ALGORITHMS FOR LARGE-SCALE INTERNET MEASUREMENTS

A Dissertation

by

DEREK ANTHONY LEONARD

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

DOCTOR OF PHILOSOPHY

December 2010

Major Subject: Computer Science

ALGORITHMS FOR LARGE-SCALE INTERNET MEASUREMENTS

A Dissertation

by

DEREK ANTHONY LEONARD

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

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Dmitri Loguinov
Committee Members,	Riccardo Bettati
	Jennifer L. Welch
	A. L. Narasimha Reddy
Head of Department,	Valerie E. Taylor

December 2010

Major Subject: Computer Science

ABSTRACT

Algorithms for Large-Scale Internet Measurements. (December 2010)

Derek Anthony Leonard, B.A., Hendrix College

Chair of Advisory Committee: Dr. Dmitri Loguinov

As the Internet has grown in size and importance to society, it has become increasingly difficult to generate global metrics of interest that can be used to verify proposed algorithms or monitor performance. This dissertation tackles the problem by proposing several novel algorithms designed to perform Internet-wide measurements using existing or inexpensive resources.

We initially address distance estimation in the Internet, which is used by many distributed applications. We propose a new end-to-end measurement framework called Turbo King (T-King) that uses the existing DNS infrastructure and, when compared to its predecessor King, obtains delay samples without bias in the presence of distant authoritative servers and forwarders, consumes half the bandwidth, and reduces the impact on caches at remote servers by several orders of magnitude.

Motivated by recent interest in the literature and our need to find remote DNS nameservers, we next address Internet-wide service discovery by developing IRLscanner, whose main design objectives have been to maximize politeness at remote networks, allow scanning rates that achieve coverage of the Internet in minutes/hours (rather than weeks/months), and significantly reduce administrator complaints. Using IRLscanner and 24-hour scan durations, we perform 20 Internet-wide experiments using 6 different protocols (i.e., DNS, HTTP, SMTP, EPMAP, ICMP and UDP ECHO). We analyze the feedback generated and suggest novel approaches for reducing the amount of blowback during similar studies, which should enable researchers to collect valuable experimental data in the future with significantly fewer hurdles.

We finally turn our attention to Intrusion Detection Systems (IDS), which are often tasked with detecting scans and preventing them; however, it is currently unknown how likely an IDS is to detect a given Internet-wide scan pattern and whether there exist sufficiently fast stealth techniques that can remain virtually undetectable at large-scale. To address these questions, we propose a novel model for the window-expiration rules of popular IDS tools (i.e., Snort and Bro), derive the probability that existing scan patterns (i.e., uniform and sequential) are detected by each of these tools, and prove the existence of stealth-optimal patterns.

To my family

ACKNOWLEDGMENTS

I would like to first acknowledge my advisor, Dr. Dmitri Loguinov, without whom this dissertation would not have been possible. He has taught me not only about the technical aspects of the field, but through his constant guidance has molded my ability to produce solid results, write technical papers, and give interesting presentations. His superior intellect and ability to quickly determine the essence of even the most difficult problem has been of great importance to me. What I have learned from him during my time at Texas A&M is immeasurable, but his example of determination and striving for better and more meaningful results will be a positive influence on me for the rest of my life.

I would like to thank Drs. Riccardo Bettati, Jennifer Welch, and Narasimha Reddy for their time serving on my committee, support throughout this process, and insightful evaluation of my research. I also acknowledge the many anonymous reviewers from IEEE INFOCOM, IEEE ICNP, ACM SIGMETRICS, ACM SIGCOMM, ACM/USENIX IMC, IEEE/ACM Transactions on Networking, etc., who significantly improved the quality of my work by taking the time to carefully read my submissions and provide useful comments.

I extend a special thanks to Willis Marti and his Networking & Information Security group at Texas A&M for their patience, helpful discussions, and understanding during the numerous network tests and measurements performed while producing this dissertation. I also thank Aaron Palermo, Blake Dworaczyk, and Nolan Flowers of the Computer Services Group in the Computer Science & Engineering Department for providing technical support for my network measurements.

One of the greatest pleasures of my time in graduate school was my interaction with fellow students in the Computer Science & Engineering Department and

particularly the Internet Research Lab. I will be forever grateful for the friendship, advice, and knowledge provided me by Min, Yueping, Xiaoming, Zhongmei, Seong, and Hsin-Tsang. I thank them for teaching me about their cultures, providing a good example of hard work, and always being willing to discuss difficult problems. I also greatly enjoyed being able to work with Clint, Matt, Ankur, Chandan, and others during my time in the lab.

On a personal note, I thank my parents, sisters, grandmothers, and parents-in-law for their unwavering support and confidence in me throughout my life. I have been extremely fortunate to have been surrounded by such people and would have likely faltered without them. I also thank Dean and Reed, whose smiles and hugs always cheer me. Finally, my greatest gratitude goes to my wife Megan, who has been my best supporter, sounding board, and helper. Even in the darkest times she always believed that I would complete this work, and she sacrificed greatly during the many days and nights I spent away. Without her love and support I would not have finished this dissertation.

TABLE OF CONTENTS

CHAPTER Page

I INTRODUCTION 1

1.1. Overview 1
1.2. Turbo King 2
1.3. IRLscanner 4
1.4. Modeling Window-based IDS and Stealth Scanning 6

II TURBO KING 11

2.1. Introduction 11
2.2. Background 14
2.3. Understanding Original King 15
2.3.1 Measurement Algorithm 15
2.3.2 Zones with Multiple Authoritative Nameservers 17
2.3.3 DNS Forwarders 17
2.3.4 Cache Pollution 18
2.4. Understanding Direct King 19
2.4.1 Measurement Algorithm 19
2.4.2 Additional Complexity 20
2.4.3 DNS Forwarders 21
2.4.4 Cache Pollution 21
2.5. Turbo King 22
2.5.1 Design 22
2.5.2 Discovering Nameservers 23
2.5.3 Measurement Algorithm 24
2.5.4 Detection and Avoidance of Forwarders 25
2.6. Evaluation 26
2.6.1 Results from Reverse DNS Crawl 26
2.6.1.1 Analyzing Zone Authority Data 28
2.6.2 Causes of Inaccuracy 29
2.6.2.1 Zones with Multiple Authoritative Nameservers . . 29
2.6.2.2 DNS Forwarders 31
2.6.3 Measurement-based Comparison 32
2.6.3.1 Turbo King versus O-King Estimates 33
2.6.3.2 Convergence of Estimates 34

CHAPTER Page

2.6.4 Overhead Analysis 34
2.6.4.1 Network Overhead 35
2.6.4.2 Cache Pollution 36
2.7. Summary 36

III IRLSCANNER 38

3.1. Introduction 38
3.1.1 Our Contributions 39
3.1.2 Ethical Implications 41
3.2. Scanner Design 42
3.2.1 Scan Scope 42
3.2.2 Scan Order 43
3.2.3 Scan Origin 45
3.2.4 Extrapolation 45
3.2.5 Implementation 46
3.2.6 Timeouts and Duration 46
3.2.7 Negative Feedback 47
3.3. IRLscanner 48
3.3.1 Scan Scope 48
3.3.2 Scan Order 50
3.3.3 Scan Origin 54
3.3.4 Extrapolation 56
3.3.5 Implementation 57
3.3.6 Timeouts and Duration 58
3.3.7 Negative Feedback 60
3.4. Experiments 61
3.4.1 Overview 61
3.4.2 UDP/ICMP Scans 63
3.4.3 TCP Scans 64
3.4.4 Remote OS Fingerprinting 65
3.4.5 Service Lifetime 68
3.5. Analysis 69
3.5.1 Email Complaints 70
3.5.2 Firewall Log Correlation 73
3.5.3 Enumerating Contributors 76
3.5.4 ACK Scans 76
3.5.5 DNS Lookups 77
3.6. Summary 79

CHAPTER Page

IV MODELING WINDOW-BASED IDS AND STEALTH SCANNING 80

4.1. Introduction 80
4.1.1 Formalization 81
4.1.2 Analysis 83
4.1.3 Stealth Scanner 84
4.2. Related Work 85
4.3. Formalizing Scanning 86
4.3.1 Scan Objectives 86
4.3.2 Scan Patterns 88
4.3.3 Window-based IDS 89
4.3.4 Stealth 90
4.3.5 Existence 93
4.3.6 Improvements 94
4.4. Analysis of Existing Methods 95
4.4.1 IP-sequential 96
4.4.2 Uniform Pattern 97
4.4.3 Uniform Detection Probability 98
4.4.4 Uniform Cover Time 104
4.4.5 Discussion 107
4.5. Stealth Scanning 109
4.5.1 Permutation 109
4.5.2 Split 112
4.5.3 Schedule 114
4.5.4 Correlation 116
4.5.5 Vertical Scans 118
4.6. Experiments 119
4.6.1 Methodology 119
4.6.2 Results 120
4.7. Defense 122
4.8. Implications 123
4.9. Summary 124

V SUMMARY AND FUTURE WORK 125

5.1. Summary 125
5.1.1 Turbo King 125
5.1.2 IRLscanner 126
5.1.3 Modeling Window-based IDS and Stealth Scanning . . . 126
5.2. Future Work 127

CHAPTER Page

5.2.1 Turbo King 127
5.2.2 IRLscanner 128
5.2.3 Modeling Window-based IDS and Stealth Scanning . . . 128

REFERENCES 130

VITA 143

LIST OF TABLES

TABLE		Page
I	Results of reverse DNS crawl (3.8 GHz Pentium 4)	26
II	Coverage of the Internet with discovered servers	27
III	Large-scale service discovery in the literature (dashes represent unreported values)	42
IV	Scan set size, Ethernet bandwidth, and packets/second in a 24 hour TCP SYN scan	49
V	Benchmark of GIW address generation	54
VI	Summary of scans performed	62
VII	Top 5 devices	66
VIII	Summary of fingerprinted devices	67
IX	General purpose (GP) devices	68
X	Emails and IPs excluded by service	70
XI	Email notices by complainant type	71
XII	DNS lookups on scanner source IPs	78
XIII	Parameters of common IDS	92
XIV	ISC reports	121
XV	Comparison of RAM Usage	122

LIST OF FIGURES

FIGURE		Page
1	Organization of this dissertation.	9
2	King estimates the latency from host A to B.	14
3	O-King query sequence.	16
4	O-King query sequence for two different server configurations.	18
5	D-King query sequence.	20
6	Turbo King query sequence.	24
7	Comparison of O-King to D-King for zone with two nameservers.	29
8	Convergence of O-King estimate for zone with two nameservers.	30
9	Forwarder affect on O-King and D-King (single nameserver zone).	32
10	T-King vs. O-King to validate implementation and show differences.	33
11	Convergence of measured latencies for O-King and T-King.	35
12	Illustration of AGT.	51
13	GIW split and extrapolation delay.	57
14	IRLscanner implementation.	58
15	Progression of blacklisted IPs.	73
16	ISC reports with our scans marked.	75
17	Illustration of permutation/split ($m = 3$).	87
18	Process $C_i^s(t)$ of IDS-A.	90
19	Process $C_i^s(t)$ of IDS-B.	90

FIGURE Page

20 Stealthy β-aware probing seen by s. 94

21 Uniform model ($m = 4, |s| = 4$). 98

22 Comparison of post-permutation IDS-A model (13) to simulations (default parameters $|s| = 2^8, \Delta_s = 60$ sec, $a_s = 4$, and $m = 1$). 100

23 Comparison of post-permutation IDS-B model (16) to simulations (default parameters $|s| = 2^8, \Delta_s = 60$ sec, $a_s = 4$, and $m = 1$). 103

24 Relative error between the binary-search SCT and its closed-form approximations ($|s| = 2^{16}, \Delta_s = 60$ sec, $m = 1$). 106

25 Ratio $\pi(\epsilon)$ for $\epsilon = 10^{-3}$. 109

26 Illustration of AGT and $SSO(2, 2)$. 112

27 Correlation in $\mathcal{B}(8)$. 117

28 Process $C_i^s(t)$ of IDS-C. 122

CHAPTER I

INTRODUCTION

1.1. Overview

The Internet and the services it supports have grown in importance both economically and socially in the last decades. However, with this growth the Internet has become more difficult to manage and improve due to the sheer number of end-hosts [35],[37],[42] and networks [92], the presence of malicious entities constantly seeking to exploit users [68], [70], [97], and a general inability to deploy global changes caused by its decentralized administration [72], [81], [102]. Given these issues, it has become increasingly difficult to perform Internet-wide experiments that are critical to verifying newly proposed algorithms (e.g., [20], [24]), tracking growth and changes in the Internet population (e.g., [10], [34]), and monitoring global metrics of interest (e.g., [35], [81]). In this dissertation we tackle the problem by proposing several novel algorithms for performing Internet-wide measurements of various types and studying their impact on remote networks. We next highlight some of the goals we developed to inform the design of each of the proposed techniques.

As the purpose of Internet measurements is to produce useful data, our first overarching goal is to design algorithms that allow for experiments to be performed in a timely fashion (i.e., days or weeks instead of months) using fairly inexpensive local hardware, which ensures that progress can be accomplished quickly and makes our methods accessible to a larger number of researchers. To ensure this is the case, we

---

The journal model is *IEEE/ACM Transactions on Networking.*

do not consider any solution that requires modifying existing protocols or deploying servers at remote locations, which can be prohibitively expensive and extremely difficult logistically. Our second goal is to design algorithms that minimize the impact on remote users and networks while still acquiring the data of interest in a timely fashion. This generally takes the form of reducing bandwidth consumption at remote networks and minimizing disk space and CPU cycles for their servers and end-hosts. Further care must be used to avoid sending large bursts of traffic to individual networks, which can congest routers and effectively cause a denial-of-service for remote users. In short, we are interested in developing algorithms that allow for efficient collection of data without requiring extensive local resources or overburdening remote networks and their users.

With these goals in mind, the next three chapters in this dissertation present algorithms with the ability to perform Internet-wide measurements and analyze their effect on remote networks. The rest of this chapter briefly introduces each of the topics in turn and describes how they relate to form a unified dissertation.

1.2. Turbo King

We start with distance estimation in the Internet, which has recently become a large field of study [7], [16], [20], [24], [28], [32], [33], [49], [54], [60], [75], [76], [80], [87], [98], [99], [109], [111], [118], [126]. Estimates or measurements of latency between end-hosts can be used to improve the efficiency of networks (e.g., content distribution networks) and service to end-users (e.g., reduce user-perceived latency or improve responsiveness in online games). Existing techniques largely consist of virtual coordinate approaches that estimate delay [20], [24], [33], [36], [49], [60], [76], [98], [99], [111], which are limited by a lack of real-world distance data for verification, and

methods that employ tracers to measure delay [1], [28], [72], [81], [102], which are inherently limited by the difficulty of deploying measurement servers throughout the Internet. Our aim is to bridge this gap by proposing a framework that allows for accurate latency measurements on an Internet-wide scale without modifying existing protocols or requiring large-scale deployment of remote sensors.

A natural choice for this endeavor is King [32], which approximates the distance between end-hosts using the delay between Domain Name System (DNS) servers responsible for mapping human-readable names to IP addresses of the hosts in question. However, through an evaluation of King we show that it suffers from non-negligible estimation bias, inundates remote servers with unnecessary requests (i.e., pollutes message caches), and incurs significant network overhead that makes it unsuitable for Internet-wide measurements. We attempt to mitigate these drawbacks by developing a new system called Turbo King (T-King) that improves on the accuracy of King, effectively eliminates cache pollution at remote servers, and significantly decreases network traffic by minimizing the number of queries needed to make distance measurements using DNS. Turbo King starts with a large set of DNS servers collected from throughout the Internet, from which is taken the closest nameserver to each end-host for use in the measurement. Once the closest nameservers are selected, T-King then uses a new measurement algorithm which not only reduces the number of queries and bandwidth overhead of King by more than 50%, but also achieves higher accuracy and a factor of N reduction in the number of polluted cache entries at each remote server for an $N \times N$ latency measurement.

We finish the chapter by evaluating both the prevalence and the effect of bias incurred by King, then describing a method for building the database of DNS servers required by Turbo King. To quantify the effect of bias in King, we use a small 50×50 delay matrix and compare the estimates produced by King to those of Turbo

King. Our results show that 15% of the measurements are different by more than 10% and 8% by more than 20%, which indicates that the severity of bias in King is non-negligible though generally mild. We then build the database of DNS servers for T-King by crawling the reverse DNS tree and discover a set of 216,843 nameservers, out of which we find 117,817 to be recursive and receptive to queries originating from our subnet. These servers reside in 174 countries, cover over 31,000 BGP prefixes, and are responsible for approximately 50% of IP addresses (i.e., 828 million) advertised in BGP [92]. However, the technique of reverse-crawling the DNS tree discovers only authoritative nameservers, which omits nearly all of the local DNS servers that are not part of the DNS tree but are often responsible for performing recursive queries. Our desire to locate these servers motivated us to search for a new discovery method, which led to the work described next.

