SCALABLE OS FINGERPRINTING: CLASSIFICATION PROBLEMS AND APPLICATIONS

A Dissertation

by

ZAIN SARFARAZ SHAMSI

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2017

Major Subject: Computer Science

ABSTRACT

The Internet has become ubiquitous in our lives today. With its rapid adoption and widespread growth across the planet, it has drawn many research efforts that attempt to understand and characterize this complex system. One such direction tries to discover the types of devices that compose the Internet, which is the topic of this dissertation.

To accomplish such a measurement, researchers have turned to a technique called *OS fingerprinting*, which is a method to determine the operating system (OS) of a remote host. However, because the Internet today has evolved into a massive public network, large-scale OS fingerprinting has become a challenging problem. Due to increasing security concerns, most networks today will block many of the probes used by traditional fingerprinting tools (e.g., Nmap), thus requiring a different approach. Consequently, this has given rise to single-probe techniques which offer low overhead and minimal intrusiveness, but in turn require more sophistication in their algorithms as they are limited in the amount of information that they receive and many parameters can inject noise in the measurement (e.g., network delay, packet loss).

This dissertation focuses on understanding the performance of single-probe algorithms. We study existing methods, formalize current problems in the field and devise new algorithms to improve classification accuracy and automate construction of fingerprint databases. We apply our work to multiple Internet-wide scans and discover that besides general purpose machines, the Internet today has grown to include large numbers of publicly accessible peripheral devices (e.g., routers, printers, cameras) and cyber-physical systems (e.g., lighting controllers, medical sensors). We go on to recover empirical distributions of network delays and loss, as well as likelihoods of users reconfiguring their devices. With our developed techniques and results, we show that single-probe algorithms are an effective approach for accomplishing wide-scale network measurements.

To my family.

ACKNOWLEDGMENTS

I must express my gratitude towards several people without whom this dissertation could not have been completed. First and foremost is my advisor, Dr. Dmitri Loguinov, who has been my patient guide from my first steps into this undertaking and my navigator for this journey during the past several years. Due to his systematic direction and sage advice, I have learned how to solve research problems using a scientific approach as well as present my results to the academic community. He provided me with the necessary independence one must have as a doctoral student, but also instruction and supervision when it was needed. I have seen him strive for perfection in understanding the most minute details, and his passion for tackling difficult problems has been infectious, leading me to seek the same answers in my own work and bring forth new ideas and discoveries that paved the way for this research to flourish. I will certainly benefit from his teachings for the rest of my career.

I am thankful to Dr. Daren Cline for his help with the mathematical proofs required for Chapter 5, as well as Dr. Riccardo Bettati, Dr. Radu Stoleru and Dr. Narasimha Reddy for serving on my committee and providing their insightful feedback on my research. This sentiment also extends to the anonymous reviewers from ACM SIGMETRICS and IEEE/ACM Transactions on Networking, who contributed valuable judgment and suggestions for improving previous versions of this work.

Completing the various large scale Internet studies for this research required the help of Willis Marti and the network security team at Texas A&M, as well as Brad Goodman and his IT team in our CSE department. They accommodated me with the equipment, technical support, and freedom required to conduct my measurements for which I am immensely grateful. My lab mates – Xiaoyong Li, Tanzir Ahmed, Yi Cui, and Di Xiao played a significant part in this work as well. Besides constructive conversations about research, they helped me take breaks from the long hours of study in the lab and enjoy graduate student life.

While those mentioned above ensured this endeavor could be completed, this journey would have never begun if it were not for my family. My father, who has my utmost respect for showing me how success can be built from the ground up, my mother, who instilled in me the values of perseverance and education, and my wife, who inspired me to pursue my ambitions, have formed the backbone on which this work stands. They encouraged me to ignite a spark, and have been instrumental in keeping the fire lit these past years. Their unending love kept me going during all the ups and downs, and their boundless support made this goal look achievable even from its most distant point. Without them, this work would certainly not exist.

Finally, I wish to acknowledge *you*, the reader. For it is people like you, scholars in their fields, that I aspire to support in the never-ending pursuit of knowledge with this dissertation. If I can impart to you even a small bit of what I have learned over the course of writing this dissertation, I will consider my mission a success. Thank you.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professors Dmitri Loguinov, Riccardo Bettati and Radu Stoleru of the Department of Computer Science and Professor A.L. Narasimha Reddy of the Department of Electrical Engineering.

The data analyzed for Section 3 was obtained from previous efforts by Derek Leonard [45]. Assistance and advice for the proofs presented in Section 5 was provided by Professor Daren Cline of the Department of Statistics. All other work conducted for the dissertation was completed by the student.

Funding Sources

Graduate study was supported by a graduate assistantship from Texas A&M University.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Overview

With the rapid growth of the Internet, our world has become a connected grid of heterogenous 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 interesting topic that has gained traction in the literature [28], [32], [42], [55], [61], [67], [77]. To categorize the devices that compose today's networks, researchers have employed a technique called *OS fingerprinting*, which aims to determine the operating system of remote hosts using their responses to external stimuli.

While the signals used in OS fingerprinting can be based on network protocols such as DNS [62], ICMP [4], [73] and DHCP [51], [78], our focus is on the TCP/IP stack. This is because the TCP/IP implementation greatly differs not only between OS families (e.g., Linux, Windows and Mac), but also versions and patches of the same OS (e.g., Windows XP vs. Vista and Linux 2.4 vs. 3.0). This is explained by the freedom allowed in selection of certain default stack parameters, ambiguities in IETF RFCs [13], [81], [83] as well as a lack of standardization for responses to malformed requests. The methods using TCP/IP can be partitioned into three categories – *banner-grabbing* via plain-text protocols (e.g., telnet, HTTP, FTP) [92], *multi-probe* tools that elicit OS-specific responses from various non-standardized combinations of flags and/or unexpected usage of protocol fields (e.g., SinFP [6], [10], [38], [65], Nmap [73], [91], [104], SYNSCAN [105], Xprobe [120], [121], p0f [122]), and *single-probe* methods that send only one legitimate SYN to each host (Snacktime [7], RING [112]).

For usage at large scale over the Internet, banner-grabbing is no longer considered viable due to frequent removal of OS-identifying strings by administrators for security purposes, high bandwidth overhead, and common interaction with non-platform-specific application-layer software (e.g., apache, nginx). Multi-probe tools have their own challenges – heavy load on the target (e.g., over 100 packets in Nmap), massive complaints about intrusive activity when used at wide scale, and reduced accuracy when firewalls block auxiliary probes (e.g., UDP to a closed port, rainbow flags in TCP headers) or the destination IP is load-balanced across a server farm (i.e., different packets hit different machines). As we show later in Section 4, multi-probe OS classification over the public Internet is a complex and poorly understood problem, with certain pitfalls and unintended side-effects.

Thus, in this dissertation we focus on examining the scalability of OS fingerprinting to millions of target hosts. With this goal in mind, the next subsection presents the structure of the rest of this dissertation.

1.2 Dissertation Structure

Figure 1.1 shows the three main topics we will study in this work. Since our objective is to examine the viability of OS fingerprinting on a large scale such as the Internet, we require a classifier that is fast, low-overhead and does not trigger IDS systems and harass network administrators. We turn to the methodology used by single probe classifiers, which use only one outbound TCP SYN packet and require a response from an active port on the host. However, this approach introduces several challenges due to limited features, loss of packets, and non-negligible queuing and processing delays encountered in communication with the target.

To overcome these challenges, our first topic in Section 3 describes our approach to building a stochastic model to handle these obstacles. We then combine our models into a classifier we call *Hershel* and subject it to various scenarios in simulations to test its accu-

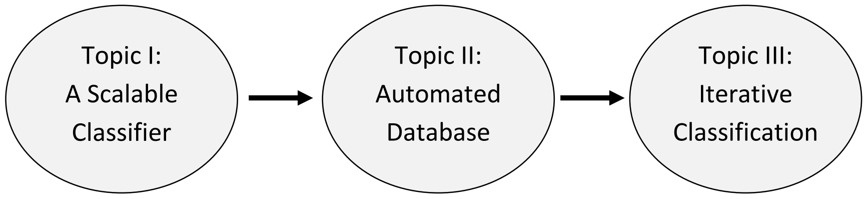


Figure 1.1: Dissertation structure.

racy. This is accomplished by building a database of 116 OS stacks, which were manually installed within our lab or identified within our department network, and adding noise to each signature. We show that our models can provide reliable classification even under extreme scenarios (e.g., half of all responses are lost, multiple second delays). Satisfied with our results, we then use Hershel to classify 37M hosts from an Internet scan, showing the distribution of devices we encountered in different countries and AS regions of the world. Finally, we also show that our classifier is robust against scrubbers that aim to confuse OS classification.

To be adaptable to the ever-evolving Internet, we require our techniques to be versatile and allow for different sizes and compositions of networks. Thus, for our next topic, we target the issue of building a database flexible to the network to be measured. Since our previous effort of 116 stacks was a manual process that was susceptible to human error and poor repeatability, our goal in Section 4 is to develop an automated, repeatable process for building a database. We propose a novel unsupervised clustering algorithm called *Plata* to separate unique signatures and discard duplicate ones. We show how this works by applying Plata to a scan of our university network, and automatically create a database of 420 OSes, which are labelled by a banner download from compliant hosts. We also update Hershel to correctly treat independence between the delays observed for each packet, giving rise to a new classifier we call Hershel+. We close out this section by conducting a large Internet study to fingerprint 66 million webservers, the first such effort to use an automatically built database, and comparing our results with Nmap, a well known multi-packet classifier.

The final topic builds on both the previous by building a complete classification system that does away with user defined heuristics. For example, Hershel relies on assumed probability distributions of noise observed in the Internet measurement, such as network one-way delays, packet loss, popularity of each OS, and user modification of network stacks. Section 5 develops an iterative classifier under the Expectation-Maximization (EM) framework called *Faulds*, and shows that the OS fingerprinting problem can successfully be modeled under EM to leverage its convergence properties. Using extensive simulations, we also show that recovery of the true distributions of observed noise is not only possible, but results in much more precise classification. Finally, we conclude by conducting a measurement where besides outputting the correct OS popularity, we obtain packet loss/delay and feature modification probabilities from 63 million hosts across the Internet – a study from a unique perspective which opens up new angles for Internet measurement in the future.

2. RELATED WORK

Besides use in various applications of Internet measurement [11], [30], [52], [59], OS fingerprinting is a well known approach used by network administrators for securing their networks. It has been used to discover vulnerable network services [63], [100], detect rogue systems and stealth intruders [1], defend against target-based fragmentation attacks [74], [97], and even expose botnets [56], [69]. It is also used by industry analysts to understand trends in OS usage [5], [31] and market share analysis by public tools [70], [71]. Below we take a look at the previous work done in this field.

2.1 Multi-Probe Techniques

OS fingerprinting has roots in *banner grabbing*, which relies on application-layer protocols (e.g., HTTP, SSH, SMTP, FTP, telnet) to provide a textual description of the OS as part of the communication sequence after successful handshakes. While this worked well in the 1990s, banner grabbing today faces many impediments, including high overhead, administrator ban on OS-identifying strings in responses, generic software (e.g., apache, nginx) that can run on multiple platforms without exposing the underlying OS, and purposefully incorrect banners that aim to mislead the various fingerprinting tools.

The second wave of OS classification started in 1997 with the release of Nmap [73], which pioneered TCP/IP tricks that would elicit different responses from different implementations. By default, it sends 1032 probes to the target, including a vertical port scan and certain malformed packets that trigger popular IDS such as Snort [89]. Nmap ideally expects the target to accept a TCP connection, send ICMP port unreachable on a closed UDP port, and respond to a ping. Under bandwidth-optimized settings for OS classification, Nmap requires no fewer than 38 different probes; however, due to mandatory retransmission, this in practice corresponds to well over 100 packets per host.

Due to its popularity, Nmap has received a great deal of attention in the literature, which includes usage of neural networks to differentiate between versions of the same OS [91], detection of unknown devices [64], and techniques for reducing the number of sent probes [35]. Additional work includes fuzzy matching [120], application of formal testing methods to the detection problem [38], and classification using lengthy observations (up to 100K packets) of Initial Sequence Numbers (ISNs) from the TCP header [65], [121].

Another direction in multi-packet classification uses clock drift in the kernel, which can be derived from observing the timestamp option in streams of reply packets [50] or variation in timer frequency [17]. This approach requires sending a steady stream of requests to discern the accurate skew, which can add up to a large overhead (i.e., thousands of packets) and requires handling of randomness introduced in the replies by OS scheduling. Thus, while this approach has its uses in localized networks, it is hardly scalable to millions of targets.

Besides the amount of traffic generated by multi-packet tools in large scale-scans, another problem is the prevalence of load balancers in the Internet today. These devices, commonly placed in front of servers, may disperse consecutive probes to different physical machines or perform certain elements of the handshake themselves, leading to jumbled fingerprints. This can be avoided by scanning techniques that rely on one outgoing packet, which we describe next.

2.2 Single Probe Algorithms

RING [112] and Snacktime [7] are the only tools that perform classification using temporal features from a single outbound probe.[1] As shown in Fig. 2.1, each measurement consists of a SYN packet, server processing delay T needed to accept the connection, and a stream of n SYN-ACK responses from the target OS, followed by an optional TCP reset

[1]Nmap [73] used to rely on temporal features, but later stopped supporting them due to classification difficulties.

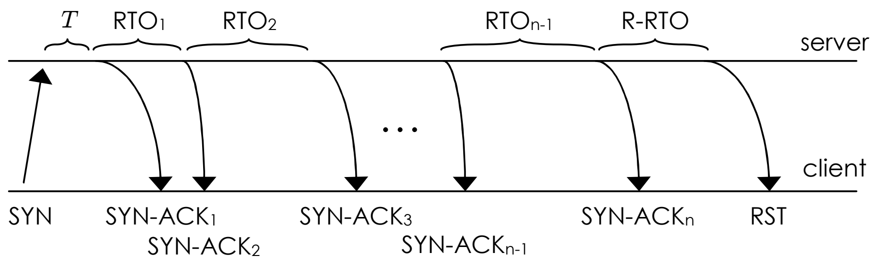


Figure 2.1: Retransmission timeouts (RTOs) between SYN-ACK packets.

(RST) with its own RTO. RING uses the $n-1$ values in the RTO vector and presence of the final RST packet in classification. Snacktime ignores the RST feature, but instead uses the default TCP window size and TTL carried in the SYN-ACKs, which allows it to differentiate between 25 operating systems [7]. We analyze its classification process in more detail later in Section 3. A simplified version of Snacktime and extension to 98 signatures was offered in [45], [55]; however, no accuracy analysis, modeling, or verified improvement was provided.

Another tool with a related capability is p0f [122]. In addition to passive fingerprinting, it can actively generate SYN packets and profile remote network stacks based on a set of fixed features from the SYN-ACKs (i.e., window size, TTL, IP flags, and TCP options); however, it does not leverage the RTOs and by default is quite verbose (i.e., sends eight copies of the same SYN per target). The current version can differentiate between 18 operating systems.

2.3 Common Defenses

There exist many approaches to thwart remote OS fingerprinting. The most basic tools tweak Windows registry [20], [75] or implement plugins [8], [9], [90] for the Unix packet-mangling module Netfilter [68]. Their objective is to modify the fixed features of departing packets to no longer resemble those of the underlying host. A similar direction is to deploy network honeypots [85], [110] or standalone systems [115] that spoof arbitrary operating systems and their services. Placing obfuscation into the network gives rise to intermediate devices known as *fingerprint scrubbers* [84], [101].

While these techniques can effectively deal with static header fields, they are not well suited for distorting the temporal features of departing packets, which requires expensive buffering of packets and per-flow state. Additionally, lack of technical support and possibility for various side-effects (e.g., disabling SACK in TCP may lead to significantly lower throughput) raises questions about deployment of these tools in production systems and/or at large-scale. Nonetheless, we study the impact of these scrubbers on our work in more detail in Section 3.5.

3. LARGE-SCALE OS CLASSIFICATION*

3.1 Introduction

The Internet has been the target of numerous measurement studies, with the trend recently shifting from covering a small subset of destinations [77], [86] to scanning the entire IP space [22], [42], [55], [87]. This allows researchers to enumerate live hosts, detect vulnerabilities, and shed light on deployment of new protocols. Over the years, network scanning has become progressively faster – from 4 months [87] down to 30 days [42], then one day [55], and now 45 minutes [29]. In conjunction with these studies, low-overhead OS fingerprinting can allow significantly better understanding of the systems researchers interact with and improve our general knowledge about the Internet.

OS fingerprinting consists of two approaches – *passive* and *active*. The former [50], [122] monitors ongoing communication (inbound and/or outbound) with remote hosts, but does not generate traffic of its own. Unless each studied device voluntarily connects to the measurement server, this technique is difficult to use for classifying each IP on the Internet. The latter approach, which is our topic of interest, actively sends packets to targets and infers their operating system from the collected responses.

One important aspect that differentiates between the active methods is the potential maliciousness of probing traffic, where certain nonsensical combinations of TCP flags (e.g., SYN-FIN-RST-ACK) or intrusive actions (e.g., trying to delete the root directory in HTTP fingerprinting [92]) may harm or crash the target. Additionally, these packets are easily detected and dropped by IDS [102], which leads to complaints against research institutions using these methods and possibly reduced accuracy of the results.

The second aspect is the amount of outbound traffic required by the classifier, which

*Reprinted, with permission, from "Hershel: Single-Packet OS Fingerprinting" by Z. Shamsi, A. Nandwani, D. Leonard and D. Loguinov which appeared in [95] and [96]. © 2016 IEEE.

ranges from a single SYN probe [7], [112] to lengthy multi-packet exchanges [65], [73], [92], [105], [120]. Ideally, fingerprinting should be performed with no extra overhead to scan traffic, which rules out techniques [73], [120] that expect to reach the target on multiple open ports, using different protocols (e.g., ICMP, TCP, UDP), and elicit responses on closed ports. While LAN environments can tolerate high traffic rates and may allow multi-protocol access to each host, these conditions are generally difficult to satisfy when scanning the entire Internet.

The third aspect is the ability of the underlying estimator to correctly identify the target OS under realistic network conditions and without using retransmission. Since prior single-packet techniques [7], [112] were mainly developed for local use, they are not well provisioned to overcome high amounts of fluctuation and loss in temporal features. They also lack resilience to OS tuning, which can be applied by end-users in hopes of optimizing network performance or obfuscating the default parameters of the stack. Either way, the modified OS features may exhibit little correlation to those originally present at the host, which cripples estimation accuracy of existing tools.

3.1.1 Contributions and Ethical Implications

Given the many open issues in wide-scale fingerprinting and lacking performance analysis in the literature, our first goal is to formalize the estimation problem in single-packet OS classification and study the pitfalls of existing techniques. We then develop a low-overhead framework we call *Hershel*[1] for overcoming the various randomization effects (i.e., queuing delays, packet loss, manual tuning) and apply it as proof-of-concept in a measurement study that classifies every visible webserver on the Internet.

We next discuss the ethical implications of this work. Our main objective is to benefit researchers studying the Internet at wide scale and provide a solution to an interesting

[1]William J. Herschel invented forensic usage of fingerprints in 1858.

mathematical problem. However, one may become concerned that intruders can use our algorithms for detection of vulnerable operating systems and better tailor the attack payload to particular configurations (e.g., patch levels) of the targets. As opposed to Nmap, our techniques require no additional bandwidth during port scanning, which makes them completely stealthy against IDS and other security monitors.

While hypothetically this may be true, we do not believe there is great cause for concern. With modern botnets, large-scale port scanning can be performed in a highly decentralized fashion, with very little traffic originating from each hijacked IP. This affords the attackers a luxury of using more verbose OS fingerprinting tools (i.e., Nmap) and still remaining undetected. Researchers, on the other hand, are typically constrained to a single subnet whose generation of disruptive volumes of highly anomalous traffic is bound to attract negative attention.

Additionally, we are not aware of any evidence confirming that attackers are interested in profiling discovered devices using only SYN packets. Recent studies [124] show that once an open port is found, bots either perform more extensive testing of the open service or attempt all known exploits (some outdated by decades) against the port without discrimination. Eliminating Nmap from the picture and directly interacting with the service is much quicker and more informative in that context. We therefore do not see OS fingerprinting as a practical technique for increasing maliciousness of the Internet ecosystem.

3.2 Stochastic Model

We assume a single-packet scanner similar to Snacktime in Fig. 2.1. While this approach has minimal intrusiveness, lowest transmission overhead, and non-malicious operation, it also exhibits several fundamental challenges. These arise due to the complex ways in which the RTOs can be modified by packet traversal across wide-area networks, scarcity of information about the target host contained in the samples, and user tuning of features, all of which has a strong influence on one's ability to detect the underlying OS.

It should be noted that straightforward application of machine-learning methods [108] to our problem is difficult. Experimentation with support vector machines, neural networks, and decision trees has led to the realization that they perform poorly when the measured samples contain missing data (i.e., the RTO vector is corrupted by packet loss). Statistical imputation [34] is a common technique for dealing with these problems; however, it requires knowing *which* features are missing and ability to accurately reconstruct the *remaining* (non-missing) features. In our case, lost packets go completely unnoticed and additionally modify the following RTOs to produce feature vectors that have little resemblance to the original (see below).

Our contribution in this subsection is to formalize single-packet OS fingerprinting, set forth clear goals for the classifier, study the impact of network delay and loss on the measured samples, analyze the existing methods, and outline the assumptions under which the classification problem is tractable.

3.2.1 Objectives

Assume a database $\mathcal{D} = (1, 2, \ldots, M)$ of $M \geq 1$ known operating systems, where each OS j has some vector-valued fingerprint y_j collected during a-priori measurement of the OS. The fingerprint consists of multiple features, which we partition into those modified only by the network (e.g., RTOs) and those only by the user (e.g., TCP window size). Suppose the former are described by some vector δ_j and the latter by another vector u_j. While the length of δ_j normally depends on j, that of u_j is constant across all operating systems.

As both vectors undergo random modification before being observed by the scanner, the response of OS j to probe traffic is some random variable that is a function of y_j. Given an observation $x = (\delta; u)$ from an Internet host, a typical estimation problem is to find the most likely OS $s(x)$ that could have produced that vector:

$$\begin{aligned} s(x) := \operatorname*{argmax}_{j \in \mathcal{D}} \; p(y_j|x) &= \operatorname*{argmax}_{j \in \mathcal{D}} \frac{p(x|y_j)p(y_j)}{p(x)} \\ &= \operatorname*{argmax}_{j \in \mathcal{D}} \; p(x|y_j)p(y_j), \end{aligned} \tag{3.1}$$

where notation $p(x|y)$ refers to the probability (or conditional density, if more convenient) of x given y. Observe that the probability $p(x)$ that some OS in $\mathcal{D}$ has produced x is constant for a given observation and can be omitted from the optimization. If the fraction of Internet hosts $p(y_j)$ running OS j is unknown, it is common to set each value to $1/M$, which removes this term from the optimization as well.

The more interesting component of (3.1) is the probability $p(x|y_j)$ that OS j has produced the observation, or equivalently that y_j has become distorted into x. Before investigating this metric further, observe that network and user modifications to the OS features can be treated as independent, from which it follows that:

$$p(x|y_j) = p(\delta|\delta_j)p(u|u_j). \tag{3.2}$$

This means that the two terms can be dealt with separately, which we do in the rest of the section.

3.2.2 Network Features: Jitter

For single-packet techniques [7], [112] described in 2.2, the vector of temporal features δ_j consists of individual RTOs generated by network stack j. Classification based on δ_j is possible not only because some devices deviate from TCP algorithms (e.g., exponential timer backoff), but also because RFCs that govern TCP retransmission [13], [81], [83] do not specify the initial RTO or how many SYN-ACKs must be generated. As a result, a

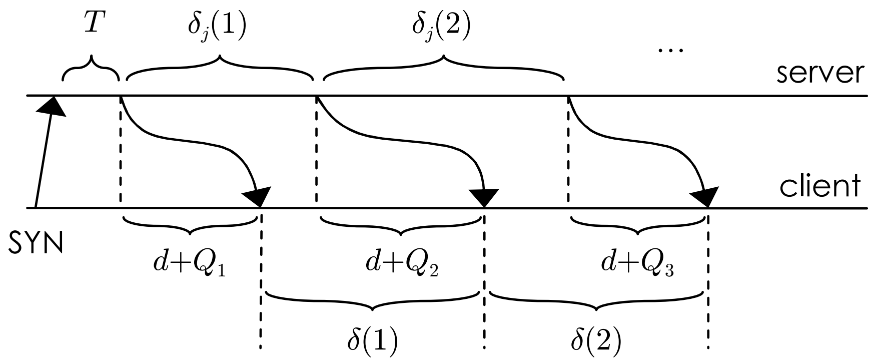


Figure 3.1: Effect of jitter on observed RTOs.

wide variety of unique RTO patterns exists.

For the time being, assume loss-free conditions. During collection of sample x, suppose d is the sum of propagation and transmission delays along the path from the server back to the scanner. Note that d is a constant due to the fixed size of SYN-ACKs. Now define Q_m to be a random queuing delay of the m-th packet in the return path. As shown in Fig. 3.1, the RTO vector δ_j undergoes distortion that is independent of the forward path, server think time T, and propagation delay d:

$$\delta(m) = \delta_j(m) + Q_{m+1} - Q_m, \quad m = 1, 2, \ldots, |\delta_j| \tag{3.3}$$

Defining OWD (one-way delay) jitter $J_m = Q_{m+1} - Q_m$ and considering that the gap between subsequent SYN-ACKs is sufficiently large (i.e., at least several seconds), it follows that back-to-back packets arriving from the server are not likely to encounter the same busy period of the queues they traverse. In that case, it is reasonable to assume that sequence $Q_1, Q_2, \ldots$ consists of independent and identically distributed (iid) random variables. Furthermore, since the number of hops and congestion of the path is not affected by j, the distribution of each Q_m does not depend on the OS being profiled. This leads to:

$$p(\delta|\delta_j) = \begin{cases} \prod_{m=1}^{|\delta|} f(\delta(m) - \delta_j(m)) & |\delta| = |\delta_j| \\ 0 & \text{otherwise} \end{cases}, \tag{3.4}$$

where $f(.)$ is the PDF (probability density function) or PMF (probability mass function) of OWD jitter, depending on whether J_m is treated as continuous or discrete. It should also be noted that $Var[J_m] = 2Var[Q_m]$, while $f(.)$ is zero-mean and symmetric. For certain models of OWD, jitter can be obtained in closed-form. For example, exponential Q_m produces the Laplace distribution with the same parameter λ and Gaussian $N(\mu, \sigma^2)$ becomes $N(0, 2\sigma^2)$.

We next contrast (3.4) with the RTO classifier in Snacktime [7], which is a tool that is the closest to our objectives and most advanced in single-packet OS fingerprinting. For each RTO m, this method first computes the number of matching digits (limited to 6 decimal places of precision) between the sample and all known fingerprints j:

$$Y_{jm} = \max(\lceil -\log_{10}(\max(|\delta(m) - \delta_j(m)|, 10^{-6}))\rceil, 0).$$

It then assigns score W_j to OS j using the sum of these weights across all available RTOs:

$$W_j = \sum_{m=1}^{|\delta|} Y_{jm}. \tag{3.5}$$

For the example in Table 3.1, which exemplifies the common pitfalls of Snacktime, (3.5) scores six for the first OS and two for the second OS, indicating that jitter combination $(0, 12)$ is more likely than $(0.1, 0.1)$. Taking the $\log$ of (3.4), our model can also be

	RTO$_1$ (sec)	Y_{j1}	RTO$_2$ (sec)	Y_{j2}	W_j
Observation δ	3.0		24.0		
Fingerprint δ_1	3.0	6	12.0	0	6
Fingerprint δ_2	2.9	1	23.9	1	2

Table 3.1: Snacktime example.

reduced to optimizing a summation:

$$\log p(\delta|\delta_j) = \sum_{m=1}^{|\delta|} \log f(\delta(m) - \delta_j(m)); \tag{3.6}$$

however, it differs from (3.5) in two important ways. First, the $\log$ is applied to the distribution function $f(.)$ rather than the jitter itself. Second, there is no loss of precision due to rounding to the nearest integer or capping the jitter at 10^{-6}.

