last iteration. It should be noted that the second case S_{32} modifies Win, TTL, and MSS in 100% of the samples. Identifiability in such conditions is helped by the fact that OPT is constrained to a subset of the original string, which makes a certain fraction of randomly generated values feasible for only one OS. This allows EM to learn to ignore (Win, TTL, MSS) and focus decisions on (DF, OPT). Furthermore, when guessing is involved, EM uses its knowledge of α to correctly pick the most-likely OS. It is also interesting that S_{33} is classified with 100% accuracy once EM gets a grasp on the new values in Table IX and their probability of occurrence.

VI. COMPLETE SYSTEM

A. Reset Packets

Because loss of RST packets causes the corresponding user features (i.e., ACK/RST flags, ACK sequence number, window size [46]) to be wiped out, there is dependency between distortion applied by the network and the user. As a result, this case should be handled separately. The first modification needed is to increase the length of network vectors $\mathbf{d}_i$ and $\mathbf{d}_j'$ to accommodate the RST timestamp. The second change is to add RST values into user features. Since it is currently believed that RST fields are unmodifiable independently of each other [46], they can be combined into a single integer and appended to user vectors $\mathbf{u}_i$ and $\mathbf{u}_j'$ in position $b+1$.

There are four possible scenarios for handling RST packets. They are shown in Table XII, each with a certain probability ζ_{ij}^t that must be factored into the formulas developed earlier. When both the observation and candidate signature contain a RST, the only multiplier needed is the probability that the received feature was produced by that OS. If the sampled

TABLE XI
CLASSIFICATION RESULTS IN D_3

Case	Hershel+	EM		
	ρ^1	ρ^1	ρ^∞	ϕ_v^∞
S_{31}	0.76	0.79	0.96	$(0.30, 0.20, 0.50, 0.40, 0.40)$
S_{32}	0.29	0.32	0.91	$(0.00, 0.00, 0.10, 0.20, 0.00)$
S_{33}	0.31	0.50	1	$(0.20, 0.20, 0.20, 0.20, 0.20)$

TABLE XII
HANDLING OF RST PACKETS

RST present		Action	Multiplier ζ_{ij}^t
$\mathbf{d}_j'$	$\mathbf{d}_i$		
yes	yes	–	$\pi_{i,b+1}^t(u_{j,b+1}')$
yes	no	ignore RST in $\mathbf{d}_j'$	$\pi_{i,b+1}^t(u_{j,b+1}')$
no	yes	–	1
no	no	–	1

OS has a RST, but the signature does not, this indicates a possible interference from an intermediate device (e.g., IDS after expiring connection state, scrubbers). In this case, it is likely meaningless to use the temporal characteristics of the RST, which is why we omit it from $\mathbf{d}_j'$ before computing the loss and delay probabilities. However, multiplication by $\pi_{i,b+1}^t(u_{j,b+1})$ is still warranted since we must assign a proper weight to this mismatch. The third row of the table corresponds to packet loss, which is handled automatically in $p_i^t(\gamma)$, i.e., no additional actions or multipliers are needed. Finally, the last row is identical to the setup assumed in preceding sections.

B. Final Model

We now combine the developed network, user, and RST models into a single framework. Redefining (20) as

$$p_{ij\tau\gamma}^t = \alpha_i^t \zeta_{ij}^t \Big(\prod_{v=1}^{b} \pi_{iv}^t(u_{jv}') \Big) f_T^t(\tau) p_i^t(\gamma) \prod_{r=1}^{|\mathbf{d}_j'|} f_\Delta^t(\delta_{ij\tau\gamma r}) \tag{49}$$

allows us to compute $\beta_{ij\tau\gamma}^t$ still via (21), as well as reuse (22)-(25). However, (48) requires an update to

$$\pi_{iv}^{t+1}(y) = \frac{\sum_{j=1}^m \mathbf{1}_{u_{jv}=y} \sum_{\tau\gamma} \beta_{ij\tau\gamma}^t}{m\alpha_i^{t+1}}, \tag{50}$$