1.3. IRLscanner

A more comprehensive technique for discovering remote hosts is *horizontal scanning* [107], which is a method for enumerating (in some set $\mathcal{S}$) all remote hosts that support a given protocol/service p. This is accomplished by sending packets to destinations in $\mathcal{S}$ and counting positive responses within some time interval. Besides discovering DNS servers for Turbo King, such a technique has wide applicability and can also be used to study publicly available services in the Internet (e.g., end-hosts [35], [37],[42], web sites [10], [34], [52]), understand how botnets are created by Internet worms [17], [45], [62], [108], and evaluate the prevalence of known security flaws (e.g., DNS [25], SSH [82]).

Though several Internet-wide service discovery measurements have been presented in the literature [10], [25], [35], [83], they have been largely focused on ob-

taining data in some finite amount of time (often months) rather than designing a high-performance scanner or maximizing politeness at remote networks. In accordance with the high-level goals we set for the algorithms developed in this dissertation, we maintain three objectives for a good Internet-wide scanning solution. The first requires efficient usage of local resources, which will ensure that the implementation supports scan durations T on the order of hours or even minutes. The second objective is to provide accurate extrapolation of interesting metrics when partial scans, which are often useful when only the *number* of hosts is desired instead of their actual IPs, are performed. The last objective is to maximize politeness at remote networks, which is accomplished by reducing the burstiness (i.e., instantaneous load of traffic) sent to target subnets. This serves the dual purpose of avoiding overload for intermediate routers and lowering incidents of wasted investigation effort, false alarms by Intrusion Detection Systems (IDS), and general administrator annoyance.

After initially showing that previous work does not satisfy our objectives, the second part of the chapter presents our design of IRLscanner, which is a high-performance and source-IP scalable framework for service discovery in the Internet. We show that achieving optimal politeness at remote networks requires a novel algorithm for controlling the order in which IPs are targeted (i.e., a permutation) as well as an algorithm for parceling targets to local scanning nodes (i.e., a split). The resulting techniques space probes to each CIDR subnet s evenly throughout scan duration $[0, T]$, even with multiple source IPs. The permutation is also designed to provide accurate extrapolations using partial scans, which is exhibited by the 1% estimation error in the number of live hosts IRLscanner incurs after only a 10 second scan (at a sending rate that would cover the Internet in $T = 24$ hours). Finally, the goal of allowing for arbitrarily fast scan durations leads us to several optimizations that help achieve our objectives. We show that the scope of measurements (i.e., the number of

IP addresses scanned) can be decreased significantly over that of previous scanners without significantly affecting results, that retransmissions are largely ineffective and can be omitted, and that by using much larger timeouts for unresponsive targets we can capture a wider array of busy/slow hosts in the Internet than was possible before.

Using IRLscanner, in the third part of the chapter we perform 20 Internet-wide scans that run over 20 times faster than any prior scanner, span three protocols, and encompass several ports such as DNS (port 53), HTTP (port 80), SMTP (port 25), EPMAP (port 135), and UDP ECHO (port 7). We perform several experiments that have never been attempted in the literature on an Internet-wide scale, including targeting several novel ports (i.e., ECHO, EPMAP, SMTP), using different types of packets (e.g., ACK scanning), and performing the first large-scale OS fingerprinting study of 44M web servers that respond to port 80. We finish the chapter by presenting the feedback received during our experiments in an effort to inform others of what to expect when performing similar studies. Included are a detailed analysis of email complaints, techniques for using firewall log correlation data to understand the impact of individual measurements on IDS detection in the Internet, advice for predicting the potential number of complaints when scanning a particular port, and other methods for reducing the perceived maliciousness of scans. In the next section we extend our analysis of the impact of scanning on remote networks in a more formal manner by focusing on IDS and modeling detection rates in light of various scan patterns.

1.4. Modeling Window-based IDS and Stealth Scanning

One of the difficulties of understanding the impact of horizontal scanning on remote networks is a lack of information about what administrators consider to be harmful traffic. Fortunately, as the number of malicious entities in the Internet has grown over

time [78], [108], many networks now use Intrusion Detection Systems (IDS) to monitor traffic and detect various forms of potentially malicious activity, including horizontal scanning that we employ with IRLscanner. When an IDS detects unwanted traffic, it can cooperate with firewalls to block offending hosts and alert administrators of the actions taken. To formally analyze horizontal scanning, we model two popular IDS implementations (i.e., Snort [105] and Bro [12]), study their ability to detect existing techniques [3], [41], [59], [62], [82], [83], [108], demonstrate that more stealthy algorithms can be used to scan faster than previous methods while avoiding detection, and suggest modifications to current IDS that mitigate such algorithms.

While there are several algorithms that can be used by IDS to detect scanning once past history has been established [43], [95], [113], most existing IDS tools [12], [44], [74], [84], [105] keep per-flow statistics only for a limited period of time to establish this history, which is called *window-based* processing of traffic. Given the large quantity of data commercial IDS must process, this maintains scalability [56] and ensures that state will not grow to infinity. To avoid a large number of false positives, IDS also typically require for a configurable number of packets, called a *threshold*, to be received during a particular window before raising an alarm or initiating estimators such as TRW [43]. Given the purely regenerative [89] nature of window-based processing and the need to exceed the threshold, it then becomes possible for a scanner to avoid detection simply by staying beneath the threshold throughout each window. However, it is unclear how well current methods avoiding tripping IDS and whether there are better techniques, which we tackle next.

We start by developing two models for window expiration based on deployed IDS solutions [12], [44], [74], [84], [105]. The first we call IDS-A, which is based on Snort [105] and expires the state of *all* scanning sources every Δ_s time units for subnet s. The second we call IDS-B, which is based on Bro [12] and expires state for each

individual source i Δ_s time units after the last target hit by i. IDS-B selectively tracks sources that continuously scan and never expires their state, which allows IDS-B to perform much better than IDS-A at detecting slow scanners. We then analyze existing scan patterns in light of these IDS models and develop optimal strategies for avoiding detection.

To aid our analysis, the key metric we introduce is *stealth cover time* (SCT), which is the minimum scan duration T that allows a particular Internet-wide scan pattern X (i.e., permutation, split, and schedule of time instances when packets are sent) to avoid detection at s. Using SCT, we then define a scanner to be *stealth-optimal* (SO) if it simultaneously minimizes the SCT of all CIDR subnets under both IDS-A/B. We then show that SO patterns exist, define their properties, and derive their SCT in relation to IDS-A/B so that existing scan patterns (i.e., IP-sequential [108] and uniform [62], [82], [83], [108]) can be analyzed for their stealth ability. After deriving the probability that both IDS-A/B detect each scan pattern, we derive their respective SCT's and show that the uniform permutation is generally much stealthier than IP-sequential. However, contrary to common belief [3], [41], [59], we demonstrate that in some cases IP-sequential is stealthier than uniform against IDS-A (i.e., in all networks larger than /20) and against IDS-B (i.e., in those larger than /21). As expected, IDS-B is significantly harder to avoid than IDS-A, and both permutations require a constant-factor slower scanning as a result. Most significantly, in comparison to SO patterns we show that the uniform permutation is orders of magnitude slower under all practical conditions and SO patterns cover both IDS-A/B with the same SCT. For example, using /16 subnets and default Bro settings, SO scanning has an SCT that is $1,209$ times smaller than uniform, which results in a reduction in scan duration T from 3.3 years to 1 day without increasing the probability of detection.

The next part of the chapter deals with implementing the stealth-optimal pat-

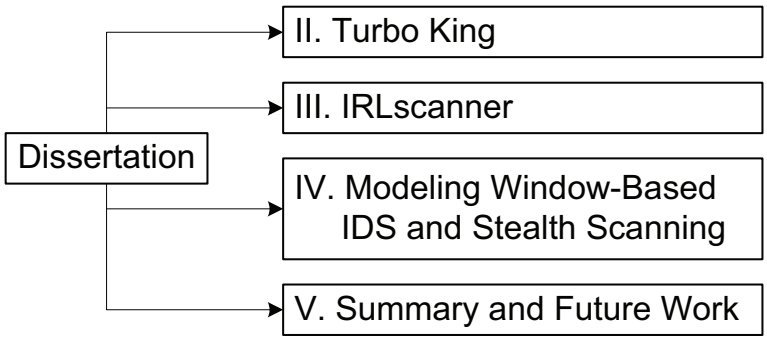


Fig. 1. Organization of this dissertation.

terns derived previously and testing them in the Internet. In practice, SO patterns can be achieved using the same permutation and split proposed for IRLscanner, but in order to achieve stealth-optimality against IDS-B a new schedule is required to allow for variable sending rates. In fact, the algorithm used in IRLscanner is a special SO case called *unaware* with an SCT that is slower against IDS-B than IDS-A. Using IRLscanner and the newly designed algorithm that is stealth-optimal against both IDS-A/B, we run three Internet-wide HTTP (port 80) scans using $T = 24$ hours and monitor the resulting number of scanning reports generated at the SANS Internet Storm Center (ISC) [94]. We demonstrate a nearly 40% reduction in reports from SO scanning over the original unaware IRLscanner, which implies that IDS-B is being actively used by IDS in the Internet. Our testing also allows us to determine the optimal parameters for minimizing the number of ISC scan reports, which agree with our analysis and confirm the proposed models. To round out the chapter, we propose a new model of scan detection called IDS-C that dynamically increases Δ_s with every packet received and results in significantly reduced effectiveness of SO scan patterns.

The rest of the dissertation is organized as illustrated in Fig. 1. In Chapter II we present the Turbo King framework for distance estimation in the Internet. Chapter III contains our design of the IRLscanner Internet-wide service discovery tool

and subsequent experiments, then Chapter IV tackles modeling Intrusion Detection Systems and the existence of stealth-optimal scan patterns. We finish with Chapter V, which summarizes the dissertation and discusses future work.

CHAPTER II

TURBO KING

2.1. Introduction

Widespread interest in distance estimation in the Internet has recently evolved into a large field [7], [16], [20], [24], [28], [32], [33], [49], [54], [60], [75], [76], [80], [87], [98], [99], [109], [111], [118], [126]. The purpose of this research is to estimate or measure the latency between hosts, which can then be leveraged to provide better service to end-users and construct more efficient networks. Examples include increasing the responsiveness of online games, efficiently locating the closest server in a content distribution network, and building topologically-aware P2P networks. While the existing approaches are promising, obtaining a large-scale[1] Internet distance map for verification of virtual-coordinate approaches [20], [24], [33], [36], [49], [60], [76], [98], [99], [111] and actual use in deployed applications has proven to be a difficult task. The aim of this chapter is to introduce a first step in this direction and propose a framework that allows such a service to be transparently enabled in the current Internet.

Due to the difficulty of deploying tracers [1], [28], [72], [81], [102] in every possible network, we choose to build upon an existing technique called King [32] that does not require any changes to existing protocols or access to remote computers. King approximates the distance between end-hosts using the delay between DNS servers

[1]The scale considered in this chapter assumes building an all-to-all delay matrix between approximately $220,000$ BGP prefixes advertised in the Internet. This is in contrast to the frequently-used latency maps today [24], [61], [76], [126] that rely on $100-400$ nodes in PlanetLab or $1700-2500$ nodes in the DNS tree.

authoritative for IP addresses of the hosts in question. While generally accepted as a sound methodology for estimating delay and used in many papers [5], [6], [15], [21], [23], [24], [29], [53], [57], [86], [88], [91], [96], [109], [118], [124], King has not been analyzed for accuracy and pitfalls since the original paper [32], nor has it been involved in measurements larger than 2500×2500 nodes. Furthermore, some of the advanced techniques suggested in [32] have never been implemented and their feasibility in practice has not been assessed in the literature.

We start the chapter by identifying causes of King's inaccuracy and evaluating its suitability for large-scale measurements. We first argue that King incorrectly estimates delay when the target DNS zone contains multiple nameservers that are not geographically close to each other (e.g., outside the target domain and its BGP network). We also find that King can estimate entirely wrong delays when the source DNS zone uses forwarders, which are stand-alone servers that aggregate queries from multiple domains. In such cases, King fails to detect the presence and location of forwarders, in addition to incorrectly measuring the forwarder's query-processing delay that must be subtracted from the final measurement. In regard to overhead, King utilizes a complex multi-step process (see below for the algorithm) that requires numerous queries for each delay measurement and seeding of source DNS servers with a large number of unwanted entries. As the scale of the experiment increases, cache pollution becomes a non-trivial issue.

To overcome these drawbacks, we propose a new system called Turbo King (T-King) that streamlines the process of making distance measurements using DNS, improves their accuracy, reduces overhead, and almost entirely eliminates cache pollution. The first component of T-King is a large collection of nameservers distributed throughout the Internet, from which the closest nameserver to each end-host A is selected for use in the measurement. In the current implementation, we use periodic

crawls of the DNS tree to find nameservers that can be used in the measurement and maintain this information in our server. The second component of T-King is a new measurement algorithm based on several improvements we have made to the advanced techniques in [32] that mitigate problems caused by forwarders and zones with multiple authoritative nameservers. Our approach not only reduces the number of queries and bandwidth overhead of King by more than 50%, but also achieves higher accuracy and a factor of N reduction in the number of polluted cache entries at each remote server for an $N \times N$ latency measurement.

We finish the chapter by showing how to build the current database of DNS servers in Turbo King, examining how likely King is to experience its drawbacks in practice, and assessing the effect of these drawbacks on King's delay estimation. We first perform a reverse DNS crawl to discover a set of $216,843$ nameservers, out of which we find $117,817$ to be recursive and accepting queries from outside networks.[2] These servers reside in 174 countries, cover over $31,000$ BGP prefixes, and are responsible for approximately 50% of IP addresses (i.e., 828 million) advertised in BGP [92]. Further analyzing the data, we find that 33% of reverse DNS zones utilize a nameserver that neither belongs to the same BGP prefix nor the same domain as the other servers. Additionally, over 32% of recursive servers found in this study use a hidden forwarder, which suggests that a large fraction of King's measurements may be affected by the drawbacks identified in this work. We finish the chapter by quantifying the effect of this bias using a small 50×50 delay matrix and comparing the estimates of King to those of Turbo King. Our results show that 15% of the measurements are different by more than 10% and 8% by more than 20%, which suggests that the magnitude of bias in King is generally mild, but nevertheless non-negligible.

[2]Other techniques (such as those in Chapter III) can significantly expand this database.

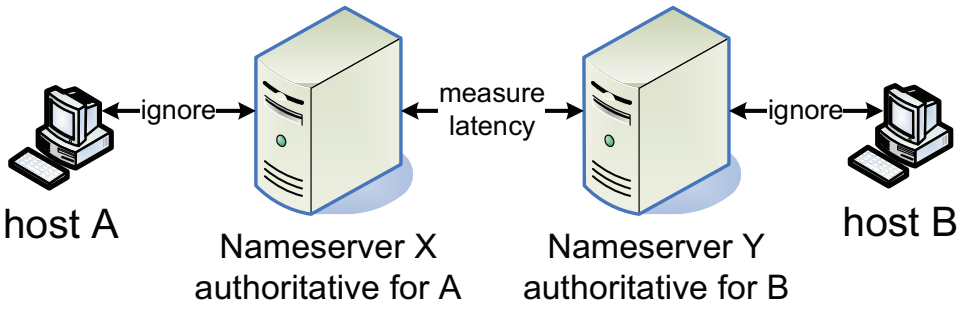


Fig. 2. King estimates the latency from host A to B.

The rest of the chapter is organized as follows. Section 2.2. studies previous work. Sections 2.3. and 2.4. outline issues with King. Section 2.5. introduces T-King and Section 2.6. evaluates our method, comparing it to King. Section 2.7. concludes the chapter.

2.2. Background

The Domain Name System (DNS) [66], [67] is a distributed tree-based database that allows for the resolution of domain names to various types of data, most notably IP addresses. The DNS standard [67] also provides for reverse lookup of IP addresses, which is accomplished through the `IN-ADDR.ARPA` domain tree. There are several types of servers and clients that operate on DNS and to avoid confusion we introduce the following terminology. In this chapter, a *recursive resolver* is a server that queries the DNS and returns answers to end-hosts. *Nameservers* are DNS servers that maintain authoritative data about a subset (i.e., zone) of the domain space. *Recursive nameservers* act as both a recursive resolver and a nameserver simultaneously. An *open resolver* is either a recursive nameserver or a recursive resolver that responds to recursive queries for arbitrary zones from hosts *outside* its local network.