Nevertheless, while (3.4) is a good starting point, it does not work in real networks due to the lacking robustness against packet loss. This is our next topic.

3.2.3 Network Features: Loss

The main problem with (3.4) is that loss-free conditions are impossible to satisfy during Internet scans. Besides congestion, routing loops, and various checksum violations, the RTOs may be altered by the target server crashing or shutting down during the measurement, which affects the tail of the RTO vector and appears similar to packet loss. Since single-packet fingerprinting *by definition* cannot retransmit SYN probes, OS detection must be performed using only the features available in observation x, which calls for more sophistication in the model.

To exacerbate the situation, packet loss creates more dramatic changes to the RTO vector than delay jitter. For example, consider a scenario with $\delta_j = (3, 6, 12)$, where all delays are given in seconds. Even with a relatively large $E[Q_m] = 100$ ms, delay jitter remains small compared to each RTO. On the other hand, the loss of a single packet produces one of four dissimilar combinations – $(3, 6)$, $(3, 18)$, $(6, 12)$, or $(9, 12)$ – while that of two packets leads to six additional options – (3), (6), (9), (12), (18), or (21). The RTO swing in these cases is significantly higher, which makes mapping x to the correct OS more challenging.

We now examine how to model the combined probability that loss and jitter transform δ_j into observation δ. This will allow us to solve such dilemmas as whether $\delta = (3, 18)$ is a more likely match to $(3, 6, 12)$ with one lost packet or to some other signature $(2.6, 17.9)$ without any loss. To deal with these cases, we propose to generalize the concept of RTO. First, let τ_j be a vector of $|\delta_j| + 1$ packet-transmission timestamps from OS j:

$$\tau_j(m) = \begin{cases} 0 & m = 1 \\ \tau_j(m-1) + \delta_j(m-1) & m \geq 2 \end{cases} \tag{3.7}$$

and τ be the corresponding random vector observed in x after the packets have traversed the network. Then, a generalized $(m, m+k)$-RTO is the distance $\tau_j(m+k) - \tau_j(m)$, which is illustrated in Fig. 3.2 for $m = 1$ and $k = 2$. Note that $k = 1$ produces the usual RTO and that all timestamps are given using local clocks (i.e., τ_j at the server and τ at the client).

Now suppose set $\Gamma(\tau, \tau_j)$ contains all subsets of size $|\tau|$ of integer sequence $(1, 2, \ldots, |\tau_j|)$. We can view each $\gamma \in \Gamma(\tau, \tau_j)$ as a mapping of received packets in τ to their position in the original vector τ_j, i.e., $\gamma(m) = k$ means that the m-th received SYN-ACK was initially in position k. For the example in Fig. 3.2, we have $\gamma(1) = 1$ and $\gamma(2) = 3$. Assuming no reordering of SYN-ACKs, which is reasonable given at least several seconds between them, each γ is a vector of strictly increasing integers.

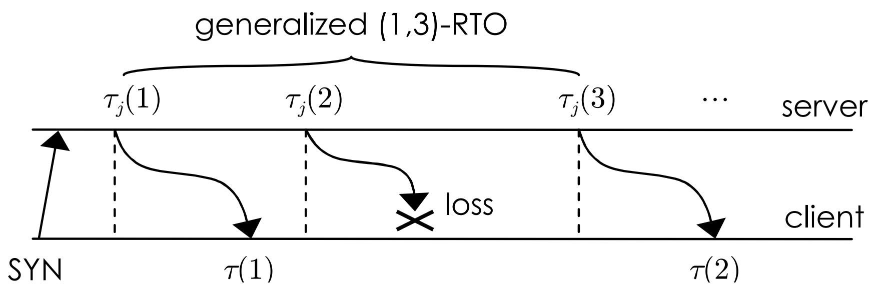


Figure 3.2: Generalized RTOs under packet loss.

Armed with these definitions, we get:

$$p(\tau|\tau_j) = \begin{cases} \sum_{\gamma \in \Gamma(\tau,\tau_j)} p(\gamma)p(\tau|\tau_j,\gamma) & |\tau| \le |\tau_j| \\ 0 & \text{otherwise} \end{cases}, \tag{3.8}$$

where the number of summation terms equals the number of ways to select $|\tau|$ objects from $|\tau_j|$ available options and (3.8) is non-zero only if the number of received packets does not exceed that in the fingerprint. This is in contrast to (3.4), where the two vectors had to have equal length.

Again leveraging the large spacing between server responses, we can treat congestion events affecting SYN-ACKs as independent, which allows one to approximate packet loss as an iid Bernoulli process with some probability q. Since each loss combination is equally likely, we get:

$$p(\gamma) = q^{|\tau_j|-|\tau|}(1-q)^{|\tau|}, \tag{3.9}$$

which can be moved outside the summation in (3.8). To deal with $p(\tau|\tau_j,\gamma)$, which is the probability to observe τ from OS j under loss pattern γ, notice that the gap between each adjacent pair of received packets is determined by the generalized RTO:

$$\tau(m) - \tau(m-1) = \tau_j(\gamma(m)) - \tau_j(\gamma(m-1)) + J'_m, \tag{3.10}$$

where $m \geq 2$ and generalized jitter J'_m is given by:

$$J'_m = Q_{\gamma(m)} - Q_{\gamma(m-1)}. \tag{3.11}$$

Rearranging the terms in (3.10), define the m-th jitter sample under pattern γ as:

$$R^{\gamma}_{jm} = \tau(m) - \tau(m-1) - \tau_j(\gamma(m)) + \tau_j(\gamma(m-1)). \tag{3.12}$$

Noticing that J'_m has the same distribution as J_m yields:

$$p(\tau|\tau_j, \gamma) = \prod_{m=2}^{|\tau|} f(R^{\gamma}_{jm}). \tag{3.13}$$

We thus get for $|\tau| \leq |\tau_j|$:

$$p(\tau|\tau_j) = q^{|\tau_j|-|\tau|}(1-q)^{|\tau|} \sum_{\gamma \in \Gamma(\tau,\tau_j)} \prod_{m=2}^{|\tau|} f(R^{\gamma}_{jm}), \tag{3.14}$$

which replaces $p(\delta|\delta_j)$ in (3.2).

3.2.4 User Features

OS tuning is common practice in the current Internet, with numerous online guides recommending optimizations to network settings [76], [109] and automated software offering tuning capabilities to the TCP/IP stack to achieve better performance [27]. A number of fixed header parameters in general-purpose kernels (e.g., Unix, Windows) can be changed through registry or using command-line tools. Unlike jitter-induced noise, where small distortions are generally more likely that large ones, the main difference with OS tuning is that *there may be no correlation between the manually selected values of the user and those installed in the OS by default*. For example, TCP window size may be more likely to jump from 8192 to 65535 than to 8193.

While accurate modeling of manual modification and human psychology is difficult, it makes sense for the analysis to at least take into account whether a given feature under user control has been changed. Suppose that π_m is the probability of such modification in feature m and assume that user tuning is applied independently to each available parameter. Defining $I_{jm} = \mathbf{1}_{\{u(m)=u_j(m)\}}$ to be an indicator of the event that the m-th measured feature matches the original of OS j, we get:

$$p(u|u_j) = \prod_{m=1}^{|u|} \Big[(1-\pi_m)I_{jm} + \pi_m(1-I_{jm})\Big]. \tag{3.15}$$

Besides user interference, vector u_j may be modified by intermediate devices along the path (e.g., NAT, IDS, fingerprint scrubbers [20], [84], [90], [101], [115]), whose actions can be clumped under the same umbrella of (3.15). Since buffering packets for periods of time comparable to RTO (i.e., $3-6$ seconds) and per-flow state are expensive, it is often safe to assume that these devices do not alter the RTO pattern in significant ways and thus leave enough features by which the OS can still be identified. This underscores the importance of having a robust RTO estimator.

The Snacktime algorithm for scoring user-modified features can be generalized as a sum of weights assigned to each match:

$$W_j' = \sum_{m=1}^{|u|} w_m I_{jm} = \sum_{I_{jm}=1} w_m, \tag{3.16}$$

which is added to the RTO score W_j in (3.5) for a final result. One open issue, however, is selection of proper weights, which need to be somehow correlated with feature volatility. Our model is much simpler since π_m directly provides this probability. To better understand the difference between (3.15) and (3.16), assume that $\pi_m > 0$ for all m and write:

$$\log p(u|u_j) = \sum_{I_{jm}=1} \log(1-\pi_m) + \sum_{I_{jm}=0} \log \pi_m. \tag{3.17}$$

For $\pi_m \approx 1$, we get $\log \pi_m \approx 0$, the second term of (3.17) disappears, and our model reduces to Snacktime with weights $w_m = \log(1-\pi_m)$. However, in more realistic cases of $\pi_m \ll 1$, the second term of (3.17) becomes non-negligible and serves the role of balancing non-matching features against those that do match. Snacktime has no such mechanism.

3.2.5 Final Result

We now consolidate the various models into one formula. Combining (3.14) and (3.15) in (3.2) and (3.1), dropping terms that do not depend on j, and performing straightforward manipulations, we get:

$$\begin{aligned} s(x) = \operatorname*{argmax}_{j \in \mathcal{D}: |\tau| \le |\tau_j|} \Bigg\{ & p(y_j) q^{|\tau_j|-|\tau|} \sum_{\gamma \in \Gamma(\tau,\tau_j)} \prod_{m=2}^{|\tau|} f(R^{\gamma}_{jm}) \\ & \times \prod_{I_{jm}=1} (1-\pi_m) \prod_{I_{jm}=0} \pi_m \Bigg\}. \end{aligned} \tag{3.18}$$

Although (3.18) maximizes the OS-detection probability under the assumptions stated above, its performance with a-priori-unknown q, π_m, $f(.)$, and $p(y_j)$ is an open question. We return to it later in the section; in the meantime, we outline the various remaining issues.

3.2.6 Limitations

First, the SYN packet may be lost and never reach the target. Since there is no way to verify this, the host will automatically be considered non-responsive and will be excluded from fingerprinting. Not much can be done to overcome this problem unless SYN retransmission is allowed. If we relax the single-packet assumption, the estimator will face the problem of determining which of the SYNs triggered which SYN-ACK response, without which the RTOs cannot be computed correctly. This problem can be solved in the future by encoding the retransmission attempt into the source port of the SYN.

Second, our model allows only the *network* to modify the received RTOs; however, this may not hold if users manage to alter SYN-ACK spacing during OS tuning. This is not of wide-spread concern as few optimization guides target the RTO pattern. With enough effort, scrubbers and obfuscation tools can disrupt inter-SYN-ACK delays; however, we do not consider development of end-to-end methods to combat such approaches a fruitful objective. A related problem arises with middleboxes and caches that accept the connection on behalf of the server [43], in which case any fingerprinting tool is bound to classify only the visible side of the TCP stream (i.e., the OS of the middlebox).

Third, Hershel's accuracy may deteriorate if the network jitter process J_m becomes non-iid or deviates from the predicted bounds, e.g., due to significant kernel scheduling latency during CPU overload. Similar issues may surface if network loss depends on j, users modify different operating systems with different probability, or there is correlation in loss events within a single stream of SYN-ACKs. Solving these problems requires a per-OS set of parameters $(q_j, f_j(.), \pi_{jm})$, which is our focus in Section 5.

3.3 Classifier

Our next contribution is to enhance Snacktime's feature vector, describe a working classifier based on the theory developed above, bring awareness to RTO randomization performed by certain OSes, and explain how to collect signature databases under these conditions.

3.3.1 Features

Snacktime uses only two non-RTO features – TCP advertised window size and TTL; however, additional parameters are readily available from the SYN-ACKs. Following Table 3.2, these include the Do Not Fragment (DF) flag in the IP header, four different fields from the RST packet (more on this below), the Maximum Segment Size (MSS) declared by TCP, the order in which the OS assembles the option fields (OPT), SYN-ACK RTOs (SA-RTO), and the RST RTO (R-RTO). Some of these features are self-explanatory, but others require additional elaboration.

First, it should be noted that the initial TTL cannot be reconstructed exactly at the receiver. We use the common technique of rounding this value up to the nearest "likely" boundary, which includes four values used by the OSes in our database $\mathcal{D}$ – 32, 64, 128, and 255. Second, the reset features are quite rich. In Table 3.2, the binary flag RST is 1 for the fingerprints that contain a reset packet, RA indicates whether the RST has the ACK bit set, RN is 1 if the ACK sequence is non-zero, and RW records the window of the reset packet. RST features represent peculiarities of internal stack operation and cannot be modified via OS tuning. However, fingerprint scrubbers, NAT/IDS, and kernel recompilation can still change them.

Third, as seen in the table, support for TCP options differs between the operating systems since no specific subset is required to be implemented [46]. More importantly, users have the freedom to disable them as needed. As certain options are considered security risks (e.g., timestamps), they may be disabled by default, although users can still re-enable them. Certain devices (e.g., printers) do not allow OPT tweaking at all, while newer versions of popular operating systems tend to support fewer choices. For example,

Operating system	Win	TTL	DF	Reset				MSS	OPT	SA-RTO	R-RTO
				RST	RA	RN	RW				
Windows 7	8192	128	1	1	0	1	0	1460	MNWST	3, 6	12
Linux 2.6	5792	64	1	0	–	–	–	1460	MSTNW	3.8, 5.9, 12.1, 24, 48.2	–
Linux 2.0	32736	64	0	0	–	–	–	1414	M	3, 6, 12, 24, 48, 96	–
Mac OS 10.3	33304	64	1	1	1	1	32768	1460	MNWNNT	2.92, 6, 12, 24	30
NetBSD 4.0.1	32768	64	1	0	–	–	–	1460	MNWNNTSNN	2.92, 6, 12, 24	–
VxWorks 5.4.2	8192	64	0	1	1	1	8192	512	MNW	5.58, 24	45
Juniper Netscreen	8192	64	0	1	0	0	8192	1380	M	1.67, 2, 2, 2, 2, 2, 2, 2	2

Table 3.2: Sample signatures.
(M = MSS, N = NOP, W = window scale, S = SACK, T = timestamp)

even though Windows 7/2008 provides registry keys to disable TCP timestamps, the modification does not work. Similarly, SACK can be disabled only if the entire TCP stack is offloaded to the NIC [66].

What makes OPT a good feature is not the specific string, but rather *the order in which non-padding options appear*. This is illustrated in Table 3.3, where we progressively disable various combinations of options and observe the resulting SYN-ACK packets. For example, Windows XP supports four options MWTS. Turning off W produces MTS interspersed by NOPs as padding. Simplicity of implementation and lacking reasons to reorder the options suggests that this phenomenon likely exists in other stacks.

As a result, OPT requires a more advanced classification logic than straight comparison. Specifically, a match is registered if the observed sample x contains a *feasible* string, which we examine by taking an intersection of non-NOP options between x and each fingerprint, followed by verification that the order of the resulting letters is the same. For example, MTW is a match to Linux, VxWorks, and Juniper in Table 3.2, but not the other OSes.

Fourth, the reset RTO (R-RTO) helps in resolving additional ambiguities, such as between Mac OS 10.3 and NetBSD 4.0.1 in Table 3.2, which otherwise have identical SA-RTO patterns. Additionally, we expand Snacktime's default measurement time limit from 65 seconds to 120, the latter of which is the MSL (Maximum Segment Lifetime) of TCP [83]. For instance without considering the 96-second RTO of Linux 2.0 in Table 3.2, it might be hard to differentiate it from Linux 2.6.

Table 3.4 summarizes the features used in our classification and compares them to those in Nmap, p0f, and Xprobe [73], [105], [112], [120], [122]. We have four novel features and one match type (subset) never used in fingerprinting before.

Operating system	All enabled	Drop S	Drop T	Drop W	Drop ST	Drop SW	Drop WT	Drop all
Linux 2.6	MSTNW	MNNTNW	MNNSNW	MST	MNW	MNNT	MNNS	M
Windows XP/2003	MNWNNTNNS	MNWNNT	MNWNNS	MNNTNNS	MNW	MNNT	MNNS	MNW
Windows 7/2008	MNWST	–	–	MST	–	–	–	–
FreeBSD 8.2	MNWST	MNWNNT	–	–	–	–	MSE	M
Solaris 10	NNTMNWNNS	NNTMNW	–	–	–	–	–	–

Table 3.3: Examples of transformations applied by the OS to TCP options (dashes indicate impossible cases).

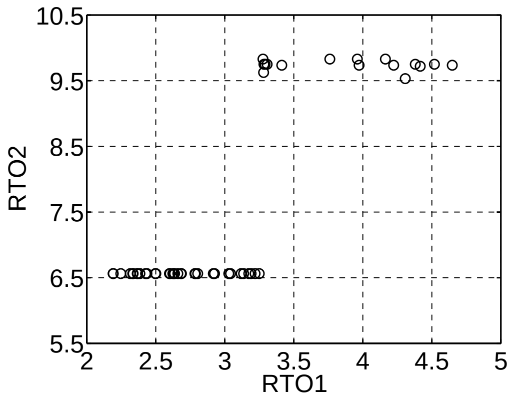
Feature	Description	Appeared In
Win	Receiver window	[7], [73], [105], [120], [122]
TTL	Time-to-live field	[7], [73], [105], [120], [122]
DF	Do Not Fragment	[73] [105] [120], [122]
SA-RTO	RTO sequence	[7], [105], [112]
RST	True if RST packet	[112]
MSS	Max segment size	[73], [105], [122]
OPT	TCP options (exact)	[73], [122]
RA	ACK bit in RST	New
RN	ACK seq $\neq 0$ in RST	New
RW	Window in RST	New
OPT	TCP options (subset)	New
R-RTO	RTO of RST packet	New

Table 3.4: Enhanced feature vector.

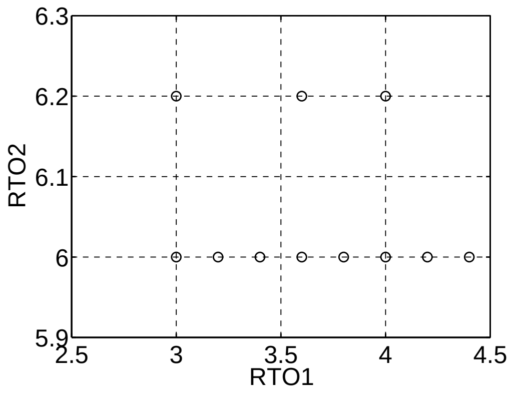
3.3.2 Stochastic Timers

Table 3.2 shows SA-RTOs from a single captured sample of the OS; however, it turns out that many kernels naturally exhibit significant RTO variation, sometimes by as much as 50%. Two examples are shown in Fig. 3.3 using a 2D scatter plot of the first two SA-RTOs. For Server 2003 in subfigure (a), there are two distinct patterns – the lower left corner, with RTO$_1$ distributed in $[2.2, 3.3]$ and RTO$_2$ frozen at 6.56, and the upper section, with RTO$_1$ scattered in $[3.3, 4.6]$ and RTO$_2$ in $[9.5, 9.8]$. Furthermore, the two scenarios are not equally likely as the bottom one occurs 68% of the time. This shows that the temporal model must take into account not just the possible RTO regions, but also their likelihoods.

A similar picture emerges for Linux 2.6 in subfigure (b). The mass of the RTO is now concentrated on 11 distinct points, where RTO$_1$ ranges from 3 to 4.4 seconds and RTO$_2$ from 6 to 6.2. Again, the popularity of individual points is non-uniform, swinging from 2% to 16%. Note that both cases in Fig. 3.3 have been collected from idle hosts over a single-hop network consisting of one switch, *which makes this behavior part of the fingerprint itself rather than an artifact of the sampling environment*.



(a) Windows Server 2003



(b) Linux 2.6

Figure 3.3: RTO randomness in TCP/IP scheduler.

Possible reasons for this fluctuation are the absence of per-connection RTO timers during the SYN-ACK phase and discretization of retransmission delays. What these examples show is that internal OS operation is a complex stochastic system that requires measuring the RTO *distribution* (rather than a single snapshot) during creation of the signature database. This is necessary because such large variations are not taken into account by the jitter model, which normally assumes OWDs on the order of tens or hundreds of milliseconds, with similarly sized jitter.

Our approach is to treat RTOs as random variables, unlike prior work that has always considered them deterministic. Specifically, suppose OS j has w_j unique types of behavior, each occurring with probability β_{jr}, where $r = 1, 2, \ldots, w_j$. We call each of these types a *subOS* and assign it a separate RTO vector τ_{jr}, which updates (3.13) to:

$$p(\tau|\tau_j, \gamma) = \sum_{r=1}^{w_j} \beta_{jr} p(\tau|\tau_{jr}, \gamma). \tag{3.19}$$

A simpler technique is to measure each host w times and let each obtained RTO vector

τ_{jr} be a subOS with $\beta_{jr} = 1/w$. In that case, (3.19) becomes:

$$p(\tau|\tau_j, \gamma) = \frac{1}{w} \sum_{r=1}^{w} \prod_{m=2}^{|\tau|} f(R_{jrm}^{\gamma}), \tag{3.20}$$

where R_{jrm}^{γ} is the generalized jitter of the m-th RTO under subOS r of OS j and loss-pattern γ. Note that summations involving $\Gamma(\tau, \tau_j)$ remain the same since all subOSes within a given OS send a fixed number of SYN-ACKs. They also exhibit deterministic user features, which keeps (3.15) unchanged.

3.3.3 Fingerprint Database

In order to produce an accurate fingerprint τ_j, the OS must be measured in some isolated testbed with low end-to-end delays and idle conditions at the server. To avoid loss-related bias, each host must be sampled multiple times to determine the longest vector of RTOs it produces, which should then be used to collect w loss-free samples for the database. Following these guidelines, we installed a variety of commodity operating systems in our lab, determined the proper size of their RTO vectors, and collected $w = 50$ clean fingerprints from each. We also captured a number of embedded devices found in our department LAN.

While Snacktime ships with 25 signatures and [55] uses 98, our database contains 116 network stacks. We can distinguish not only between different operating systems (e.g., Windows, Linux, FreeBSD), but also sometimes identify their versions and patches (e.g., Windows Server 2003 with and without SP1, MacOS 10.3 vs MacOS 10.4).

3.3.4 Hershel

Our classification method, which we call *Hershel*, builds upon (3.18) and (3.20), where we treat all $w = 50$ subOSes as deterministic. Common sense suggests that users, scrubbers, and network devices are not likely to directly tweak individual RST features RA,

Win	TTL	DF	OPT	MSS	RST
39.2	4.7	1.8	1.1	5.5	14.4

Table 3.5: Classification accuracy (percent) of isolated features.

RN, and RW; instead, these fields (if modified at all) will be simultaneously replaced with another set that comes from a different OS. We thus combine all four RST values in Table 3.2 into one atomic feature for classification purposes. This makes vector u_j consist of six fields – Win, TTL, DF, MSS, OPT, and aggregated RST. Table 3.5 shows the accuracy of individual features across the entire database (all ties are broken uniformly randomly).

RTO vectors τ and τ_j include timestamps of all SYN-ACKs and the first RST (if present). To account for resets that might be injected by firewalls/IDS after they time out the connection, (3.8) and (3.15) require a revision. Specifically, if the measured vector τ contains a reset, but τ_j does not, the RST is removed from τ prior to computing (3.8). To account for the mismatch in the RST feature, (3.15) gets multiplied by π_6. In the opposite case, i.e., τ_j contains a RST, but τ does not, it is important to avoid mistaking packet loss for changes in the RST feature and improperly penalizing $p(u|u_j)$ with π_6. Next, if both vectors contain a reset packet, (3.15) gets hit with either π_6 or $1 - \pi_6$ depending on the match in (RA, RN, RW). Finally, if neither vector has a RST, then (3.15) enjoys multiplication by $1 - \pi_6$.

3.4 Simulations

Our contribution in this subsection is to explain how to select the parameters of the model and examine Hershel's accuracy in simulations in comparison to Snacktime.

3.4.1 Parameters

For lack of a better assumption, we suppose that all OSes are equally likely to appear in the trace and set $p(y_j) = 1/M$ to be a uniform PMF. While it is possible to consider multiple iterations and refine this value after each pass, the resulting system sometimes exhibits instability and divergence into inferior states. We analyze these stability issues further in Section 5, but for now perform a single iteration in our evaluation below.

We use $\pi_m = 0.01$ for RST and OPT, while keeping $\pi_m = 0.1$ for the other features. The rationale is that RST behavior and option ordering can be changed only through kernel source-code modifications and usage of aggressive intermediate devices, neither of which we believe is that common in today's Internet compared to stack tuning. For queuing delay, we use a simple exponential distribution with CDF $1 - e^{-\lambda x}$ whose mean is set to 0.5 seconds (rate $\lambda = 2$). This produces Laplace jitter density:

$$f(z) = \frac{\lambda}{2} e^{-\lambda |z|}. \tag{3.21}$$

Note that usage of $\lambda = 2$ is fairly pessimistic, with the majority of paths likely exhibiting significantly smaller delays. For example, this model assumes 82% of the paths produce over 100 ms queuing delays, 37% over 500 ms, and 14% over 1 second. For packet loss, we use Google's study [18] to set $q = 3.8\%$, which was their highest rate of SYN-ACK loss.

3.4.2 Results

Our next goal is to examine Hershel's robustness in the presence of OWD jitter, packet loss, and random feature modification by the user. We also aim to assess the sensitivity of results to our choices of default parameters above. We simulate a FIFO queue between the server and the client with a given delay distribution. Each packet is dropped by the router with some probability q_{real} and each feature is independently modified with another probability π_{real}. Since these are per-packet and per-feature metrics, it also makes sense to examine the fraction $\chi = E[(1 - q_{real})^{|\tau_j|}](1 - \pi_{real})^6$ of all generated samples that do not have any loss or feature modification, where the expectation is taken over all j.

The distribution of popularity $p_{real}(y_j) \sim j^{-\alpha}$ is set to Zipf with shape parameter $\alpha = 1.2$, which approximates the fact that some OSes are much more popular than others. We do not attempt to make our assignment of index j to each physical OS such that its $p_{real}(y_j)$ closely follows that in the Internet (which is unknown anyway); instead, the simulation simply verifies performance of the proposed estimator when the OS frequency is highly non-uniform. For that purpose, random ordering of OSes in the database is sufficient.

Table 3.6 shows classification accuracy for several scenarios of interest. We examine three types of OWD with mean μ in the first column – Pareto $1 - (1 + x/\beta)^{-\alpha}$ with $\alpha = 3$ and $\beta = \mu(\alpha - 1)$, exponential with rate $1/\mu$, and uniform in $[0, 2\mu]$. We use the original Snacktime since the simplified version from [45] performs worse. Using just the RTOs, Snacktime in the table starts at close to 13%, but then deteriorates below 1% near the bottom. This amounts to essentially guessing across the 116 available options (i.e., $1/116 = 0.86\%$). Augmented with Win and later TTL, Snacktime begins at a more healthy $52 - 58\%$, but then eventually reduces to single digits.

The next six columns show Hershel with its default $\lambda = 2$. Classifying just based on the RTO vector, Hershel doubles Snacktime's accuracy in the first three scenarios (i.e., the first 12 rows of the table), triples it in the next one, and improves by an order of magnitude in the last one. As additional features are added, Hershel becomes even better, with significant gains seen at the Win and OPT boundaries. This shows that unlike DF, option strings form an orthogonal dimension to Win/TTL. The MSS improves the result further by 3% and the RST packet by an additional $0.5 - 3\%$, with the impact mostly limited to high-loss cases.