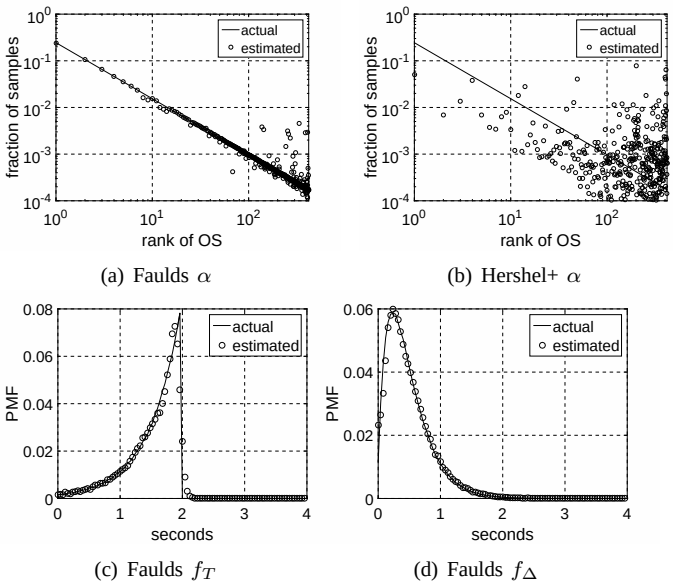
where $v = 1, 2, \ldots, b+1$. The final classifier, which we call *Faulds*[1], is applied after EM has converged and is given by

$$p(\omega_i | \mathbf{x}_j', \theta^\infty, \alpha^\infty) = \sum_{\tau\gamma} \beta_{ij\tau\gamma}^\infty. \tag{51}$$

It is easy to generalize our earlier results to cover the complete model, as given in the next theorem without proof.

Theorem 5: Under both network and user distortion, estimator (21)-(25), (49)-(50) is the EM algorithm for (θ, α).

[1]Henry Faulds was a Scottish scientist who extended the ideas of William Herschel and proposed the first usable forensic fingerprint-identification method in 1880.



(a) Faulds α (b) Hershel+ α

(c) Faulds f_T (d) Faulds f_Δ

Fig. 5. Results in D_4.

C. Scaling the Database

Due to the large number of features it combines, Faulds is not challenged by the previous toy databases. We therefore switch to a more realistic set of signatures created by Plata in [45]. We call this database D_4 and note that it contains 420 stacks, among which some have the same exact RTO vector and others overlap in *all* user features. The database was constructed to ensure that signatures were sufficiently unique under delay distortion, but packet loss and user modifications were not taken into account. As a result, the database contains a number of entries that would be difficult to distinguish under the types of heavy distortion considered in this paper. Nevertheless, these tests should indicate how well Faulds scales to larger databases and whether its recovery of the unknown parameters (α, θ) is affected by an increased uncertainty during the match.

We set popularity α to the Zipf distribution with shape parameter 1.2 and continue using $m = 2^{18}$ observations, which gives us 64K samples from the most common OS and just 49 from the least. We borrow the delay from case S_{13} (i.e., reverse-exponential T with mean 1.5 sec, Erlang(2) Δ with mean 0.5) and packet loss from S_{23} (i.e., reverse-binomial). Finally, we use RAND with stay probability $\phi_v = 0.8$.

The first iteration of Faulds produces a respectable $\rho^1 = 0.42$. This is gradually improved with each step, until convergence to a more impressive $\rho^\infty = 0.70$. To make sense out of α^∞, we sort all signatures in rank order from the most popular to the least and plot the result in Fig. 5(a). There is a strong match in the top-100, while the random noise in the tail is explained by the scarcity of these OSes in the observation (i.e., below 250 samples each). For comparison, the outcome of Hershel+ is displayed in part (b) of the figure. To complete the big picture, subfigures (c)-(d) show estimates of f_T and f_Δ. Despite an overall 30% classification mismatch, these PMFs are no worse than previously observed in Fig. 4, which indicates that incorrect decisions overwhelmingly went to signatures with similar RTO vectors as the true OS.

TABLE XIII
INJECTION CLASSIFICATION SUMMARY

Size of D_4	Injected samples	ρ^1_*	ρ^∞_*	p^∞_{loss}	$E[\phi^\infty_v]$
378 (90%)	7,089 (2.8%)	0.88	0.91	0.10	0.80
336 (80%)	49,648 (19.0%)	0.87	0.89	0.11	0.74
294 (70%)	60,058 (22.9%)	0.87	0.89	0.11	0.73
210 (50%)	91,408 (34.9%)	0.91	0.91	0.11	0.72
126 (30%)	189,293 (72.2%)	0.95	0.93	0.17	0.60

Instead of scrutinizing 21 different loss PMFs, suppose we compute a single metric – the fraction of packets dropped within the entire observation $\mathbf{x}'$, conditioned on at least one packet surviving. To this end, define during step t

$$L^t_k = \sum_{\ell=1}^{k-1} \ell q^t_k(\ell) \tag{52}$$

to be the average number of lost replies in signatures with k packets. Then, taking an estimated ratio of all dropped packets to the total transmitted yields the expected loss rate

$$p^t_{loss} = \frac{\sum_{i=1}^n \alpha^t_i L^t_{|\mathbf{d}_i|}}{\sum_{i=1}^n \alpha^t_i |\mathbf{d}_i|}. \tag{53}$$

Recall that the simulation allowed loss to affect at most $k-1$ packets in OSes with $|\mathbf{d}_i| = k$. Therefore, its ground-truth packet loss should represent the same quantity as (53). Traces show that 70.1% of the packets were dropped, which matches quite well against $p^\infty_{loss} = 69.3\%$.