King [32] uses existing DNS infrastructure to measure the latency between two

hosts on the Internet. The method relies on the fact that open recursive nameservers on the Internet will attempt to resolve any valid request, which forces them to query *remote* nameservers for the proper response. The time that these queries take to be processed can be measured to determine the distance between the two nameservers. In order for the measurements to apply to *arbitrary* hosts, Gummadi *et al.* assume that end-hosts on the Internet are within close proximity to the authoritative DNS nameserver that maintains DNS information about their IP address. Given this assumption, King approximates the delay between hosts A and B using the latency between their authoritative servers X and Y as shown in Fig. 2. Heuristics are used to choose which authoritative nameserver to include in the measurements, the details of which can be found in [32].

2.3. Understanding Original King

We refer to the main technique proposed by Gummadi *et al.* in [32] as *Original King* (O-King). O-King has been used extensively in the literature [5], [6], [15], [21], [23], [24], [29], [53], [57], [86], [88], [91], [96], [109], [118], [124] as a way to easily collect latency information from the Internet; however, no formal or detailed analysis of its pitfalls exists to date. We first describe the measurement algorithm used by O-King, which is necessary for understanding its limitations and our proposed system later in the chapter.

2.3.1 Measurement Algorithm

We start by defining terminology. Throughout the rest of the chapter, a *query* is defined as a single DNS request sent to a remote server and an *answer* is the response to a query. Queries are either recursive or iterative as defined by the DNS specification

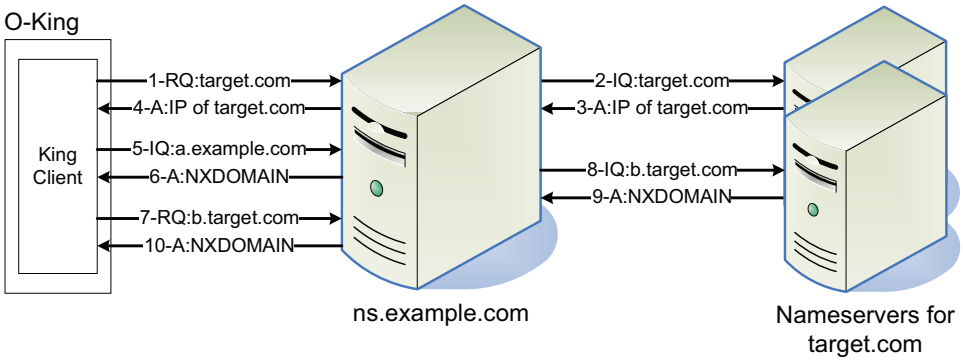


Fig. 3. O-King query sequence.

[66]. Given time t_s when a query is sent and t_r when the answer is received for that query, we define a sample s to be $t_r - t_s$. We are now ready to detail the O-King algorithm that measures delay between two nameservers.

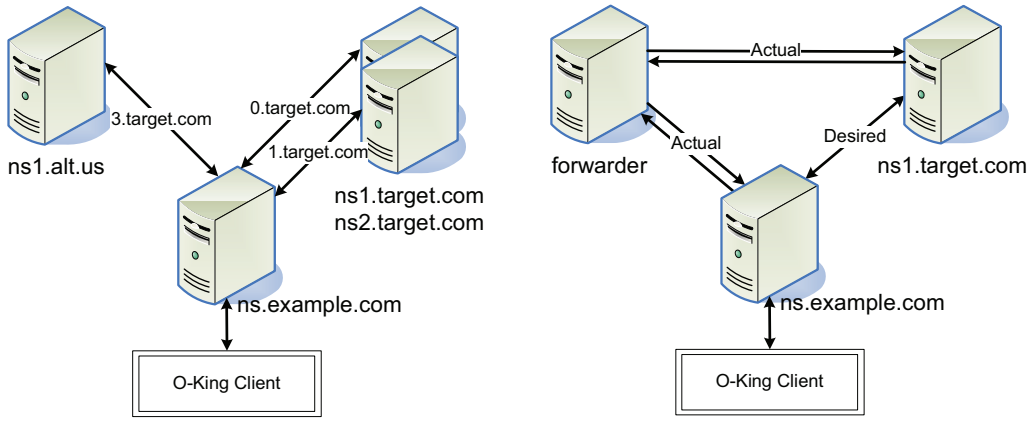
The O-King process is illustrated in Fig. 3, where `ns.example.com` is a recursive nameserver chosen by O-King as "close" to the desired IP. In the figure, each query is labeled as either `RQ` for recursive query or `IQ` for iterative query. Answers are labeled with `A`. A *seed* recursive query, which is represented by message numbers 1–4, is sent to `ns.example.com` for the `target.com` domain to ensure direct contact between the two for subsequent measurements. Messages 5 and 6 show the *local latency sample* L_i between the O-King client and `ns.example.com`, which is accomplished by a simple iterative query that can be repeated to improve accuracy. Illustrated by messages 7–10 is the *remote latency sample* R_i, which uses a recursive query to measure the delay from the O-King client to `target.com` (via `ns.example.com`) and also can be repeated. The resulting latency estimate between `ns.example.com` and `target.com` produced by O-King is $\min\{R_i\} - \min\{L_i\}$. One of the features that makes O-King so attractive is its ease of use; however, it has certain drawbacks that we discuss in the remainder of this section.

2.3.2 Zones with Multiple Authoritative Nameservers

In the original specification for DNS [66], [67], it is recommended that authoritative nameservers be placed in geographically diverse locations on separate networks. Thus, if connectivity is interrupted at one of the sites, the remaining nameservers would maintain availability for the zone. Queries for a particular zone are sent to one address in the group of nameservers, but the decision about *which* nameserver to query is left up to the individual resolver implementation. As O-King requires at least four [32] samples to converge to an accurate measurement, *different* nameservers are potentially used for each sample. While this is of little consequence if all nameservers for a zone are on the same network, in cases where the DNS specification is strictly followed the samples could be very different, leading to inaccuracy in the final latency estimation. This issue is illustrated in Fig. 4(a) for three samples taken by the O-King client, where the authoritative nameservers for the `target.com` zone are `ns1.target.com`, `ns2.target.com`, and `ns1.alt.us`.

2.3.3 DNS Forwarders

Another potential issue for O-King measurements is the use of forwarders on the Internet by system administrators. A forwarder serves as an aggregation point for DNS queries initiated from within a network that target external destinations. If a recursive nameserver that is configured to forward messages receives a recursive query for a zone it has no authority over, it sends the query to the forwarder *without notifying* the end-user. The forwarder then resolves the query instead of the recursive nameserver. This process is illustrated in Fig. 4(b), where direct contact is intended between `ns.example.com` and `ns1.target.com`, but the query is routed through the forwarder instead. The presence of forwarders is undetectable by O-King and



(a) multiple nameservers (b) use of forwarder

Fig. 4. O-King query sequence for two different server configurations.

compromises the assumption that there is direct contact between `ns.example.com` and `ns1.target.com`, leading to an invalid latency estimate.

2.3.4 Cache Pollution

The final concern that arises from the use of O-King is the impact it has on the nameservers used for latency estimates. While the purpose of an authoritative DNS nameserver is to provide accurate information about the data under its control to the global Internet, the purpose of a DNS cache is to reduce latency strictly for *local* users, those end-hosts that principally rely on the nameserver to resolve queries on their behalf. Given that DNS caches are intended to benefit these users, we define *cache pollution* to be the insertion of DNS zone data that has not been requested by a local user into the cache of a nameserver.

O-King uses a seed query to force the recursive nameserver to cache the `NS` (nameserver) and `A` (IP address) records of *all* target authoritative nameservers. While this is unlikely to cause performance problems on a small scale, initiating billions of O-King queries could lead to a large proportion of the cache containing information that

was not requested by local users. Furthermore, local administrators are unlikely to view this intrusion as benign and may take preventative steps, jeopardizing future measurements using O-King.

2.4. Understanding Direct King

We refer to the second technique proposed in [32] as *Direct King* (D-King), which involves a modification of the O-King measurement algorithm that allows for specification of a single nameserver from the target zone. While not mentioned explicitly in the original paper, all other aspects of D-King (i.e., nameserver selection, end-to-end estimation assumptions) we assume to be equivalent to O-King. To our knowledge, only Ballani *et al.* [6] have partially implemented D-King, which was required for their study of IP Anycast as deployed by DNS root servers. There was no study or analysis of D-King in [32]. We start by describing the D-King algorithm and later discuss some of its drawbacks.

2.4.1 Measurement Algorithm

The D-King latency estimation process is illustrated in Fig. 5, where `ns.example.com` is again a recursive nameserver. In contrast to O-King, where the query is sent to one or more nameservers responsible for the `target.com` domain, D-King allows the user to pick a single authoritative nameserver, which in this case is `ns1.target.com`. To accomplish this, D-King requires that a domain name be registered and a nameserver set up to resolve queries for said domain. In the figure, `king.com` is the example domain and `ns.king.com` is its authoritative nameserver.

D-King first requires a seed query to guarantee direct contact between `ns.example.com` and `ns1.target.com`, which is illustrated in the figure by messages 1–4. To do this,

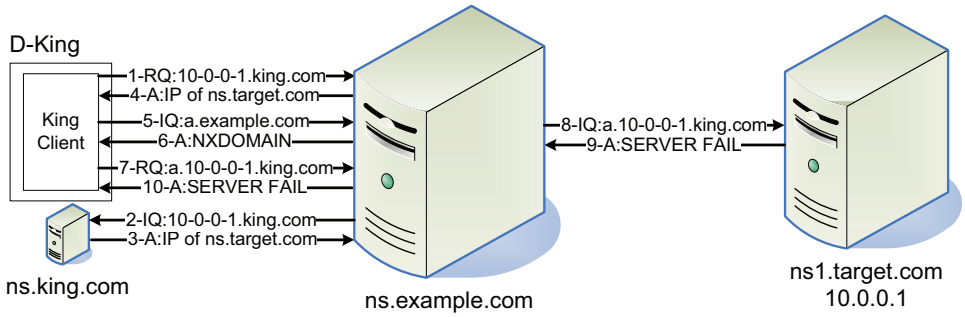


Fig. 5. D-King query sequence.

D-King initiates a recursive query to `ns.example.com` for `10-0-0-1.king.com`, which encodes the IP address of `ns1.target.com` into the query. As `ns.king.com` is authoritative for the `king.com` domain, it receives the query and uses the encoded address to respond that `ns1.target.com` is authoritative for the query, which is then *cached* at `ns.example.com`. By doing so, `ns.example.com` will now automatically forward queries for `10-0-0-1.king.com` directly to `ns1.target.com`. Once the cache is seeded, the actual latency measurements can be taken. Local sample L_i, represented as messages 5 and 6 in the figure, is taken in the same fashion as O-King. Remote sample R_i, illustrated by messages 7–10, is recorded by sending queries for random sub-domains of `10-0-0-1.king.com` to `ns.example.com`, which directly queries `ns1.target.com` as a result. Since `ns1.target.com` is not actually authoritative for the zone, it responds with an error indication, which is then echoed back to the D-King client. The final latency estimate produced by D-King is calculated in the same manner as that in O-King.

2.4.2 Additional Complexity

While D-King indeed eliminates the issue of zones with multiple authoritative nameservers affecting the latency estimate, the cost of this improvement is that the D-King

client must explicitly specify the target nameserver, which is not required by O-King. It is not mentioned in [32] exactly how this should be accomplished, but the same heuristic approach for discovering a close recursive nameserver applies in this case as well. Furthermore, a domain must be registered and a nameserver set up to respond to queries in the way D-King requires. One of the major benefits of O-King is that latency estimates can be obtained from any machine with an Internet connection, whereas D-King requires this extra infrastructure. The individual must decide whether the additional complexity is worth the improved accuracy.

2.4.3 DNS Forwarders

Along with O-King, the use of forwarders on the Internet affects D-King latency estimates as well. The D-King client and authoritative nameserver for the measurement (e.g., `ns.king.com` in Fig. 4(b)) are separate entities that only communicate through the query encoded with the IP address of the target nameserver. It is inconsequential to `ns.king.com` that a *different* nameserver (i.e., the forwarder) than the one intended by the D-King client sends it the query and caches the response. Because of this lack of communication between the components of the D-King latency estimates, forwarders remain undetected and affect the results in the same manner as discussed in the O-King case.

2.4.4 Cache Pollution

The seed query required by D-King plants authoritative data for the registered domain (e.g., `king.com`) at the recursive nameserver in a similar fashion to that required by O-King. However, there are differences in the impact on local DNS caches. The O-King seed query forces the caching of data for *all* authoritative nameservers of the target zone, whereas D-King caches data for a single nameserver. In contrast to

O-King, where the cached entries might have some future use to the local users, the D-King entry is *only* useful to the latency estimate. At the scale of billions of queries, if D-King is used the nameserver's cache would contain fewer entries than O-King, but those entries would be entirely useless to local users.

2.5. Turbo King

In this section we propose Turbo King (T-King) to address the drawbacks previously highlighted. We start by giving a high-level overview of the system then finish the section with detailed descriptions of the various components.

2.5.1 Design

Turbo King is a stand-alone service that accepts as arguments the IP addresses of end-hosts A and B from the Internet and returns the estimated latency from host A to B. It is currently implemented to resolve single estimate requests for end-host pairs.

To accomplish this goal, Turbo King maintains a large list S of N nameservers positioned throughout the Internet, which includes both recursive nameservers and non-recursive authoritative nameservers found in the DNS hierarchy. This list allows us to discover the closest nameserver without relying strictly on heuristic methods or assuming that the authoritative nameserver responsible for A's IP address is the closest nameserver to A. Turbo King first uses BGP data [92] to match the IP address of A to a recursive nameserver. If a match is found, the two are likely to reside in the same network. If no matching nameserver is found in the same network as the end-host, we simply find the recursive nameserver that has the longest matching prefix to A or select the default nameserver authoritative for A's IP address. Turbo King

then repeats the same process for B but expands the set of possible nameservers to include those that are not recursive. This is done because the target nameserver need not resolve recursive queries for the estimate to succeed.

Given the two nameservers, Turbo King then generates a latency estimate between them (the algorithm is described below) and returns the result. T-King operates in one of two modes. The default is *passive*, whereby T-King waits for requests before generating latency estimates. Estimates are cached for a configurable amount of time (e.g., 30 minutes) such that subsequent requests using the same two nameservers do not trigger a new measurement. This mode puts the least strain on resources as it only visits popular destinations. The optional mode we call *active*, in which Turbo King preemptively takes latency estimates between nameservers on the list so as to eventually obtain an entire $N \times N$ delay matrix.[3] This mode consumes more resources and possibly produces estimates that are never used, but it reduces the user-perceived delay and allows the matrix to be directly downloaded for use in applications and other research studies.

2.5.2 Discovering Nameservers

Turbo King is most effective when its list S of nameservers is large, such that at least one nameserver is "close" to every IP address that is currently in use on the Internet. The current version of T-King compiles its list of nameservers by performing exhaustive crawls[4] of the `IN-ADDR.ARPA` reverse DNS tree using the techniques

[3]For $N = 117,863$ used in the current version and one query per 22 seconds per DNS server, the entire matrix consisting of 13.8 billion measurements can be built in 30 days.

[4]Analysis shows that 85% of nameservers found by T-King in Nov. 2006 were active in Dec. 2007, which suggests that monthly or even annual re-scanning of the tree should keep the DNS server set relatively fresh.

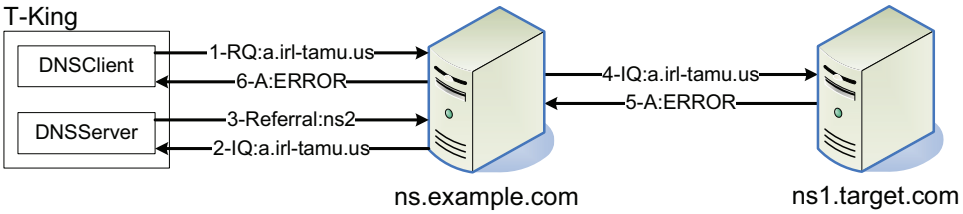


Fig. 6. Turbo King query sequence.

introduced in [42]. In contrast to [42], which accepts *cached* (i.e., non-authoritative) entries to queries because they are only interested in the number of hosts represented in the tree, our crawler probes the entire depth of the reverse tree by accepting *only* authoritative answers, which maximizes the number of nameservers found. Results from this crawl are presented in the next section, but we should note that significantly larger datasets can be built using other techniques (e.g., $N = 333,963$ in [64], over $580,000$ in [114], and approximately 4.4M stable servers as shown in Chapter III). As these results represent a large jump in S and contain more than 10M unreliable responses, further study is required to validate and use such servers.