Staying with $\lambda = 2$, observe that Hershel is quite insensitive to selection of $f(z)$. Specifically, classification accuracy improves *not* when λ equals $1/\mu$ or the PDF of real delay matches (3.21), but *as μ gets smaller or the tail of the delay gets lighter*. This can

OWD	μ	Snacktime			Hershel $\lambda = 2, q = 0.038$							Hershel	Hershel
distribution	(sec)	RTO	+Win	+TTL	RTO	+Win	+TTL	+DF	+OPT	+MSS	+RST	$\lambda = 10$	$q = 0.1$
$q_{real} = 0, \pi_{real} = 0$ ($\chi = 100\%$)													
Pareto	0.5	12.6	51.8	58.3	22.1	81.4	86.2	88.5	96.2	99.72	99.72	94.62	99.69
Exp	0.5	12.8	51.8	58.3	21.9	82.6	86.9	89.4	96.5	99.92	99.94	96.21	99.82
Uniform	0.5	13.0	51.9	58.4	21.7	84.1	87.4	89.8	96.8	99.99	99.99	98.50	99.99
Pareto	0.1	16.3	56.9	62.9	33.1	93.0	94.9	96.7	99.0	99.99	99.99	99.69	99.99
$q_{real} = 3.8\%, \pi_{real} = 0$ ($\chi = 84\%$)													
Pareto	0.5	10.0	43.4	49.0	21.4	78.5	85.1	87.7	96.1	99.69	99.69	94.68	99.66
Exp	0.5	10.1	43.4	49.0	21.5	80.1	85.6	88.1	96.3	99.76	99.82	96.21	99.80
Uniform	0.5	10.3	43.4	49.0	21.7	81.1	86.4	89.0	96.7	99.96	99.96	98.50	99.96
Pareto	0.1	13.1	47.9	53.2	31.6	89.6	93.6	95.6	98.8	99.96	99.96	99.66	99.97
$q_{real} = 3.8\%, \pi_{real} = 10\%$ ($\chi = 49\%$)													
Pareto	0.5	10.0	39.9	44.4	21.4	72.7	77.7	78.6	91.4	94.93	95.37	90.13	95.25
Exp	0.5	10.1	39.9	44.4	21.5	73.8	78.3	79.1	91.6	95.02	95.55	91.78	95.34
Uniform	0.5	10.3	39.9	44.4	21.7	75.1	78.9	79.7	91.9	95.20	95.63	93.97	95.57
Pareto	0.1	13.1	44.3	48.5	31.6	83.8	87.3	87.7	95.0	96.54	96.92	96.67	96.87
$q_{real} = 10\%, \pi_{real} = 10\%$ ($\chi = 34\%$)													
Pareto	0.5	6.9	29.9	33.4	20.1	68.1	76.2	77.1	91.2	94.84	95.22	90.01	95.14
Exp	0.5	7.0	29.9	33.4	20.1	69.2	76.8	77.7	91.5	94.98	95.43	91.76	95.20
Uniform	0.5	7.2	29.9	33.4	20.1	70.4	77.4	78.3	91.7	95.13	95.51	93.82	95.46
Pareto	0.1	9.3	33.5	36.8	29.4	78.4	85.3	85.7	94.5	96.38	96.71	96.46	96.67
$q_{real} = 50\%, \pi_{real} = 50\%$ ($\chi = 0.13\%$)													
Pareto	0.5	0.82	2.37	2.49	10.4	23.7	28.1	35.6	53.7	56.65	59.95	58.95	60.23
Exp	0.5	0.83	2.37	2.49	10.5	24.1	28.4	35.9	53.8	56.74	60.12	60.40	60.31
Uniform	0.5	0.84	2.37	2.49	10.6	24.5	28.6	36.5	54.0	56.89	60.25	60.79	60.46
Pareto	0.1	1.11	2.90	2.95	14.4	28.3	32.0	40.5	56.8	59.45	62.68	64.84	63.06

Table 3.6: Classification accuracy (percent) in simulations of 2^{18} samples.

be seen by contrasting the two Pareto cases ($\mu = 0.1$ and $\mu = 0.5$) and comparing Pareto, exponential, and uniform cases (all with $\mu = 0.5$). As the difference between the last three scenarios is quite small, we conclude that the distribution of network jitter, as opposed to its mean, generally has a minor effect on accuracy. Therefore, keeping the Laplace model (3.21) appears reasonable.

To shed additional light on selection of parameters, the next column of the table re-runs Hershel with all available features and $\lambda = 10$. While this slightly improves the $\mu = 0.1$ case, this happens only under 50% packet loss and at the expense of significant reduction in accuracy in other rows, which suggests that $1/\lambda$ should *overestimate*, rather than *underestimate*, the real network delay. To this end, our previous conservative choice $\lambda = 2$ seems quite appropriate. The last column of the table reverts to $\lambda = 2$ and demonstrates that the model is insensitive to selection of q. We thus keep $q = 3.8\%$ for the Internet classification below.

3.5 Experiments

Our contribution in this subsection is to apply Hershel to a wide-scale Internet scan and provide an assessment of the obtained classification.

3.5.1 Dataset Properties

We use Internet scan data from [45], which is based on a 2010 survey of webservers in [55]. These IPs were discovered by sending port-80 SYN packets from Windows Server 2008 (with all TCP options enabled) to every address in BGP. The experiment garnered 37.8M samples x that contained at least one SYN-ACK, which we later feed into Hershel. We start by examining occurrence of various features in the dataset and their mapping to signatures in $\mathcal{D}$. We qualitatively group them into four types – linux, windows, embedded (routers, modems, cameras, hardware gadgets), and other (BSD, Mac, AIX, NetApp, Big-IP, SunOS).

RTOs	Hosts	Sigs	Group
3	9,639,810	27	all
2	9,070,991	16	windows, embedded
5	7,834,027	23	linux, embedded, other
4	5,066,940	16	unix, embedded
1	2,669,222	1	Dell printer
0	1,992,196	0	–
6	540,042	9	linux, embedded, other
19	202,733	2	embedded
18	162,442	0	–
17	110,335	0	–

Table 3.7: Top RTO counts (99% of total).

Window	Hosts	Sigs	Group
5,792	10,143,772	4	linux
16,384	7,051,858	6	windows, embedded, other
8,192	4,266,370	17	windows, embedded
65,535	3,551,640	9	windows, other
5,760	2,643,274	0	–
5,840	981,136	3	embedded
16,000	781,225	5	embedded
4,096	775,473	5	embedded
1,024	758,230	4	embedded
2,800	677,211	1	TP-Link router

Table 3.8: Top window sizes (87% of total).

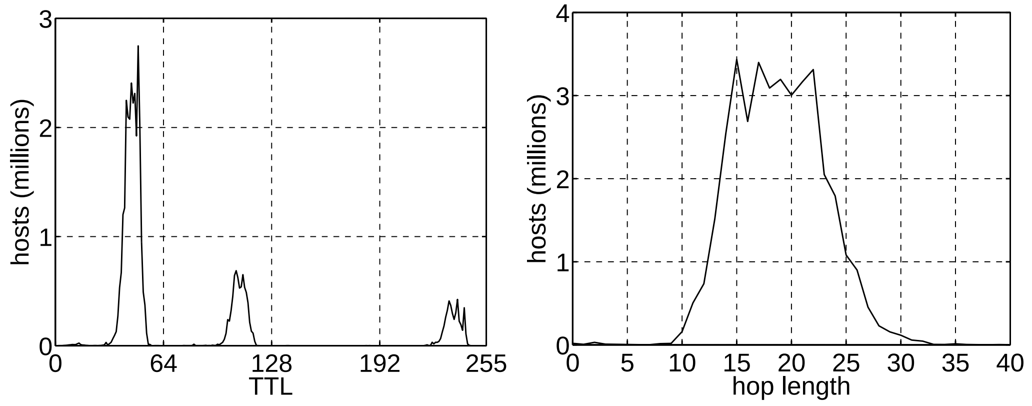
To first step is to ensure that packet loss has not produced totally unworkable temporal features in the dataset. Table 3.7 shows the number of available RTOs per destination. It is encouraging to see that the top four spots retain enough information for a meaningful match and the most difficult case (i.e., single SYN-ACK) follows in sixth place. While the average number of received packets was 5, one host transmitted over 3M SYN-ACKs. We next analyze sanity of the remaining features and build intuition for what to expect from Hershel classification.

The scan contains a staggering 3,815 unique window sizes, while our fingerprint collection $\mathcal{D}$ has only 51. While users tuning their stacks and scrubbers modifying the OS signature are possible reasons, we also found that the advertised window of SYN-ACKs can be easily changed at the application layer by resizing the socket buffer (i.e., calling `setsockopt` with the `SO_RCVBUF` option) before the connection is accepted. This highlights the need for a flexible classifier that allows features to mismatch.

The good news is that the distribution of window size is heavily skewed towards well-known values, as seen in Table 3.8. The most common window is unique to Linux variants, while the most ambiguous is split across 17 operating systems. Interestingly, window size 5760 in position #5, which we later discovered belongs to Ubuntu, is absent not just from ours, but also other fingerprinting databases (e.g., p0f, xprobe). We come back to these hosts later and examine how Hershel classifies them. Ideally, unknown devices should be mapped to the same OS family (i.e., Linux in this case).

Another peculiar case are 168K hosts with zero window size, which in our database corresponds to a single device related to building automation. This particular stack forces the sender to finish the 3-packet handshake (SYN, SYN-ACK, ACK) and wait for the window to move before sending the first GET request. Immediately after the sender's ACK, the window expands to 12,288 bytes. Closed receiver windows can be an artifact of rate-limiting firewalls or site policies related to congestion control. One notable example is a popular host craigslist.com that prior to 2006 was completing all TCP handshakes with window size zero [58]. Other usage of this technique comes from network tarpits [3], which aim to slow down scanners by advertising small windows in SYN-ACKs. All of this suggests that the true window size may remain "hidden" from the fingerprinting tool for reasons unrelated to users, scrubbers, or TCP socket options.

The TTL values of received packets are plotted in Fig. 3.4(a), covering 251 unique points out of the 255 possible. A vast majority of the hosts are clustered on the values just before the initial TTL defaults 64, 128, and 255. Fig. 3.4(b) shows the distribution of



(a) received TTL
(b) reverse distance

Figure 3.4: Received TTL and reverse path length.

TTL	Hosts	Sigs	Group
64	26,275,301	70	linux, embedded, other
128	7,129,667	17	windows, embedded, other
255	4,214,927	22	linux, embedded, other
32	190,697	7	embedded

Table 3.9: Initial TTL distribution (100% of total).

reverse hop length for each host back to the scanner, calculated by subtracting the received TTL from the nearest well-known initial value. This distribution appears reasonable, with less than 0.4% of the mass below 10 or above 30 hops. This suggests the number of non-standard initial TTLs (if any) is small. Table 3.9 shows the distribution seen by Hershel and the corresponding number of signatures in $\mathcal{D}$.

A good number of hosts (69%) set the DF flag, indicating they intend to perform path-MTU discovery, which matches 45% of the signatures. Out of 37.8M responsive targets, 5.9M (16%) send at least one reset packet (in addition to the SYN-ACKs), which is consistent with 56 OSes. The reset window (RW) deviates from that in the SYN-ACK for 20.8% of the IPs and 8 fingerprints in $\mathcal{D}$.

Feature	Hosts	Sigs	Group
RA= 1, RN= 1	4,368,098	35	embedded, other
RA= 0, RN= 1	1,167,761	11	windows, embedded
RA= 0, RN= 0	367,915	10	embedded
RA= 1, RN= 0	37,113	0	–

Table 3.10: Breakdown of 5.9M hosts with RSTs.

Options	Hosts	Sigs	Group
MSTNW	13,156,171	8	linux
MNWNNT	6,214,837	18	embedded, other
MNWNNTNNS	5,579,866	12	windows, other
M	5,431,682	41	embedded
MNW	2,656,342	5	linux, embedded, other
MNWST	1,107,935	2	windows, unix
MNWNNTSEE	762,593	4	other
MNNSWNNNT	412,602	0	–
MST	370,699	1	Windows Vista/7
MNNSNW	339,215	1	Akamai linux

Table 3.11: Top options strings (95% of total).

Table 3.10 examines the interplay between RA and RN in reset packets. In the most common scenario, hosts indicate that the ACK sequence is valid and correctly acknowledge values one larger than transmitted by the scanner in the SYN packet (which encodes the destination IP); however, there are also 37K hosts (last row) with broken implementations that indicate a valid ACK, but set the field to zero. None of our signatures exhibit this behavior.

We have 21 unique combinations of options in $\mathcal{D}$; however, the dataset shows 264 different strings, with the top 10 provided in Table 3.11. Similar to Table 3.8, a few popular cases account for the majority of IPs and Linux variants hold a clear lead, but now the most ambiguous combination splits across 41 embedded devices. While Akamai currently reports 137K servers [2], it seems reasonable that multiple NICs and IP aliasing

MSS	Hosts	Sigs	Group
1460	21,969,799	70	all
512	3,523,272	9	embedded
1452	3,512,626	2	embedded
1380	1,633,852	3	windows, embedded
1440	1,472,969	2	linux, embedded
1400	1,074,502	2	embedded
536	620,013	7	embedded
1448	562,961	0	–
1420	431,720	1	Avocent KVM switch
768	419,326	2	embedded

Table 3.12: Top MSS values (93% of total).

can produce 339K samples in last row.

Practically every host (99.5%) supports the MSS option, with Table 3.12 showing the top 10 cases out of the 1,021 observed in the dataset. The most common MSS 1460 does not provide much information about the OS, but the other values appear useful at partitioning the dataset into small groups. On the downside, general-purpose OSes often set the MSS as a function of the underlying data-link layer (i.e., MSS = MTU – 40), which creates some interesting dilemmas. For example, MSS 1452 in third place can be classified as one of two embedded devices or as home computers with 1492-byte MTUs commonly seen over PPP links such as DSL. This emphasizes importance of Hershel's probabilistic matching (3.15) and explains the significantly smaller number of unique MSS values in $\mathcal{D}$ (i.e., only 20).

3.5.2 Classification Overview

We run Hershel on the scan dataset and obtain a non-zero classification probability for 37.4M devices. Before showing these results, we perform additional sanity checks by examining how often individual features of each IP matched those in the most-likely OS suggested by Hershel.

Feature	Fraction	RST possibilities	Fraction
Win	70.3%	Neither has RST	80.9%
TTL	95.2%	Both have RST, match	10.4%
DF	96.2%	Missing RST	4.2%
MSS	70.6%	Both have RST, non-match	3.5%
OPT	99.4%	Bogus extra RST	1.0%

Table 3.13: Hershel's feature match rate.

Starting with the first two columns of Table 3.13, observe that window size is quite volatile, with 30% of the decisions going to signatures with a different window. This was expected given the numerous reasons to modify this field and the large amount of unique values seen earlier. Additionally, these 30% cover unknown devices whose RTOs and other features may match some OS in $\mathcal{D}$, but not the window size. Hershel remains robust in these cases and simply identifies the closest signature based on the available information. For example, 98.4% of Ubuntu cases with the unknown window 5760 are classified to Linux 2.4/2.6. These 2.6M hosts account for 25% of all window mismatch.

TTL and DF both exhibit match rates over 95%, while MSS comes in much lower at 71%. This is not surprising in light of its dependency on the MTU. The OPT string proves extremely reliable, where 77.4% of the cases match exactly and 22% are feasible subsets/supersets of the original. The five possible cases with RST packets are shown in the other two columns of Table 3.13. Combining the first two rows, we can conclude that 91% of the hosts have a matching RST feature. The next row with missing RSTs allows us to ballpark network packet loss at $q_{real} = 4.2\%$, not too far from the model's 3.8%. The majority of non-matching combinations (RA, RN, RW), responsible for 3.5% in the table, are caused by RW. Some of this behavior was also expected since tweaking of window size causes certain OSes to alter RW as well. Finally, we see 1% of the cases with extra RST packets, which we suspect are injected by firewalls, NAT boxes, and other devices as indication that they have expired the per-flow state.

OS	Count
Linux 2.6 / 2.4	9,610,732
VxWorks embedded systems	4,179,583
Windows Server 2003 SP1 SP2	2,316,590
VxWorks 5.4.2 / Xerox embedded	1,890,585
Linux 2.6 / Debian / CentOS / SonicWall	1,196,143
Embedded Linux / Mikrotik routers	1,190,102
Windows Server 2008 SP1 SP2 R2 / Vista / 7	1,146,609
TP-Link / Iball / Huawei home routers	1,046,985
Windows Server 2003 / 2000 / XP SP1	1,001,343
Cisco / Scientific Atlanta cable modems	827,285

Table 3.14: Top individual signatures (65% of total).

Group	Count
Linux	13,882,999
Embedded	13,590,803
Windows	7,561,839
Other	2,396,455

Table 3.15: Common families of operating systems.

3.5.3 Results

Having verified the general soundness of Hershel's output, we show it in Table 3.14. Linux attracts the most classification decisions, accounting for nearly a quarter of the webservers. This signature is quite unique, which makes accidental lumping of unknown devices or misclassified hosts into this category highly unlikely. In second and fourth place is VxWorks, which is an embedded OS extensively used in routers, modems, cameras, and printers. Interestingly, Windows 2003 is third, well above Server 2008 in seventh position. More Linux, home routers/modems, and Server 2003/XP make up the remaining OSes.

Table 3.15 groups fingerprints by type. Linux not just takes the first spot, but it dominates all other types of unix combined by a factor of 6. Embedded systems continue in second place, while windows is firmly in third. Interestingly, these results differ quite a bit

	IPs	Result	Count
Consensus	429	Both correct	424
		Neither correct	3
		Indeterminate	2
Disagreement	571	Hershel correct	476
		Snacktime correct	9
		Neither correct	6
		Indeterminate	80

Table 3.16: Manual verification.

from those in prior application of Snacktime to this dataset [55], with the most noticeable difference being 9M hosts moving from windows to embedded. This is not surprising as Snacktime's ability to overcome noise, packet loss, and feature corruption is quite weak. Further, as shown above, Microsoft OSes often share the window size and TTL with embedded devices, making this distinction even more difficult for Snacktime.

To better understand the difference between these methods, we carry out comparison using manual analysis of 1,000 random targets for which we had an HTTP header from a separate download process that grabbed the root page of each replying IP (this was done in real-time during the 2010 scan). Table 3.16 shows the result. The first category in the table breaks down 429 hosts on which both methods produce the same exact OS. Out of these, 424 are correct matches, 3 incorrect, and 2 indeterminate. The last option occurs for devices inadequately represented in the database (i.e., no resemblance to any signature) or when multiple OSes appear to be probable (e.g., due to extensive packet loss or missing/ambiguous "Server:" field in the HTTP response header). Among the 571 disputed hosts, Hershel delivers 476 correct results and Snacktime 9.

We can make a decision for 918 cases, out of which Hershel's accuracy is 98% and Snacktime's is 47%. The 9 cases where Hershel is wrong, but Snacktime is right, are caused by bogus RSTs, which Snacktime ignores, but Hershel takes into account. Overall, we find that when the two methods disagree, Hershel is overwhelmingly more accurate.

Country	Hosts	Windows	Linux	Embedded	Other
US	16,187,542	16.1%	30.4%	47.2%	5.2%
CN	2,345,462	54.1%	14.2%	14.6%	16.1%
ES	1,620,920	4.1%	89.2%	5.5%	0.8%
JP	1,614,724	11.3%	37.0%	35.8%	14.7%
DE	1,043,699	19.9%	57.7%	15.7%	5.7%
GB	862,571	32.1%	34.8%	25.0%	5.9%
CA	849,285	25.9%	45.3%	12.8%	14.7%
IT	810,104	14.3%	53.3%	29.1%	1.7%
BR	685,597	14.5%	52.8%	25.2%	5.3%
TW	644,645	35.9%	47.2%	10.8%	5.3%

Table 3.17: Top countries running webservers (71% of total).

AS	Size	Owner	Hosts	Windows	Linux	Embedded	Other
7922	71.0M	Comcast Cable	3,444,634	3.3%	6.2%	89.8%	0.3%
4134	109.7M	Chinanet	988,397	50.7%	13.3%	15.9%	18.5%
3352	10.9M	Telefonica de Espana	861,222	2.3%	92.0%	4.8%	0.5%
4837	54.5M	CNC Group China	595,931	53.2%	9.4%	15.0%	21.7%
20001	5.7M	Time Warner Cable	485,766	2.4%	1.5%	95.4%	0.3%
11351	4.9M	Time Warner Cable	436,329	2.0%	1.1%	96.3%	0.2%
2914	7.7M	NTT America	429,648	25.6%	20.6%	20.3%	33.1%
22773	11.9M	Cox Comm.	426,807	4.8%	2.8%	90.8%	0.6%
7018	75.2M	AT&T Services	373,068	31.0%	36.2%	18.9%	11.0%
7155	988K	Viasat Comm.	370,821	39.0%	0.0%	60.9%	0.0%

Table 3.18: Top ASes running webservers (22% of total).

3.5.4 World View

Next, we use the MaxMind GeoIP database [23] to glean trends in OS usage around the globe. Table 3.17 shows the top countries in the measurement. The US leads the list, accounting for almost half of the discovered web servers (i.e., 16M out of 37M) and exceeding China in second place by a factor of 8. The distribution of OS popularity is quite diverse, with only Italy and Brazil exhibiting similar vectors. Interestingly, Linux prevails over Windows in all countries except China; Spain stands out with 90% Linux, far more than any other locale in the list; and the US has the highest fraction of embedded devices among the entries in the table.

Table 3.18 breaks down the data by AS, shedding additional light on the results. Home access providers in the US (i.e., Comcast, Time Warner, Cox) are full of embedded devices, likely consumer routers and modems. In combination, these 4.8M boxes represent 30% of the discovered servers in the US, which helps explain the high percentage of embedded stacks seen earlier. Similarly, Telefonica de Espana, a large telecommunications provider in Spain and South America, is responsible for 50% of Spanish webservers in our dataset. This company is known for collaborations with RedHat and a cloud-computing emphasis [103]. Its 92% bias towards Linux is consistent with an earlier observation that Spain is dominated by this operating system. China's propensity towards Windows may stem from lax software-piracy laws, with 67% of its devices coming from two ISPs in Table 3.18, each replete with Microsoft OSes.

3.5.5 Scrubbers

While the Hershel's main purpose is large-scale measurement, where OS scrubbing is not likely to be prevalent, it still makes sense to examine its performance in such scenarios. Table 3.19 lists four obfuscators mentioned in existing literature and available for testing.

The first is Linux iptables, part of the packet-filtering framework called netfilter [68]. It is commonly used to inspect packets, modify routing tables, and configure the kernel firewall. It has extensions that 'mangle' packets and change certain header fields; however, the only ones of interest to Hershel are TTL and MSS. OSfuscate [20] is a Windows scrubber that thwarts fingerprinting tools by changing the registry. It can modify Win, TTL, MSS, and certain options (i.e., drop SACK and timestamps). Along similar lines, TCP Optimizer [75] gives its users ability to change the same five registry values, in hopes of improving TCP transfer speed. Finally, IPPersonality [90], built on top of the netfilter

Tool	Win	TTL	DF	TCP Options	MSS
Netfilter iptables	–	X	–	–	X
OSfuscate	X	X	–	Drop S and T	X
SG TCP Optimizer	X	X	–	Drop S and T	X
IPPersonality	X	X	X	Replace or reorder	X

Table 3.19: Capability of OS obfuscation tools.

Loss	Snacktime		Hershel				
(%)	All	RTO	-RST-RTO	-RST	RTO	All	RTO+RST
0.0	9.9	12.6	0.0	11.8	22.1	36.6	47.6
3.8	7.8	10.0	0.0	11.4	21.4	34.8	45.3
10	5.2	6.9	0.0	10.9	20.1	31.9	41.8
50	0.6	0.8	0.0	6.0	10.4	15.6	20.5

Table 3.20: Scrubbed accuracy (percent) among all OSes.

framework, is the most sophisticated scrubber in the list. It can modify all Hershel features except RST and RTO.

To evaluate performance against scrubbers, we simulate the worst-case scenario – IPPersonality with an adversary who mimics the signature with the closest RTO vector from another OS family (i.e., windows, linux, embedded, other). Table 3.20 shows the result using Pareto OWDs ($\mu = 0.5$ sec) and the Zipf setup from Table 3.6. Snacktime stays in the single digits, showing performance slightly below that of using just the RTOs. Hershel with only the fixed features from previous literature (i.e., all except RTO and RST) produces the expected 0% match rate. Adding the RTO pushes accuracy to 6-12%, but this far from impressive – the RTO alone works better, achieving 10-22%. Employing all Hershel features almost doubles the result; however, the real winner in this comparison is the RST+RTO combination, which reaches as high as 47%.

Limiting the simulation to 26 Windows/Linux signatures that the scrubber modifies using the same rules produces a more challenging case outlined in Table 3.21. There is

Loss	Snacktime		Hershel				
(%)	All	RTO	-RST-RTO	-RST	RTO	All	RST+RTO
0.0	2.5	5.9	0.0	5.0	14.6	31.8	41.7
3.8	2.0	5.0	0.0	4.7	14.1	29.8	39.4
10	1.4	3.8	0.0	4.5	12.9	26.9	35.7
50	0.1	0.5	0.0	2.2	6.2	12.0	16.3

Table 3.21: Scrubbed accuracy (percent) among Windows/Linux.

an accuracy reduction in all categories, but the scrubber-resilient version of Hershel still manages to correctly pinpoint over 41% of the samples that sustain no loss.

3.6 Conclusion

In this section, we modeled the problem of single-packet OS fingerprinting and developed novel approaches for tackling delay jitter, packet loss, and user modification to SYN-ACK features. Based on this theory, we created a classification method called Hershel, that significantly increased the accuracy of existing techniques, both in simulation and the real Internet. We employed Hershel on a large Internet dataset, obtaining classification of 37.4M hosts, and broke down the results to show OS usage of different countries and ASes. Finally, we also verified Hershel's robustness to scrubbers, showing that respectable accuracy can still be maintained by ignoring the scrubbed features.

4. AUTOMATED DATABASE CREATION*

4.1 Introduction

For classifiers such as Hershel to work, there must be a process that establishes signatures for known types of behavior and builds a database that contains all sufficiently different specimens found in the wild. To keep results up-to-date, new signatures must be periodically acquired and merged into the existing database. This is often a manual process that suffers from human error, poor repeatability, heuristic decisions, and database compositions incompatible across different classification methods.

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

4.1.1 Motivation and Contributions

Performance of each classifier depends on not only its internal algorithms, but also database $\mathcal{D}$ and types of volatility experienced during measurement. This makes comparison between different approaches (e.g., Nmap [73], Snacktime [7], p0f [122], Hershel) fairly complicated, especially if they utilize incompatible sets of features, databases, or assumptions on feature determinism. For example, consider method $\mathcal{M}_1$ with n signatures and $\mathcal{M}_2$ with $m \ll n$. It may appear that $\mathcal{M}_1$ is more powerful because its $\mathcal{D}$ is bigger; however, its classification accuracy may be lower due to the larger number of options to choose from and/or less reliable decision-making. Additionally, the specific model of distortion $\mathcal{X}$ (i.e., noise in certain features) applied during the experiment may have a dra-

*Reprinted, with permission, from "Unsupervised Clustering Under Temporal Feature Volatility in Network Stack Fingerprinting" by Z. Shamsi and D. Loguinov which appeared in [93] and [94]. © 2017 IEEE.

matic impact on the result. In such cases, it is possible that $\mathcal{M}_1$ resorts to random guessing and makes inferior choices to those of $\mathcal{M}_2$.

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

We apply these concepts to Hershel, which allows random OS behavior and provides probabilities, rather than heuristic weights, for the match across any pair of samples. We focus on its temporal network features (i.e., delay jitter) since they are highly volatile and fairly well-understood, but difficult to separate using manual analysis.

This leads to our third contribution that consists of building a Plata database using 9.7K webservers discovered in our campus network and passing all HTTP headers through simhash [60] to label the elements of $\mathcal{D}$. Using only delay features, we show that Hershel achieves 80%-separation under 500-ms random distortion on 117 signatures. Adding deterministic header values, this number jumps to $d(0.8,\mathcal{X})=398$, which is 3.4 times larger than the database we used in Section 3.

While Plata works well, its Monte Carlo simulations require a large amount of CPU

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

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

We finish the section by scanning the Internet on port 80 and applying Hershel+ to the result. Among Internet-wide studies, this is the largest population to be fingerprinted (i.e., 66M IPs), using the most extensive database (i.e., 420 signatures), and the first such attempt with an automatically generated $\mathcal{D}$. Compared to the scan dataset from 2010 that we used with the Hershel classification earlier, we find that the number of Linux and embedded devices has almost doubled, while that of Windows has remained stable. We compare some of our results with those of Nmap and discover a major flaw in the operation of the latter that surfaces in scenarios with non-ideal network conditions (e.g., firewalls). More importantly, however, we conclude that stochastic network effects do not impede the use of temporal features, but they require a more careful database construction process. Our proposed framework of Plata and Hershel+ is a step in the direction of automated, repeatable, and streamlined classification of massive datasets.