Since $\phi_v = 0.8$ was a constant in this simulation, it makes sense to compare it against feature-modification estimates averaged across all fields and all OSes, i.e.,

$$E[\phi^t_v] = \frac{1}{b+1}\sum_{v=1}^{b+1} \phi^t_v = \frac{1}{b+1}\sum_{v=1}^{b+1}\sum_{i=1}^{n} \alpha_i \phi^t_{iv}. \tag{54}$$

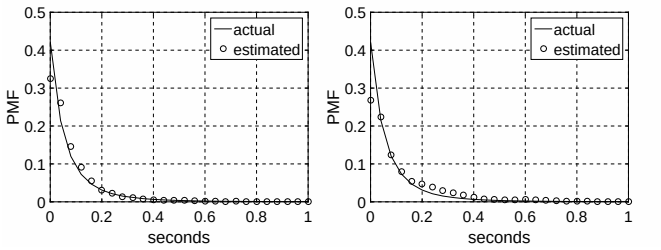
Results show that $E[\phi^\infty_v] = 0.802$, which is very close to the actual value. While there is some variation in individual ϕ_{iv}, it is of little concern due to the small number of samples seen by Faulds from these OSes.

D. Unknown Signatures

We recognize that having a database that knows all devices on the Internet is near impossible. Therefore, infiltration of samples from unknown signatures into $\mathbf{x}'$, which we call *injections*, is inevitable in practice. Understanding their impact is our next topic.

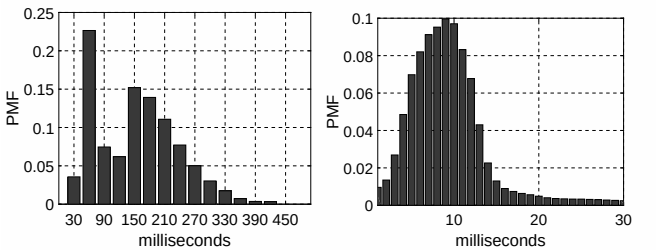
Suppose $\mathbf{x}'_j$ is produced by some unknown OS ω that does not belong to the database. If $\mathbf{x}'_j$ is so different from the known signatures that $p(\mathbf{x}'_j|\theta^t, \alpha^t) = 0$, i.e., it matches each OS with probability 0, its injection into the observation will contribute nothing to updates of (α^t, θ^t) and thus will have no impact on classification decisions. In order to achieve a flat-out mismatch of this type, either delay $\delta_{ij\tau\gamma}$ must be negative for all i, τ, γ or the product in (49) must be smaller than the precision of floating-point arithmetic.

For injections with $p(\mathbf{x}'_j|\theta^t, \alpha^t) > 0$ the situation is less clear-cut. In some cases, ω may be close to an existing signature ω_i, which makes injections minimally different from



(a) f_T (b) f_Δ

Fig. 6. Recovered delay under 72% injection.



(a) f_T (b) f_Δ

Fig. 7. Internet delay distributions.

distorted instances of $\mathbf{x}_i$. As a result, they do not negatively impact EM or its convergence point. On the other hand, it is also possible that $\mathbf{x}'_j$ is a potential match to multiple unrelated OSes and the amount of distortion needed to make them appear as $\mathbf{x}'_j$ is much greater than the underlying θ. If the volume of injections is high, how likely is EM to introduce bias into distributions of delay/loss to the point of impacting classification accuracy for *non-injected* samples?

We do not consider encountering of adversarial injections (i.e., special signatures crafted to cause maximum harm for a given database and classifier) to be likely in practice and instead focus on evaluating the effect of random subset removal from D_4. Specifically, assume the simulator produces distorted observations using all 420 network stacks; however, Faulds has access to only some of the original signatures. For the next simulation, we use Pareto f_T and f_Δ, both with mean 0.1 seconds, iid packet loss at 10%, and $\phi_v = 0.8$.