2.5.3 Measurement Algorithm

Our proposed algorithm is illustrated in Fig. 6, where Turbo King operates as a multi-threaded application with both the client and server operations communicating seamlessly. This allows timestamps to be taken for every packet sent or received by our software, which reduces the number of queries required to complete an estimate. During step 1, `DNSClient` takes a timestamp and initiates a query to `ns.example.com` for the domain we control, namely `irl-tamu.us`. Since `DNSServer` is listed with the `.us` registrar as the authoritative nameserver for this domain, host `ns.example.com` recursively queries `DNSServer` in step 2. At step 3, our software takes another timestamp and answers with a referral saying that `ns1.target.com` is authoritative for

the query, but sets the TTL for this information to zero, meaning that it should *not* be cached [67].[5] Nameserver `ns.example.com` then directly queries `ns1.target.com`, which answers with some form of error indication (steps 4–5). That error indication is forwarded back to `DNSClient` in step 6, which then takes the third and final timestamp, allowing us to estimate the latency between nameservers `ns.example.com` and `ns1.target.com` as $d_{36} - d_{12}$ where d_{ij} is the delay between steps i and j. Thus, Turbo King is able to determine the latency between `ns.example.com` and `ns1.target.com` without seeding the cache of `ns.example.com` by judiciously taking timestamps at every point of communication between `ns.example.com` and the T-King software.

2.5.4 Detection and Avoidance of Forwarders

While both O-King and D-King are unable to detect forwarders, they are simple to detect with Turbo King due to its integrated infrastructure and can be eliminated from the measurement. Because T-King acts as both the client *and* the server application for the latency estimate, it simply compares the IP addresses that are used to contact `DNSClient` and `DNSServer` respectively for a particular query. If different IP addresses are used, T-King excludes the original IP from the list of recursive nameservers and determines if the forwarder allows for recursion, adding it to the list of possible nameservers if so. A new closest server to the IP is retrieved from the list of recursive nameservers and the latency estimate restarts. While there is some small additional delay in returning an answer to the end-user when in passive mode, the resulting estimate is not tainted by the presence of a forwarder.

---

[5] We found that 35 of the 117,817 discovered recursive nameservers were either misconfigured or non-compliant with [67] and ignored zero TTL.

Table I. Results of reverse DNS crawl (3.8 GHz Pentium 4)

	T-King	ISC [42]
Month run	Nov. 2006	Jul. 2006
Duration (hours)	33.8	240
Queries/Sec	5,300 (2.3 mb/s)	751 (0.3 mb/s)
Queries Completed	649,270,000	N/A
IPs Discovered	439,431,355	439,286,364
Nameservers	216,843	89,592
Recursive Nameservers	117,817	N/A

2.6. Evaluation

In this section we evaluate the effectiveness of Turbo King for providing accurate latency measurements and its suitability for large-scale studies compared to O-King and D-King. We start by discussing our efforts to discover a large number of nameservers, then perform several real-world measurements to compare the three algorithms.

2.6.1 Results from Reverse DNS Crawl

Because of the large number of queries required to complete the `IN-ADDR.ARPA` crawl, we designed and implemented a multi-threaded DNS resolver to collect a list of nameservers and authority data for *all* zones in the reverse lookup tree. The results of one particular crawl executed in November 2006 are summarized in Table I, where both Turbo King and ISC [42] found roughly the same number of IP addresses in the tree; however, our crawler was approximately seven times faster and discovered 2.4 times more nameservers. Examination of the nameservers we discovered revealed that 117,817 of them support recursive queries. Using the T-King client, we profiled each

Table II. Coverage of the Internet with discovered servers

	All	Recursive	Total
Countries	190	174	232 [40]
AS	13,017	10,895	23,773 [37]
BGP Prefixes	48,196	31,059	219,110 [92]
IPs covered	1,031,736,562	828,675,500	1,642,441,178
Web servers	3,192,918	2,659,379	3,638,433
Gnutella peers	1,734,483	1,338,217	3,534,300

of the recursive nameservers in our list and found that 32% use a forwarder to resolve queries for zones not under their control.

We next study the coverage of the Internet by all discovered nameservers and the subset of nameservers that are recursive, which is illustrated in Table II. This data shows that Turbo King contains a nameserver in 190 countries, covering over 13 thousand ASes, 48 thousand BGP prefixes [92], and 1.03 billion IP addresses out of 1.6 billion advertised by BGP [92]. We performed further analysis of how well the discovered BGP prefixes cover 3.5 million Gnutella peers found in prior work [116] and 3.6 million web servers (hosting over 6.3 billion webpages) found by our unrelated web-crawling project. These numbers show that 49% of peers and 88% of web servers reside in BGP prefixes that contain at least one nameserver discovered by T-King.

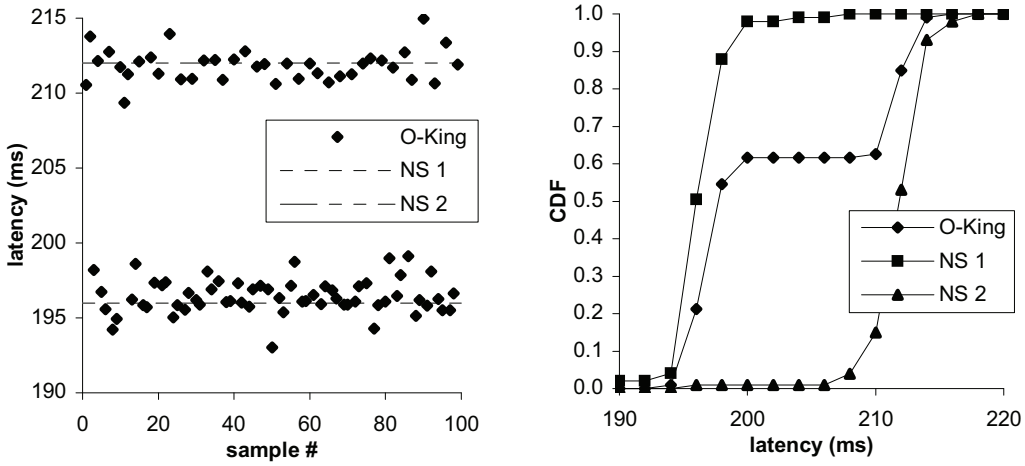
For the subset of nameservers that are recursive, Turbo King found nameservers in 174 countries, representing nearly 11 thousand ASes, 31 thousand BGP prefixes, and 828 million IP addresses. This resulted in a coverage of 37% of Gnutella peers and 73% of webservers. While T-King is able to find a nameserver in the same network for a large percentage of end-hosts (especially web servers), the relatively low percentage of Gnutella peers covered indicates that we should aim to discover more recursive

resolvers used by home-based Internet connections in the future.

2.6.1.1 Analyzing Zone Authority Data

Of further interest is the percentage of zones in the reverse lookup tree that contain multiple authoritative nameservers, which we examined by recording the set of nameservers authoritative for every zone during the `IN-ADDR.ARPA` crawl. The accuracy of O-King estimates is only significantly affected if one or more of the nameservers for a zone is in a *different* network, making it likely for O-King to produce conflicting results over multiple samples. We downloaded 219,110 BGP prefixes from RouteViews [92] and matched each nameserver's IP to one or more prefixes, then examined the nameserver set for every zone in the reverse lookup tree. We found that 49% of reverse lookup zones contain at least one nameserver in their set that is in a different network. While this is a striking result, it is possible that the unmatched nameserver could be in another network under the same administrative control that is well-connected to the rest of the nameservers in the set.

Accurately determining administrative control for a large number of networks is difficult, but it stands to reason that if all nameservers for a zone share a single domain name, they are more likely to be under one organization's administrative control. While the process is easy for generic top-level domains (gTLDs), it is significantly more complex for country-coded TLDs (ccTLDs) as most countries created sub-domains from which people could purchase their own domains (e.g., `.com.es`). We compiled a comprehensive list of these sub-domains for each ccTLD and hereby refer to this list and the set of gTLDs as pay-level domains (PLDs). We again evaluated the nameserver set for each zone and found that 33% have at least one nameserver in their authority set that both resides in a different network and has a different PLD than the other nameservers. It is very likely that the accuracy of O-King queries for



(a) sample sequence (b) CDF of latencies

Fig. 7. Comparison of O-King to D-King for zone with two nameservers.

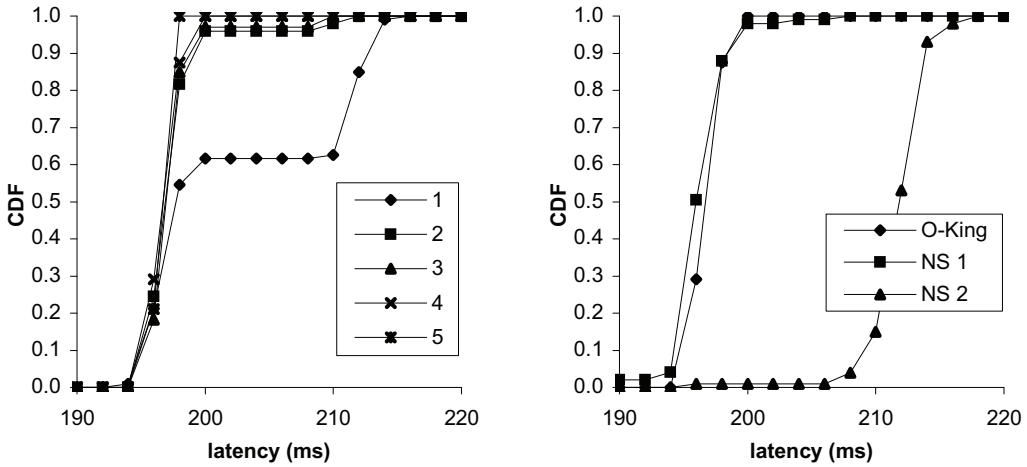
these zones will be negatively impacted.

2.6.2 Causes of Inaccuracy

In this section we compare O-King and D-King to Turbo King using latency estimates they produce from the Internet. To remove variability caused by differences in architecture, we implemented all three algorithms using the same timing and socket mechanisms and ran all of the tests from a single Windows 2003 x64 machine. To highlight the differences in accuracy, we focus only on the actual latency estimate between nameservers and note that T-King should perform *no worse* than O-King or D-King in selecting a "close" recursive nameserver. In many cases it will perform better, but we leave such analysis for future work.

2.6.2.1 Zones with Multiple Authoritative Nameservers

To illustrate the impact of authoritative nameservers in different networks on O-King estimates, we chose a zone with two authoritative nameservers and used O-King



(a) latency estimate per sample size (b) four-sample O-King vs. D-King

Fig. 8. Convergence of O-King estimate for zone with two nameservers.

to generate 100 latency estimates to a target IP in the zone. We then used D-King to estimate the latency to the two individual authoritative nameservers for the zone. In Fig. 7(a) the sequence of O-King estimates are individual points and the two D-King estimates are represented as lines. O-King performs as expected and vacillates between the two nameservers arbitrarily. This is further demonstrated in Fig. 7(b), where roughly 60% of the time O-King chose `NS 1` as the preferred server. We confirmed that this behavior also occurs in zones with more than two nameservers, but omit these examples for brevity.

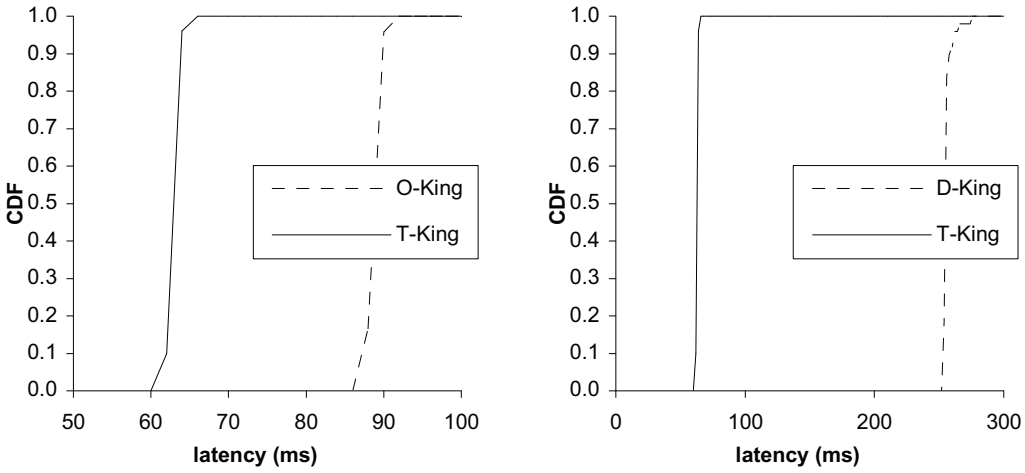
We next analyze the convergence properties of O-King measurements for zones with multiple authoritative nameservers. To determine exactly what happens to the latency estimate when the number of samples increases, we use the measurement data from above and plot in Fig. 8(a) the CDF of latency estimates for sample sizes one through five. From inspecting the figure, it quickly becomes apparent that the higher latency samples are ignored and are effectively removed from the overall estimate as the sample size increases. This is further illustrated in Fig. 8(b), where the latency

estimated by O-King using four samples is plotted with D-King measurements using one sample. As the figure clearly shows, the O-King measurement is nearly identical to the measurement given by D-King to `NS 1`.

There are two insights that can be gained from this behavior. The first is that the requirement of at least four samples per measurement proposed in [32] for O-King is at least partially due to the natural differences in latency between multiple authoritative nameservers for a particular zone. In contrast, D-King provides the same latency estimate with one sample in this case. The second issue is that O-King always biases its estimates towards the *lowest* latency nameserver of a zone. While in some cases this might be the server located closest to the target end-host, there is no evidence that this happens in the general case.

2.6.2.2 DNS Forwarders

The impact of forwarders on both O-King and D-King latency estimates largely depends on the proximity of the forwarder to the original recursive nameserver. If the two servers are on the same local network, any additional latency should be rather small. To quantify the likelihood of this event, we matched both the forwarder and the original recursive nameserver to their advertised BGP prefixes from RouteViews [92] and discovered that 45% of the time the two servers did not reside on the same network. To demonstrate the inaccuracy introduced by the presence of forwarders, we took 100 latency estimates using all three algorithms from a recursive nameserver known to use a forwarder to a zone with a single authoritative nameserver (this rules out effects from multiple authoritative nameservers on O-King). The resulting latency estimates are illustrated in Fig. 9(a), which compares O-King to T-King, and in Fig. 9(b), which compares D-King to T-King. In both figures O-King and D-King overestimate latency due to the presence of the forwarder, whereas T-King does not.



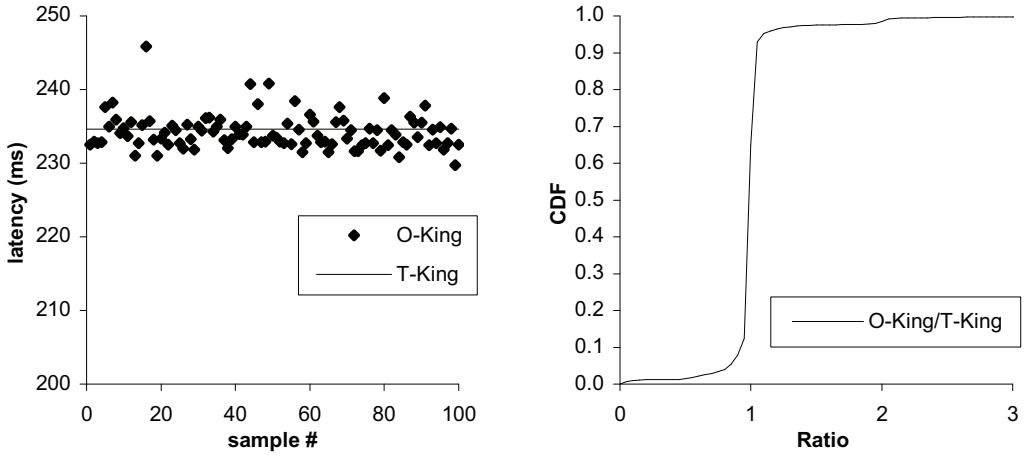
(a) O-King with forwarder (b) D-King with forwarder

Fig. 9. Forwarder affect on O-King and D-King (single nameserver zone).