4.2 Background

The majority of efforts in stack fingerprinting [6], [7], [17], [50], [65], [73], [105], [112], [122] concentrate on introducing new features and designs to further distinguish between the OSes, thus improving the classification step; however, they universally rely on manual effort to construct databases. Since nearly all of them rely only on deterministic features, database creation is fairly uncomplicated.

The closest related problem to ours is automatic discovery of features that can be used to differentiate one OS from another. For example, [15] proposes a set of rules built from sending out a large number of probes (i.e., 300K) to controlled hosts and randomly varying header fields to detect patterns that produce OS-specific responses. The authors show that this method can reliably differentiate between three stacks (i.e., Windows XP, Linux 2.6, and Solaris 9) in a LAN environment.

In [88], this idea is explored at a larger scale by increasing the number of network stacks and applying a wider range of machine-learning algorithms from the Weka tool [40]. However, their results from scaling this approach to more signatures are quite pessimistic – the authors conclude that over-fitting to non-deterministic header fields, training bias towards certain implementations, and lacking semantics lead to confusion for the learning algorithms.

4.3 Overview

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

4.3.1 Terminology

Classifiers rely on vectors of distinctive features that identify each specimen, either uniquely or with some reasonably high probability. The former case arises when the features are *deterministic*, meaning all inspections of a given system produce the same result (e.g., the order of TCP options). The latter case occurs when the features are inherently *random* due to some non-deterministic processes running within the specimen (e.g., SYN-

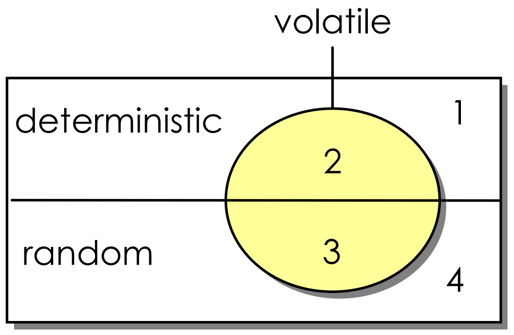
ACK retransmission delays). Features of either type may undergo additional modification due to influence of system owners or as byproduct of the measurement process, in which case we call them *volatile* (e.g., users tuning the TCP window size, queuing delays affecting packet spacing). All four types are illustrated in Fig. 4.1(a).

Note that volatility and randomness are not the same – the former arises due to forces *external* to the object being classified, while the latter due to *internal*. This distinction is important when internal disturbances exhibit substantially larger variance than external, or produce patterns that cannot be accounted for in the volatility model alone. With this in mind, we call classifiers *simple* if they operate using only non-volatile deterministic features (i.e., type-1 in Fig. 4.1(a)) and *complex* otherwise (i.e., types 2-4).

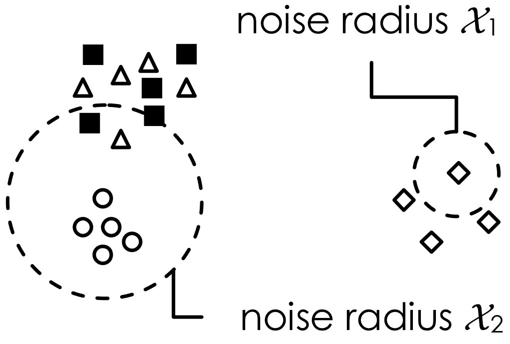
Consider an automaton that performs classification decisions for measurements x using some database $\mathcal{D}$. We call the matching process *membership* if it returns the probability that $x \in \mathcal{D}$, where determination of the most-likely match is not important. One example is intrusion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

4.3.2 Challenges

We are now ready to describe the problem of creating $\mathcal{D}$. Assume a measurement of several, possibly duplicate, specimens. Membership classifiers are not overly concerned with high-precision duplicate elimination as these have no effect on accuracy, only on speed and memory consumption. Simple identification classifiers can construct $\mathcal{D}$ by retaining the observations with unique combinations of features, which makes the problem trivial. However, complex identification classifiers must instead ensure *separability* among



(a) types



(b) separability

Figure 4.1: Classifier features.

the signatures, keeping only those that can be reliably distinguished from each other under various types of distortion $\mathcal{X}$. Inseparable specimens in $\mathcal{D}$ drop classification accuracy and increase overhead, while offering no tangible benefit.

To visualize this better, Fig. 4.1(b) plots random features of four hypothetical systems – circles, squares, diamonds, and triangles – where each point is a random observation of the corresponding system. Assuming uniformly random noise centered at each sample, distortion $\mathcal{X}_1$ keeps circles and diamonds separable, but not necessarily triangles and squares. Dropping either of the last two leads to a separable 3-signature database. For larger radius of noise (e.g., $\mathcal{X}_2$ in the figure) the database may consist of only two separable signatures – diamonds and one of circles/squares/triangles.

Our goal in this section is to study separation algorithms for volatile and/or random features, with application to inter-packet delays in wide-scale stack fingerprinting. This problem arises in single-packet techniques [7], [112] whose classifier must heavily rely on temporal features. The general appeal of these methods includes low bandwidth consumption (i.e., no extra packets beyond those sent by the scanner), a reduced probability of tripping IDS, no requirement that the target respond on closed ports or multiple protocols, and good scalability in Internet-wide classification. However, unlike traditional tools (e.g., Nmap [73]) that rely on deterministic features, single-packet classifiers require prohibitively expensive manual effort to construct databases of non-trivial size. Since this problem has not been studied before, we address it below.

4.4 Database Creation Using Plata

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

4.4.1 Preliminaries

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

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

$$p_i(\delta) := P(\Delta_i = \delta) \tag{4.1}$$

is built from observation. Note that $\delta = (\delta_1, \delta_2, \ldots)$ is a deterministic feature vector that consists of multiple scalar values. Using a pair of initial RTOs (SYN-ACK retransmission timeouts), Fig. 4.2(a) shows the distribution of Δ_i for two Xerox printers in our dataset. Depending on the target jitter model $\mathcal{X}$, these two hosts may very well be $(1-\epsilon)$-separable; however, doing so manually for hundreds of thousands of points is close to impossible. To compound the issue, the majority of systems use random vectors with at least 3 dimensions and some with over 20.

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

$$p(\Delta_i|\Delta_j, \mathcal{X}) = \sum_{\delta} \sum_{\delta'} p(\delta|\delta', \mathcal{X}) p_i(\delta) p_j(\delta'). \tag{4.2}$$

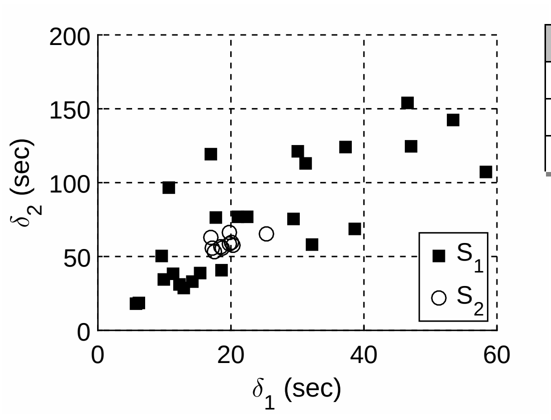
For a given i, classifiers are typically concerned with finding j that produces the largest value in (4.2). However, we are facing a different problem that requires normalization. Let $\pi_i(\mathcal{X}) := \sum_{j=1}^{n} p(\Delta_i|\Delta_j, \mathcal{X})$ be the total similarity weight of system S_i across all available options j. Depending on the classifier, π_i may not always be 1. To handle such cases, define

$$q(\Delta_i|\Delta_j, \mathcal{X}) = \frac{p(\Delta_i|\Delta_j, \mathcal{X})}{\pi_i(\mathcal{X})} \tag{4.3}$$

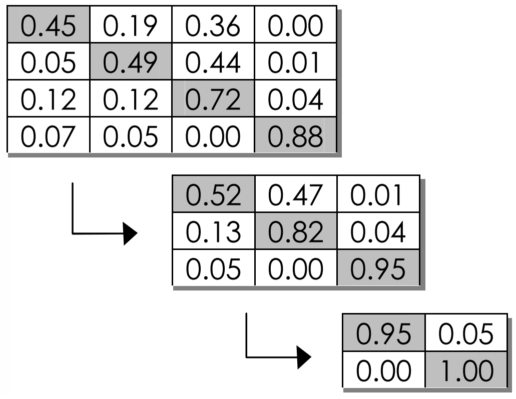
to be the probability that S_i gets classified as S_j. Now suppose systems $S_1, \ldots, S_n$ are deployed in a production environment (e.g., wide-area Internet) and measured using remote probing. Therefore, instead of seeing Δ_i, the observer now samples $\Delta_i + \theta$, where random vector θ is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

4.4.2 Matrix Construction

We next describe our database-construction framework, which we call Plata. It starts by building a confusion matrix $M = (M_{ij})$, where each cell $M_{ij} = E[q(\Delta_i + \theta|\Delta_j, \mathcal{X})]$



(a) random features



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

0.52	0.47	0.01
0.13	0.82	0.04
0.05	0.00	0.95

0.95	0.05
0.00	1.00

(b) matrix reduction

Figure 4.2: Randomness of RTO features and elimination of duplicates in Plata.

and the expectation is taken over θ. In general, classification decisions and vectors θ may be available only as output of some algorithm. For example, the former might be a C4.5 decision tree and the latter may require simulations of a specific queuing discipline. In such cases, the only solution is to run Monte-Carlo simulations that repeatedly distort Δ_i, classify the resulting observations, and average the result to obtain an approximation to M_{ij}.

To this end, suppose we generate r vectors $\theta_1, \ldots, \theta_r$ by simulating $\mathcal{X}$. Using the PMF in (4.1), we obtain the same number of instances from random variable Δ_i, which we call $\delta_i^1, \ldots, \delta_i^r$. Then, the approximate matrix is given by

$$\tilde{M}_{ij} = \frac{1}{r} \sum_{m=1}^{r} q(\delta_i^m + \theta_m | \Delta_j, \mathcal{X}). \tag{4.4}$$

Since this expands to

$$\tilde{M}_{ij} = \frac{1}{r} \sum_{m=1}^{r} \sum_{\delta'} q(\delta_i^m + \theta_m | \delta', \mathcal{X}) p_j(\delta'), \tag{4.5}$$

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

4.4.3 Separation

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

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

$$\tilde{M}_{ij} = \tilde{M}_{ij} + \frac{\tilde{M}_{ij}}{1 - \tilde{M}_{ik}} \tilde{M}_{ik}. \tag{4.6}$$

Note that if none of i's classifications went to system k, i.e., $\tilde{M}_{ik} = 0$, row i does not change. This process continues until all diagonal values are above $1 - \epsilon$. The remaining systems at that stage are added to the database and their number establishes the $(1 - \epsilon, \mathcal{X})$-dimension of the classifier. An example of this reduction process is shown in

Fig. 4.2(b), where the rows are sorted in ascending order of $\tilde{M}_{ii}$ for convenience of presentation. Setting $\epsilon = 0.2$, there are three rows that violate separability constraints. Since (S_1, S_2) are both similar to S_3, but none of them resembles S_4, intuition suggests the initial measurement may contain only two separable specimens. After removal of the first row, all diagonals receive a boost, but (S_2, S_3) are still inseparable. Another iteration produces the expected two vectors that match themselves with probability 0.95 or better.

Note that $1 - \epsilon$ can be used as a tuning parameter – larger values reduce the number of eventual vectors in the database, while smaller values preserve more, but at the risk of having more duplicates and poor classification accuracy. Although only $\tilde{M}_{ii}$ is compared against $1 - \epsilon$, the entire matrix needs to be recomputed after each iteration. This is necessary in order to properly distribute the weights of eliminated systems using (4.6). Thus, the complexity of each step is n^2, repeated $n - d$ times, where $d := |\mathcal{D}|$ is the size of the final database.

4.4.4 Labeling

Once database $\mathcal{D}$ is created, Plata needs to assign system-identifying labels to the available signatures. Assume a process that collects mappings from each S_i to the corresponding label l_i using some type of download (e.g., port-80 HTTP requests), oracle input, or other means, but possibly for a subset of the known specimens. Incomplete labeling may occur due to bandwidth constraints, obfuscation of certain systems by their administrators, and generic software names (e.g., apache) that fail to identify the underlying system. Since labels might be available for hosts that have been discarded during the matrix-separation step, we must again consider the entire set $S_1, \ldots, S_n$. To this end, we classify each known specimen using $\mathcal{D}$ and produce a set of clusters $C_1, \ldots, C_d$, where d is the $(1 - \epsilon, \mathcal{X})$-dimension of the database/classifier obtained earlier by Plata.

To eliminate duplicate labels, a separate procedure clusters them into multiple cate-

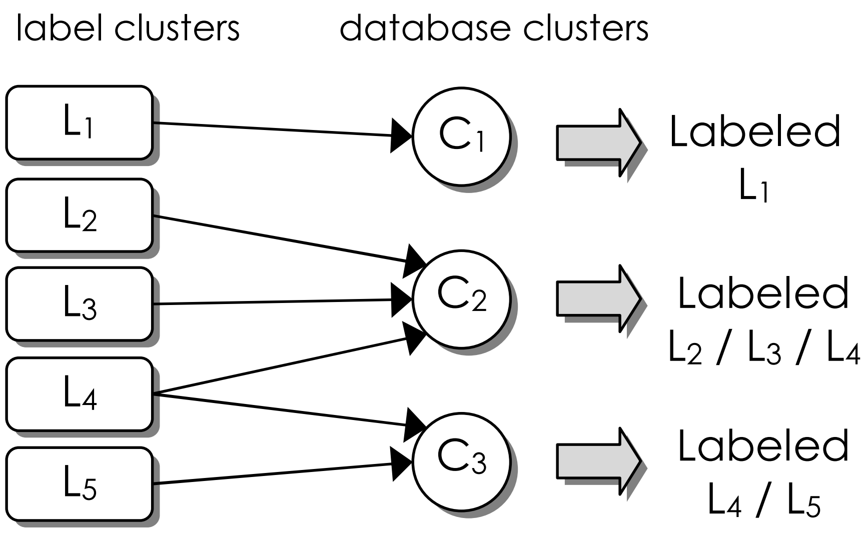


Figure 4.3: Applying labels to database clusters.

gories $L_1, L_2, \ldots$ using some type of string-similarity matching. As shown in Fig. 4.3, there is a directed edge between clusters L_k and C_j if there exists a system $S_i \in C_j$ such that its label $l_i \in L_k$. Note that this forms a bipartite graph in which L_k may point to multiple clusters C_j. Plata leaves the specifics of choosing the right label for each C_j to the application. One option is to combine the labels of all in-neighbors, as done in Fig. 4.3. Another option is to assign weights to edges (e.g., equal to the number of corresponding S_i's) and enforce some minimum frequency before a label is considered valid. This can be further extended to allow for majority voting. For example, 100 hosts with label "Linux 2.4" and two with "Windows 7" mapping to C_j probably indicate the former is more appropriate than the latter.

4.5 OS Fingerprinting Database

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

4.5.1 Classifier

The database for single-packet fingerprinting tools has evolved from 25 signatures in [7] to 98 in [55], and eventually to the set of 116 signatures we built in Section 3, but the corresponding $(1-\epsilon, \mathcal{X})$-dimensions of the underlying classifiers remain unknown. So far, manual construction of $\mathcal{D}$ in these tools has relied on separation only across deterministic features (e.g., window size, TTL, RST bit) and never examined how to determine whether two hosts with the same fixed header values have sufficiently distinct RTO vectors. To address this issue, we next apply Plata to temporal features of single-packet classifiers and build the first OS-fingerprinting database that is separable across random/volatile features.

4.5.2 Data Collection

We scan our campus network (three /16 blocks) on port 80 to obtain observations $\Delta_1, \ldots, \Delta_n$ from responsive hosts $S_1, \ldots, S_n$. Since each Δ_i may be random due to kernel-scheduling peculiarities, as in Fig. 4.2(a), we persist in gathering $w = 50$ RTO vectors from each host, which is typically enough to capture whatever variation Δ_i may exhibit. Additionally, to exclude lossy vectors from being included in the database, the scanner continues until it receives w samples of the maximum length seen so far. Since packet loss in our network is low, quick convergence follows – the average number of SYN probes per responsive IP was 50.14.

As each S_i is a public server, care needs to be exercised to not overload the target with w back-to-back requests and cause unnecessary side-effects (e.g., rejected connections, CPU overload). However, as it turned out, even conservative 1-second inter-SYNs delays were too small. One such problem surfaced with certain printers, whose SYN-backlog queue [125] was smaller than w. When the queue was full, the printers terminated the oldest ongoing sequence of SYN-ACKs and started a new one. This caused the corresponding

Δ_i to exhibit random truncation and presented difficulties in obtaining w loss-free observations. We eventually settled on delaying SYN probes by 240 seconds, i.e., double the TCP MSL (maximum segment life), which solved the problem.

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

Along with the scan, a separate process opens a connection to each responsive host and attempts to download its root page over HTTP. While banner grabbing is not generally considered reliable because any identification strings may be replaced by OS-oblivious names (e.g., apache) or altogether removed, it works for our purpose since admins have no incentive to obfuscate OS names behind our campus firewall, and Plata only needs a subset of $S_1, \ldots, S_n$ to be labeled. This provides a fast, repeatable process that requires no manual intervention.

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

4.5.3 Separating Features

Single-packet OS-fingerprinting tools use both deterministic and random features. For each S_i, we move the former into vector u_i and the latter into Δ_i. In general, Hershel treats u_i as volatile, which means it allows users to change TCP/IP header values without making the OS fundamentally different. However, there is no even remotely accurate model for distortion $\mathcal{X}$ applied by users to these features. We therefore limit our efforts to the better-

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

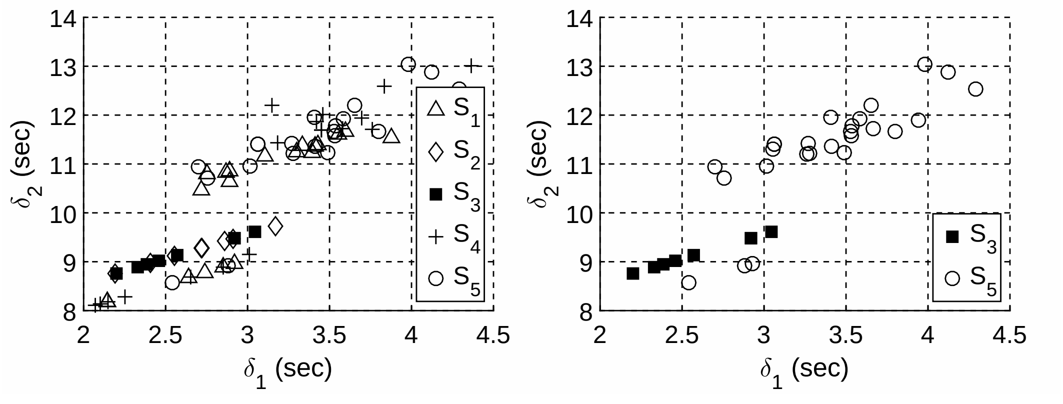
Table 4.1: Database dimensions.

understood network delay jitter and its volatility. If a realistic noise model $\mathcal{X}$ becomes available for u_i, Plata can be used to compact duplicate hosts even further.

The simplest way to achieve separation on the deterministic features is to combine u_i with the size of RTO vector Δ_i. Splitting the available hosts $S_1, \ldots, S_n$ into clusters based on the deterministic pair $(u_i, |\Delta_i|)$ produces the first row of Table 4.1, with 28 signatures for RING [112], 209 for Snacktime [7] and IRLsnack [55], and 344 for Hershel. Note that hosts within each cluster have same-length RTO vectors and our next goal is to further subdivide them into smaller groups that are $(1-\epsilon, \mathcal{X})$-separable.

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

We generate $r = 1$K random noise vectors $\theta_1, ..., \theta_r$ and add them to each observation of Δ_i, resulting in $wr = 50$K disturbed samples per host S_i. We run Plata for each candidate classifier using their similarity function p and compute (4.5), in which $p_j(\delta') = 1/w$. This creates one matrix $\tilde{M}$ for each unique combination of deterministic features,



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

Figure 4.4: Plata example.

which is fed to Plata's separation algorithm with $1 - \epsilon = 0.8$. After all matrices are compacted, we combine the surviving specimens into the final database $\mathcal{D}$.

Going back to Table 4.1, the second row shows that RTO features alone allow single-packet tools to differentiate between 23-117 stacks under this combination $(\epsilon, \mathcal{X})$. Hershel more than doubles the dimension of its nearest competitor, which stems from its more sophisticated model for $p(\delta|\delta')$. Combining both deterministic and random features, Hershel ends up with 398 signatures, which is quite significant given the limited scope of the initial scan. Due to its higher accuracy and better separation ability, the rest of this section stays with Hershel as the underlying classifier for Plata.

To demonstrate how matrix reduction works in practice, consider five actual Windows hosts in Fig. 4.4(a) with $|\Delta_i| = 2$. While all of these OS kernels produce noisy RTOs, there are two distinct patterns. Fig. 4.4(b) shows the result of Plata separation, which successfully extracts both patterns (Windows Server 2003 with two different service packs) out of the group and represents them using hosts (S_3, S_5).

4.5.4 Label Clustering

Note that Plata does not specify how to assign labels to clusters $\{L_j\}$. Besides ground-truth obtained from device owners, which may be infeasible for large decentralized networks, some of this information can be collected automatically. Our approach is to proceed along this route. Recall that HTTP headers contain the "Server:" string that sometimes identifies the version of the web server and uniquely ties it to a particular OS (e.g., Windows IIS). However, in other cases, the operating system can be inferred only from the HTML content of the page, as is the case with certain embedded devices (e.g., printers, cameras). We thus combine the "Server:" field with the entire HTML page and perform clustering using simhash [60], which is a well-known technique for detecting similar webpages. This creates 515 clusters $L_1, L_2, \ldots$, which we match to $d = 398$ Hershel signatures $C_1, \ldots, C_d$ using the procedure in Fig. 4.3.

The final step is to perform manual verification of label sanity, determine which tags in the HTML to use (e.g., head, title), and convert low-level software versions to the corresponding OS name (e.g., IIS 7.5 to Windows Server 2008 R2). With enough coding effort to account for the various formats, most of this can be automated [107], but we found it easier to just show each page to a human and let them decide which of the found labels is appropriate. Plata does this by sequentially rendering one page from each L_k and recording the user's response. Even for $n \to \infty$, the number of unique clusters should remain reasonably small.

Results reveal that our label clustering works quite well – 326 out of 398 signatures (82%) receive a meaningful description. They are responsible for 98% of $n = 9{,}701$ measured hosts. Table 4.2 shows the top five most-popular signatures on our campus, where Plata successfully shrinks the most common Windows RTO pattern from 3,803 hosts down to 1. Heavy usage of Windows (43% of all servers) and Linux (12%) is no

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

Table 4.2: Top 5 database signatures gathered from our campus scan
(Win = window size, TTL = time to live, DF = do not fragment, MSS = max segment size, RST = reset packet features).

surprise, but we also find a large amount of HP LaserJet printers in fifth place. The 398-326=72 unlabeled cases belong to network elements that either fail to provide a banner or supply one that contains no clue about the underlying OS. The latter case often happens with extremely rare devices for which we have only one banner to analyze. If Plata is exposed to additional data collection and user input (i.e., outside of our network), these gaps can be eliminated. The main benefit of our framework is that only a small fraction of n (i.e., $72/9701 = 0.7\%$) requires further attention.

Note that using automated banners for labeling does limit our ability to distinguish between OS versions. For example, the two Linux signatures in Table 4.2 are likely from different kernel versions. However, if the application requires more fine-granular labeling, additional effort – installing each OS in a test environment or contacting the owner – is needed in conjunction with Plata.

4.6 Optimizing Plata

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

Analyzing (4.5), there are two obvious ways to reduce complexity – lowering r and making function $p(.)$ faster. However, for Hershel, we can attempt to do even better – replace Monte-Carlo simulations with a directly evaluated model that produces the expected probability that S_i gets classified as S_j under random noise θ. The rest of the subsection treats $\theta = (\theta_1, \theta_2, \ldots)$ as a vector consisting of scalar random variables, with respect to which all expectations are taken. Since $M_{ij} := E[p(\Delta_i + \theta|\Delta_j, \mathcal{X})]$ can be written as

$$\sum_{\delta} \sum_{\delta'} E[p(\delta + \theta|\delta', \mathcal{X})] p_i(\delta) p_j(\delta'), \tag{4.7}$$

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

4.6.1 Closed-Form Plata-Hershel Matrix

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

$$p(\delta|\delta', \mathcal{X}) = \prod_{m=1}^{|\delta|} f(e_m). \tag{4.8}$$

Note that (4.8) treats error values $(e_1, e_2, \ldots)$ as iid random observations. For the default model of $\mathcal{X}$, recall from Section 3 that Hershel uses exponential one-way delay. This produces Laplace jitter with density $f(x) = (\lambda/2)e^{-\lambda|x|}$, where parameter λ should conservatively reflect the amount of jitter anticipated in the network during actual measurement (i.e., $1/\lambda$ should upper-bound the real mean). With this in mind, our goal is to derive the following expectation

$$E[p(\delta + \theta|\delta', \mathcal{X})] = E\Big[\prod_{m=1}^{|\delta|} f(e_m + \theta_m)\Big], \tag{4.9}$$

where each θ_m is a random variable.

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

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

$$E[p(\delta+\theta|\delta', \mathcal{X})] = \Big(\frac{\lambda\mu}{4}\Big)^{|\delta|} \prod_{m=1}^{|\delta|} \begin{cases} g_m & \lambda \neq \mu \\ h_m & \lambda = \mu \end{cases}, \tag{4.10}$$

where

$$g_m = \frac{2(\lambda b_m^{\mu} - \mu b_m^{\lambda})}{\lambda^2 - \mu^2}, \quad h_m = b_m^{\lambda}\Big(|e_m| + \frac{1}{\lambda}\Big). \tag{4.11}$$

Proof. Using (4.8),

$$\begin{aligned} E[p(\delta+\theta|\delta')] &= E\Big[\prod_{m=1}^{|\delta|} f(\delta_m + \theta_m - \delta'_m)\Big] \\ &= \prod_{m=1}^{|\delta|} E[f(\delta_m + \theta_m - \delta'_m)] \\ &= \Big(\frac{\lambda\mu}{4}\Big)^{|\delta|} \prod_{m=1}^{|\delta|} \int_{-\infty}^{\infty} e^{-\lambda|e_m+z|-\mu|z|} dz. \end{aligned} \tag{4.12}$$

First assume $\lambda \neq \mu$. Given a constant $c < 0$, we get

$$\begin{aligned}\int_{-\infty}^{\infty} e^{-\lambda|c+z|-\mu|z|}dz &= \int_{-\infty}^{0} e^{\lambda(c+z)+\mu z}dz \\ &+ \int_{0}^{-c} e^{\lambda(c+z)-\mu z}dz \\ &+ \int_{-c}^{\infty} e^{-\lambda(c+z)-\mu z}dz \\ &= \frac{e^{\lambda c}}{\lambda+\mu} + \frac{e^{\mu c}-e^{\lambda c}}{\lambda-\mu} + \frac{e^{\mu c}}{\lambda+\mu}.\end{aligned} \tag{4.13}$$

When $c \geq 0$, we have

$$\begin{aligned}\int_{-\infty}^{\infty} e^{-\lambda|c+z|}e^{-\mu|z|}dz &= \int_{-\infty}^{-c} e^{\lambda(c+z)+\mu z}dz \\ &+ \int_{-c}^{0} e^{-\lambda(c+z)+\mu z}dz \\ &+ \int_{0}^{\infty} e^{-\lambda(c+z)-\mu z}dz \\ &= \frac{e^{-\mu c}}{\lambda+\mu} + \frac{e^{-\lambda c}-e^{-\mu c}}{\mu-\lambda} + \frac{e^{-\lambda c}}{\lambda+\mu}.\end{aligned} \tag{4.14}$$

Combining the two cases, notice emergence of $|c|$

$$\int_{-\infty}^{\infty} e^{\lambda|c_m-z|}e^{\mu|z|}dz = \frac{e^{-\lambda|c|}+e^{-\mu|c|}}{\lambda+\mu} + \frac{e^{-\mu|c|}-e^{-\lambda|c|}}{\lambda-\mu}.$$

For the special case $\lambda = \mu$, we obtain

$$\begin{aligned}\int_{-\infty}^{\infty} e^{\lambda|c+z|}e^{-\lambda|z|}dz &= \frac{e^{-\lambda|c|}}{\lambda} + |c|e^{-\lambda|c|} \\ &= e^{-\lambda|c|}\Big(|c| + \frac{1}{\lambda}\Big).\end{aligned} \tag{4.15}$$

Simplifying using b_m, we get (4.10). ☐

The next logical step is to investigate whether matrix M built using (4.10) matches the Monte-Carlo version $\tilde{M}$. We consider a simple scenario with $|\delta| = 2$, $\delta = \delta'$, and $\lambda = \mu = 10$. This represents some diagonal cell M_{ii}, i.e., similarity score of S_i to itself, for a deterministic Δ_i. Setting $e_m = 0$ for all m, (4.10) produces 6.25, while Monte-Carlo simulations yield $\tilde{M}_{ii} = 6.7$. The error increases with RTO vector length and is more difficult to predict for off-diagonal values M_{ij}.