Define ρ^t_* to be the classification accuracy among non-injected observations during step t and consider Table XIII, which shows the shrunk database size, number of injected samples among $m = 2^{18}$ observations, and the output of Faulds. The result shows that removal of signatures does not carry a significant negative impact on accuracy of classification for the known OSes. In fact, ρ^t_* slightly rises as the database shrinks since it becomes easier to classify among fewer options. Packet loss p^∞_{loss} also appears immune, except in the last row where 72% of $\mathbf{x}'$ contains observations from unknown OSes. Its increase to 17% is explained by more frequent matches that require high packet loss to be feasible. Finally, the feature-stay probability in the last column is the most affected, which was also expected due to the increased header-field mismatch.

Fig. 6 shows the two delay PMFs estimated by Faulds in the last row of Table XIII. Recovery is quite good, except for a slight bump in f_Δ between 200 and 400 ms. This shows that removing 70% of the signatures in D_4 still leaves enough unique RTO vectors to produce highly accurate results. In the actual Internet, however, we do not expect injection conditions to be anywhere near these levels because D_4 contains an array of major network stacks (e.g., Windows, Unix), printer firmware (e.g., HP, Lexmark, Brother), Cisco equipment, and various derivative implementations that run on embedded devices. See [45] for more details.

VII. Internet Measurement

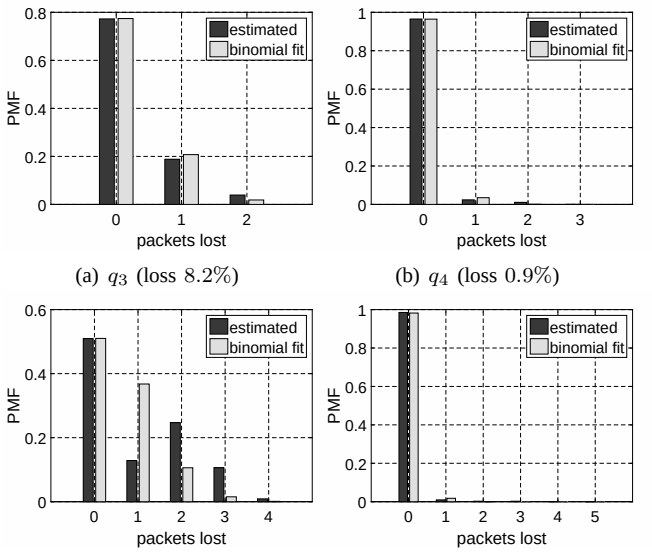
A. Overview

On December 14, 2016, we conducted a port-80 SYN scan of all BGP-reachable IPv4 addresses on the Internet. Of the 2.8B IPs contacted, we gathered responses from 67.6M hosts. Using a 16-core AMD Opteron @ 2.8 GHz, a parallelized C++ version of Faulds was able to process $3{,}801$ hosts per second. In large-scale classification, such as the one attempted here, Faulds produces a huge volume of information in the form of various PMFs and estimates. Due to limited space, we present only a brief review of the obtained results and leave more detailed analysis (including attempts to uncover injections and correct for them) for future work.We start with basic sanity checks of the estimated distortion θ and then delve into classification result α.

B. Network Distortion

Fig. 7(a) shows the recovered distribution f_T using bin size 30 ms. Delays below 60 ms (29%) represent unloaded servers in close proximity to the scanner, most likely within the continental US. Those in the $120 - 200$ ms range (40%) indicate targets whose RTTs are consistent with destinations in Europe and Asia. The remaining cases covers longer paths, OS scheduling delays, non-trivial CPU load on the server, and involvement of various backend databases to set up the connection. Overall, we obtain $E[T_j] = 148$ ms, 80% of the samples below 200 ms, and 99.2% below 450 ms. Fig. 7(b) plots the distribution of one-way delay f_Δ, in which 92% of the mass concentrates below 30 ms and 97% below 100 ms. The average queuing delay $E[\Delta_{jr}] = 15$ ms also sounds quite reasonable.