The D-King estimate is larger than O-King due to multiple attempts to resolve the query by the forwarder, a problem that is mentioned in [32] and accounted for in T-King.

2.6.3 Measurement-based Comparison

To study Turbo King in more depth, we performed $2,450$ latency estimates on the Internet using 50 recursive nameservers from the previously discovered set for both T-King and O-King over various measurement sample sizes. From this data we show that T-King measurements are indeed different from those produced by O-King. We then show that Turbo King converges to a consistent latency estimate in two samples instead of the four suggested in [32]. D-King is omitted from this section, but the results are consistent with those found in the previous section. D-King is more accurate than O-King, but less so than T-King due to its inability to detect forwarders.



(a) single nameserver target zone (b) ratio of O-King/T-King

Fig. 10. T-King vs. O-King to validate implementation and show differences.

2.6.3.1 Turbo King versus O-King Estimates

Before comparing Turbo King to O-King in a general case, we first consider the two algorithms for a target zone with a *single* authoritative nameserver using a recursive nameserver that we verified does not use a forwarder. Because we eliminated all of the factors that skew O-King estimates, the two should produce the same value. The result is illustrated in Fig. 10(a), with the estimates produced by O-King as data points and the average estimate by T-King as a line to allow the reader to distinguish between the two. The figure clearly shows that T-King produces latency estimates that are equivalent to O-King in such idealized cases.

We next compare the $2,450$ latency estimates produced by Turbo King to those by O-King, which is shown in Fig. 10(b). To highlight differences between the two, we generated a ratio of the estimates by dividing O-King's value by T-King's, so that if O-King and T-King produced identical values, the CDF would be a straight line at one on the x-axis. Note that in this case we used four samples for each estimate

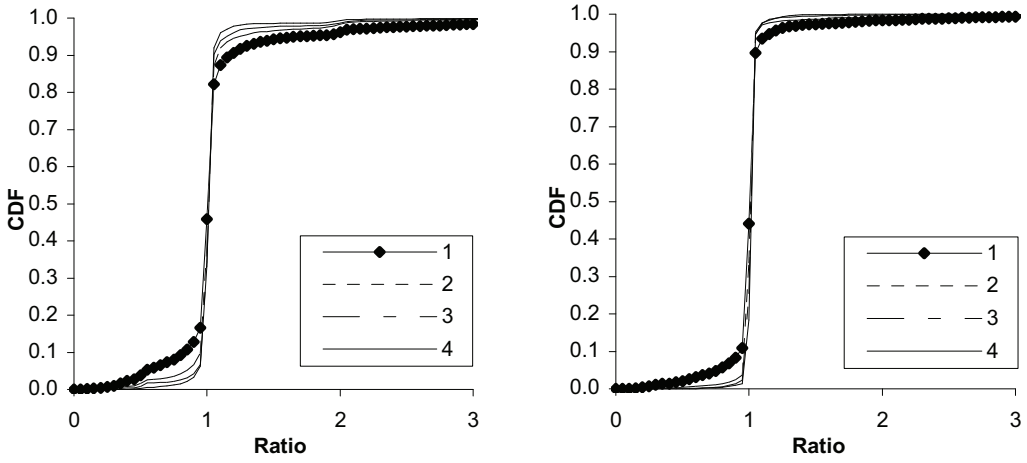
as suggested in [32]. From the data, 15% of O-King estimates are more than 10% different from T-King measurements, and 8% of O-King measurements are more than 20% different from those generated by T-King.

2.6.3.2 Convergence of Estimates

Previously, we showed that zones with multiple authoritative nameservers are one of the reasons O-King requires at least four samples to produce a latency estimate. In this section, we expand that study by examining the convergence properties of both T-King and O-King, showing that over a wide range of estimates Turbo King converges to a consistent estimate with fewer samples than required by O-King. To accomplish this, we repeated the above $2,450$ latency estimates using sample sizes varying from one to four for both algorithms. We collected two estimates for each sample size and calculated the ratio of both O-King to O-King and T-King to T-King. The goal is to provide *consistent* estimates for latency, so we plotted the CDF of the ratio in Fig. 11(a) for O-King and Fig. 11(b) for T-King, with each line representing the number of samples used to produce the estimate. In the O-King case, illustrated in Fig. 11(a), improvement in the consistency of estimates is apparent as the number of samples increases to four, whereas in the Turbo King case (Fig. 11(b)) the greatest improvement is from one sample to two, with little afterward. From these graphs we conclude that the recommendation of four samples in [32] is sound for O-King and that T-King produces an accurate sample using only two samples.

2.6.4 Overhead Analysis

In this section we study the resources required of DNS nameservers and the Internet for all three algorithms. In particular, we are interested in how the three compare for large-scale estimates involving more than $100,000$ recursive nameservers discovered



(a) O-King/O-King by sample size (b) T-King/T-King by sample size

Fig. 11. Convergence of measured latencies for O-King and T-King.

by Turbo King. We start by examining the number of queries sent to capture the network overhead, then discuss cache pollution for large-scale measurements.

2.6.4.1 Network Overhead

To study network overhead, we consider the number of queries required to perform all-to-all latency estimates for the $100,000$ recursive nameservers, which is 10 billion estimates. In this calculation, we included every query initiated either by or on behalf of the measurement client, and used the number of samples required to produce consistent latency estimates: four for O-King and two for D-King and T-King. Due to the lack of seeding, Turbo King requires 70 billion queries to complete 10 billion latency estimates. D-King needs 100 billion queries for the measurement, which is 1.43 times more than required by T-King. Finally, O-King uses 150 billion queries, or 2.14 times more than Turbo King and 1.5 times more than D-King. Thus, designing T-King to be more accurate and to avoid seeding led to a significant reduction in bandwidth usage.

We next consider the impact each algorithm has on DNS caches under the same measurement conditions.

2.6.4.2 Cache Pollution

We examine cache pollution by calculating the total number of DNS records inserted into the cache of the 100,000 recursive nameservers. Each entry includes two records, an `NS` record and an `A` (IP address) record. Since O-King causes recursive nameservers to seed the cache with *every* authoritative nameserver for a zone, we used the reverse crawl data to find an average of 2.4 nameservers per zone. Thus, O-King would cause the insertion of 48 billion entries into cache for the nameservers used in the measurement. D-King needs a single set of records for each latency estimate, which means that 20 billion entries would be saved in caches on its behalf. Turbo King only requires that the local domain (e.g., `irl-tamu.us`) be cached at each recursive nameserver, which implies merely 200,000 total cache pollution entries. To compare, Turbo King requires 0.0004% of the total entries caused by O-King and 0.001% of those initiated by D-King, clearly making Turbo King much more appropriate for large-scale measurement studies.

2.7. Summary

In this chapter we showed that King, a previous distance estimation method, suffers from non-negligible error when DNS zones employ geographically diverse authoritative servers or utilize forwarders, both of which are very common in the existing Internet. We also showed that King requires insertion of numerous unwanted DNS records in caches of remote servers (which is called cache pollution) and requires large traffic overhead when deployed in large-scale. To overcome these limitations, we pro-

posed the Turbo King latency estimation framework that obtains end-to-end delay samples without bias in the presence of distant authoritative servers and forwarders, while consuming half the bandwidth needed by King and reducing the impact of cache pollution by several orders of magnitude. We also demonstrated through several experiments that Turbo King is more accurate that prior methods.

CHAPTER III

IRLSCANNER

3.1. Introduction

Characterizing visible services in the Internet (e.g., web sites [10], [34], [52], end-hosts [35], [37],[42]), discovering and patching servers with critical security vulnerabilities (e.g., SSH [82], DNS [25]), and understanding how Internet worms create massive botnets [17], [45], [62], [108] are important research topics that directly benefit from efficient and scalable scanning techniques that can quickly discover available services in the Internet.

Our focus in this chapter is on *horizontal scanning* [107], which is a method for enumerating (in some set $\mathcal{S}$) all remote hosts that support a given protocol/service p. This is accomplished by sending packets to destinations in $\mathcal{S}$ and counting positive responses within some time interval. We call a scan *complete* if each address in $\mathcal{S}$ is probed and *partial* otherwise. The latter type of scan significantly reduces the burden on remote networks and is useful when an estimate of the number of responsive hosts (rather than their IP addresses) is sufficient.

While several large-scale measurements have been conducted in the past [10], [25], [35], [83], researchers initially considering a similar project are often faced with delays on the order of months for individual tests to run [10], [83]. During this time, computational resources of potentially dozens [10] of local machines that could be put to other uses are tied up. Further complicating the issue is the possibility of facing a significant number of complaints from hostile network administrators [10], [13], [18], [25], [30], [35], [82], [83], even for partial measurements. Given these issues, questions

arise about the feasibility of service discovery [14], especially on sensitive TCP ports. Evidence suggests that sometimes [35] researchers are even forced to abort planned activities due to the negative publicity generated by scan traffic.

Since previous work has not explicitly aimed to design a high-performance scanner and/or maximize its politeness, there is no standard by which to judge the quality or intrusiveness of a scanner. Our first step then is to propose three main objectives that a good Internet-wide scanning solution must satisfy: 1) efficient usage of resources (i.e., bandwidth, CPU, memory) in complete scans; 2) accuracy of extrapolation in partial scans; and 3) maximum politeness at remote networks. The first objective ensures that the implementation scales well when scan duration T is reduced to hours or even minutes. The second objective delivers a platform for extremely fast partial scans with accurate extrapolation of metrics of interest. The last objective maximally reduces the instantaneous load (i.e., burstiness) applied to target subnets and controls the rate of IDS activity (i.e., false-positive alarms, wasted investigation effort, and dynamic firewall blocks against the scanner network) in response to scan traffic.

We next build a scanner that satisfies these goals and evaluate its performance in real scans.

3.1.1 Our Contributions

The first part of the chapter analyzes the approaches exposed in the literature to understand whether they can be used to optimize performance, politeness, and extrapolation ability of an Internet-wide scanner. In addition to realizing that prior work was not driven by any particular objectives in designing their scanners (besides obtaining the data of interest in some finite amount of time), we also reach the conclusion that there is no consensus on such important parameters as scan scope,

permutation, split among source IPs, timeouts, handling of complaints, and monitoring of the scan's intrusiveness at remote networks.

To overcome this problem, the second part of the chapter presents our design of IRLscanner, which is a high-performance and source-IP scalable framework for service discovery in the Internet. The central element of the scanner is a novel permutation/split algorithm, which we show is optimally polite at each CIDR subnet s as it spaces probes arriving to s equally throughout scan duration $[0, T]$, even with multiple source IPs. Extrapolation results with IRLscanner running at its default rate r (at which it covers the Internet in $T = 24$ hours) demonstrate that partial scans of our approach are unbiased, leading to 1% estimation error in the number of live hosts in just 10 seconds.

Due to the goal of allowing faster scan durations (i.e., minutes/hours) and politeness concerns, IRLscanner incorporates additional features that help it achieve our objectives. These include a significantly reduced scope of measurements (i.e., one billion packets fewer) compared to previous scanners, absence of largely ineffective retransmissions, ability to run with any number of IPs aliased to the same server, capture of all back-scan and bogus traffic (e.g., from hackers and buggy implementations), and significantly higher timeouts for unresponsive targets, which allows it to capture a wider variety of busy/slow hosts in the Internet than was possible before.

Armed with IRLscanner, the third part of the chapter highlights 20 Internet-wide scans across a wide range of protocols and ports, including DNS (port 53), HTTP (port 80), SMTP (port 25), EPMAP (port 135), and UDP ECHO (port 7). In addition to running over 20 times faster than any prior scanner and probing ports that have never been scanned in the literature (i.e., SMTP, ECHO, EPMAP), we experiment with several novel techniques (e.g., ACK scanning) and perform the first large-scale OS fingerprinting study of 44M hosts responding to port 80.

We finish the chapter by analyzing the feedback generated from our experiments. This includes a detailed complaint analysis, techniques for monitoring the impact of scan traffic on IDS activity in the Internet, approaches for a-priori predicting the amount of blowback in response to scanning a particular port, and various ways for reducing the perceived maliciousness of the scan.

3.1.2 Ethical Implications

Over the last 2 years, we have closely worked with university officials and taken numerous steps (see below) in an effort to reduce investigation effort and aggravation for remote administrators. While our interest in this chapter is purely to expose the underlying issues of service discovery and make it more accessible to researchers without damaging remote networks, one concern might be that our scan techniques are not only maximally polite, but also optimally stealthy against popular IDS packages. As a result, one could argue that attackers could benefit from our work and thus inflict certain damage that would not otherwise be possible.

However, we do not believe this to be the case. First, as hackers must constantly remain two steps ahead of the security community to be able to exploit the implemented defenses, our results are not necessarily novel or useful to them. Instead, we believe that the discussion and techniques exposed in this chapter might be useful in building future defenses against scanning worms. Second, stealthier scanning by itself does not compromise hosts; in contrast, intrusion using malicious payload (e.g., delivered through unsolicited packets or email) does. As a result, many networks with patched hosts and up-to-date IDS signatures of various malware should remain well protected despite the findings of this chapter. Third, botnets afford hackers such a diverse pool of IPs that they often do not care to remain stealthy and rely on the most basic sequential scanning [3], which apparently is sufficient for their purposes.

Another concern might be that we are collecting information about remote networks that their administrators do not wish to make public. We contend that such information is well-protected by firewalls, and this chapter makes no attempt to trick or confuse network monitoring devices to gather sensitive data. Further, given the constant background scanning performed by attackers [78], any data (and likely much more) we collect is already available to them. However, to ensure privacy, we do not publicize any information about individual networks collected during our scans and instead rely only on summary statistics.

3.2. Scanner Design

Beyond the measurement-specific choice of the protocol/port pair that uniquely characterize a service, every researcher considering a horizontal scan must answer a common set of questions before proceeding. In this section, we turn to several recent studies [10], [25], [35], [83] that have performed large-scale service discovery to determine whether these design questions have been definitively answered and our objectives met.

3.2.1 Scan Scope

We start with the issue of which IP addresses to target when scanning. Define $\mathcal{F}$ to be the Internet IPv4 address space, which consists of $n = 2^{32}$ addresses available for scanning. While intuition may suggest to probe the entire space to ensure completeness, certain IPs may not be suitable for scanning. Before delving into details, define set $\mathcal{NR} \subseteq \mathcal{F}$ to be all non-reserved destinations [38], $\mathcal{I} \subseteq \mathcal{NR}$ to be all IANA-allocated blocks [39], and $\mathcal{B} \subseteq \mathcal{I}$ to be the set of IPs advertised in BGP prefixes [128] at the border router of the scanner network.

Table III. Large-scale service discovery in the literature (dashes represent unreported values)

Scanner	Scope	Permutation	Servers	Protocol	Port	Timeout	Duration	Blacklist	.0/.255	Exclude
Pryadkin [83]	$\mathcal{I}$	uniform	3	ICMP/TCP	–	10s	123d	yes	no	no
Benoit [10]	$\mathcal{NR}$	uniform	25	TCP	80	30s	92d	no	yes	no
Dagon [25]	$\mathcal{I}$	uniform	–	UDP	53	–	30d	–	yes	US Gov
Heidemann [35]	$\mathcal{I}$	RIS	8	ICMP	echo	5s	52d	yes	no	no

To capture the choice of which destinations to target, we define scan *scope* $\mathcal{S}$ to be the subset of $\mathcal{F}$ probed during measurement. As shown in the second column of Table III, all previous Internet-wide service discovery projects scanned at least the IANA allocated space $\mathcal{I}$, which is justified [35], [65] by churn in BGP routing tables and desire to avoid losing responses during the period of the experiment (i.e., 30+ days in Table III). In [10], however, no reason is given for scanning unallocated space, although there is a slight possibility of new blocks being allocated by IANA during the measurement.

As we discuss later, sets $\mathcal{I}$ and $\mathcal{NR}$ may be appropriate for slow scans; however, faster scanners have little incentive to utilize sets larger than $\mathcal{B}$ in the current Internet since performance concerns (i.e., volume of sent traffic) usually outweigh completeness of scan results.

3.2.2 Scan Order

The next factor we consider is the order in which IP addresses are scanned. This is determined by the *permutation* [108] of space $\mathcal{S}$, which is simply a reordering of target addresses to achieve some desired result. The chosen permutation controls the burstiness of traffic seen by remote networks and is a significant factor in both the perceived politeness of scan traffic and estimation accuracy of Internet-wide metrics from partial scans. For all discussion below, we assume that target subnets s are full CIDR blocks (i.e., given in the $/b$ notation).