Further analysis uncovers that the source of this bias lies in Hershel's assumption on delay jitter. To illustrate this point, consider distorting a two-RTO vector δ using $\theta = (\theta_1, \theta_2)$. From the queuing model of Hershel, consecutive Laplace jitter values can be expressed using three iid exponential one-way delays X, Y, Z, i.e., $\theta_1 = Y - X$ and $\theta_2 = Z - Y$. While we were reasonable in arguing that X, Y, Z are independent due to the large gaps between SYN-ACKs, the same logic unfortunately does not apply to jitter because θ_1 and θ_2 share a common variable Y. For $e_m = 0$ and $|\delta| = 2$, the correct expectation of (4.9) is $E[f(\theta_1)f(\theta_2)]$. On the other hand, Theorem 1 uses Hershel to deduce the result as $E[f(\theta_1)]E[f(\theta_2)] = \lambda^2/16$. We next expand the former term and show that it deviates from the latter for all λ.

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

$$E[f(\theta_1)f(\theta_2)] = \frac{29\lambda^2}{432}. \tag{4.16}$$

Proof. Considering jitter dependent, we must look at three separate cases. For the first one, define

$$\chi_1 = E[f(J_1)f(J_2)|X > Y, Z > Y] \tag{4.17}$$

and notice that event $X > Y, Z > Y$ happens with probability $1/4$. Now observe

$$\begin{aligned}\chi_1 &= \lambda^5 \int_0^\infty \int_y^\infty \int_y^\infty e^{-\lambda(x-y)} e^{-\lambda(z-y)} e^{-\lambda(x+y+z)} dxdzdy \\ &= \lambda^5 \int_0^\infty e^{-3\lambda y} y^2 dy = \lambda^5 \frac{2}{(3\lambda)^3} = \frac{2\lambda^2}{27}. \end{aligned} \tag{4.18}$$

For the second case, we have

$$\chi_2 = E[f(J_1)f(J_2)|X < Y, Z < Y], \tag{4.19}$$

where event $X < Y, Z < Y$ also happens with probability $1/4$. This leads to

$$\begin{aligned}\chi_2 &= \lambda^5 \int_0^\infty \int_0^y \int_0^y e^{-\lambda(y-x)} e^{-\lambda(y-z)} e^{-\lambda(x+y+z)} dxdzdy \\ &= \lambda^5 \int_0^\infty e^{\lambda y} \int_y^\infty e^{-2\lambda} dx \int_y^\infty e^{-2\lambda} dzdy \\ &= \frac{\lambda^3}{4} \int_0^\infty e^{-3\lambda y} dy = \frac{\lambda^2}{12}. \end{aligned} \tag{4.20}$$

The remaining two cases $X > Y > Z$ and $Z > Y > X$ are identical to each other. Without loss of generality, we use the former and define

$$\chi_3 = E[f(J_1)f(J_2)|X > Y, Z < Y], \tag{4.21}$$

which leads to

$$\begin{aligned}\chi_3 &= \lambda^5 \int_0^\infty \int_0^x \int_0^y e^{-\lambda(x-y)} e^{-\lambda(y-z)} e^{-\lambda(x+y+z)} dzdydx \\ &= \lambda^5 \int_0^\infty e^{-2\lambda x} \int_0^x e^{-\lambda y} ydydx \\ &= \lambda^3 \int_0^\infty e^{-2\lambda x}(1-(1+\lambda x)e^{-\lambda x})dx = \frac{\lambda^2}{18}. \end{aligned} \tag{4.22}$$

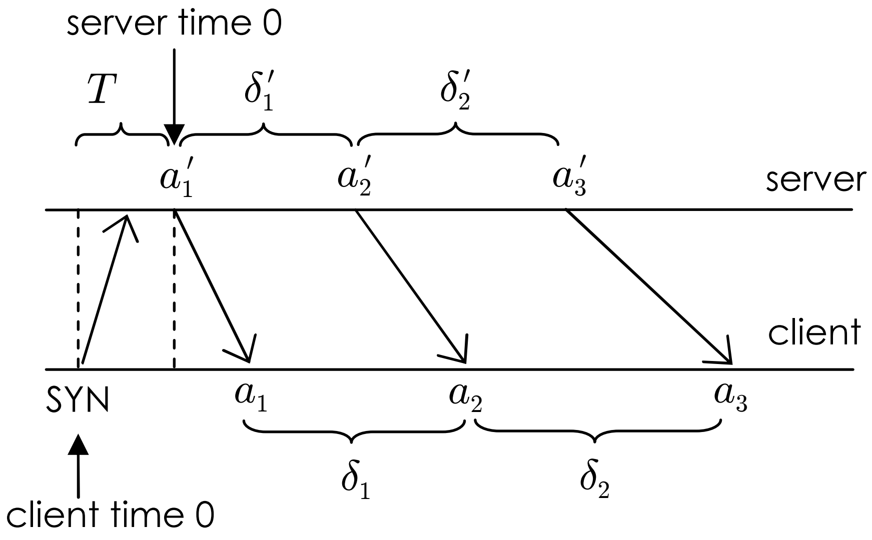


Figure 4.5: Features in Hershel (δ) and Hershel+ (a).

Combing these cases with respective weights $1/4, 1/4$, and $1/2$, we get the overall expectation in (4.16). $\square$

Using $\lambda = 10$ in (4.16) produces 6.7 observed in simulations. While we succeeded in correctly modeling M_{ii} for two RTOs, doing the same for $i \neq j$ and longer vectors δ is very tedious.

4.6.2 Hershel+

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

Now suppose that the scanner finds a remote host on the Internet and obtains a vector of SYN-ACK arrival instances as A, which are relative to the transmission time of the SYN. The main caveat here is that the forward SYN delay and server think time, which we collectively call T, are not just unknown in the public Internet, but also likely non-negligible. Consequently, the classifier must consider all options for T in its decision whether the observed A could have been produced by some known vector A_i. Delay randomness is handled similar to (4.7), which means that it is again sufficient to consider only deterministic pairs of delay vectors, i.e., by conditioning on $A = a = (a_1, a_2, \ldots)$ and $A_i = a' = (a'_1, a'_2, \ldots)$. This is illustrated in Fig. 4.5. Supposing that Q_m is the m-th OWD from the server to the client, we have $a_m = T + a'_m + Q_m$.

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

$$p(a|a', \mathcal{X}) = E\Big[\prod_{m=1}^{|a|} f_Q(e_m - T)\Big], \tag{4.23}$$

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

$$p(a|a', \mathcal{X}) = \int_0^s \Big[\prod_{m=1}^{|a|} f_Q(e_m - x)\Big] f_T(x)dx. \tag{4.24}$$

We apply Hershel's exponential OWD with $f_Q(x) = \lambda e^{-\lambda x}$ and additionally represent T as a sum of two exponential variables (i.e., forward SYN delay and server think time), which leads to $f_T(x) = \nu^2 x e^{-\nu x}$, i.e., Erlang(2) distribution with some rate ν and mean $2/\nu$. The OWD classifier (4.24) is more complex than Hershel's as it requires numerical integration of a computationally expensive product of shifted density functions. Our next result shows that this can be avoided through additional derivations.

Theorem 3. *The closed form for* (4.24) *is*

$$p(a|a',\mathcal{X}) = \mathbf{1}_{s\geq 0}\nu^2\lambda^{|a|}\psi\prod_{m=1}^{|a|}e^{-\lambda e_m}, \tag{4.25}$$

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

$$\psi = \begin{cases} \frac{1-e^{-(\nu-\lambda|a|)s}(1+(\nu-\lambda|a|)s)}{(\nu-\lambda|a|)^2} & |a| \neq \frac{\nu}{\lambda} \\ \frac{s^2}{2} & |a| = \frac{\nu}{\lambda} \end{cases}. \tag{4.26}$$

Proof. If $s < 0$, there exists m such that $e_m - T$ is less than zero. Since OWD cannot be negative, the corresponding term $f_Q(e_m - T) = 0$. Consequently, we need to consider only $s \geq 0$, in which case all e_m are non-negative. Substituting the densities of Q and T into (4.24), we get

$$\begin{aligned} p(a|a',\mathcal{X}) &= \mathbf{1}_{s\geq 0}\int_0^s \Big[\prod_{m=1}^{|a|}\lambda e^{-\lambda(e_m-x)}\Big]\nu^2 x e^{-\nu x}dx \\ &= \mathbf{1}_{s\geq 0}\nu^2\lambda^{|a|}\Big[\prod_{m=1}^{|a|}e^{-\lambda e_m}\Big]\int_0^s xe^{(\lambda|a|-\nu)x}dx. \end{aligned}$$

Using WolframAlpha's integral solver [117] yields (4.25). □

Replacing Hershel's $p(\delta|\delta',\mathcal{X})$ with (4.25) and keeping the rest of the method unchanged gives rise to a technique we call Hershel+. Our next step is to verify that its accuracy is no worse than that of Hershel even when the assumed Erlang model for T, which uses $\nu = 4$ in all computation below, does not match the true distribution. To this end, we use the simulation setup from Section 3.4, where the only new parameter is T. In the first scenario, we keep T uniform in $[0, 1]$ seconds, maintain zero packet loss, and run both methods over Hershel's original database with 116 stacks. The result is shown

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

Table 4.3: Accuracy on the Hershel database.

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

Table 4.4: Accuracy on the Plata database.

in Table 4.3. As the new model only changes the RTO classifier, the most important comparison involves the first three rows of the table, which confirm superiority of Hershel+. In the second scenario, we fix the OWD to be uniform in $[0, 1]$ and use the larger Plata database. Table 4.4 shows that Hershel+ again edges out Hershel, despite its higher uncertainty related to T.

4.6.3 Closed-Form Plata-Hershel+ Matrix

Armed with the new classifier, we revisit the issue of obtaining a Plata matrix without Monte-Carlo simulations. To model $\mathcal{X}$, we disturb each A_i using a random OWD vector $V = (V_1, V_2, \ldots)$, where all V_i are iid exponential with rate λ. We additionally apply noise to the forward SYN delay and server think time, which are collectively given by an Erlang(2) random variable W with rate ν. Note that we use λ and ν from Hershel+, although other options are possible.

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

$$\zeta(a, a') := E[p(a + V + W|a', \mathcal{X})] \tag{4.27}$$

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

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

$$\zeta(a, a') = \upsilon \int_{-\infty}^{\infty} e^{-\lambda \sum_m |e_m + z|}(1 + \nu|z|)e^{-\nu|z|}dz. \tag{4.28}$$

Proof. We first require the following Lemma.

Lemma 1. *Define $Z = W - T$, where W and T are Erlang(2) with rate ν. The density of Z is then*

$$f(z) = \frac{\nu}{4}e^{-\nu|z|}(1 + \nu|z|). \tag{4.29}$$

Proof. Notice that $W - T$ has the same distribution as $X + Y$, where X, Y are iid Laplace with the same rate ν. Their convolution for $z \geq 0$ produces

$$
\begin{aligned}
f_{X+Y}(z) &= \int_{-\infty}^{\infty} f_X(x) f_Y(x-z) dx \\
&= \frac{\nu^2}{4} \int_{-\infty}^{\infty} e^{-\nu|x|} e^{-\nu|x-z|} dx \\
&= \frac{\nu^2}{4} \Big[\int_{-\infty}^{0} e^{\nu x} e^{\nu(x-z)} dx + \int_{0}^{z} e^{-\nu x} e^{\nu(x-z)} dx \\
&+ \int_{z}^{\infty} e^{-\nu x} e^{-\nu(x-z)} dx \Big] \\
&= \frac{\nu^2}{4} \Big(\frac{e^{-\nu z}}{2\nu} + z e^{-\nu z} + \frac{e^{-\nu z}}{2\nu} \Big) \\
&= \frac{\nu^2}{4} \Big(\frac{2e^{-\nu z}}{2\nu} + z e^{-\nu z} \Big) = \frac{\nu^2}{4} \Big(\frac{e^{-\nu z} + \nu z e^{-\nu z}}{\nu} \Big).
\end{aligned}
$$

Combining with the symmetric case $z < 0$, we get (4.29). □

Now we are ready to establish Theorem 4. The general form of this expectation is

$$
\begin{aligned}
\zeta(a, a') &= E\Big[\prod_{m=1}^{|a|} f_Q(a_m + V_m + W - a'_m - T) \Big] \\
&= E\Big[\prod_{m=1}^{|a|} f_Q(e_m + V_m + W - T) \Big] \\
&= E\Big[\prod_{m=1}^{|a|} f_Q(e_m + V_m + Z) \Big], \qquad (4.30)
\end{aligned}
$$

where $Z = W - T$. Note that e_m can be negative as long as the sum $e_m + V_m + Z \geq 0$. Condition on $Z = z$ and define

$$
\begin{aligned}
\zeta_z(a, a') &:= E\Big[\prod_{m=1}^{|a|} f_Q(e_m + V_m + z) \Big] \\
&= \prod_{m=1}^{|a|} \int_{d_m}^{\infty} f_Q(x - c_m) f_V(x) dx, \qquad (4.31)
\end{aligned}
$$

where $c_m = -(e_m+z)$, $f_V(x) = \lambda e^{-\lambda x}$ is the density of each V_m, and the integration range starts at $d_m = \max(c_m, 0)$ to ensure the terms inside the density f_Q are non-negative. This leads to

$$\begin{aligned}\zeta_z(a,a') &= \prod_{m=1}^{|a|} \int_{d_m}^{\infty} \lambda e^{-\lambda(x-c_m)} \lambda e^{-\lambda x} dx \\ &= \prod_{m=1}^{|a|} \frac{\lambda}{2} e^{-\lambda(2d_m - c_m)}. \end{aligned} \tag{4.32}$$

Since $2\max(x,0) - x = |x|$, this yields

$$\zeta_z(a,a') = (\frac{\lambda}{2})^{|a|} e^{-\lambda \sum_m |c_m|}. \tag{4.33}$$

Unconditioning Z and recalling $c_m = -(e_m + z)$,

$$\begin{aligned}\zeta(a,a') &= \int_{-\infty}^{\infty} \zeta_z(a,a') f_Z(z) dz \\ &= (\frac{\lambda}{2})^{|a|} \int_{-\infty}^{\infty} e^{-\lambda \sum_m |e_m+z|} f_Z(z) dz, \end{aligned} \tag{4.34}$$

which leads to (4.28) after invoking Lemma 1. ☐

Note that (4.28) can be computed by splitting the integral into $|a|+2$ regions such that $|z|$ and $|e_m + z|$ are conclusively resolved as being either positive or negative. Each of these smaller integrals expands in closed-form; however, due to the large number of terms involved and lacking structure, this result is difficult to represent symbolically. Algorithmically, however, this is simple to code using a bit-vector of size $|a|+1$ that keeps track of which of the terms $(z, e_1 + z, e_2 + z, \ldots)$ is positive. Moving from one interval to the next flips one bit from 0 to 1 and switches to the corresponding integral.

After verifying that (4.28) and its $|a|+2$ sub-integrals produce correct results, we run

Plata separation over H instead of $\tilde{M}$ and obtain 420 signatures, out of which 79 come out unlabeled. Recalling Table 4.1, notice that Hershel+ increases the dimension of its predecessor by 22 entries, indicating a more powerful classifier. Performance improvement is remarkable as well – the runtime reduces from over 24 hours to just 12 minutes. Added benefits include higher accuracy of Hershel+ decisions and alleviation of uncertainty if r is large enough to keep Monte-Carlo results convergent.

4.7 Internet Scan

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

4.7.1 Classification Results

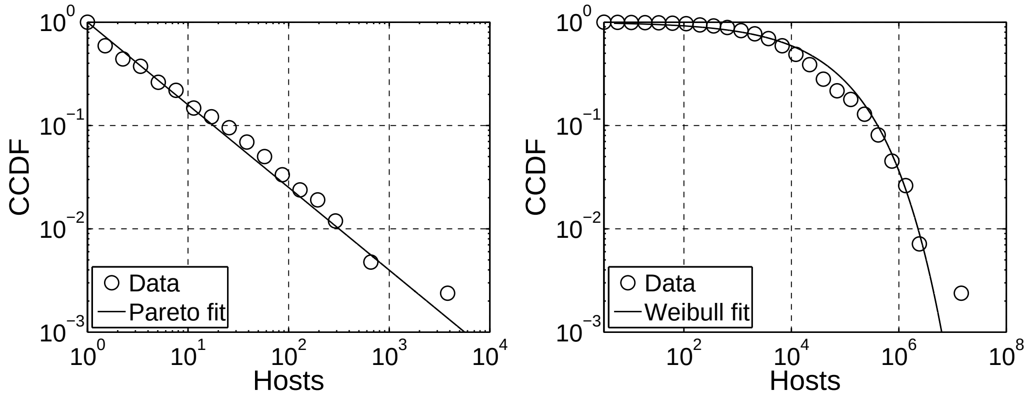
In July 2015, we sent 2.7B SYN probes on port 80 to every IP address advertised in BGP and obtained SYN-ACK responses from 66.4M hosts. This is almost double the 37M IPs used in the Hershel study. The scan lasted 6 hours and operated at 125K packets per second.

Table 4.5 shows the Hershel+ output on the Internet data. We break down the result by OS category, showing the 5 most-popular signatures in each. Not surprisingly, Linux still dominates the webserver market. Although its top-5 signatures are separable at the feature level, limitations of our banner-based labeling do not allow identification of the specific version of these OSes. In second place, there is a large number of embedded devices, mostly routers and printers. This finding agrees with those in previous measurements at this scale [42], [55]. In third place, we combine hosts that map to a signature without a useful banner and those with a zero probability of matching to anything in $\mathcal{D}$. The former category is responsible for 94% of these cases, where 79 "mystery" signatures in $\mathcal{D}$ catch almost 12% of all Internet classification, despite being rare on our campus. Future work will attempt to uncover their OS.

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

Table 4.5: OS classification of the Internet dataset.

Next, there is Windows in fourth place with 7M hosts. Unlike the previous categories, we can identify the specific type of Windows from its IIS version in the HTTP header. While it is by far the most popular desktop OS [70], its penetration of the webserver domain has been lagging behind Linux. This is in contrast to our campus scan, which was dominated by Windows. One explanation for Unix prevalence is migration of online services to enterprise clouds, which have traditionally favored Linux installations. Another is the possibility that Linux distributions more commonly enable a webserver in their de-



(a) campus (b) Internet

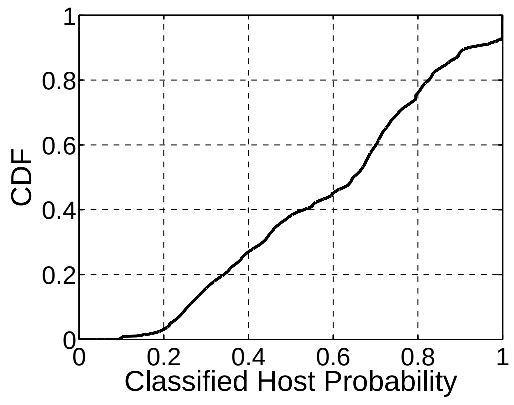
Figure 4.6: OS popularity distributions.

fault configurations or alias more IPs to the same physical server. And yet another is a higher percentage of Unix computers not being protected by a firewall (either corporate or host-level).

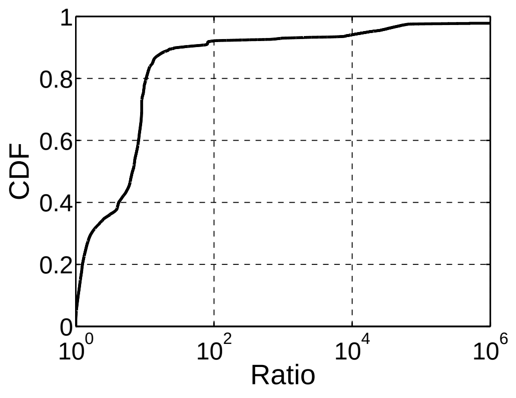
The table ends with 752K devices (1.1% of the total) in the "other" category that includes BSD, Mac, Novell, and Solaris. Compared to our previous large-scale fingerprinting effort that used scans from July 2010, the table shows that Linux and embedded have doubled their numbers (i.e., from 13-14M to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

4.7.2 OS Popularity and Confidence

To better understand device deployment at different scales, we next examine the distribution of cluster size W for each of the 420 signatures in our database. Fig. 4.6(a) shows the CCDF $P(W > x)$ using the initial campus scan. This plot is a close match to Pareto tail $(x/\beta)^{-\alpha}$, where $\alpha = 0.8$ and $\beta = 1$. Interestingly, the bottom 40% of the signatures



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



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

Figure 4.7: Hershel+ classification confidence.

map to a single host each. In contrast to the well-known stacks in Table 4.2, these clusters enjoy more esoteric items such as security cameras, room-temperature controllers, UPS (uninterruptable power supplies), tape backup, humidity sensors, and even discontinued oscilloscopes. Fig. 4.6(b) plots the same tail for the Internet scan, which is a good match to the Weibull distribution $\exp(-(x/\lambda)^k)$, where $k = 0.4$ and $\lambda = 45$K. Each of the top-14 signatures accounts for at least 1M hosts and the top-17 are responsible for 60% of IPs. The bottom 204 signatures match a combined 1% of the servers (i.e., 664K).

Another interesting issue is the amount of confidence with which Hershel+ selects the best OS during classification. Assuming a is a measurement from some IP, (4.24)-(4.25) can be used to obtain similarity score $p(a|A_j, \mathcal{X})$ for each OS j, the highest of which is selected as the match after normalization. Fig. 4.7(a) plots the distribution of this probability across all 66.4M IPs. Observe that almost no classifications occur with less than 20% likelihood and over half the hosts fit some signature with probability at least 65%. The far end of the CDF shows 7% of the IPs with a 100% match, which are devices with truly unique combinations of features. In the same vein, to determine if the second-best match follows closely the top signature and how often the classifier might be "guessing," Fig. 4.7(b) shows the CDF for the ratio of the two highest probabilities. In 17% of the cases, the second-best match is pretty close, i.e., within a factor of 1.2. Afterwards, the curve sharply rises and yields over 68% of IPs with a decisive winner (i.e., ratio 2:1 or better).

4.8 Comparison with Nmap

Since ground-truth for millions of Internet hosts is difficult to obtain, we next perform comparison against Nmap v6.49 [73]. During the scan, we randomly selected 1% of responsive hosts and invoked Nmap to fingerprint them as soon as the first SYN-ACK was received. Real-time execution was needed to minimize the possibility they left the network and other hosts appeared in their place (e.g., due to DHCP). We used Nmap's least-verbose mode to limit the traffic and complaints from target networks; however, this still resulted in 80 sent and 60 received packets per IP, as well as several notifications to campus network administrators about intrusive activity coming from our subnet. The complaints identified Nmap by name, which raises questions how often IDS tools not just detect, but drop its traffic.

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

4.8.1 Agreement

We first investigate how well Nmap and Hershel+ agree on the classification of the selected subset of hosts. Comparison with Nmap is far from straightforward since its stack names are human-created and rather fine-granular. The most detailed category in our Plata

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

Table 4.6: Internet subsample classification.

database is Windows, while the majority of other hosts are marked with just the name of the OS and/or device. Thus, it makes sense to separately consider whether Hershel+ matches the exact signature string of Nmap or just the category.

Table 4.6 shows the result of this process, where we group hosts based on Hershel+ classification. In the category match, we achieve over 98% agreement in Linux, 95% in Windows, and 100% in "other." With embedded systems, Nmap often claims the host is running Linux, whereas we have a specific (non-Linux) device name from the banner. Without tedious manual effort, it is difficult to know if Nmap has been exposed to these devices and whether it can reliably identify them. With that said, we still mark these cases as a mismatch, which drops the agreement rate to 75%.

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

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

Table 4.7: Hershel+ classification and features.

4.8.2 Disagreement

We now analyze the peculiar case of the four Windows hosts from Table 4.6 where Nmap and Hershel+ disagree. We call these observations $S_1, \ldots, S_4$. Table 4.7 shows their features and the corresponding database signatures $D_1 - D_2$ for the Hershel+ classification. Notice that S_1 is an easy classification decision because the RTT is small (i.e., ≈ 220 ms) and D_1 matches all of its features. For S_2, Hershel+ prefers the same OS, overcoming a change in TTL/MSS and a loss of the RST packet at 21 sec. For the other two hosts, both matching to D_2, the only discrepancy is the MSS, which is a highly volatile field that depends on the MTU. Judging from the OPT and RTO features, the accuracy of these decisions is not in doubt.

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

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

$$\frac{\sum_{i=1}^{l}(Y_i \mathbf{1}_{R_i=R'_i} w_i + R_i R'_i \sum_j Z_{ij} \mathbf{1}_{F_{ij}=F'_{ij}} w_{ij})}{\sum_{i=1}^{l}(Y_i w_i + R_i R'_i \sum_j Z_{ij} w_{ij})}, \tag{4.35}$$

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

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

Armed with this insight, consider in Table 4.8 the Nmap features of $S_1 - S_4$, as well as their best matches – a network boot card, modem jail-break firmware, a decade-old OpenBSD 4.3, and an ancient D-link switch – where S_1 scores over 85% with both D_1 and D_2. From the table, notice that Nmap sampled the same SYN features as Hershel+, meaning they contacted similar network stacks. For inexplicable reasons, the database allows $\emptyset$ for mandatory values (e.g., Win, TTL, DF), where all four entries $D_1 - D_4$ contain at least one such case.

Based on Table 4.8, it is pretty clear that Nmap decisions are heavily influenced by the R vector and empty fields. Indeed, iPXE/Tomato have no features F_{ij} in common with S_1, OpenBSD 4.3 matches S_2 only in three fairly generic fields TTL/DF/MSS, and D-Link agrees with S_3/S_4 in just the DF bit. We thus find no evidence to suggest that Nmap signatures $D_1 - D_4$ are statistically probable, let alone better than the Hershel+ result in Table 4.7. In fact, D_3 and S_2 are conclusively different stacks judging from their ordering

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

Table 4.8: Nmap classification and features.

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

Table 4.9: Popularity of Nmap signatures.

of non-NOP TCP options (i.e., MSWT vs MWST).

From a broader perspective, Table 4.9 shows the number of hosts for which Nmap decides that $D_1 - D_4$ exceed the 85% threshold. Remarkably, Tomato appears in 21% of the cases and OpenBSD in 13%. These results raise questions about Nmap's ability to provide meaningful classification, not just in the four cases we dissected, but generally in wide-area networks, where R is easily distorted by IDS, host-level packet filters, and network firewalls.

4.9 Conclusion

In this section, we introduced a novel unsupervised clustering algorithm called Plata, which can be used to separate gathered signatures according to known noise model, eliminate duplicates, and allow user tuning of the separability desired. We applied Plata to a scan of our university campus, capturing 420 unique signatures, labelled them automatically, and used our new automatically built database to accomplish the largest OS fingerprinting effort ever achieved in the wild. We concluded by comparing our results with Nmap, showing that Hershel+ classification from using just a single probe agrees with Nmap in most cases, and provides more accurate results in cases where they do not.

5. ITERATIVE LEARNING OF FEATURE DISTORTION

5.1 Introduction

Now that we have built a scalable classifier in Hershel+ and a framework for constructing an automated database with Plata, in this section we turn our attention to solving the existing issues remaining in our system, as well as looking at some measurement applications of our work. One point of concern is that Hershel+ 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 is to understand these limitations and offer novel avenues for increasing both the classification accuracy and amount of information that can be recovered from responses to a SYN packet.