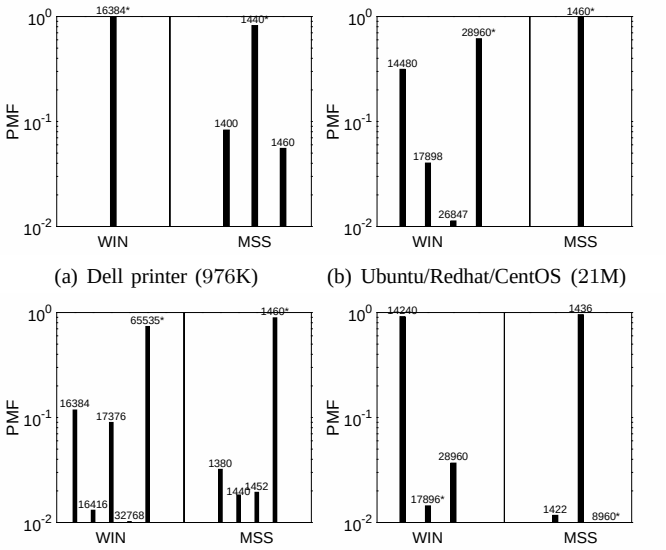
To examine packet loss, define $\eta^t_k = \sum_{i=1}^n \alpha^t_i \mathbf{1}_{|\mathbf{d}_i|=k}$ to be the estimated fraction of observations that use an OS with k packets. The top values of k are four ($\eta^\infty_4 = 0.42$, 112 stacks in Plata database D_4), six ($\eta^\infty_6 = 0.31$, 80 stacks), three ($\eta^\infty_3 = 0.07$, 72 stacks), and five ($\eta^\infty_5 = 0.04$, 54 stacks). Fig. 8 plots the recovered loss PMFs for these values of k, each fitted with an iid binomial model and accompanied by the average loss rate L^∞_k/k from (52). First, it is interesting that the loss rate is heterogeneous, ranging from 0.3% in q_6 to 12.6% in q_5. This phenomenon may be inherent to the signatures that map to each k (e.g., certain printers cut the SYN-ACK sequence when their tiny SYN backlog queue overflows [45]), the load on the corresponding OSes, and host location on the Internet, all of which suggests there is an extra benefit to estimating q_k



(a) q_3 (loss 8.2%) (b) q_4 (loss 0.9%)

(c) q_5 (loss 12.6%) (d) q_6 (loss 0.3%)

Fig. 8. Internet packet-loss PMFs.



(a) Dell printer (976K) (b) Ubuntu/Redhat/CentOS (21M)

(c) Mac OS X Server (67K) (d) CentOS (161K)

Fig. 9. Internet distributions π_{i1} and π_{i5} (default values have an asterisk).

independently for different k. Second, while in a few of the plots the binomial model shows a reasonable fit, this does not universally hold. Finally, computing (53) for the Internet scan yields an average loss rate of 3.3% across all observations. This is consistent with 3.8% found in a Google study of SYN-ACK retransmission rates [11].

C. User Distortion

Computing (54), we obtain $E[\phi_v^\infty] = 0.81$, i.e., the average probability to encounter a non-default value was 19%. Faulds produced $420 \times 6 = 2{,}520$ distributions of user features, among which we highlight several interesting cases, focusing on the two most volatile fields – Win and MSS – and limiting all PMFs to values above the 1% likelihood. Since MSS sometimes depends on the MTU of the underlying data-link layer and/or tunneling protocol (e.g., IPv6), this field may experience fluctuation even if the OS does not allow explicit means for changing this value.

We expected devices with firmware restrictions that prevent user access to the configuration of SYN-ACK parameters to exhibit high ϕ_{iv}. One example is shown in Fig. 9(a) for a popular Dell printer. Among 976K occurrences on the Internet, this device keeps the default window with probability 1. Intuition also suggests that general-purposes OSes are more susceptible to modification and/or existence of alternatively patched versions. One example is 21M hosts with Ubuntu Linux, where Fig. 9(b) shows that Faulds discovers 31% of the cases with window size exactly half of the default (i.e., $14{,}480$ instead of $28{,}960$). A more dispersed case is Mac OS X Server in part (c), which exhibits noticeable variation in both Win and MSS. Its default values remain with probability 73% and 89%, respectively. Finally, in subfigure (d), CentOS (enterprise Linux) has its original combination ($17{,}896$, $8{,}960$) occurring in only 1% of the cases. We conjecture that the Plata database [45], which was constructed from production devices in a large campus network, captured a non-standard version of this stack with jumbo Ethernet frames enabled. Since this is an inherent property of any database, it is important to allow great flexibility in the match process to accommodate such scenarios.

D. Classification Results

We define Faulds to be successful for sample j if the denominator of (1) is non-zero, i.e., $p(\mathbf{x}_j|\theta^t, \alpha^t) > 0$. This means that at least one OS matches $\mathbf{x}'_j$ with a non-zero probability. Using the Plata database with 420 network stacks [45], Faulds successfully classified 63.1M hosts (i.e., 93%). From a pure statistical point of view, the remaining 4.5M devices should be assigned to the OS with the highest α_i^∞. But it is also likely these cases come from unknown stacks or observations with too much packet loss, in which case excluding them from classification might be prudent as well, which is our approach below.