The most basic approach, which we call *IP-sequential*, does not shuffle the address space and probes it in numerical order (e.g., 10.0.0.1, 10.0.0.2, 10.0.0.3, etc.). It is not only simple to implement, but also routinely used in the Internet [3], [41], [59] and measurement studies [8], [34]. IP-sequential targets individual subnets s with a burst of $|s|$ consecutive packets at the rate of n/T before moving on to another network.

Besides extremely high sending rates to s regardless of its size (e.g., 37 Kpps for $\mathcal{S} = \mathcal{I}$ and $T = 24$ hours), IP-sequential also suffers from poor extrapolation ability.

The main alternative to IP-sequential is the *uniform* permutation [108] also extensively used in the literature [10], [25], [82], [83]. This approach draws targets uniformly randomly from the full address space $\mathcal{S}$ and intermingles probes to many subnets, which reduces instantaneous load on individual networks and produces unbiased random sampling during partial scans. In the literature, the uniform permutation is usually accomplished by either an LCG (linear congruential generator) [55] or some encryption algorithm applied sequentially to each element of $\mathcal{S}$ (e.g., TEA [82]), both of which ensure that no IP is probed twice.

The final approach proposed in [35] we call *Reverse IP-sequential* (RIS) due to its reversal of bits in the IP-sequential permutation and targeting the same address in each subnet (e.g., *.*.127.10, *.*.8.10, *.*.248.10, etc.) before moving on to another address. Intuition suggests that RIS is poorly suited for extrapolation (which we confirm below), while the uniform permutation fails to deliver packets with maximum spacing to each subnet. Since no analysis exists to further evaluate the differences between these three permutation algorithms, making an informed choice remains an open problem.

3.2.3 Scan Origin

In a bid to obtain multiple vantage points [35] and decrease the time required to complete the scan [10], past measurement studies have often distributed the burden of scanning amongst several hosts. We call this process a *split*, which in the literature parcels blocks of either contiguous [8], [10] or permuted [35], [83] IP addresses to m scanning nodes. Column four of Table III contains values for m used previously, with [35] being the only study that used multiple hosts residing on two different networks

(four at each location).

The current consensus in the literature is that multiple scanning hosts on a single network are necessary only if the full assigned scanning bandwidth cannot be utilized by one host, a condition that is implied in [83] and mentioned explicitly in [8], [10], [35]. However, it is not clear from these studies how many hosts are needed to efficiently utilize a link or provide reasonably short scan durations. Further, the literature does not consider the split's impact on the perceived politeness of the scan, which we tackle later in the chapter.

3.2.4 Extrapolation

In many research applications, especially those that monitor growth of the Internet [35], [42], it is sufficient to obtain the number of live hosts or estimate their characteristics (e.g., mean uptime) rather than a list of their exact IPs. The best approach in such cases is a partial scan, which produces a tiny footprint at remote networks and in many cases allows accurate extrapolation of metrics of interest. This requires that targets within each subnet be randomly selected, without any bias being given to certain parts of $\mathcal{S}$ or particular patterns within probed IP addresses (because the density of live hosts varies both across the Internet and the last $1-2$ bytes of the IP). In addition, non-random probing is often seen by administrators as purposefully malicious, which in turn leads to unnecessary investigation overhead, firewall blocks, and complaints.

3.2.5 Implementation

The next pressing issue of service discovery is the method used to send/receive packets, which significantly impacts the efficiency of the scanner. The easiest implementation method uses scripts that execute pre-written utilities [34], [83] or existing

scanners [8]. An alternative is to write a custom scanner for a particular measurement, which opens the possibility of using connectionless sockets [25], [35] for ICMP or UDP-based scans, connection-oriented TCP sockets [10], and finally raw IP sockets for TCP SYN scans [82], [83]. While there is no consensus in the scanning literature on what method to use, [26] suggests that software limitations on packet sending rates can be overcome using a network subsystem that bypasses the default network stack.

3.2.6 Timeouts and Duration

The next two issues are when to mark a host as unresponsive and what aspects should determine scan duration T. The former issue comes down to two choices: 1) waiting a "safe" amount of time before retransmitting [10], [82], [83]; and 2) when to finally time out and declare targets dead [35]. Note that both incur substantial overhead due to the need to remember all covered destinations and to maintain numerous timers. Furthermore, it is unclear what benefit retransmission carries given the low packet loss on the backbone and whether the increased overhead (i.e., doubling or tripling the number of sent packets) justifies the potentially minuscule accuracy gains.

Table III lists timeout values that range from 5 to 30 seconds for previous measurements studies. Given the limited number of outstanding sockets, finite bandwidth and CPU power, and a wide range of possible choices, it remains unclear how to choose timeouts to simultaneously allow for efficiency and accuracy (e.g., certain busy servers respond with a 60-second delay, but should they be captured by the scanner?).

After settling the above problems, it is important to ensure that the scan will complete in such amount of time T that produces the most relevant data without overburdening local/remote network resources. Of the measurement studies listed in Table III, only [25] was unencumbered by software/hardware restrictions, while for others these issues dominated the choice of T. As such, previous measurement

studies have generally been limited to a tradeoff between small T, few servers m, large timeouts, and large scan scope. Instead, our goal is to develop a scanner in which the researcher can control T independently of all other listed parameters.

3.2.7 Negative Feedback

Due to the unsolicited nature of the packets sent by service discovery measurement studies and the diversity of networks in the Internet, it is inevitable that some targeted hosts will take offense. This often manifests itself in the form of email/phone complaints from network administrators [10], [18], [25], [35], [82], [83], though the literature is lacking in details on the exact nature of complaints (e.g., frequency of legal threats) and specific techniques for dealing with them.

In Table III, we list three of the methods used by previous studies to mitigate complaints. The first is the use of a blacklist by [35], [82], [83] to exclude the networks of sensitive/suspicious administrators, which generally avoids repeated complaints from the same network. The other two approaches boil down to a further reduction in scope by the omission of network/broadcast IP addresses (i.e., *.*.*.0 and *.*.*.255) [35], [83] and preemptively blacklisting networks before they complain (e.g., U.S. government) [25]. While blacklisting complaining parties is undoubtedly a sound approach, no reasoning or motivating factors have been provided for the other two methods and it is unclear whether they are indeed necessary.

3.3. IRLscanner

Based on our analysis of scanning literature in the last section, it appears that researchers interested in service discovery projects are faced with scan durations on the order of months, tying up several machines that could be dedicated to other

projects, and the likelihood of significant negative feedback that could easily lead to the measurement being terminated. For example, [35] reports that TCP scans produce 30 times more complaints than ICMP scans, which has precluded the authors from conducting them after the first attempt.

Assuming a fixed amount of bandwidth available for scanning, in this section we seek to alleviate these concerns by designing a service discovery tool we call IRLscanner that allows for very small scan durations T, originates from a single inexpensive host, and minimizes aggravation of network administrators (both remote and local) by scanning as politely as possible for a given T.

3.3.1 Scan Scope

We start by determining the scope of IRLscanner. While firewalls and routers routinely use the Bogons list [22] to filter nonsensical traffic (i.e., reserved and unallocated blocks), packets destined to unadvertised BGP blocks are also dropped by the scanner's border router, but only after unnecessarily increasing router load and wasting scanner resources. Therefore, one expects that only set $\mathcal{B}$ should normally produce valid results or be used for discovering hosts responsive to unicast traffic. However, given that BGP tables change dramatically in the long-term [65], restricting the scope to only routable addresses either requires a live BGP feed or potentially allows for inaccurate representation of the Internet in the resulting measurement.

While this is definitely a concern for slow scanners (i.e., T is weeks or months), our goal is to complete measurements in much shorter periods (i.e., hours) during which BGP changes can often be neglected. For fast scans, updates pulled from RouteViews [93] at start time sufficiently approximate the routable space during the entire experiment. Our analysis of BGP tables during August 2009 discovered less than 0.1% difference over a 10-day period, with proportionally fewer changes during

Table IV. Scan set size, Ethernet bandwidth, and packets/second in a 24 hour TCP SYN scan

Set $\mathcal{S}$	Size	Reduction	Rate (Mbps)	Rate (Kpps)
$\mathcal{F}$	4.29B	–	33.4	49.7
$\mathcal{NR}$	3.7B	14%	28.8	42.8
$\mathcal{I}$	3.27B	24%	25.4	37.8
$\mathcal{B}$	2.11B	51%	16.4	24.4

$T = 24$ hours used in our experiments. While for IRLscanner it makes sense to only probe $\mathcal{B}$, the tradeoff between scope, duration T, and BGP table accuracy must be determined on a case-by-case basis.

To gauge the potential gains from restricting the scope to routable destinations, we determine [39], [93] the current state of sets $\mathcal{B}$, $\mathcal{NR}$, and $\mathcal{I}$ in late August 2009 and list them in Table IV. While previous scanners achieve a significant reduction (i.e., by 24%) in the number of sent packets by omitting the reserved/unallocated space, probing only set $\mathcal{B}$ removes almost one billion *additional* targets and doubles the performance gains of previous work to 51%. The table also shows the bandwidth necessary to complete the scan in 24 hours, where all 40-byte SYN packets are padded to 84-byte minimum-size Ethernet frames, and the corresponding pps (packets per second) rate.

To implement a scanner with scope $\mathcal{B}$, it is necessary to obtain a timely BGP dump from either the RouteViews project [93] or the local border router. Given the desire for small scan durations on inexpensive hardware, checking individual IP addresses against a list of roughly $300,000$ prefixes must be very efficient. While IP checking can be accomplished with a balanced binary tree [25] with logarithmic lookup complexity, IRLscanner uses a 512 MB hash table, where each bit indicates whether

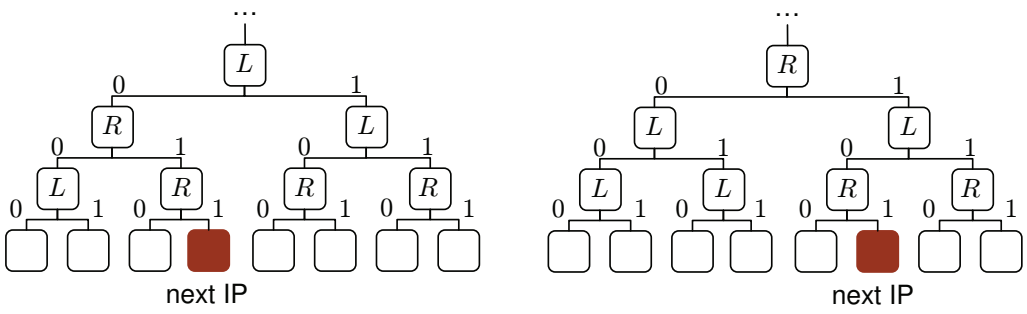
the corresponding IP is allowed or not. This ensures that checks are accomplished in $O(1)$ time and improves lookup speed from 923 Kpps (balanced tree) to 11 Mpps (using a single core of a 2.2 GHz Opteron). Given that most commodity machines have at least 1 GB of RAM and the rest of our scanner requires only 2 MB of main memory, this tradeoff allows us to dedicate more computational power to sending packets and performing other processing as needed.

3.3.2 Scan Order

Despite the constant volume of scanning traffic in the Internet [78], network administrators generally view this activity as malicious and periodically complain to networks that originate such traffic [25], [35] [82]. Furthermore, many IDS tools [12], [105], [125] automatically generate firewall rules against scanning hosts, whether detected locally or through distributed collaborative systems [71], [94]. With this perception in mind, researchers must first weigh the benefit gained from performing a service discovery measurement with the possibility of negative publicity for their institution and/or its address space being blacklisted at remote networks.

Upon determining to proceed, these negative effects can be reduced for all involved parties by using an address permutation that avoids targeting individual subnets with large bursts of traffic, which often triggers Intrusion Detection Systems (IDS) and raises concerns of malicious/illegal activity. Since IDS predominantly operates on a per-IP basis, additional reduction in false-alarms is possible by using multiple source IPs at the scanner host, which we discuss later in this section. While the uniform permutation [108] is routinely used in scanning applications, no previous paper has examined the issue of achieving maximal politeness and whether such methods could be implemented in practice. We address this open problem next.

For a given CIDR subnet s in the Internet, our goal is to maximally reduce the



(a) IP ends with 011 (b) IP ends with 101

Fig. 12. Illustration of AGT.

burstiness of scan traffic seen by s, which is equivalent to maximizing inter-packet delays for *all* probes arriving to s. Recalling that $n = 2^{32}$, we define permutations that return to s with a period $n/|s|$ to be *IP-wide* at s and those that achieve this simultaneously for all possible CIDR subnets to be *globally IP-wide* (GIW). Note that GIW permutations spread probes to each s evenly throughout $[0, T]$, which ensures that all networks are probed at a constant rate $|s|/T$ proportional to their size and that no s can be scanned slower for a given value of T. This makes GIW optimally polite[1] across the entire Internet.

The simplest GIW technique, which we call an *alternating gateway tree* (AGT), is a binary tree of depth 32 where target IPs reside in leaves and all edges are labeled with 0/1 bits. Traversing the tree, the scanner accumulates individual bits along the edges into the next IP. Decisions to move left or right at internal nodes (gateways) v depend on their states θ_v, which are flipped during each visit to ensure that no IP is probed twice and that packets alternate between left/right children of each gateway. Fig. 12 shows the bottom four levels of some random AGT, where the tree in part (a) generates an IP address ending with bits 011. Part (b) of the figure illustrates the

[1]While completely refraining from scanning is even more polite, it does not produce any useful service-discovery results.

next IP produced by this portion of the AGT, which results in the address ending with bits 101.

Since balanced binary trees have well-defined rules for calculating the offset of each internal node, AGTs do not require storing child pointers. Thus, their RAM overhead is $(n-1)/8 = 512$ MB needed to store tuple $(\theta_1, \ldots, \theta_{n-1})$ and their computational complexity is 26 memory reads/writes (i.e., 52 total) per generated IP (assuming depth-31 traversal and 64-bit lookups that yield the first 5 levels of the tree in one RAM access).

Note that AGT provides the scanner with 2^{n-1} possible GIW permutations, which is enormous. In practice, one does not require this much diversity and other GIW algorithms may be sufficient. One reason to seek alternatives is that AGT requires saving 512 MB during checkpointing and transmission of the same amount of seed data to other scanner hosts in distributed implementations. Another reason is that AGT's CPU complexity is quite high, which we reduce in our next method.

Assume that s has depth b in the AGT (i.e., $n/|s| = 2^b$) and observe that GIW patterns must visit all remaining $2^b - 1$ subnets at depth b before returning to s. In practice, this means that the permutation must exhibit a full period in the upper b bits. Since for GIW this holds for all s, the full period must be simultaneously maintained at all depths $1 \leq b \leq 32$. Reversing the bits in each IP, we can replace this condition with a much simpler one – the full period must hold in the lower b bits. Define ${}_b x$ to be the lower b bits of an integer x and $\mathcal{R}(x)$ to be the bit-reversal function. Then, we have the following result.

Theorem 1. *Given a sequence of integers $\{x_k\}_{k=1}^n$, suppose sequence $\{{}_b(x_k)\}_{k=1}^n$ has a full period for all $b = 1, 2, \ldots, 32$. Then, sequence $\{\mathcal{R}({}_{32}x_k)\}_k$ is GIW.*

While there are many possible ways to construct $\{x_k\}_k$, an LCG of the form

Table V. Benchmark of GIW address generation

Type	Bit reversal	Rate (IP/sec)	State & seed
AGT	–	661,247	512 MB
LCG	Bit shifts	10,729,920	4 bytes
	Two-byte hash	21,263,889	4 bytes

$x_k = ax_{k-1} + c$ is a natural choice due to its computational efficiency and need for only a single integer of state. To establish its suitability for Theorem 1, we note that the conditions for achieving a full period in $\{x_k\}_k$ with an LCG are well-known and require that $a - 1$ be divisible by 4 and c be odd [47]. We call the resulting algorithm *Reversed LCG* (RLCG) and use it with $a = 214,013, c = 2,531,011$, which are well-known constants that produce an uncorrelated sequence of random variables. The random initial seed x_0 can then be used to change the scan order across multiple runs.