Assume a database of known fingerprints $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ and an observation $\mathbf{x}' = (\mathbf{x}'_1, \dots, \mathbf{x}'_m)$ from a large number of Internet hosts. Suppose vector $\alpha = (\alpha_1, \dots, \alpha_n)$ 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 [7], [112], 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 unstable states and inferior results compared to staying with the default parameters, as we discovered in Section 3.

5.1.1 Contributions

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) [24], [41]. We call this approach Faulds[1] 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 $\mathbf{x}'$.

Throughout the section, we provide a combination of simulations and discussion of the various caveats. For example, letting T be the forward latency and Δ be the reverse SYN-ACK delay, one of the properties of active OS fingerprinting is that it cannot observe these variables individually. Instead, they are always coupled into a summation $T + \Delta$ that is carried in $\mathbf{x}'$. We show that under certain conditions Faulds can successfully deconvolve these distributions without having any a-priori knowledge about them and explain

[1]Henry Faulds was a Scottish scientist who proposed the first usable forensic fingerprint-identification method in 1880.

the rationale behind this seemingly impossible act.

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 shows non-trivial changes compared to Hershel+, but also provide estimated distributions of one-way return delay, forward latency, and one-way SYN-ACK packet loss across 63.5M webservers.

5.2 Background

Characterizing Internet packet loss and delay is a long-standing problem in computer networks [12], [25], [80], [119], where availability of realistic models can fuel protocol development, provide theoretical insight, and improve simulations. Besides traditional usage in congestion control (e.g., retransmission-timer tuning, stability analysis), knowledge of delay/loss are important in research of many modern applications (e.g., content distribution networks, video streaming, VoIP, cloud computing).

One particularly hard issue is estimating one-way path properties, especially at large scale. Perhaps the simplest approach is to measure round-trip parameters [123] and then infer their one-way counterparts using certain assumptions. However, path asymmetry and the influence of the remote server on the measured parameter (e.g., delay in the reply) may lead to bias in the result [47], [79], [114]. An alternative is to deploy a measurement platform that allows control of both sides of a connection (e.g., PlanetLab [82], IDmaps [33], CDNs [44], [48], [116]). Clock-synchronization issues notwithstanding [39], this leads to accurate estimation, but requires a significant effort to build the underlying infrastructure and extrapolate the measurements to other parts of the Internet. The final technique places an observer inside routers that monitor ongoing connections [36]; however, this does not make estimation any less expensive.

Other broader, but related, topics include distance measurement using coordinate systems [16], [21], [57], [72], [98], [106], involvement of DNS for estimation of round-trip delay/loss between remote servers [37], [54], [113], and application of EM to various networking problems (e.g., tomography [14], [19], [99], [111], flow sampling [26], [53]).

5.3 Learning from Observation

5.3.1 General Problem

Suppose the OS database consists of $n \geq 1$ known stacks $(\omega_1, \ldots, \omega_n)$, each with some vector-valued fingerprint $\mathbf{x}_i = (x_{i1},\, x_{i2}, \ldots)$. These contain a combination of features, including temporal information about the SYN-ACK retransmission timer of each OS and default header values used for incoming connections. Further assume a set of observations $\mathbf{x}' = (\mathbf{x}'_1, \ldots, \mathbf{x}'_m)$, where each $\mathbf{x}'_j = (x'_{j1}, x'_{j2}, \ldots)$ is also a vector. Note that $\mathbf{x}'$ is typically obtained by scanning the Internet and eliciting responses from every live server. For the type of OS fingerprinting considered here, i.e., single-probe, this is done using a SYN probe to every IP address in BGP and collection of SYN-ACKs/RSTs from the contacted servers.

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 presence of distortion that randomly modifies the original features of the system. This typically involves 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 and/or obscure the end-system.

Define $\alpha_i = p(\omega_i)$ to be the fraction of hosts with OS i and let $\alpha = (\alpha_1, \ldots, \alpha_n)$ be the corresponding vector. Suppose θ denotes the distribution of distortion and $p(\mathbf{y}|\omega_i, \theta)$ is the probability that the fingerprint of signature i has been changed into $\mathbf{y}$ under θ. On the other hand, assume $p(\omega_i|\mathbf{y}, \theta, \alpha)$ 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, the classifier must determine for each j the one database entry ω_i with the largest

$$p(\omega_i|\mathbf{x}'_j, \theta, \alpha) = \frac{\alpha_i p(\mathbf{x}'_j|\omega_i, \theta)}{p(\mathbf{x}'_j|\theta, \alpha)}, \tag{5.1}$$

where

$$p(\mathbf{x}'_j|\theta, \alpha) = \sum_{\ell=1}^{n} \alpha_\ell p(\mathbf{x}'_j|\omega_\ell, \theta). \tag{5.2}$$

Analysis of (5.1) in our earlier sections assumed that α was uniform (i.e., $\alpha_i = 1/n$) and θ was known ahead of time (e.g., exponential one-way delays with mean 500 ms). Therefore, both α_i and denominator $p(\mathbf{x}'_j|\theta, \alpha)$ were independent of i and could be omitted from the optimization, leaving only $p(\mathbf{x}'_j|\omega_i, \theta)$ as the target. 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 subsection deals with estimating α, the next one covers network distortion, and the one after that focuses on modification to fixed header features.

5.3.2 EM Principles

Suppose $\mathbf{X}(\alpha, \theta)$ is a random variable that models the feature vector observed from a uniformly selected system on the Internet. Note that randomness in $\mathbf{X}(\alpha, \theta)$ arises due to both selection of the host and feature distortion during observation. Then, knowing the distribution of $\mathbf{X}(\alpha, \theta)$ allows us to write a set of (generally non-linear) equations

$$P(\mathbf{X}(\alpha, \theta) = \mathbf{y}_\ell) = \lim_{m \to \infty} \frac{1}{m} \sum_{j=1}^{m} \mathbf{1}_{\mathbf{x}'_j = \mathbf{y}_\ell} = p(\mathbf{y}_\ell|\theta, \alpha), \tag{5.3}$$

i.e., one equation for each unique vector $\mathbf{y}_\ell$ from the domain of $\mathbf{X}(\alpha, \theta)$. Extraction of α and θ from (5.3) commonly involves the Expectation-Maximization (EM) method, which is a technique that solves this system using fixed-point iteration [24], [41]. At every step t, it maximizes the expected log-likelihood function conditioned on the parameters obtained during the previous iteration $t - 1$.

Assuming (5.3) is *identifiable* (i.e., each combination (α, θ) produces a different distribution for $\mathbf{X}(\alpha, \theta)$), EM can accurately recover the unknown parameters [118]. Identifiability is equivalent to (5.3) having a unique solution (α, θ) for any valid distribution of observations, which is a strong condition; however, EM is also usable in *partially identifiable* cases where (α, θ) is a locally-stable maximum for which common-sense knowledge about the system allows selection of the initial state in close-enough proximity. If not, multiple restarts and/or other heuristics (e.g., simulated annealing) can be deployed as well.

Stability, convergence, and numerical computation of EM under multiple fixed points is a topic beyond our scope, but it should be noted that as long as the number of non-redundant equations is larger than the number of unknown variables, EM works well for many problems in practice.

5.3.3 Fingerprint Popularity

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 (5.3) forms a linear system around the unknown parameters. It has a unique solution as long as the rank of the matrix with elements $A_{i\ell} = p(\mathbf{y}_\ell|\omega_i)$ is n.

Throughout the section, 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.,

$$\alpha_i^{t+1} = \frac{1}{m}\sum_{j=1}^{m} p(\omega_i|\mathbf{x}'_j, \theta, \alpha^t). \tag{5.4}$$

Note that this is markedly different from updating the popularity vector 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 [49]. In fact, all previous fingerprinting tools [7], [10], [73], [120], [122] 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.

It is not difficult to see that for $m \to \infty$, fixed points of recurrence (5.4) are solutions to (5.3). Additionally, there is a stronger result. While it is fairly well-known, its proof methodology is needed for later parts of the section.

Theorem 5. *For a classifier with fixed θ,* (5.4) *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

$$p(\mathbf{x}'|\theta, \alpha) := \prod_{j=1}^{m} p(\mathbf{x}'_j|\theta, \alpha) = \prod_{j=1}^{m}\sum_{i=1}^{n} \alpha_i p(\mathbf{x}'_j|\omega_i, \theta). \tag{5.5}$$

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 (5.5), 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

$$\begin{aligned} p(\mathbf{x}', z|\theta, \alpha) &:= \prod_{j=1}^{m} p(\mathbf{x}'_j, z_j|\theta, \alpha) = \prod_{j=1}^{m} p(\mathbf{x}'_j|z_j, \theta)p(z_j|\alpha) \\ &= \prod_{j=1}^{m} \alpha_{z_j} p(\mathbf{x}'_j|\omega_{z_j}, \theta). \end{aligned} \tag{5.6}$$

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

$$\begin{aligned} \log p(\mathbf{x}', z|\theta, \alpha) &= \sum_{j=1}^{m} (\log \alpha_{z_j} + \log p(\mathbf{x}'_j|\omega_{z_j}, \theta)) \\ &= \sum_{j=1}^{m} \sum_{i=1}^{n} (\log \alpha_i + c_{ij}) \mathbf{1}_{z_j = i}, \end{aligned} \tag{5.7}$$

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 (5.7) with respect to z, conditioned on the previous values α^t and the available observations, producing

$$\begin{aligned} Q(\alpha|\alpha^t) &:= E_z[\log p(\mathbf{x}', z|\theta, \alpha)|\mathbf{x}', \theta, \alpha^t] \\ &= \sum_{j=1}^{m} \sum_{i=1}^{n} (\log \alpha_i + c_{ij}) E[\mathbf{1}_{z_j = i}|\mathbf{x}', \theta, \alpha^t] \\ &= \sum_{j=1}^{m} \sum_{i=1}^{n} (\log \alpha_i + c_{ij}) \beta_{ij}^t, \end{aligned} \tag{5.8}$$

where

$$\beta_{ij}^t := p(\omega_i|\mathbf{x}'_j, \theta, \alpha^t) = \frac{\alpha_i^t p(\mathbf{x}'_j|\omega_i, \theta)}{\sum_{\ell=1}^{n} \alpha_\ell^t p(\mathbf{x}'_j|\omega_\ell, \theta)}. \tag{5.9}$$

The M-step maximizes (5.8) with respect to the unknown parameter α and entails solving

$$\frac{\partial Q(\alpha|\alpha^t)}{\partial \alpha_i} = 0. \tag{5.10}$$

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

$$\sum_{j=1}^{m} \Big(\frac{\beta_{ij}^t}{\alpha_i} - \frac{\beta_{nj}^t}{\alpha_n} \Big) = 0. \tag{5.11}$$

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

$$\alpha_i = \frac{1}{c} \sum_{j=1}^{m} \beta_{ij}^t. \tag{5.12}$$

From normalization $\sum_{i=1}^{n} \alpha_i = 1$, it follows that c must be m and that additionally (5.12) applies to $i = n$. We therefore get (5.4). $\square$

5.3.4 Discussion

We now address the question of whether (5.4) 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.

To accomplish this, we use the simulation setup described in Section 3.4. For the scenario we call S_1, there are four different cases for the distribution of foward/reverse

Case	Forward latency		One-way delay		Loss
	Distribution	Mean	Distribution	Mean	
S_{11}	Erlang(2)	0.5	Exp	0.5	0.038
S_{12}	Pareto	0.5	Pareto	0.5	0.5
S_{13}	Reverse exp	1.5	Erlang(2)	0.5	0.1
S_{14}	Pareto	0.1	Uniform	0.1	0.5

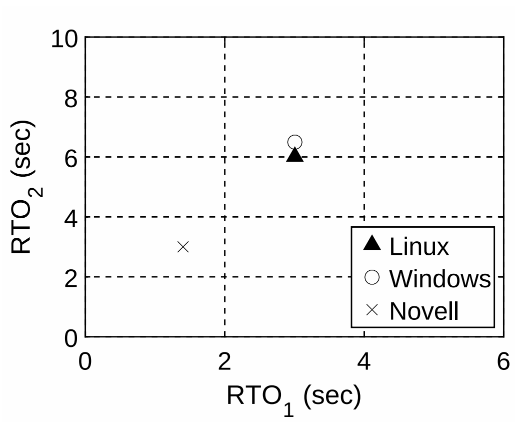
Table 5.1: Network distortion in scenario S_1.

delays and packet-loss probability. These are shown in Table 5.1 and discussed in more detail next.

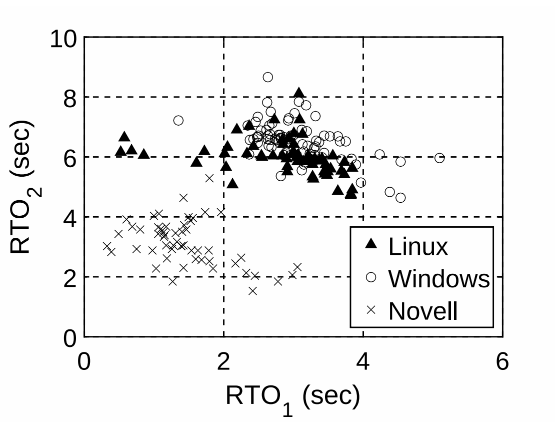
The first row matches exactly the assumed parameters θ in Hershel+. The second row uses Pareto delays with mean 500 ms and 50% loss, emulating highly volatile network conditions. The next row uses a shifted *reverse-exponential* forward latency with CDF $e^{-\lambda(2-x)}$, defined for $-\infty < x \leq 2$, 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 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). We retain the first two retransmission timeouts (RTOs), remove all fixed header features, and obtain the fingerprints in Figure 5.1(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 200 points per plot and remove observations with lost packets.

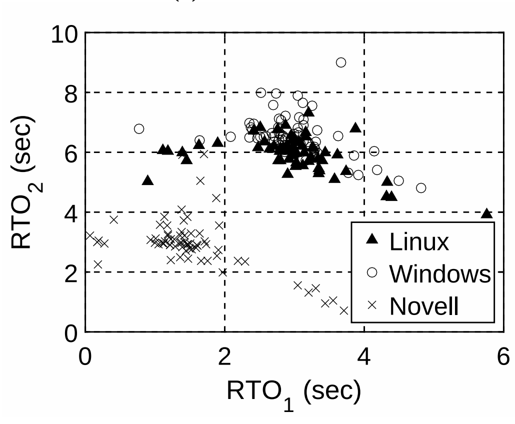
Define ρ^t to be the fraction of correct classifications for a given method during iteration t, where $t = \infty$ represents the convergence point of the underlying estimator (usually 20-40 iterations). If the method does not perform iteration, only ρ^1 is meaningful. We consider



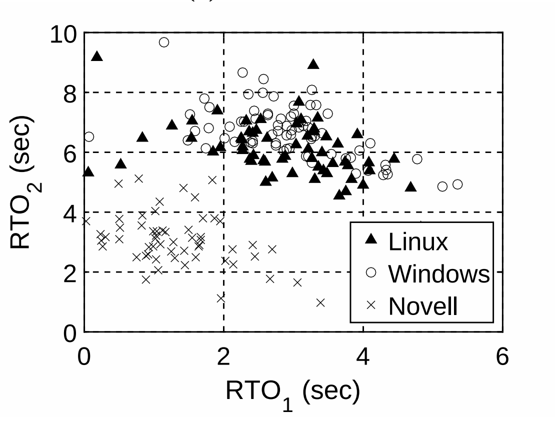
(a) database D_1



(b) case S_{11}



(c) case S_{12}



(d) case S_{13}

Figure 5.1: Database and distorted observations.

three techniques – Hershel+, hard EM with multiple iterations, and soft EM in (5.4), all using the same function $p(\mathbf{x}'_j|\omega_i, \theta)$ and starting from uniform popularity $\alpha_i^0 = 1/n$. Note that the former two methods estimate α using

$$\alpha_i^{t+1} = \frac{1}{m}\sum_{j=1}^{m} \mathbf{1}_{\operatorname{argmax}_i p(\omega_i|\mathbf{d}_j,\theta,\alpha^t)=i}. \tag{5.13}$$

Results of this process with $m = 2^{18}$ observations are shown in Table 5.2. 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 (5.4) solves this issue.

Swapping (α_1, α_2), the second simulation in Table 5.2 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 (5.4) 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 (5.4) 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 [7], [10], [73], [120], [122], may generally 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.

5.4 Network Features

5.4.1 Distortion Model

Our goal in this subsection 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 our classification assumes that distortion is applied to each subvector independently, it follows that

$$p(\mathbf{x}'_j|\omega_i, \theta) = p(\mathbf{d}'_j|\omega_i, \theta_d)p(\mathbf{u}'_j|\omega_i, \theta_u), \tag{5.14}$$

Case	α	Hershel+		Hard EM		EM in (5.4)	
		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

Table 5.2: Classification results in D_1.

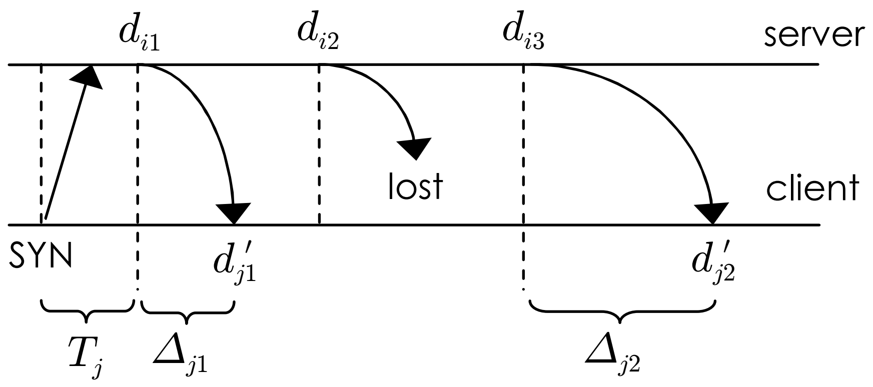


Figure 5.2: Delay features.

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 subsection 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 (5.14), we present a re-illustration of RTOs in Figure 5.2, where a host with network signature $\mathbf{d}_i$ generates an observation $\mathbf{d}'_j$. Measurement begins with a random forward latency T_j, which has some unknown distribution $f_T(\tau) = P(T_j = \tau)$. This includes the time needed for the SYN packet to reach the server and for it to process the request. Along the return path, one-way delays $(\Delta_{j1}, \Delta_{j2}, \ldots)$ are iid random variables with another unknown distribution $f_\Delta(\delta) = P(\Delta_{jr} = \delta)$. 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.

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 is what we considered in Hershel. However Hershel+ switched to usage of absolute timestamps d_{ir} as it identified these as having certain modeling advantages (i.e., independence between delays after conditioning on T_j), and we retain this approach. 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 Figure 5.2, for example, we have $\gamma_j = (1, 3)$. Then, if the j-th observation comes from a system with fingerprint ω_i, it follows that

$$d'_{jr} = T_j + d'_{i,\gamma_j(r)} + \Delta_{jr}, \quad r = 1, 2, \ldots, |\mathbf{d}'_j|. \tag{5.15}$$

As in our earlier sections, we keep the assumption of no reordering due to the large spacing between the packets (often several seconds), which implies $\gamma_j(r+1) > \gamma_j(r)$. Letting $\Gamma(i, j)$ be the set of all monotonic loss vectors that start with $|\mathbf{d}_i|$ packets and finish with $|\mathbf{d}'_j|$, the Hershel+ network classifier can be summarized by

$$\begin{aligned} p(\mathbf{d}'_j|\omega_i,\theta_d) &= \sum_{\tau} f_T(\tau) p(\mathbf{d}'_j|\omega_i,\tau,\theta_d) \\ &= \sum_{\tau} f_T(\tau) \sum_{\gamma\in\Gamma(i,j)} p_i(\gamma) p(\mathbf{d}'_j|\omega_i,\tau,\gamma,\theta_d) \\ &= \sum_{\tau} f_T(\tau) \sum_{\gamma\in\Gamma(i,j)} p_i(\gamma) \prod_{r=1}^{|\mathbf{d}'_j|} f_\Delta(d'_{jr}-\tau-d_{i,\gamma(r)}), \end{aligned} \tag{5.16}$$

where $p_i(\gamma)$ is the probability to observe loss pattern γ under $|\mathbf{d}_i|$ transmitted packets. To avoid clutter, we omit here 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.

5.4.2 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.,

$$f_T^{t+1}(\tau) = \frac{1}{m}\sum_{j=1}^{m} P(T_j=\tau|\mathbf{d}'_j,\theta_d^t,\alpha^t). \tag{5.17}$$

Next, each database signature with k original packets admits 2^k-1 unique loss patterns γ, where k goes as high as $k_{max}=21$. Estimating the probability $p_i(\gamma)$ for each possible option γ is likely to produce too many unknown variables in (5.3) 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 $k = 1, 2, \dots, k_{max}$. 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 quite a bit more general scenarios than the traditional iid Bernoulli model we used in Section 3 and 4.

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. Then, we get

$$q_k^{t+1}(\ell) = \frac{\sum_{j=1}^{m} P(Y_j = k|\theta_d^t, \alpha^t)\mathbf{1}_{|\mathbf{d}'_j|=k-\ell}}{\sum_{j=1}^{m} P(Y_j = k|\theta_d^t, \alpha^t)\mathbf{1}_{|\mathbf{d}'_j|\le k}}, \tag{5.18}$$

from which the estimated probability of pattern γ can be expressed as

$$p_i^t(\gamma) = \frac{q^t_{|\mathbf{d}_i|}(|\mathbf{d}_i| - |\gamma|)}{\binom{|\mathbf{d}_i|}{|\gamma|}}. \tag{5.19}$$

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.,

$$f_\Delta^{t+1}(\delta) = \frac{\sum_{j=1}^{m}\sum_{s=1}^{|\mathbf{d}'_j|} P(\Delta_{js} = \delta|\mathbf{d}'_j, \theta_d^t, \alpha^t)}{\sum_{j=1}^{m} |\mathbf{d}'_j|}. \tag{5.20}$$

5.4.3 Analysis

To make the framework outlined above usable, our next task is to expand it into an explicit recurrence using the distributions contained in θ_d^t, i.e., $(f_T^t, f_\Delta^t, q_k^t)$. Let

$$\delta_{ij\tau\gamma r} = d'_{jr} - \tau - d_{i,\gamma(r)} \tag{5.21}$$

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

$$p^t_{ij\tau\gamma} := \alpha_i^t f_T^t(\tau) p_i^t(\gamma) \prod_{r=1}^{|\mathbf{d}'_j|} f_\Delta^t(\delta_{ij\tau\gamma r}), \tag{5.22}$$

$$\beta^t_{ij\tau\gamma} := p(\omega_i, \tau, \gamma | \mathbf{d}'_j, \theta_d^t, \alpha^t) = \frac{p^t_{ij\tau\gamma}}{\sum_{i\tau\gamma} p^t_{ij\tau\gamma}} \tag{5.23}$$

and consider the next result.

Theorem 6. *Under network distortion, estimators* (5.4), (5.17), (5.18), *and* (5.20) *can be* *written as*

$$\alpha_i^{t+1} = \frac{1}{m}\sum_{j\tau\gamma}\beta_{ij\tau\gamma}^t, \tag{5.24}$$

$$f_T^{t+1}(\tau) = \frac{1}{m}\sum_{ij\gamma}\beta_{ij\tau\gamma}^t, \tag{5.25}$$

$$q_k^t(\ell) = \frac{\sum_{ij\tau\gamma}\beta_{ij\tau\gamma}^t \mathbf{1}_{|\mathbf{d}_j'|=k-\ell,|\mathbf{d}_i|=k}}{\sum_{ij\tau\gamma}\beta_{ij\tau\gamma}^t \mathbf{1}_{|\mathbf{d}_j'|\le|\mathbf{d}_i|=k}}, \tag{5.26}$$

$$f_\Delta^t(\delta) = \frac{\sum_{ij\tau\gamma s}\beta_{ij\tau\gamma}^t \mathbf{1}_{\delta_{ij\tau\gamma s}=\delta}}{\sum_j |\mathbf{d}_j'|}. \tag{5.27}$$

Proof. We start with the recurrence on α. Keeping distortion limited to network features, (5.4) 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 (5.16), we get

$$p(\mathbf{d}_j'|\omega_i,\theta_d^t) = \sum_{\tau\gamma} f_T^t(\tau)p_i^t(\gamma)\prod_{r=1}^{|\mathbf{d}_j'|} f_\Delta^t(\delta_{ij\tau\gamma r}), \tag{5.28}$$

which leads to

$$\alpha_i^t p(\mathbf{d}_j'|\omega_i,\theta_d^t) = \sum_{\tau\gamma} p_{ij\tau\gamma} \tag{5.29}$$

and, leveraging (5.2) for the denominator of (5.28),

$$\alpha_i^{t+1} = \frac{1}{m}\sum_{j=1}^{m}\frac{\sum_{\tau\gamma}p_{ij\tau\gamma}^t}{\sum_{i\tau\gamma}p_{ij\tau\gamma}^t} = \frac{1}{m}\sum_{j\tau\gamma}\beta_{ij\tau\gamma}^t. \tag{5.30}$$

Moving on to the forward latency, notice that (5.17) becomes

$$f_T^{t+1}(\tau) = \frac{1}{m}\sum_{j=1}^{m} \frac{p(\mathbf{d}'_j|\tau,\theta_d^t,\alpha^t)p(\tau|\theta_d^t)}{p(\mathbf{d}'_j|\theta_d^t,\alpha^t)}$$
$$= \frac{1}{m}\sum_{j=1}^{m} \frac{\sum_i \alpha_i^t p(\mathbf{d}'_j|\omega_i,\tau,\theta_d^t) f_T^t(\tau)}{p(\mathbf{d}'_j|\theta_d^t,\alpha^t)}$$
$$= \frac{1}{m}\sum_{j=1}^{m} \frac{\sum_{i\gamma} p_{ij\tau\gamma}^t}{\sum_{i\tau\gamma} p_{ij\tau\gamma}^t} = \frac{1}{m}\sum_{ij\gamma} \beta_{ij\tau\gamma}^t. \tag{5.31}$$

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

$$P(Y_j = k|\theta_d^t,\alpha^t) = \sum_{i=1}^{n} p(\omega_i|\mathbf{d}'_j,\theta_d^t,\alpha^t)\mathbf{1}_{|\mathbf{d}_i|=k}$$
$$= \sum_{i=1}^{n} \frac{\alpha_i^t p(\mathbf{d}'_j|\omega_i,\theta_d^t,\alpha^t)\mathbf{1}_{|\mathbf{d}_i|=k}}{p(\mathbf{d}'_j|\theta_d^t,\alpha^t)}. \tag{5.32}$$

Using this, the numerator of (5.18) expands to

$$\sum_{j=1}^{m} \frac{\sum_i \alpha_i^t p(\mathbf{d}'_j|\omega_i,\theta_d^t,\alpha^t)\mathbf{1}_{|\mathbf{d}'_j|=k-\ell,|\mathbf{d}_i|=k}}{p(\mathbf{d}'_j|\theta_d^t,\alpha^t)}$$
$$= \sum_{ij\tau\gamma} \beta_{ij\tau\gamma}^t \mathbf{1}_{|\mathbf{d}'_j|=k-\ell,|\mathbf{d}_i|=k}. \tag{5.33}$$

Applying the same logic to the denominator of (5.18), we get (5.26). Finally, updates to one-way delay admit the following interpretation

$$P(\Delta_{js} = \delta|\mathbf{d}'_j,\theta_d^t,\alpha^t) = \frac{\sum_{i\tau\gamma} p_{ij\tau\gamma}^t \mathbf{1}_{\delta_{ij\tau\gamma s}=\delta}}{p(\mathbf{d}'_j|\theta_d^t,\alpha^t)}$$
$$= \sum_{i\tau\gamma} \beta_{ij\tau\gamma}\mathbf{1}_{\delta_{ij\tau\gamma s}=\delta}, \tag{5.34}$$

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 (5.27). $\square$

While the result of Theorem 6 may appear daunting due to the number of nested summations, its implementation in practice can be done with little extra overhead compared to Hershel+. Specifically, usage of (5.16) in (5.1) for all i, j already requires five nested loops. In the inner-most loop of that algorithm, (5.27) adds one increment to a hash table that maintains the PMF of one-way delay. Updates in (5.24)-(5.26) are performed much less frequently and, in comparison, consume negligible computation time. The only caveat is that Hershel+ can be optimized to remove the outer summation in (5.16) when f_T is Erlang(2) and f_Δ is exponential, as we described in Section 4.6.2. This 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 7. *Iteration* (5.24)-(5.27) *is the EM algorithm for recovering* (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

$$\begin{aligned} p(\mathbf{d}', H|\theta_d, \alpha) &:= \prod_{j=1}^{m} p(\mathbf{d}'_j, H|\theta_d, \alpha) \\ &= \prod_{j=1}^{m} p(\mathbf{d}'_j|H_j, \theta_d, \alpha) p(H_j|\theta_d, \alpha), \end{aligned} \tag{5.35}$$

where

$$p(\mathbf{d}'_j|H_j, \theta_d, \alpha) = \prod_{r=1}^{|\mathbf{d}'_j|} f_\Delta(d'_{jr} - \tau_j - d_{z_j, \gamma_j(r)}) \tag{5.36}$$

$$p(H_j|\theta_d, \alpha) = \alpha_{z_j} f_T(T_j) p_{z_j}(\gamma_j). \tag{5.37}$$

Define

$$p_{ij\tau\gamma} = \alpha_i f_T(\tau) p_i(\gamma) \prod_{r=1}^{|\mathbf{d}'_j|} f_\Delta(d'_{jr} - \tau - d_{i,\gamma(r)}). \tag{5.38}$$

Following the proof of Theorem 5, the log-likelihood expands to

$$\begin{aligned}\log p(\mathbf{d}', H|\theta_d, \alpha) &:= \sum_{j=1}^{m} \log(p_{z_j,j,T_j,\gamma_j}) \\ &= \sum_{j=1}^{m} \sum_{i\tau\gamma} \log(p_{ij\tau\gamma}) \mathbf{1}_{z_j=i,T_j=\tau,\gamma_j=\gamma}.\end{aligned} \tag{5.39}$$

The expected log-likelihood function is then given by

$$\begin{aligned}Q(\theta_d, \alpha|\theta_d^t, \alpha^t) &= \sum_{ij\tau\gamma} \log(p_{ij\tau\gamma}) p(\omega_i, \tau, \gamma|\mathbf{d}'_j, \theta_d^t, \alpha^t) \\ &= \sum_{ij\tau\gamma} \log(p_{ij\tau\gamma}) \beta_{ij\tau\gamma}^t.\end{aligned} \tag{5.40}$$

Taking partial derivatives with respect to α_i and $f_T(\tau)$, we get a set of equations similar to (5.10)-(5.11). Their solution is trivially given by (5.24)-(5.25). A more interesting case is the loss PMF. Using substitution

$$q_k(k-1) = 1 - \sum_{\ell=0}^{k-2} q_k(\ell), \tag{5.41}$$

in (5.40), we get for $\ell = 0, 1, \ldots, k-2$ that

$$\begin{aligned}\frac{\partial Q(\theta_d, \alpha|\theta_d^t, \alpha^t)}{\partial q_k(\ell)} &= \sum_{ij\tau\gamma} \frac{\mathbf{1}_{|\mathbf{d}'_j|=k-\ell, |\mathbf{d}_i|=k}}{q_k(\ell)/\binom{k}{\ell}} \beta_{ij\tau\gamma}^t \\ &\quad - \sum_{ij\tau\gamma} \frac{\mathbf{1}_{|\mathbf{d}'_j|=1, |\mathbf{d}_i|=k}}{q_k(k-1)} \beta_{ij\tau\gamma}^t.\end{aligned} \tag{5.42}$$

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

Table 5.3: Classification results of network EM in D_1.