The left side of Table XIV shows the top ten OSes after one iteration of Faulds. Note that the Plata database was *auto-generated* from a pool of devices found at a university network. Even though this process [45] produced only a high-level description of each OS, additional *manual* effort can be used to provide each signature with a more specific kernel version and/or physical device. We consider this issue orthogonal to the topic of the paper since Faulds operates on TCP/IP signatures and its accuracy does not depend on the name affiliated with each fingerprint $\mathbf{x}_i$.

The dominance of Linux and embedded devices in Table XIV (left) matches the statistics reported in prior work [27], [45], [46], although a more interesting result is the amount of relative change occurring in the classification as Faulds goes through its iterations. Table XIV (right) shows the α vector after 100 steps. The top Linux signature gains 52%, Windows 7 in third place increases by 25%, and two other Linux stacks drop 17% each. Further down the list, there is significant movement as well, where certain embedded systems, such as

TABLE XIV
FAULDS CLASSIFICATION AT ITERATION 1 (LEFT) AND 100 (RIGHT)

OS	α^1	Count
Ubuntu / Redhat / CentOS	0.224	14,098,093
Ubuntu / SUSE / CentOS	0.111	8,896,622
Embedded Linux	0.082	6,326,349
Windows 7 / 2008 / 2012	0.047	2,942,254
Ubuntu / Redhat / SUSE	0.037	2,408,386
Schneider / APC Embedded	0.022	1,587,396
Windows XP / 2003	0.021	1,314,967
Redhat / CentOS / SUSE	0.018	1,254,797
Embedded Linux	0.015	1,044,028
Windows 2008 R2 / 2012	0.013	907,167

$\longrightarrow$

OS	α^{100}	Count	Change
Ubuntu / Redhat / CentOS	0.334	21,361,956	0.52
Embedded Linux	0.103	6,467,303	0.02
Windows 7 / 2008 / 2012	0.056	3,669,372	0.25
Schneider / APC Embedded	0.055	3,632,638	1.29
Ubuntu / Redhat / SUSE	0.031	2,001,329	−0.17
Windows XP / 2003	0.018	1,248,619	−0.05
Redhat / CentOS / SUSE	0.016	1,046,567	−0.17
Dell Laser / Xerox WorkCenters	0.015	976,717	0.25
Windows 2008 R2 / 2012	0.014	837,466	−0.08
Cisco Embedded	0.013	824,039	2.29

TABLE XV
TYPES OF DEVICES RUNNING WEBSERVERS

Device Type	Count	Fraction
General purpose	42,277,294	67%
Switch/router/gateway/network controller	8,854, 290	14%
No label in database	7,038, 785	11%
Printers	2,813, 292	4.5%
RAID controller/NAS	1,348, 895	2.1%
Video conferencing/telepresence	603,035	1.0%
Cyberphysical systems	91,033	0.14%
IP phones	61,400	0.10%

TABLE XVI
UNPROTECTED INDUSTRIAL AND ENTERPRISE DEVICES

Device	Count	Type
Polycom HDX 8000 HD	266,565	Telepresence
Hickman ITV 450D	67,091	Telepresence
Cisco Unified IP Phone 7900 Series	27,151	IP Phone
AVTech RoomAlert/Rockwell Automation	21,756	Cyberphysical
Loytec L-DALI Lighting Control Systems	20,517	Cyberphysical
Codian Telepresence MCU	20,036	Telepresence
Polycom RealPresence Server 4000	18,977	Telepresence
AdTran IP Phone Manager	11,909	IP Phone
HWg-STE: Ethernet thermometer	11,826	Cyberphysical
D-Link DCS Series Internet Camera	9,279	Telepresence

Schneider APC (data-center hardware solutions), Dell printers, and Cisco, increase their membership by $25-229\%$. There is even more shuffle outside the top-10, which underscores the importance of using proper algorithms for estimating α.

Table XV splits all classified hosts into eight categories. The top two signatures are desktop/server OSes and various stacks from network-device manufacturers (i.e., switches and routers). In third place, there are 7M hosts with no label, which means Faulds finds a matching signature for each of them, but Plata does not know what these devices are. The bottom half of the table, with a substantial count of cyber-physical systems and office equipment, is more alarming. These oftentimes run on default manufacturer passwords and allow reconfiguration using a built-in webserver. Investigating further, Table XVI shows the top-ten signatures from these categories, which include camera systems, building lighting controllers, and temperature monitors. They present high security risks to organizations because malicious actors may be able to use these systems to gain access to workplace audio/video recordings, printed documents, and environmental settings of critical infrastructure (e.g., cooling in data-centers).