To efficiently reverse the bits, we use a 2-byte hash table that flips the order of bits in 16-bit integers. Therefore, any 32-bit IP can be processed in two memory lookups (i.e., 26 times faster than AGT); however, the CPU cache often makes this operation run even faster in practice. Table V benchmarks IP generation of AGT, naive bit-shifts (32 shifts down and 32 up), and the hash-table technique. Observe that RLCG with a hash-table runs at double the speed of bit-shifts and beats AGT by a factor of 32, which is slightly faster than 26 predicted by the analysis above.

3.3.3 Scan Origin

While previous work has split the scan burden among m nodes to decrease total duration [8], [10], [35], [83] or obtain multiple vantage points [35], no apparent consideration has been given to the possible effect it has on the perceived politeness of the

measurement. The main objective of a polite split in this chapter is to maintain the GIW pattern across scanner IPs, which requires a mechanism for not only parceling the address space to m scanning hosts without burdensome message-passing, but also ensuring that each subnet s sees scanner IPs in a *perfectly alternating and equally-spaced fashion* (e.g., IP1, IP2, IP3, IP1, IP2, IP3, ...).

The rationale for using all m sources to scan each s lies in the fact that IDS (both open-source [12], [105] and commercial [44], [74]) detect scan traffic and throw alarms in response to perceived malicious activity based on individual source IPs. Therefore, a particular IP address sending its packets to s faster than other IPs is more readily detected as it simply stands out from the others. The reason for maximally spacing probes from different IPs is the same as before – reducing the overall burstiness at remote subnets – which for large m (i.e., hundreds or thousands) may become non-trivial. One example of an extremely impolite split is IP-sequential, which scans each s from a *single* source IP at rates similar to those in Table IV (i.e., megabits per second and thousands of pps), regardless of subnet size.

Analysis shows that GIW split does not require a new permutation; however, individual source IPs must now return to s every $mn/|s|$ packets (i.e., alternating in some order with a full period m). Synchronizing m hosts using the block-split algorithms of previous work [8], [10], [35], [83], while sustaining the GIW split is a difficult problem. We instead introduce a new split algorithm that satisfies our conditions and requires low overhead/state.

The intuition behind our split, which we call *round-robin* (RR), is to generate a single RLCG permutation $\{z_k\}$ and assign target z_k to host k mod m. Assuming $\mathcal{M}$ is the set of scanning hosts, RR sends the initial seed x_0 to every host $i \in \mathcal{M}$, its position i, and the number of sources m. Each host then generates the entire sequence $\{z_k\}_k$ locally and hits target z_{i+jm} at time $(i+jm)T/n$ for $j = 0, 1, \ldots, n/m$, the simplicity

Algorithm 1 RLCG/RR at host $i \in \mathcal{M}$

```
1: x_0 = rand()                                   ▷ Set initial seed x_0
2: for k = 1 to n do                              ▷ Iterate through all IPs
3:     ip = k mod m                               ▷ Assigned source IP
4:     x_k = a x_{k-1} + c                        ▷ Advance LCG
5:     if (ip == i) then                          ▷ Our IP?
6:         target = R(x_k)                        ▷ Reserve bits
7:         if (BGP[target]==VALID) then           ▷ In BGP?
8:             probe(target)                      ▷ Hit destination
9:         end if
10:    end if
11:    sleep(T/n)                                 ▷ Wait for next packet
12: end for
```

of which is demonstrated in Algorithm 1. Even with $T = 24$ hours, subnets are visited so infrequently (e.g., every 337 seconds for a /24) that perfect synchronization of start times is not necessary. Furthermore, in scanners running from a single location, all m IPs can be aliased to the same host and RR-split can be used locally to ensure perfect synchronization, which is the approach taken by IRLscanner later in the chapter.

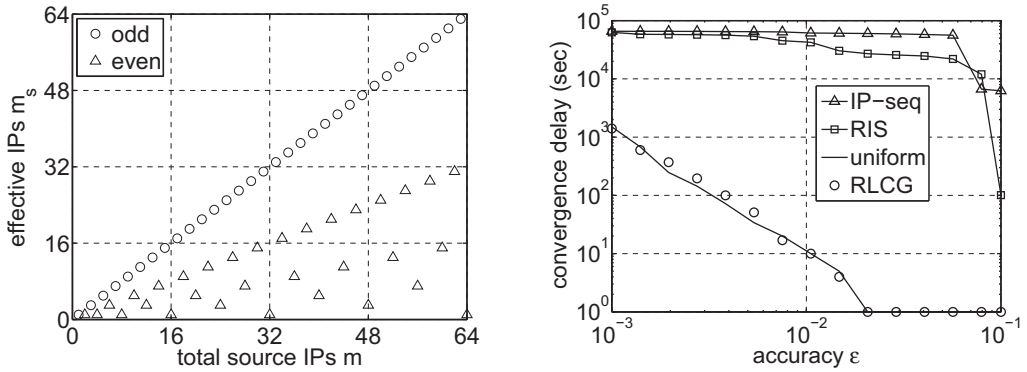
From the well-known properties of LCGs [9], we immediately obtain the following crucial result.

Theorem 2. *RR-split with any GIW permutation scans s with* $\min(|s|, m_s)$ *sources, where*

$$m_s = \frac{m}{\gcd(\frac{n}{|s|}, m)} \tag{1}$$

and $\gcd(a, b)$ *is the greatest common divisor of* (a, b).

To better understand this result, examine Fig. 13(a) that shows one example of (1) for $|s| = 65536$ (i.e., a /16 target subnet). Notice that even values of m lead to $m_s \leq m/2$ (triangles in the figure), which reduces the effective number of IPs seen by each subnet at least by half. The worst choice of m is a power of two, in which case $m_s = 1$ regardless of m. On the other hand, odd values of m produce the ideal



(a) m_s in (1) (b) extrapolation delay

Fig. 13. GIW split and extrapolation delay.

$m_s = m$ (circles in the figure) and thus achieve a GIW split. We rely on this fact later in Section 3.4..

3.3.4 Extrapolation

Given the goal of being able to extrapolate the number of responsive hosts and other properties of open ports from a severely abbreviated scan (e.g., $1-10$ seconds instead of 24 hours), we next examine how the existing and proposed approaches handle this problem. We split the allocated IANA space into three blocks (i.e., ARIN, RIPE, and APNIC), roughly corresponding to different geographical zones, and build three distributions of live IPs from our Internet measurements. Specifically, PMF function $p_j(x_3, x_4)$ specifies the probability that IP $x_1.x_2.x_3.x_4$ is alive in geographical zone $j \in \{1, 2, 3\}$. We then generate a Bernoulli random variable for each IP in the IANA space and make it alive using the corresponding probability $p_j(x_3, x_4)$.

Using a simulation with $T = 24$ hours, we scan the assigned distribution of live/dead hosts using four approaches – uniform, RLCG, IP-sequential, and RIS. Assuming A is the true number of live hosts in the assignment and $\tilde{A}(t)$ is an estimate at time t, define the relative extrapolation error $e(t) = |1 - \tilde{A}(t)/A|$. Convergence

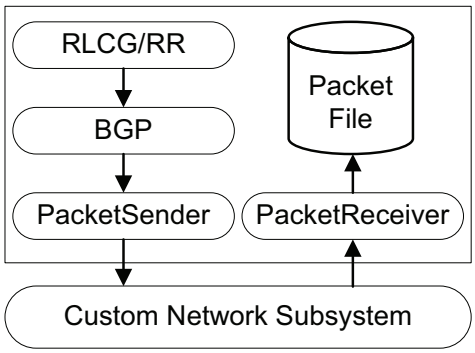


Fig. 14. IRLscanner implementation.

to threshold ϵ is established at such time t_ϵ when estimates for all $t \geq t_\epsilon$ have error smaller than ϵ.

Fig. 13(b) plots the expected convergence delay t_ϵ averaged over 100 iterations. Observe that both RLCG and uniform converge to 1% error in 10 seconds, while RIS and IP-sequential take 11 and 16 hours, respectively. This result is easy to explain since IP-sequential gets trapped in certain CIDR blocks for an extended period and RIS hits the same last octet 16M times in a row. Furthermore, 0.1% error in Fig. 13(b) can be achieved in 23 minutes for both uniform and RLCG, while the other two methods require 17+ hours. Even to arrive at 5% accuracy, which takes RLCG less than a second, RIS requires 6 hours, which makes this method unsuitable for all but most crude extrapolations.

3.3.5 Implementation

Fig. 14 shows the general structure of IRLscanner. IPs generated by RLCG/RR are first checked against BGP prefixes and then delivered to the sending module, which forms raw Ethernet frames and transmits them to a custom network driver (detailed in [103]), which is capable of saturating a gigabit link with SYN packets (1.4Mpps) from one Intel Pro/1000 NIC on a 2.2 GHz AMD Opteron system using only 60%

of a single core. Through this subsystem, we are also able to intercept arbitrary incoming/outgoing packets and suppress RSTs from the OS TCP/IP stack, which we make use of later when profiling remote operating systems. All received packets are saved to disk without inspection and are processed offline. After completing each scan, this framework continues to listen for incoming packets for several hours to capture any extremely slow hosts, as well as record any back-scanning packets from hackers and other potentially interesting entities.

3.3.6 Timeouts and Duration

Previous measurement studies [10], [82], [83] used retransmissions to the unresponsive set of target hosts to minimize false negatives, which we now evaluate in light of politeness and efficiency. Cursory inspection shows that retransmitting probes to unresponsive hosts is the violation of the GIW pattern, which is undesirable. Combining this with the likelihood that many false negatives in the unresponsive set are likely to be from persistently congested links or over-burdened servers [2] with potentially sensitive network administrators, politeness concerns suggest that retransmission is not generally advisable.

From an efficiency standpoint, it should also be noted that the unresponsive set accounts for $90 - 99\%$ of $\mathcal{S}$ (depending on the protocol), which means that a single timeout-based retransmission would require almost doubling the number of sent packets. Our experiments show that retransmission not only yields a negligible increase in found hosts (i.e., by $0.3-1.7\%$ depending on the port and time of day), but also introduces bias by capturing hosts that come online within the retransmission timeout.

We next turn to the issue of *when* the status of an IP address can be determined, which in related work [10], [35], [82], [83] has occurred at some timeout after the initial

probe was sent. Considerable effort has been spent deciding on appropriate timeout values [10], the choice of which affects the number of false negatives due to slowly responding hosts and the overhead of keeping large amounts of state for outstanding targets. Given that retransmissions are not required, we avoid this tradeoff entirely by delaying the classification of IP addresses until *after* the scan completes.

In practice, we accomplish this by saving all packets incoming to our scanning IP addresses to disk for later analysis. As there are many packets that are not relevant to the scan, we note that certain information can be embedded in the packets themselves to correlate responses with hosts scanned. This option has been used by encoding the target IP address in ICMP ID fields [35] and DNS queries [25]. For TCP scans, we take advantage of the sequence number field to encode the target IP, which allows us to detect invalid and/or malicious replies. While this approach does raise concerns about hard-disk space and I/O speed, in our experience the 25 GB required would not be a factor for even very short scan durations (e.g., given 100 MB/s write speed of modern drives, this volume of data requires a meager 250 seconds of disk I/O).

3.3.7 Negative Feedback

Throughout this section, we have explored and implemented several techniques to reduce the sending rate (i.e., BGP scope reduction), minimize the burden on remote networks (i.e., GIW), lower IDS false-alarm rates (i.e., RR-split), and avoid probing busy servers and non-existent/firewalled hosts with repeat packets (i.e., no retransmission).

In addition to technical solutions outlined above, a political strategy for reducing complaints and dealing with their aftermath is beneficial. Our general approach in this pursuit is to make the non-malicious purpose of our scans as transparent as possible to those remotely investigating our traffic. This includes providing scanning IPs

with descriptive names (i.e., indicating their research purpose) in the forward/reverse DNS tree, as well as creation of TXT DNS records pointing to the project web-page with instructions on how to opt out. With over 123 IPs participating in this endeavor, special scripts have been written to manipulate IP assignment to various NIC interfaces and modify DNS records in our authoritative server.

However, the most widely-used means of investigation is through a whois lookup on offending IP addresses, followed by a direct email to the party listed therein. In the event a complaint is received, our policy is to reply as quickly as possible with an explanation of our traffic, a link to the project web-page, and an offer to blacklist the network. Dynamic blacklisting in IRLscanner is implemented through periodic reading of a flat file of blocked networks and simply removing them from the BGP hash table. Under the assumption that network administrators who complain will do so again later, blacklisted networks are maintained across scans. However, given that no analysis was provided in prior work [25], [35], [83] to justify preemptively removing subnets or addresses, our initial scan started with an empty blacklist.

The final issue one must also be aware of is that significant care should be taken to avoid negatively impacting the *local* network, where internal stateful firewalls and IDS are particularly vulnerable (from the load perspective) to large volumes of traffic destined to billions of unique destinations. We have experienced a number of issues with department and campus-wide IDS/firewall installations at our institution, which all had to be manually bypassed for this project to proceed.

3.4. Experiments

In this section, we test our design decisions by performing several Internet-wide scans using our high performance kernel-level network architecture [103] and present several

novel scan methods. We defer in-depth analysis of the actual scan data to a later paper, instead focusing on high-level observations and results.

3.4.1 Overview

As the goal of scanning is to produce the set of hosts offering a given service, each targeted IP address must eventually be classified into one of four categories. Define *open* set $\mathcal{O}$ to contain all hosts that responded positively to a scan packet (e.g., a SYN-ACK to a TCP SYN), *closed* set $\mathcal{C}$ to represent IPs responding negatively using the same protocol (e.g., a TCP RST to a SYN packet), *unreachable* set $\mathcal{U}$ to consist of IPs that return ICMP unreachable or TTL expired errors, and *dead* set $\mathcal{D}$ to designate hosts from which no reply was received at all. Note that excluding bogus responses and strange firewall/NAT behavior, $\mathcal{O} \cup \mathcal{C} \cup \mathcal{U} \cup \mathcal{D} = \mathcal{S}$ and the individual sets do not overlap.

Through development of IRLscanner and in the course of other projects, we have performed 20 Internet-wide scans since February 2008. To test a wide range of possibilities and demonstrate the general feasibility of service discovery, we targeted UDP, TCP, and ICMP protocols on both popular services (e.g., HTTP, DNS) and those often used for nefarious purposes (e.g., SMTP, EPMAP). Table VI summarizes our scanning activity. We initially started slowly with a 30-day scan duration from a single source IP to gauge the feedback, then increased the sending rate over subsequent scans until we achieved a duration of 24 hours, which is over 20 times faster than any documented scan of which we are aware [35]. The number of source IPs m varied based on their availability in our subnet and specific goals of the measurement, generally ranging from 31 to 123. In comparison, the highest IP diversity in related work was $m = 25$ in [10], followed by $m = 8$ in [35].

Table VI. Summary of scans performed

Name	Proto	Port	Type	Date	T	m	$\lvert\mathcal{O}\rvert$	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{U}\rvert$	pps	Mbps
DNS_1	UDP	53	DNS A	2-21-08	30d	1	15.2M	–	148M	709	0.48
DNS_2		53	DNS A	3-25-08	6d	5	15.2M	–	155M	3.5K	2.38
DNS_3		53	DNS A	5-07-08	1d	31	14.7M	–	168M	21.2K	14.28
DNS_4		53	DNS A	5-19-08	1d	31	14.5M	–	169M	21.2K	14.28
DNS_5		53	DNS A	5-20-08	1d	31	14.6M	–	168M	21.2K	14.28
DNS_6		53	DNS A	5-21-08	1d	31	14.5M	–	167M	21.2K	14.28
DNS_7		53	DNS A	5-22-08	1d	31	14.5M	–	169M	21.2K	14.28
ECHO		7	–	7-01-08	1d	31	322K	–	170M	22.1K	21.03
PING	ICMP	–	echo	6-24-08	1d	31	139M	–	99M	22.1K	14.85
$SMTP_S$	TCP	25	SYN	7-30-08	2d	61	17M	87.1M	119M	11.2K	7.55
$SMTP_A$		25	ACK	7-30-08	2d	61	–	116M		11.2K	7.55
$EPMAP_S$		135	SYN	8-05-08	2d	61	4.9M	40.2M	127M	11.3K	7.58
$EPMAP_A$		135	ACK	8-05-08	2d	61	–	68.4M		11.3K	7.58
$HTTP_1$		80	SYN	7-17-08	1d	123	30.3M	49.1M	78M	22.6K	15.19
$HTTP_2$		80	SYN	8-05-09	1d	61	44.3M	61.3M	97.1M	24.4K	16.39
$HTTP_3$		80	SYN	8-06-09	1d	61	44.0M	61.2M	85.1M	24.2K	16.26
$HTTP_4$		80	SYN	8-10-09	1d	123	44.2M	61.5M	94.7M	24.4K	16.39
$HTTP_5$		80	SYN	8-24-09	2d	123	44.5M	61.7M	96.4M	12.1K	8.15
$HTTP_6$		80	SYN	8-27-09	1d	61	44.1M	61.4M	80.7M	24.4K	16.37
$HTTP_{AS}$		80	ACK→SYN	9-02-09	1d	61	31.7M	49.6M	92M	25.8K	17.35

3.4.2 UDP/ICMP Scans

We started with seven DNS scans due to an interest in public recursive DNS servers. These scans produced between 14.5M and 15.2M responses in each run, which represents a 30% growth from the 10.5M found in [25] less than 9 months prior. We discovered a stable set of 4.4M servers that responded to every DNS scan over a period of three months, which indicates that the number of consistently available hosts is far fewer than might be expected from the responses to a single scan.