Setting c to be the second summation in (5.42) and equating the derivative to zero, we get

$$q_k(\ell) = \frac{1}{c} \sum_{ij\tau\gamma} \mathbf{1}_{|\mathbf{d}'_j|=k-\ell, |\mathbf{d}_i|=k} \binom{k}{\ell} \beta^t_{ij\tau\gamma}. \tag{5.43}$$

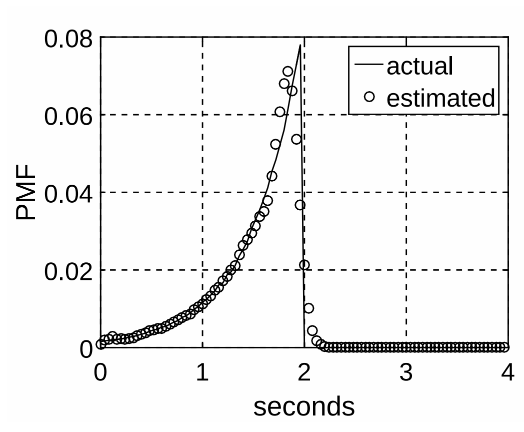
Since the PMF q_k must add up to 1, it follows that

$$\begin{aligned} c &= \sum_{\ell=0}^{k-1} \sum_{ij\tau\gamma} \mathbf{1}_{|\mathbf{d}'_j|=k-\ell, |\mathbf{d}_i|=k} \binom{k}{\ell} \beta^t_{ij\tau\gamma} \\ &= \sum_{ij\tau\gamma} \mathbf{1}_{|\mathbf{d}'_j|\le k, |\mathbf{d}_i|=k} \binom{k}{\ell} \beta^t_{ij\tau\gamma}. \end{aligned} \tag{5.44}$$

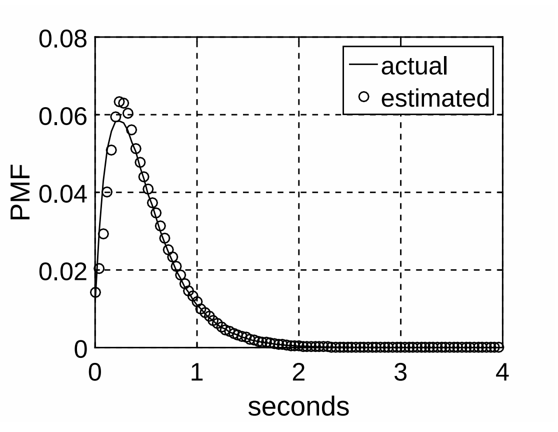
Using this in (5.43) and canceling $\binom{k}{\ell}$ yields (5.26). Note that derivation of (5.27) is very similar. We omit it for brevity. $\square$

5.4.4 Discussion

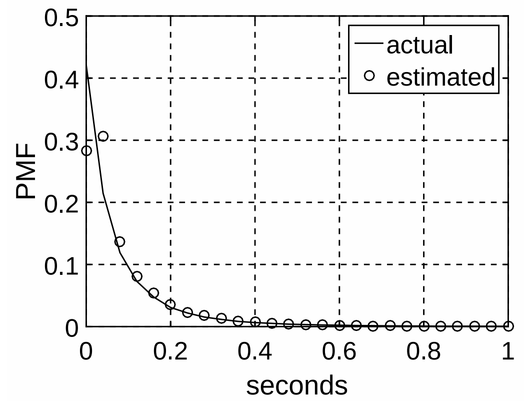
We revisit earlier simulations on dataset D_1, run (5.24)-(5.27) over the same input, and show the result in Table 5.3. Compared to Table 5.2, the derived EM estimator significantly improves the accuracy of classification and vector α, especially in the bottom two rows. Note that S_{12} and S_{14} contain 43% of the observations with just one packet, i.e., zero RTOs. In methods that rely on RTO, these samples would be either discarded as impossible to classify or assigned to a uniformly random signature. In contrast, (5.24)-(5.27) manages



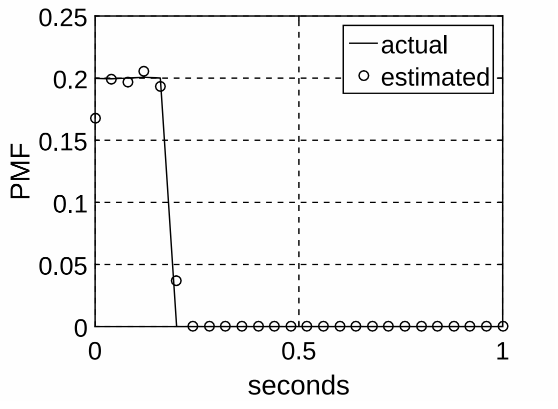
(a) reverse exp f_T (case S_{13})



(b) Erlang(2) f_Δ (case S_{13})



(c) Pareto f_T (case S_{14})



(d) uniform f_Δ (case S_{14})

Figure 5.3: Recovery of delay parameters in D_1.

to do much better because it learns distributions (f_T, f_Δ, α) and makes the best decision possible under these difficult conditions. The accuracy of estimated delay distributions is shown in Figure 5.3. 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 (5.15), where $T_j + \Delta_{jr}$ are always measured together, it may not be obvious how T can be separated from Δ and why the result in Figure 5.3 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.

Case	Delay f_T, f_Δ	q_3	q_4
S_{21}	Same as S_{12}	BinT(3, 0.3)	BinT(4, 0.3)
S_{22}	Same as S_{13}	BinT(3, 0.7)	BinT(4, 0.7)
S_{23}	Same as S_{12}	BinT(3, 0.1)	BinT(4, 0.8)
S_{24}	Same as S_{12}	RevBin(3, 0.1)	RevBin(4, 0.1)

Table 5.4: Network parameters of scenario S_2.

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 Figure 5.3 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 (5.26) 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 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* that has the following property: if $X \sim$ BinT(k, p) and $Y = k - 1 - X$, then $Y \sim$ RevBin(k, p). With this in mind, consider scenario S_2 in Table 5.4. The first two rows have iid loss at 30% and 70%, respectively. The next case applies 10% loss to signatures with 3 packets (i.e., Windows, Novell) and 80% loss to those with 4 (i.e., Linux). The final row uses reverse-binomial loss for all ω_i.

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	0.42	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
S_{23}	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_{24}	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

Table 5.5: Classification results in D_2.

Table 5.5 shows classification results for three methods – Hershel+, the partial EM framework without loss updates (5.26), and the full algorithm from Theorem 6. Not surprisingly, Hershel+ again struggles to recover α, even when its classification accuracy is pretty high. Omission of (5.26) 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 5.6. They all match ground-truth q_k with high precision.

Besides aiding fingerprinting, ability of EM to estimate *one-way* distributions of delay and loss (conditioned on at least one packet surviving) may open up interesting angles to Internet measurement and help with end-to-end sampling of these parameters in scenarios that do not have a suitable infrastructure of cooperating receivers.

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	(0.04, 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)
S_{23}	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_{24}	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)

Table 5.6: Recovery of loss PMFs in D_2.

5.5 User Features

5.5.1 Distortion Model

Our goal in this subsection is to expand the second factor in (5.14) 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 subsection. Note that each field may be allocated a different number of bits and the number of available options a_v for u'_{jv} may depend on v.

Modification to user features, 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 [20], [84], [90], [101], [115] 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 in OS fingerprinting is mostly rule-based, and omits formally modeling user volatility entirely [7], [73], [120], [122]. In Hershel, we introduced a model which assumed that u_{iv} can stay the same with some probability π_v and change to another integer with probability $1-\pi_v$. While this 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 may occur 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 frequently than others (e.g., due to firmware restrictions in printers) 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., non-default window size 57 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 comprises our user-distortion model θ_u. Then, the proposed classifier can be summarized by

$$p(\mathbf{u}'_j|\omega_i, \theta_u) = \prod_{v=1}^{b} \pi_{iv}(u'_{jv}), \tag{5.45}$$

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).

5.5.2 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 – as long as ϕ_{iv} is the largest value in each PMF π_{iv}. Coupling this with an initial state that satisfies the same constraint leads to discovery of a unique solution in (5.3).

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

$$\pi_{iv}^{t+1}(y) = \frac{\sum_{j=1}^{m} p(\omega_i|\mathbf{u}'_j, \theta_u^t, \alpha^t)\mathbf{1}_{u'_{jv}=y}}{\sum_{j=1}^{m} p(\omega_i|\mathbf{u}'_j, \theta_u^t, \alpha^t)}. \tag{5.46}$$

To simplify this expression, define

$$p_{ij}^t := \alpha_i^t p(\mathbf{u}'_j|\omega_i, \theta_u^t, \alpha^t) = \alpha_i^t \prod_{v=1}^{b} \pi_{iv}^t(u'_{jv}), \tag{5.47}$$

$$\beta_{ij}^t := p(\omega_i|\mathbf{u}'_j, \theta_u^t, \alpha^t) = \frac{p_{ij}^t}{\sum_{i=1}^{n} p_{ij}^t}. \tag{5.48}$$

The next result follows from substitution of (5.47)-(5.48) into (5.4) and (5.46), as well as earlier proofs of Theorems 5 and 6.

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

Table 5.7: User features of database D_3.

Theorem 8. *Under user distortion, estimators* (5.4) *and* (5.46) *can be written as*

$$\alpha_i^{t+1} = \frac{1}{m} \sum_{j=1}^{m} \beta_{ij}^t, \tag{5.49}$$

$$\pi_{iv}^{t+1}(y) = \frac{\sum_{j=1}^{m} \beta_{ij}^t \mathbf{1}_{u'_{jv}=y}}{m\alpha_i^{t+1}}. \tag{5.50}$$

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

5.5.3 Discussion

To evaluate the result of Theorem 8, we construct a new database D_3, shown in Table 5.7, by switching from RTOs to user features (in the OPT string, M stands for MSS, N for NOP, S for SACK, T for timestamp, and W for window scale). Note that this Windows signature ties Novell in TTL, while Linux does the same in DF and MSS. 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 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 d_{i4}. For example, MST is feasible for Linux, but not the other two signatures in Table 5.7. Note that any single option (M, S, W) and the empty set are valid for all three OSes.

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 5.8: Patched user features.

We use two models for generating the alternatives for each field. 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 5695 candidates for Win, four for TTL, two for DF, 266 for OPT, and 1082 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 examples 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 individual u_{iv} to u''_{iv} with probability ϕ_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 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 maximum field value). The result is presented in Table 5.8.

Our next scenario S_3 is detailed in Table 5.9 and the corresponding outcome in Table 5.10. Due to the differences in treatment of non-default features, Hershel+ is slightly inferior to the first iteration of EM. 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

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)

Table 5.9: Parameters of scenario S_3.

Case	Hershel+	EM	
	ρ^1	ρ^1	ρ^∞
S_{31}	0.76	0.79	0.96
S_{32}	0.29	0.32	0.91
S_{33}	0.31	0.50	1

Table 5.10: Classification results in D_3.

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 5.8 and their probability of occurrence.

To estimate vector ϕ_v^t in the classifier, we use a weighted average of feature non-modification across all OSes, i.e., $\phi_v^t = \sum_{i=1}^n \alpha_i^t \phi_{iv}^t$. The result, together with the final estimate of α, is shown in Table 5.11. Both are an excellent match to the parameters of the simulation.

5.6 Complete System

5.6.1 Reset Packets

Because loss of RST packets causes the corresponding user features (i.e., ACK/RST flags, ACK sequence number, window size) to be wiped out, there is dependency between

Case	α^{∞}	ϕ_v^{∞}
S_{31}	(0.90, 0.05, 0.05)	(0.30, 0.20, 0.50, 0.40, 0.40)
S_{32}	(0.90, 0.05, 0.05)	(0.00, 0.00, 0.10, 0.20, 0.00)
S_{33}	(0.70, 0.20, 0.10)	(0.20, 0.20, 0.20, 0.20, 0.20)

Table 5.11: Recovery of α and ϕ_v in D_3.

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

Table 5.12: Handling of RST packets.

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 record RST header values into user features. Since RST fields are unmodifiable independently of each other , 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 5.12, 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 matches that of the original OS. If the sampled OS has a RST, but the signature does not, this indicates a possible injection 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 proper weights 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 earlier.

5.6.2 Final Model

We now combine the developed network, user, and RST models into a single framework. First, redefine (5.22) as

$$p_{ij\tau\gamma}^t = \alpha_i^t \zeta_{ij}^t \prod_{v=1}^{b} \pi_{iv}^t(u'_{jv}) f_T^t(\tau) p_i^t(\gamma) \prod_{r=1}^{|\mathbf{d}'_j|} f_\Delta^t(\delta_{ij\tau\gamma r}). \tag{5.51}$$

This allows us to compute $\beta_{ij\tau\gamma}^t$ still via (5.23), as well as reuse (5.24)-(5.27); however, (5.50) requires an update to

$$\pi_{iv}^{t+1}(y) = \frac{\sum_{j=1}^m \mathbf{1}_{u_{jv}=y} \sum_{\tau\gamma} \beta_{ij\tau\gamma}^t}{m\alpha_i^{t+1}}, \tag{5.52}$$

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

$$p(\omega_i | \mathbf{x}'_j, \theta^\infty, \alpha^\infty) = \sum_{\tau\gamma} \beta_{ij\tau\gamma}^\infty. \tag{5.53}$$

It is fairly straightforward to generalize our earlier results to cover the complete model. We thus present the next theorem without proof.

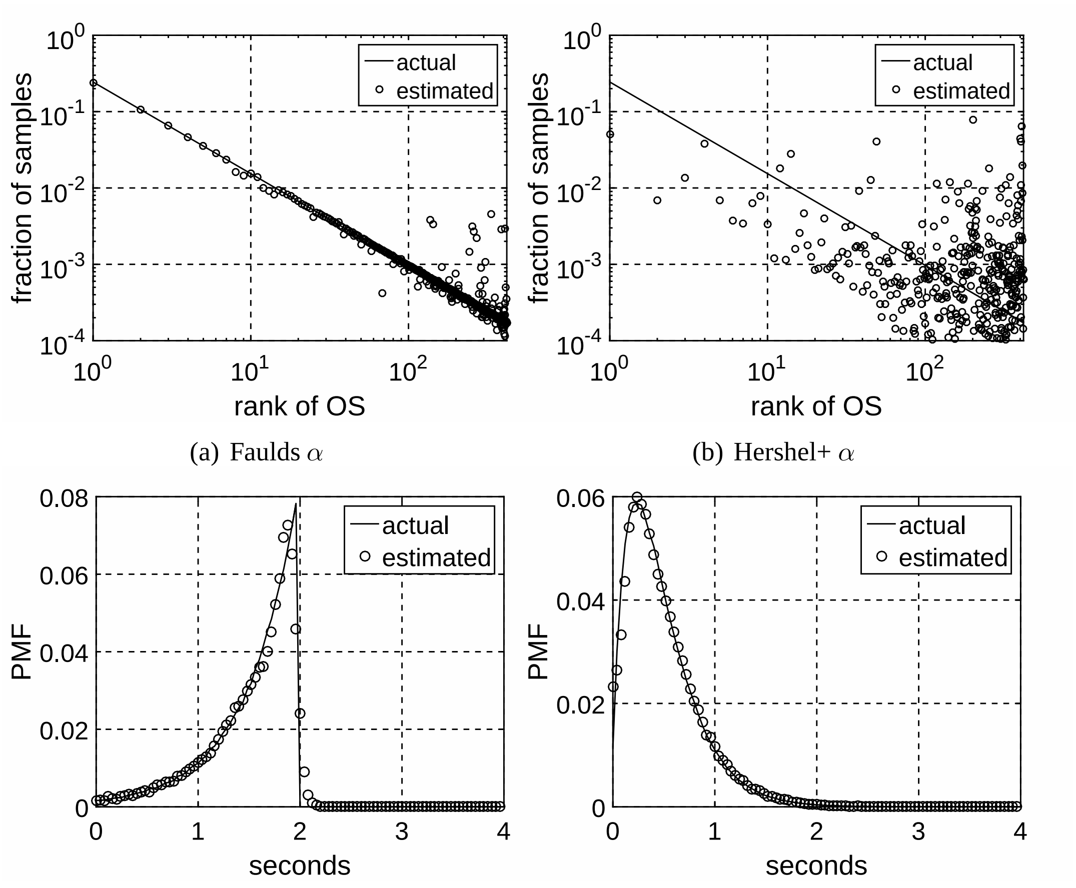
Theorem 9. *Under both network and user distortion, estimator* (5.23)-(5.27), (5.51)-(5.52) *is the EM algorithm for* (θ, α).

5.6.3 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 – our Plata database. To keep continuity in the notation, 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. We constructed this database 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 section. 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_{24} (i.e., reverse-binomial with probability 0.1). Finally, we use RAND with stay probability $\phi_v = 0.8$ for all v.

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 Figure 5.4(a). This 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). Next, 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 Figure 5.3, which indicates



(a) Faulds α (b) Hershel+ α

(c) Faulds f_T (d) Faulds f_Δ

Figure 5.4: Results in D_4.

that incorrect decisions overwhelmingly went to signatures with similar RTO vectors as the true OS.

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

$$L_k^t = \sum_{\ell=1}^{k-1} \ell q_k^t(\ell) \tag{5.54}$$

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

$$p^t_{loss} = \frac{\sum_{i=1}^n \alpha^t_i L^t_{|\mathbf{d}_i|}}{\sum_{i=1}^n \alpha^t_i |\mathbf{d}_i|}. \tag{5.55}$$

Recall that the simulation allowed packet loss to affect at most $k-1$ packets in an OS with $|\mathbf{d}_i| = k$. Therefore, its ground-truth packet loss should represent the same quantity as (5.55). 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.,

$$E[\phi^t_v] = \frac{1}{b+1}\sum_{v=1}^{b+1} \phi^t_v = \frac{1}{b+1}\sum_{v=1}^{b+1}\sum_{i=1}^{n} \alpha_i \phi^t_{iv}. \tag{5.56}$$

Results show that $E[\phi^\infty_v] = 0.802$, which is again 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.

5.6.4 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 the impact of these cases 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

New size	Injected	ρ_*^1	ρ_*^∞	p_{loss}^∞	$E[\phi_v^\infty]$
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

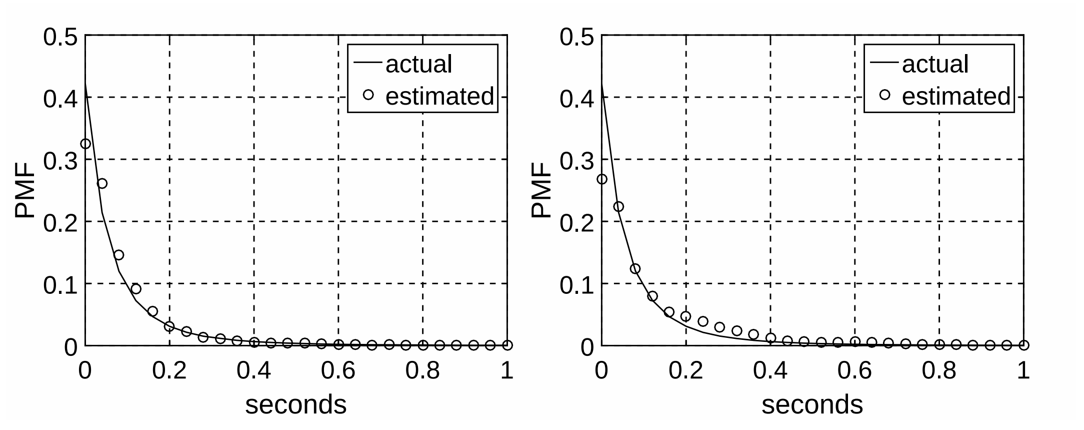
Table 5.13: Injection classification summary.

product in (5.51) 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 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 taint the PMFs π_{iv}^t of multiple OSes and 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 5.13, 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



(a) f_T (b) f_Δ

Figure 5.5: Recovered delay under 72% injection.

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. This loss of accuracy is explained by matches that now require more 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.

Figure 5.5 shows the two delay PMFs estimated by Faulds in the last row of Table 5.13. 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.

5.7 Internet Measurement

5.7.1 Overview

To experiment with Faulds, we conducted a port-80 SYN scan of all BGP-reachable IPv4 addresses on the Internet in December 2016. Of the 2.8B IPs contacted, we gathered responses from 67.6M hosts (compared to 66.4M in our previous scan, the Internet web-server population appears to have reached saturation). 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 vector α, then examine parameters of network distortion θ_d, and finish with those of user modification θ_u.

5.7.2 Classification Results

Define classification to be successful for sample j if the denominator of (5.1) is non-zero, i.e., $p(\mathbf{x}_j|\theta^t, \alpha^t) > 0$. Using the Hershel+ database, Faulds successfully classified 63.5M hosts (i.e., 94%). From a pure statistical point of view, the remaining 4.1M 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.

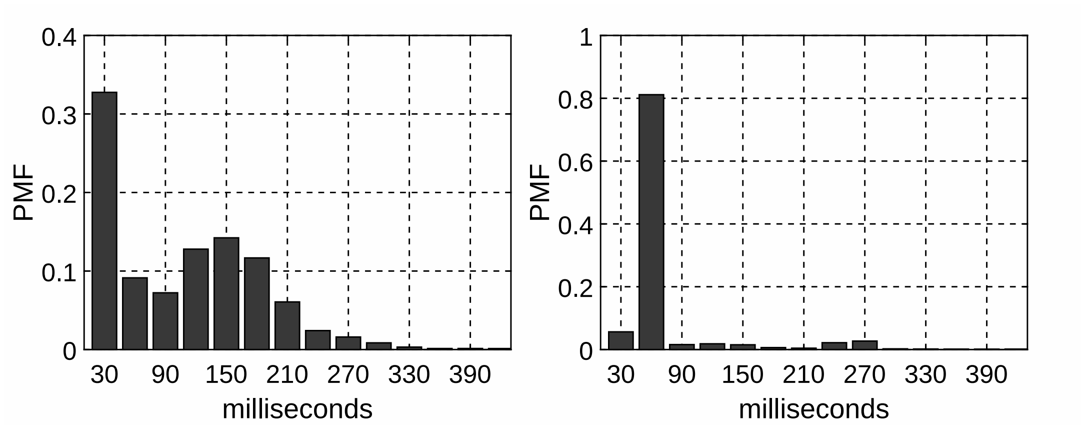
The left side of Table 5.14 shows the top ten OSes after one iteration of Faulds. As the database of Hershel+ is auto-generated and does not contain fine-granular details about individual OS versions, many signature names appear similar; however, these often correspond to different kernel versions and/or physical devices. The dominance of Linux and embedded devices in Table 5.14 matches the statistics we found in our previous studies in Section 3 and Section 4; however, a more interesting result is the amount of change occurring in the classification as Faulds goes through its iterations. The right side of Table 5.14

OS	α^1	Count
Ubuntu / Redhat / CentOS	0.227	14,662,315
Ubuntu / Redhat / SUSE	0.108	8,388,020
Windows 7 / 8 / 2008 / 2012	0.048	2,938,499
Embedded Linux	0.033	2,401,181
Ubuntu / Debian / Embedded	0.028	1,848,388
Embedded Linux	0.025	1,672,805
Ubuntu / Redhat / Sci. Linux	0.019	1,320,081
Windows XP / 2003	0.018	1,190,617
3Com Routers	0.015	1,013,943
Cisco Embedded	0.013	991,881

$\longrightarrow$

OS	α^{10}	Count	Change
Ubuntu / Redhat / CentOS	0.226	14,639,486	-0.002
Ubuntu / Redhat / SUSE	0.102	6,669,700	-0.20
Embedded Linux	0.067	4,384,225	0.82
Windows 7 / 8 / 2008 / 2012	0.045	2,948,567	0.003
Cisco Embedded	0.022	1,497,269	0.51
Ubuntu / Redhat / Sci. Linux	0.018	1,148,008	-0.13
3Com Routers	0.018	1,128,655	0.11
Embedded Linux	0.017	1,057,361	-0.37
Dell Laser / Xerox WorkCenters	0.015	982,973	0.15
Ubuntu / Debian / Embedded	0.013	844,958	-0.54

Table 5.14: Faulds classification at iteration 1 (left) and 10 (right).