With the recent leaks of NSA exploits and massive worldwide infection by ransomware WannaCry [24], [34], outdated

TABLE XVII
OSES WITH EXPIRED SUPPORT LIFE CYCLES

OS	Count	Released
Windows 2000 / XP / 2003	1,512,725	2000 / 2001 / 2003
FreeBSD 7.3 / 8.0	433,978	2010 / 2009
Windows Server 2003 SP1 SP2	195,169	2005 / 2007
Windows Server 2000 SP4/XP SP3	146,421	2003 / 2008
FreeBSD 6.4	71,190	2008
Solaris 9 / Solaris 10	78,269	2003 / 2005
Mac OS X 10.4	36,834	2005
Windows 2000/XP SP1	9,623	2001 / 2002
Novell Netware OES 2 SP1	1,108	2005

operating systems (i.e., Windows XP/Server 2003) gained renewed attention. In Table XVII, we show several signatures that have reached the end of support and are no longer being patched to keep up with the latest vulnerabilities. These are obvious security threats; however, we find over 1.8M old Windows hosts still visible over the public Internet, 500K FreeBSD, and 78K Solaris. Faulds not only allows for a timely measurement of such devices, but also paves the way for scalable, low-overhead Internet characterization, robust device identification, and better modeling of distortion θ experienced by the numerous hardware artifacts found on the Internet.

VIII. CONCLUSION

In this work, we developed novel theory and algorithms for improving OS-classification accuracy in single-probe fingerprinting, measuring one-way Internet path properties, and extracting latent distributions of feature distortion. Simulations showed exceptional robustness of our EM techniques against various types of noise, as well as injection of unknown devices. Applied to Internet scans, this methodology can be used to detect vulnerable devices, as well as estimate stack popularity, network delays, packet loss, and header-tuning probabilities.

Future work involves construction of fingerprint databases with specimens that are pairwise separable under more complex distortion than just delay, detection of unknown stacks among the observations, automatic generation of signatures for them, and extensive comparison against nmap.

REFERENCES

[1] H. Abdelnur, R. State, and O. Festor, "Advanced Network Fingerprinting," in *Proc. RAID*, Sep. 2008, pp. 372–389.
[2] A. Aksoy and M. H. Gunes, "Operating System Classification Performance of TCP/IP Protocol Headers," in *Proc. IEEE LCN*, Nov. 2016, pp. 112–120.

[3] Apple Support, "OS X Yosemite: Prevent others from discovering your Mac." [Online]. Available: https://support.apple.com/kb/PH18642?locale=en_US.

[4] O. Arkin, "A Remote Active OS Fingerprinting Tool using ICMP," *USENIX login*, vol. 27, no. 2, pp. 14–19, Apr. 2002.

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

[6] T. Beardsley, "Snacktime: A Perl Solution for Remote OS Fingerprinting," Jun. 2003. [Online]. Available: http://www.packetfu.com/wp/snacktime.html.

[7] R. Beverly, "A Robust Classifier for Passive TCP/IP Fingerprinting," in *Proc. PAM*, Apr. 2004, pp. 158–167.

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

[9] 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.

[10] 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*, Nov. 2014, pp. 173–180.

[11] H. K. J. Chu, "Tuning TCP Parameters for the 21st Century," Jul. 2009. [Online]. Available: http://www.ietf.org/proceedings/75/slides/tcpm-1.pdf.

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

[13] A. Dempster, N. Laird, and D. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society*, vol. 39, no. 1, pp. 1–38, 1977.

[14] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A Search Engine Backed by Internet-Wide Scanning," in *Proc. ACM CCS*, Oct. 2015, pp. 542–553.

[15] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer *et al.*, "The Matter of Heartbleed," in *Proc. ACM IMC*, Nov. 2014, pp. 475–488.

[16] Z. Durumeric, E. Wustrow, and J. Halderman, "ZMap: Fast Internet-wide scanning and its Security Applications," in *Proc. USENIX Security*, Aug. 2013, pp. 605–620.

[17] 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.

[18] X. Feng, Q. Li, H. Wang, and L. Sun, "Characterizing Industrial Control System Devices on the Internet," in *Proc. IEEE ICNP*, Nov. 2016, pp. 1–10.

[19] S. Guoqiang and D. Lee, "Network Protocol System Fingerprinting: A Formal Approach," in *Proc. IEEE INFOCOM*, Apr. 2006, pp. 1–12.

[20] H. O. Hartley, "Maximum Likelihood Estimation from Incomplete Data," *Biometrics*, vol. 14, no. 2, pp. 174–194, 1958.

[21] 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.

[22] Kaspersky Labs, "Targeted Cyberattacks Logbook." [Online]. Available: https://apt.securelist.com.

[23] M. Kearns, Y. Mansour, and A. Ng, "An Information-Theoretic Analysis of Hard and Soft Assignment Methods for Clustering," in *Proc. Uncertainty in Artificial Intelligence*, Aug. 1997, pp. 282–293.

[24] Z. Kleinman, "Cyber-attack: Is my computer at risk?" *BBC News*, May 2017. [Online]. Available: http://www.bbc.com/news/technology-39896393.

[25] 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.

[26] E. Kollmann, "Chatter on the Wire: A Look at DHCP Traffic." [Online]. Available: http://myweb.cableone.net/xnih/download/chatter-dhcp.pdf.

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

[28] Z. Li, A. Goyal, Y. Chen, and V. Paxson, "Automating Analysis of Large-Scale Botnet Probing Events," in *Proc. ACM AsiaCCS*, Mar. 2009, pp. 11–22.

[29] 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.

[30] J. Matherly, "Shodan Search Engine." [Online]. Available: https://shodan.io.

[31] T. Matsunaka, A. Yamada, and A. Kubota, "Passive OS Fingerprinting by DNS Traffic Analysis," in *Proc. IEEE AINA*, Mar. 2013, pp. 243–250.

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

[33] J. Medeiros, A. Brito, and P. Pires, "An Effective TCP/IP Fingerprinting Technique Based on Strange Attractors Classification," in *Proc. DPM/SETOP*, Sep. 2009, pp. 208–221.

[34] Microsoft Technet, "Microsoft Security Bulletin MS17-010 – Critical." [Online]. Available: https://technet.microsoft.com/en-us/library/security/ms17-010.aspx.

[35] Microsoft Technet, "Stealth Mode in Windows Firewall with Advanced Security." [Online]. Available: https://technet.microsoft.com/en-us/library/dd448557(WS.10).

[36] A. Mirian, Z. Ma, D. Adrian, M. Tischer, T. Chuenchujitasphon, T. Yardley, R. Berthier, J. Mason, Z. Durumeric, and J. A. Halderman, "An Internet-Wide View of ICS Devices," in *Proc. IEEE PST*, Dec. 2016, pp. 96–103.

[37] NetApplications, "Market Share Statistics for Internet Technologies." [Online]. Available: http://netmarketshare.com/.

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

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

[40] G. Prigent, F. Vichot, and F. Harrouet, "IpMorph: Fingerprinting Spoofing Unification," *Journal in Computer Virology*, vol. 6, no. 4, pp. 329–342, Nov. 2010.

[41] A. Quach, Z. Wang, and Z. Qian, "Investigation of the 2016 Linux TCP Stack Vulnerability at Scale," in *Proc. ACM SIGMETRICS*, Jun. 2017, pp. 3:1–3:19.

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

[43] G. Rouallend and J.-M. Saffroy, "IP Personality." [Online]. Available: http://ippersonality.sourceforge.net/.

[44] S. Shah, "An Introduction to HTTP Fingerprinting," May 2004. [Online]. Available: http://net-square.com/httprint_paper.html.

[45] Z. Shamsi and D. Loguinov, "Unsupervised Clustering Under Temporal Feature Volatility in Network Stack Fingerprinting," in *Proc. ACM SIGMETRICS*, Jun. 2016, pp. 127–138.

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

[47] U. Shankar and V. Paxson, "Active Mapping: Resisting NIDS Evasion Without Altering Traffic," in *Proc. IEEE S&P*, May 2003, pp. 44–61.

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

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

[50] A. K. Sood and R. J. Enbody, "Targeted Cyberattacks: A Superset of Advanced Persistent Threats," *IEEE S&P*, vol. 11, no. 1, pp. 54–61, Jan. 2013.

[51] G. Taleck, "Ambiguity Resolution via Passive OS Fingerprinting," in *Proc. RAID*, Sep. 2003, pp. 192–206.

[52] G. Taleck, "SYNSCAN: Towards Complete TCP/IP Fingerprinting," *CanSecWest*, Apr. 2004.

[53] 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.

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

[55] 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.

[56] M. Zalewski, "Strange Attractors and TCP/IP Sequence Number Analysis," Apr. 2001. [Online]. Available: http://lcamtuf.coredump.cx/newtcp/.

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