(a) f_T (b) $f_\triangle$

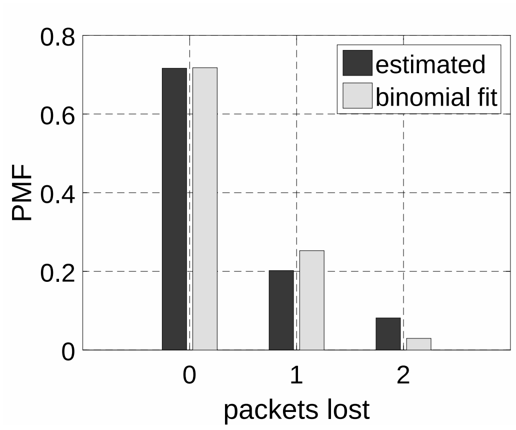
Figure 5.6: Internet delay distributions.

shows the α vector after 10 steps. With the exception of two signatures, there is significant movement in the list, including embedded Linux in third position increasing its membership by 82%, Cisco gaining 51%, 3Com 11% and Windows XP/2003 completely dropping off the top-ten. There is even more shuffle further down the list, which underscores the importance of using proper algorithms for estimating α.

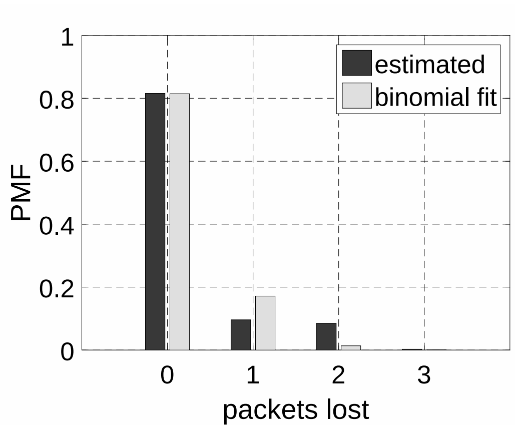
5.7.3 Network Distortion

Figure 5.6(a) shows the recovered distribution f_T using bin size 30 ms. Interestingly, 32% of delays are in the first bin, which likely represents idle servers that immediately send back the first SYN-ACK. A relatively large number (i.e., 38%) of cases belong to the 120-180 ms range, which may indicate OS scheduling delays, non-trivial CPU load on the server, longer forward paths, and involvement of various backend databases to set up the connection. Overall, we obtain $E[T] = 111$ ms, 41% of the samples below 60 ms, 90% below 180 ms, and 99.4% below 420 ms.

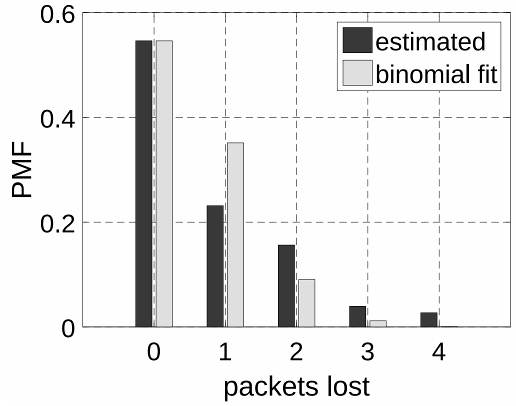
Figure 5.6(b) plots the distribution of one-way delay $f_\triangle$. The massive peak at 30-60 ms consolidates 81% of the observations and likely corresponds to fixed propagation delays



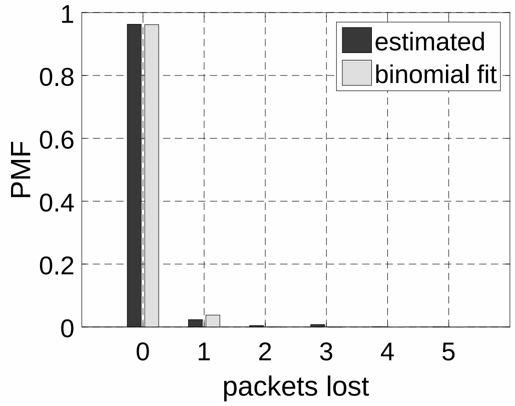
(a) q_3 (loss 12%)



(b) q_4 (loss 6.8%)



(c) q_5 (loss 15%)



(d) q_6 (loss 1.0%)

Figure 5.7: Internet packet-loss PMFs.

shared by many Internet paths leading back to our client. With $E[\Delta] = 83$ ms, 90% of the values below 120 ms, and 98.5% below 420 ms, the real Internet distortion θ_d does appear quite different from that assumed by Hershel+.

To examine packet loss, define $\eta_k^t = \sum_{i=1}^n \alpha_i^t \mathbf{1}_{|\mathbf{d}_i|=k}$ to be the estimated fraction of observations that use an OS with k packets. The top values of k are four ($\eta_4^\infty = 0.42$, 112 stacks in D_4), six ($\eta_6^\infty = 0.31$, 80 stacks), three ($\eta_3^\infty = 0.07$, 72 stacks), and five ($\eta_5^\infty = 0.04$, 54 stacks). Figure 5.7 examines 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

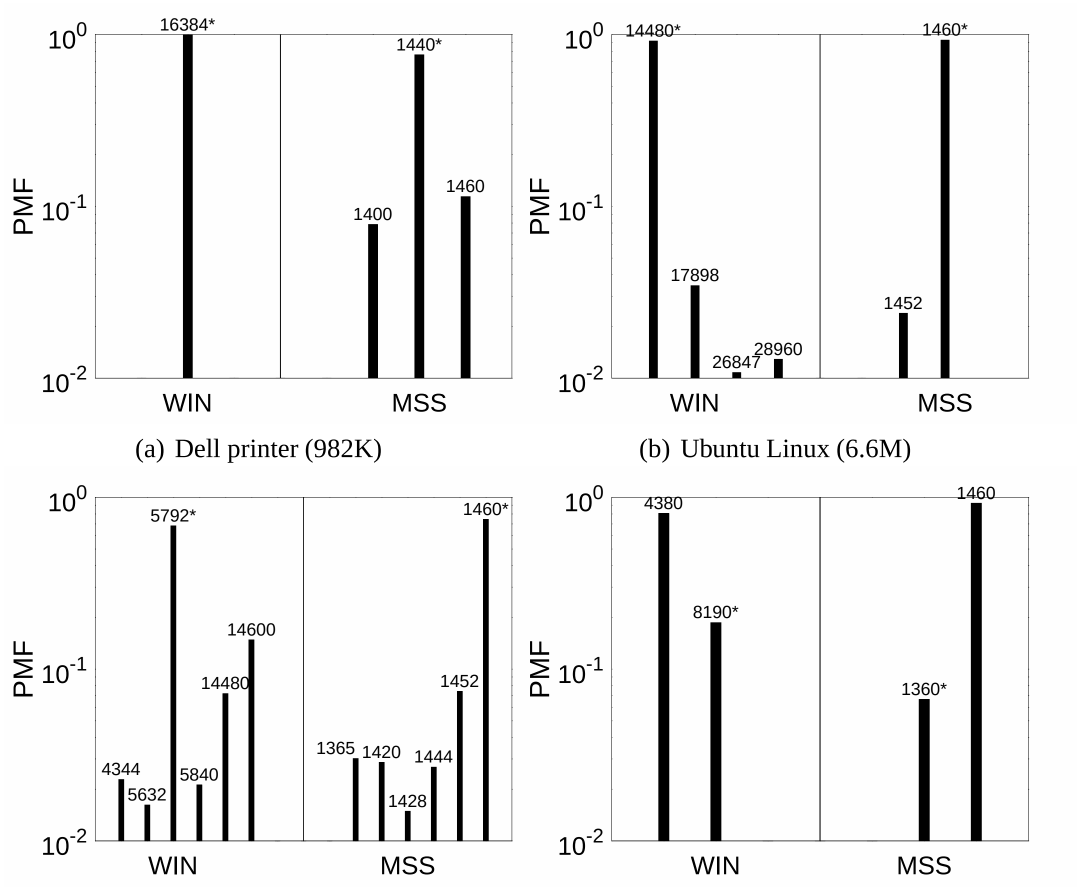
from (5.54). First, it is interesting that the loss rate is heterogeneous, ranging from 1% in q_6 to 15% in q_5. This phenomenon may be inherent to the signatures that map to each k (e.g., certain printers cut the SYN-ACK sequence when their tiny SYN backlog queue overflows) or their location on the Internet, which suggests there is an extra benefit to estimating q_k independently for different k. If injection of unknowns were responsible for the increased loss rate in q_3 and q_5, we would not expect to see a binomial-like distribution. Instead, an abnormally large spike at $\ell = 1$ or 2 would be more likely.

Second, the binomial fit in Figure 5.7 is not perfect, but it shows a similar decaying trend. Therefore, the iid loss assumption in Hershel+ may be reasonable, but with one correction that allows for heterogeneous values across different k. Third, computing (5.55) for the Internet scan yields an average loss rate of 3.7%. This is very close to the assumed model of Hershel+, whose $p_{loss} = 3.8\%$ comes from a 2009 Google study of SYN-ACK retransmission rates at their servers [18]. Apparently this magic number remains in effect for the Internet even today.

5.7.4 User Distortion

Faulds produced $420 \times 6 = 2520$ 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 Figure 5.8(a) for the Dell printer from Table 5.14. Among 982K occurrences on the Internet, this device keeps the default window with probability 1. Intuition also suggests that general-purposes



(a) Dell printer (982K) (b) Ubuntu Linux (6.6M)

(c) CentOS (762K) (d) Citrix Netscaler (28K)

Figure 5.8: Internet distributions π_{i1} and π_{i5} (default values are shown with an asterisk).

OSes are more susceptible to modification and/or existence of alternatively patched versions. An example of this is Ubuntu in Figure 5.8(b). While both features show variation, the default values dominate. A more diverse case is CentOS (enterprise Linux) in part (c), which has 29% of its samples with larger windows and 3% with smaller. In subfigure (d), Citrix Netscaler (data-center load-balancer) has its original combination (8190, 1360) overshadowed by (4380, 1460). We conjecture that our Plata database most likely captured a non-standard version of this stack. Since this is an inherent property of any database, it is important to allow great flexibility in the match process to accommodate such scenarios. Faulds does exactly that.

Computing (5.56), we obtain $E[\phi_v^\infty] = 0.89$, which confirms the accuracy of the assumed value in Hershel+ (i.e., 0.9); however, sampling the distributions in Figure 5.8 or using them in classification is only possible by introducing π_{iv}^t and iteratively refining (θ^t, α^t). As the first method to implement this functionality, Faulds paves way for scalable, low-overhead Internet characterization, robust device identification, and better modeling of distortion experienced by the numerous hardware artifacts found on the Internet.

5.8 Conclusion

In this section, 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 characterize stack popularity, network delays, packet loss, and header-tuning probabilities.

6. SUMMARY AND FUTURE DIRECTIONS

In this dissertation, we tackled the problem of large scale OS fingerprinting, a direction which is largely unexplored in the current literature. In order to build fast, low overhead algorithms that are required to measure a sizeable network, we focused on classification using a single TCP packet.

We first developed stochastic theory of single-packet OS fingerprinting, and created a classifier called *Hershel* based on our formulation. Using simulations, we showed that our algorithm is accurate even under extremely noisy conditions, and conducted a study where we successfully classified the OS of 37M Internet hosts.

Next, we turned our attention to building a scalable database of OSes to use with Hershel. We developed a framework called *Plata*, which is able to automatically create a database from a network scan using a novel unsupervised clustering algorithm. We used Plata on our university campus to discover signatures for 420 OSes, provided an improved version of our classifier (*Hershel+*) and showed its viability in an Internet measurement of 66M target hosts.

Finally, we took aim at one possible shortcoming of Hershel – the assumptions of volatility it makes for each noisy parameter. We derived a new algorithm using Expectation-Maximization called *Faulds* to recover the unknown distributions of network one-way delay, packet loss and user feature modification, using the classification process itself. After showing its reliability in simulations, we used Faulds to recover network-wide delay distributions, packet loss probabilities and likelihoods of stack tuning performed by administrators across the Internet.

6.1 Future Directions

Network stack fingerprinting has well-known pitfalls (e.g., scrubbers [20], [90], [84], [101], [115], traffic intercepts by middleboxes [43], load-balancers, RST injection by IDS), but nevertheless it is fascinating that a single SYN packet can elicit so much information about the target. With our algorithm for automated construction of databases and robust classification (i.e., Plata, Hershel+ and Faulds), our goal is to make single-packet tools a legitimate competitor for use over the public Internet. However, despite the recent developments in this field, there are still many open problems and avenues for improvement, which we discuss next.

From our classification efforts, we showed that Hershel+ and Faulds are tolerant of samples gathered from unknown devices on the Internet, either by discarding them if they are a complete non-match, or matching them to the closest possible signature. Future work can focus on more reliable detection of unknown stacks among the observations, and automatic generation of database signatures for them. This would require research to continue on discerning the separation of a "tweaked" sample versus an unknown one, and the impacts of such observations on the final accuracy.

Once this detection is possible, it leads to the next question which is whether the entire Internet dataset can be used to build a OS fingerprint database. We have considered this direction; however, Plata matrix construction has quadratic complexity and signature separation is even worse (i.e., n^3). The largest cluster in the Internet dataset formed after separation on the user features still contains over 50K RTO vectors, which will take Plata weeks to separate. Additionally, collection of loss-free samples from each IP not only requires pestering hosts with 3.3B additional packets, but also consumes a large amount of time that may result in host departure and incomplete measurement. Finally, presence of non-trivial delay T during database construction violates the current assumptions that the initial fingerprints are clean, with currently unknown consequences.

This train of thought also gives rise to another question, which is whether Plata's distortion model $\mathcal{X}$ can include the additional types of disturbance we observe in single-probe fingerprinting. Including packet loss seems like a viable direction as Plata can handle this transparently in the Monte-Carlo version; however, deriving a Hershel+ matrix in closed-form requires additional research. Including user modification would require incorporating some knowledge from the gathered labels (e.g., windows, linux, printer), as our results from Faulds show that these parameters are dependent on the class of the device. To this end, additional data mining from the Faulds classification would be required to build realistic user modification distributions.

Finally, now that we have accomplished multi-iterative classification on features obtained from a single-packet using Faulds, the next logical step to investigate is whether we can increase accuracy by abandoning the single-packet assumption and sending multiple packets to each target IP. Future work would need to assess the viability of this approach on Internet scale, and find the correct balance that would make the target host elicit enough responses without triggering IDS and harassing network administrators. Furthermore, a new database with a new distortion model would also be required. For example, $\mathcal{X}$ may be extended to include blocking of ICMP/UDP packets as done by a firewall, censorship of certain invalid flag combinations known to IDS, emulation of load-balancers, and fingerprint obfuscation by scrubbers found on the Internet.

REFERENCES

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

[2] Akamai. [Online]. Available: http://www.akamai.com/html/about/facts_figures.html.

[3] L. Alt, R. Beverly, and A. Dainotti, “Uncovering Network Tarpits with Degreaser,” in *Proc. ACM ACSAC*, Dec. 2014, pp. 156–165.

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

[5] ATInternet, “Operating Systems Barometer,” Aug. 2014. [Online]. Available: http://www.atinternet.com/uploads/Operating-Systems-August-2014.pdf.

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

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

[8] R. Beck, “Passive-Aggressive Resistance: OS Fingerprint Evasion,” *Linux Journal*, vol. 2001, no. 89, Aug. 2001.

[9] D. B. Berrueta, “A Practical Approach for Defeating Nmap OS-Fingerprinting,” 2003. [Online]. Available: http://nmap.org/misc/defeat-nmap-osdetect.html.

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

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

[12] J. Bolot, “End-to-End Packet Delay and Loss Behavior in the Internet,” in *Proc. ACM SIGCOMM*, Sep. 1993, pp. 289–298.

[13] R. Braden, “Requirements for Internet Hosts – Communication Layers,” *IETF RFC 1122*, Oct. 1989.

[14] T. Bu, N. Duffield, F. Presti, and D. Towsley, “Network Tomography on General Topologies,” in *Proc. ACM SIGMETRICS*, Jun. 2002, pp. 21–30.

[15] 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.

[16] Y. Chen, K. H. Lim, R. H. Katz, and C. Overton, “On the Stability of Network Distance Estimation,” *SIGMETRICS Performance Evaluation Review*, vol. 30, no. 2, pp. 21–30, Sep. 2002.

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

[18] 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.

[19] M. Coates and R. Nowak, “Network Loss Inference Using Unicast End-to-End Measurement,” in *Proc. ITC Conference on IP Traffic, Modeling and Management*, Sep. 2000, pp. 1–9.

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

[21] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, “Vivaldi: A Decentralized Network Coordinate System,” in *Proc. ACM SIGCOMM*, Aug. 2004, pp. 15–26.

[22] D. Dagon, N. Provos, C. P. Lee, and W. Lee, “Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority,” in *Proc. NDSS*, Feb. 2008.

[23] Maxmind GeoIP Databases. [Online]. Available: http://dev.maxmind.com/geoip/.

[24] 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.

[25] A. B. Downey, “Using PATHCHAR to Estimate Internet Link Characteristics,” in *Proc. ACM SIGCOMM*, Aug. 1999, pp. 241–250.

[26] N. Duffield, C. Lund, and M. Thorup, “Estimating Flow Distributions from Sampled Flow Statistics,” in *Proc. ACM SIGCOMM*, Aug. 2003, pp. 325–336.

[27] T. Dunigan, M. Mathis, and B. Tierney, “A TCP Tuning Daemon,” in *Proc. ACM/IEEE Supercomputing*, Nov. 2002, pp. 1–16.

[28] 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.

[29] 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.

[30] 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.

[31] Ericsson, “Ericsson Mobility Report,” Nov. 2016. [Online]. Available: https://www.ericsson.com/mobility-report.

[32] X. Feng, Q. Li, Q. Han, H. Zhu, Y. Liu, J. Cui, and L. Sun, “Active Profiling of Physical Devices at Internet Scale,” in *Proc. IEEE ICCCN*, Aug. 2016, pp. 1–9.

[33] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, “IDMAPS: A Global Internet Host Distance Estimation Service,” *IEEE/ACM Transactions on Networking*, vol. 9, no. 5, pp. 525–540, Oct. 2001.

[34] P. Garcia-Laencina, J.-L. Sancho-Gomez, and A. Figueiras-Vidal, “Pattern Classification with Missing Data: A Review,” *Neural Computing and Applications*, vol. 19, no. 2, pp. 263–282, Mar. 2010.

[35] L. G. Greenwald and T. J. Thomas, “Toward undetected operating system fingerprinting,” in *Proc. USENIX WOOT*, Aug. 2007, pp. 1–10.

[36] Y. Gu, L. Breslau, N. Duffield, and S. Sen, “On Passive One-Way Loss Measurements Using Sampled Flow Statistics,” in *Proc. IEEE INFOCOM*, Apr. 2009.

[37] K. P. Gummadi, S. Saroiu, and S. D. Gribble, “King: Estimating Latency between Arbitrary Internet End Hosts,” in *Proc. ACM IMW*, Nov. 2002, pp. 5–18.

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

[39] O. Gurewitz, I. Cidon, and M. Sidi, “One-way Delay Estimation Using Network-Wide Measurements,” *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2710–2724, Jun. 2006.

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

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

[42] 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.

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

[44] C. Huang, A. Wang, J. Li, and K. Ross, "Measuring and Evaluating Large-Scale CDNs," in *Proc. ACM IMC*, Oct. 2008, pp. 15–29.

[45] IRL Fingerprinting Dataset. [Online]. Available: http://irl.cs.tamu.edu/projects/sampling/.

[46] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," *IETF RFC 1323*, May 1992.

[47] M. Kalman and B. Girod, "Modeling the Delays of Successively-Transmitted Internet Packets," in *Proc. IEEE ICME*, Jun. 2004, pp. 2015–2018.

[48] E. Katz-Bassett, H. Madhyastha, V. Adhikari, C. Scott, J. Sherry, P. V. Wesep, T. Anderson, and A. Krishnamurthy, "Reverse Traceroute," in *Proc. USENIX NSDI*, Apr. 2010, pp. 219–234.

[49] 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*, 1997, pp. 282–293.

[50] 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.

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

[52] M. Kührer, T. Hupperich, J. Bushart, C. Rossow, and T. Holz, “Going Wild: Large-Scale Classification of Open DNS Resolvers,” in *Proc. ACM IMC*, Oct. 2015, pp. 355–368.

[53] A. Kumar, M. Sung, J. Xu, and J. Wang, “Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution,” in *Proc. ACM SIGMETRICS*, Jun. 2004, pp. 177–188.

[54] D. Leonard and D. Loguinov, “Turbo King: Framework for Large-Scale Internet Delay Measurements,” in *Proc. IEEE INFOCOM*, Apr. 2008, pp. 430–438.

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

[56] 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.

[57] H. Lim, J. C. Hou, and C.-H. Choi, “Constructing Internet Coordinate System Based on Delay Measurement,” in *Proc. ACM IMC*, Oct. 2003, pp. 129–142.

[58] J. Lippard, “Craigslist no longer uses TCP window size of 0.” [Online]. Available: http://lippard.blogspot.com/2006/07/craigslist-no-longer-uses-tcp-window.html/.

[59] 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.

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

[61] J. Matherly, “Shodan Search Engine.” [Online]. Available: https://www.shodan.io.

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

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

[64] J. Medeiros, A. Brito, and P. Pires, "A Data Mining Based Analysis of Nmap Operating System Fingerprint Database," in *Proc. IEEE CISIS*, Sep. 2009, pp. 1–8.

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

[66] Microsoft Support. [Online]. Available: http://support.microsoft.com/kb/2525390.

[67] A. Mirian, Z. Ma, D. Adrian, M. Tischer, T. Chuenchujitasphon, T. Yardley, R. Berthier, J. Mason, Z. Zakir Durumeric, and J. A. Halderman, "An Internet-Wide View of ICS Devices," in *Proc. IEEE Privacy, Security, and Trust Conference*, Dec. 2016.

[68] D. Napier, "IPTables/NetFilter – Linux's Next Generation Stateful Packet Filter," *SysAdmin Magazine*, vol. 10, pp. 8–16, Nov. 2001.

[69] A. Nappa, Z. Xu, M. Z. Rafique, J. Caballero, and G. Gu, "Cyberprobe: Towards Internet-Scale Active Detection of Malicious Servers," in *Proc. NDSS*, Feb. 2014, pp. 1–15.

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

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

[72] T. S. E. Ng and H. Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches," in *Proc. IEEE INFOCOM*, Jun. 2002, pp. 170–179.

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

[74] J. Novak and S. Sturges, “Target-Based TCP Stream Reassembly,” Sourcefire Inc., Tech. Rep., Aug. 2007.

[75] SpeedGuide TCP Optimizer. [Online]. Available: http://www.speedguide.net/downloads.php.

[76] Oracle, “Operating System Tuning.” [Online]. Available: http://docs.oracle.com/cd/E12839_01/web.1111/e13814/os_tuning.htm.

[77] J. Padhye and S. Floyd, “On Inferring TCP Behavior,” in *Proc. ACM SIGCOMM*, Aug. 2001, pp. 287–298.

[78] I. Papapanagiotou, E. Nahum, and V. Pappas, “Smartphones vs. Laptops: Comparing Web Browsing Behavior and the Implications for Caching,” in *Proc. ACM SIGMETRICS*, Jun. 2012, pp. 423–424.

[79] A. Pathak, H. Pucha, Y. Zhang, C. Hu, and Z. M. Mao, “A Measurement Study of Internet Delay Asymmetry,” in *Proc. PAM*, Apr. 2008, pp. 182–191.

[80] V. Paxson, “End-to-end Internet Packet Dynamics,” in *Proc. ACM SIGCOMM*, 1997, pp. 139–152.

[81] V. Paxson, M. Allman, J. Chu, and M. Sargent, “Computing TCP’s Retransmission Timer,” *IETF RFC 6298*, Jun. 2011.

[82] PlanetLab. [Online]. Available: http://www.planet-lab.org/.

[83] J. Postel, “Transmission Control Protocol,” *IETF RFC 793*, Sep. 1981.

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

[85] N. Provos, “A Virtual Honeypot Framework,” in *Proc. USENIX Security*, Aug. 2004, pp. 1–14.

[86] N. Provos and P. Honeyman, “ScanSSH - Scanning the Internet for SSH Servers,” in *Proc. USENIX LISA*, Dec. 2001, pp. 25–30.

[87] Y. Pryadkin, R. Lindell, J. Bannister, and R. Govindan, “An Empirical Evaluation of IP Address Space Occupancy,” USC/ISI, Tech. Rep. ISI-TR-2004-598, Nov. 2004.

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

[89] M. Roesch, “Snort – Lightweight Intrusion Detection for Networks,” in *Proc. USENIX LISA*, Nov. 1999, pp. 229–238.

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

[91] C. Sarraute and J. Burroni, “Using Neural Networks to Improve Classical Operating System Fingerprinting Techniques,” *Electronic Journal of SADIO*, vol. 8, no. 1, Mar. 2008.

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

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

[94] Z. Shamsi and D. Loguinov, “Unsupervised Clustering Under Temporal Feature Volatility in Network Stack Fingerprinting,” *IEEE/ACM Transactions on Networking*, 2017 (to be published).

[95] Z. Shamsi, A. Nandwani, D. Leonard, and D. Loguinov, “Hershel: Single-Packet OS Fingerprinting,” *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 2196–2209, Aug. 2016.

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

[97] U. Shankar and V. Paxson, “Active Mapping: Resisting NIDS Evasion Without Altering Traffic,” in *Proc. IEEE SP*, May 2003, pp. 44–61.

[98] P. Sharma, Z. Xu, S. Banerjee, and S.-J. Lee, “Estimating Network Proximity and Latency,” *ACM SIGCOMM*, pp. 39–50, Sep. 2006.

[99] M.-F. Shih and A. O. Hero, “Unicast-Based Inference of Network Link Delay Distributions with Finite Mixture Models,” *IEEE Transactions on Signal Processing*, vol. 51, no. 8, pp. 2219–2228, Aug. 2003.

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

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

[102] Snort IDS. [Online]. Available: http://www.snort.org.

[103] RedHat Customer Case Study. [Online]. Available: https://www.redhat.com/cms/managed-files/Telefonica_EN-compressed_0.pdf.

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

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

[106] L. Tang and M. Crovella, “Virtual Landmarks for the Internet,” in *Proc. ACM IMC*, Oct. 2003, pp. 143–152.

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

[108] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed. Academic Press, 2009.

[109] B. Tierney, “TCP Tuning Guide for Distributed Applications on Wide Area Networks,” *USENIX & SAGE Login*, vol. 26, no. 1, pp. 33–39, Feb. 2001.

[110] C. Valli, “Honeyd – A OS Fingerprinting Artifice,” in *Proc. Australian Computer, Network and Information Forensics Conference*, Nov. 2003.

[111] Y. Vardi, “Network Tomography: Estimating Source-Destination Traffic Intensities from Link Data,” *Journal of the American Statistical Association*, vol. 91, no. 433, pp. 365–377, Mar. 1996.

[112] 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.

[113] A. Y. Wang, C. Huang, J. Li, and K. Ross, “Queen: Estimating Packet Loss Rate Between Arbitrary Internet Hosts,” in *Proc. PAM*, Apr. 2009, pp. 57–66.

[114] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush, “A Measurement Study on the Impact of Routing Events on End-to-End Internet Path Performance,” in *Proc. ACM SIGCOMM*, Aug. 2006, pp. 375–386.

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

[116] Z. Wen, S. Triukose, and M. Rabinovich, “Facilitating Focused Internet Measurements,” in *Proc. ACM SIGMETRICS*, Jun. 2007, pp. 49–60.

[117] Wolfram Alpha, “Computational Knowledge Engine.” [Online]. Available: http://www.wolframalpha.com.

[118] C. F. J. Wu, “On the Convergence Properties of the EM Algorithm,” *The Annals of Statistics*, vol. 11, no. 1, pp. 95–103, Mar. 1983.

[119] M. Yajnik, S. Moon, J. Kurose, and D. Towsley, “Measurement and Modelling of the Temporal Dependence in Packet Loss,” in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 345–352.

[120] 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.

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

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

[123] A. Zeitoun, C.-N. Chuah, S. Bhattacharyya, and C. Diot, “An AS-level Study of Internet Path Delay Characteristics,” in *Proc. IEEE GLOBECOM*, Nov. 2004, pp. 1480–1484.

[124] Y. G. Zeng, D. Coffey, and J. Viega, “How Vulnerable are Unprotected Machines on the Internet?” in *Proc. PAM*, Mar. 2014, pp. 224–234.

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