Of further interest is the reduction in found hosts from 15.2M to 14.7M when scan duration reduced to 24 hours in DNS_3. This suggests that faster scan durations produce a lower cumulative response among the targets, which in part may be attributed to the lower possibility of counting the same host multiple times under different DHCP'ed IPs. To investigate whether previous scanning activity in some immediate past influences the response rate in subsequent scans, we probed DNS on four consecutive days in May 2008 (i.e., 96 hours of continuous scanning) and received roughly the same number of responses in each case, which indicates that the Internet is basically memoryless (at least at our scan rates).

Our last UDP scan was on ECHO port 7, which simply replies with a verbatim copy of the received packet and to our knowledge has never been scanned in the literature. We chose this port as a representative of a sensitive UDP service largely because of its notoriety for broadcast amplification attacks [63]. Later in the chapter, we deal with the huge volume of complaints and speculation that ensued in the cooperative intrusion detection community, but note that even though best practice is to disable this service, we nevertheless received replies from $321,675$ unique IP addresses.

Our lone ICMP scan was a simple echo request [35], [83] that garnered 139M

replies, representing a 20% gain over a similar scan performed in June 2007 [35].

3.4.3 TCP Scans

Our measurements targeted TCP with 11 scans on 3 different ports and two combinations of SYN/ACK flags. To our knowledge, TCP has not been scanned in the literature with T less than three months [10] or with flags set other than SYN [10], [35], [82], [83].

We start by describing the performed SYN scans in an increasing order of their sensitivity. We initially scanned HTTP with a duration more than 90 times shorter than the only previous attempt [10], discovering 30.3M hosts in July 2008 and 44.5M in August 2009, the latter of which is a 140% increase compared to 18.5M IPs found in 2006 [10]. The other two services we targeted with SYN scans were SMTP, which is frequently probed by spammers searching for open relays, and EPMAP, which is heavily scanned for network reconnaissance prior to attack [63], discovering 17M and 4.9M hosts respectively. Given the large number of Windows hosts in the Internet, the EPMAP result seems low, which suggests that many ISPs drop traffic on port 135.

To determine the feasibility of scanning with other types of TCP packets, we performed three measurements with ACK packets (i.e., SMTP_A, EPMAP_A, and HTTP_{AS}), which can be used not only to determine a host's liveness (i.e., an ACK normally elicits a RST from non-firewalled hosts), but also to bypass stateless firewalls. Both SMTP_A and EPMAP_A were executed *concurrently* with the corresponding SYN scan (i.e., two packets were sent to each IP) in order to allow us to detect and characterize firewalls (detailed analysis of results is outside the scope of this chapter). While SMTP_A returned 116M active hosts, EPMAP_A produced only 68M responses, which suggests that filtering is heavily applied on port 135 not only for SYN packets, but

Table VII. Top 5 devices

Device	Found	%
Linux (2.4 or 2.6 kernel)	13.0M	32.9
Windows XP/Server 2003	6.3M	15.8
Windows Vista/7/Server 2008	5.6M	14.0
Windows Server 2003 SP2	3.5M	8.9
FreeBSD	1.5M	3.8

for ACKs as well. For HTTP_{AS}, we scanned the entire BGP space with ACK packets, then immediately followed the resulting RST responses with a SYN packet. We present our motivation and the results from this previously undocumented approach in a later section.

3.4.4 Remote OS Fingerprinting

While service discovery projects usually focus on enumerating open set $\mathcal{O}$, further information about the hosts themselves is often critical to the depth and usefulness of measurement studies [10], [25]. With the goals of resource efficiency and maximal politeness at remote networks, in this section we focus on determining the operating system of the remote hosts in $\mathcal{O}$, which could be used to estimate the global impact of known security vulnerabilities [68], approximate Internet-wide market share [73], or track hosts with dynamic IP addresses [25]. The main difficulty in executing such a study is that most existing tools [77], [122] not only trip IDS alarms and crash older end-hosts with unusual combinations of TCP/IP flags, but also require substantial overhead (e.g., 16 packets for Nmap) in Internet-wide use [104], [115]. It is thus not surprising that large-scale OS profiling has not been attempted in the literature.

Instead of traditional fingerprinting methods, we utilize a single-packet technique

called Snacktime [104], which exploits OS-dependent features in SYN-ACKs such as the TCP window, IP time-to-live, and most importantly the length and number of retransmissions of the SYN-ACK during TCP handshakes. While initial results on accuracy were promising [104], [115], the non-trivial requirement that outgoing TCP RST packets be dropped, long period needed to produce an answer (e.g., several minutes), and limited database (i.e., 25 signatures last updated in 2003) has previously restricted its usefulness. Further work that is outside the scope of this chapter is required to rigorously confirm its accuracy in the Internet, but given that we must already send a TCP SYN packet to every host in $\mathcal{O}$, modifying the Snacktime technique for use on an Internet-wide scale would result in *no additional sent packets* to enumerate remote OSes.

To implement a scalable Snacktime, we take advantage of our custom network driver to block outgoing TCP RST packets. Since IRLscanner already captures all retransmitted TCP SYN-ACK packets, it is the perfect platform for massively parallelizing the Snacktime technique. After a scan completes, we generate the retransmission delays from the packet dump, then run a custom implementation of the Snacktime matching algorithm that gives preference to general classes of operating system in the case of ambiguity and reduces the microsecond precision of retransmission delays to manage random queuing delays in the Internet. To make the technique more useful, we processed almost 7K responsive hosts at a large university to manually verify and increase the database to more than 100 signatures, including the latest Windows versions (e.g., Vista, 7, Server 2008, Server 2003 SP2), webcams, switches, printers, and various other devices.

We applied the modified Snacktime technique to HTTP_2, which consisted of $|\mathcal{O}| = 44.3\text{M}$ hosts that responded with at least one SYN-ACK. We successfully fingerprinted 39.6M hosts, with 2.3M being excluded due to insufficient retransmis-

Table VIII. Summary of fingerprinted devices

Device Type	Found	%
General purpose	32.4M	81.8
Network device	2.7M	6.8
Printer	1.8M	4.6
Networked storage	1.5M	3.7
Media	929K	2.3
Other embedded	287K	0.7
Total	39.6M	

sions (i.e., none) and the remaining difference attributable to gaps in our signature database. The top 5 profiled OSes are given in Table VII, with Linux contributing 32.9% of the total and various Windows implementations consisting of the next several slots, which is indicative of their co-dominance in the web-server market. We provide more detail in Table VIII, where we classified each signature into one of six categories and calculated summary statistics. Note that general purpose (e.g., Linux, Windows) systems consist of nearly 82% of the total, with network devices (e.g., switches, routers, NAT boxes), networked storage (e.g., NAS, tape drives), and printers consisting of more than 1M devices each. The media category is comprised mainly of webcams and presentation devices (e.g., TVs, DVRs, projectors).

To finish this section, we present in Table IX the total number of devices and their percentage attributed to each class of OS in the general-purpose category, a result that to our knowledge has not previously been shown in the literature on an Internet-wide scale. Approximately half of the total consists of Microsoft OSes (5.6% of which belong to Windows 2000 or older), which is likely due at least partially to individuals hosting personal web-sites on their home machines. Linux hosts are

Table IX. General purpose (GP) devices

OS Class	Found	% of GP
Windows	16.3M	50.2
Linux	13.0M	40.2
BSD/Unix	2.2M	6.7
Mac	862K	2.7

responsible for 40%, which combined with the various related forms of BSD (e.g., OpenBSD, FreeBSD), SunOS, and Unix results in nearly 47% of the total and rivals Microsoft.

3.4.5 Service Lifetime

Another interesting property of Internet services is their average *lifetime* (uptime) $E[L]$, which is the mean duration of time a port stays active on a given IP. One technique [35] is to first estimate the CDF of lifetime L and then compute its mean $E[L]$. However, avoiding round-off errors and CDF tail cut-off often requires monitoring the pool of target IPs at frequent intervals (i.e., minutes) and for extended periods of time (i.e., days), all which contributes not only to higher bandwidth overhead, but also to more likely aggravation of remote network administrators.

We offer an alternative method that can estimate $E[L]$ using much lower overhead and overall delay. Modeling each host as an alternating ON/OFF process [121], a set $\mathcal{K}$ of uniformly selected live hosts exhibits a departure rate $\lambda = |\mathcal{K}|/E[L]$ hosts/sec (a similar result follows from Little's Theorem). Thus, by probing $\mathcal{K}$ twice at time t and $t + \Delta$, one can estimate λ as $p(\Delta)|\mathcal{K}|/\Delta$, where $p(\Delta)$ is the fraction of hosts that have disappeared in this interval. Solving $p(\Delta)|\mathcal{K}|/\Delta = |\mathcal{K}|/E[L]$, we obtain $E[L] = \Delta/p(\Delta)$.

The key to this technique is to uniformly randomly select $\mathcal{K}$ and simultaneously ensure maximal politeness of the scan. Leveraging the findings of Section 3.3.4 aimed exactly at this issue, we first use RLCG to scan the Internet for Δ time units at some constant rate r. We then re-generate the same sequence of IPs at the same rate, but actually send packets only to those targets that have responded in the first scan. Due to limited space, we omit simulations confirming the accuracy of this method and discuss only one extrapolation using port 80 and $\Delta = 45$ seconds. This experiment covered 1M targets, found $|\mathcal{K}| = 23.7$K live hosts, and yielded $E[L] = 50$ minutes (i.e., $p(\Delta) = 1.5\%$).

3.5. Analysis

While it would be ideal to scan the Internet using different techniques (e.g., IP-sequential, uniform, GIW) and then assess the collected feedback as a measure of intrusiveness of each scan, certain practical limitations typically prevent one from doing so (e.g., our network administrators have explicitly prohibited scanning activity using certain non-optimal permutations). Thus, comparison is often only possible through feedback analysis exposed in publications, which unfortunately is very scarce in the existing literature. To overcome this limitation, this section introduces a number of novel metrics related to the perceived intrusiveness of Internet-wide scans, studies them in detail, and unveils certain simple, yet effective, techniques for reducing the blowback.

3.5.1 Email Complaints

One of the uncertainties we encountered when initially considering a service discovery project was the number of complaints to expect, particularly as they related to

Table X. Emails and IPs excluded by service

Service	Scans	Emails	Avg	IPs excluded	Avg
DNS	7	45	6.4	3.7M	530K
Echo	1	22	22	752K	752K
Ping	1	4	4	1K	1K
HTTP	7	24	3.4	459K	66K
SMTP	2	6	3	262K	131K
EPMAP	2	2	1	65K	32K
Total	20	103	5.15	5.3M	263K

serious threats or resulted in widespread blacklisting of the scanner to the point of making Internet-wide measurements impossible. In this section, we attempt to clarify the issue by detailing the complaints we received and the effect they had on our measurements.

Table X contains a summary of email complaints broken down by service type. Over all 20 scans, we received 103 complaints for an average of 5.15 per scan. Our initial run (i.e., DNS_1) resulted in 10 complaints and more than 2.5M IP addresses blocked, which is nearly half the total of 5.3M blacklisted over the course of the project. Most of this initial number came from a single large ISP asking us to block several /16 residential networks. However, even with the initial burst removed from the calculation, DNS scans resulted in an average of 172K blacklisted IPs per scan. The most significant backlash we received was for the ECHO scan (UDP port 7), which led to 22 complaints and more than 750K blocked IP addresses. In the next section we provide an explanation for this significant increase, but note here that UDP scans account for 65% of all complaints, while being responsible for only 40% of the packets sent.

Table XI. Email notices by complainant type

Source	Cease		FYI		Total
	Human	Script	Human	Script	
Individual	14	13	7	8	42
Government	6	2	6	2	16
Corporation	11	5	7	0	23
University	5	3	10	4	22
Total	36	23	30	14	103

In contrast to the experience of [35], where the authors received 30 times more complaints for a TCP scan than ICMP pings, our TCP measurements produced a *total* of 32 complaints over 11 scans, or about three per scan. This is an even more remarkable result given that we scanned two sensitive ports, used ACK packets that penetrate stateless firewalls, and clustered six scans in less than a month. While we cannot explain this discrepancy, our numbers do not support the notion that TCP scans are more invasive than the other protocols.

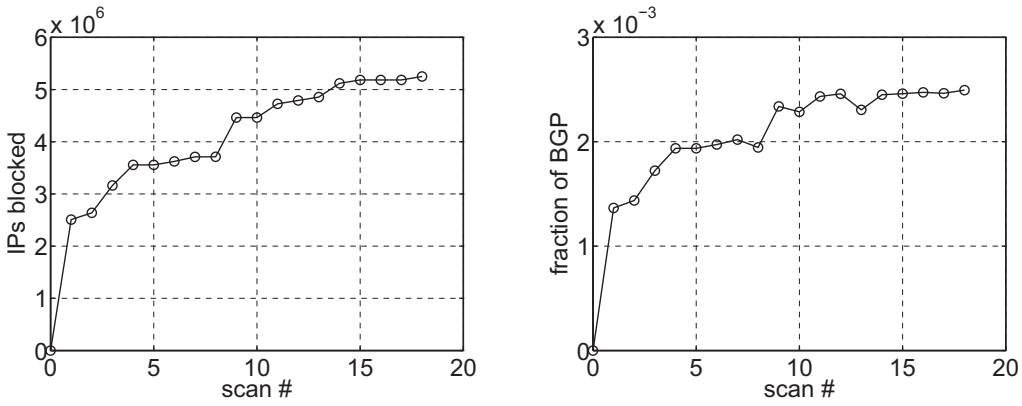
We next categorize the received emails in Table XI to show the severity and type of each complaint. Out of 103 complaints, 59 were demands to cease the activity, while the other 44 were FYI notifications about a possible virus with no expectation that the measurement stop. The first row of the table shows that individual users who monitor a single IP address with a personal firewall (e.g., ZoneAlarm, Norton) represented 41% of the total complaints (i.e., 42 out of 103), which indicates that a large portion of these emails cannot be avoided by any means. The remaining three rows of the table represent complaints received from large network entities, with universities being the most likely to send an FYI notification and worldwide government entities comprising only 16% of the total complaints.

In contrast to [25], we received only four cease demands from U.S. Federal Government entities, none of which were defense-related. Another point of interest is the number of threats to pursue legal action, though of the three received none of them turned out to be legitimate. Finally, analysis of emails generated by an automated script suggests that a large chunk of all received complaints (i.e., 36% in our case) are seldom reviewed by an actual human given the large amount of background scan traffic their networks receive [78].

We now determine the impact of email complaints on the scope of subsequent measurements (i.e., size of $\mathcal{S}$ after removing blacklisted networks) by studying the progression of blacklisted IP addresses in Fig. 15, where scans are assigned numbers in chronological order. Note that the complaints for the two simultaneous scans (i.e., SMTP and EPMAP) are encompassed in a single data point due to our inability to tell whether the SYN or the ACK portion caused the complaint. Part (a) of the figure contains the raw number of blacklisted addresses, which did not increase significantly after we stopped scanning UDP. Part (b) shows the blocked addresses as a percentage of BGP, where the total number of 5.3M represents only 0.25% of the current space (the curve is non-monotonic due to the constant expansion of BGP).

3.5.2 Firewall Log Correlation

To gain a broader view of the Internet and decrease the amount of time required to detect large-scale attacks, online collaborative systems [71], [94] have been developed to pool data from strategically placed Internet sensors and firewall/IDS logs of various networks. We focus on the SANS Internet Storm Center (ISC) [94] due to its relatively large size of 500K monitored IP addresses and detailed publicly available data. An ISC report consists of an IP address detected as a scanner, its source port, and the target's (IP, port) pair. These reports are often shared publicly, although certain fields



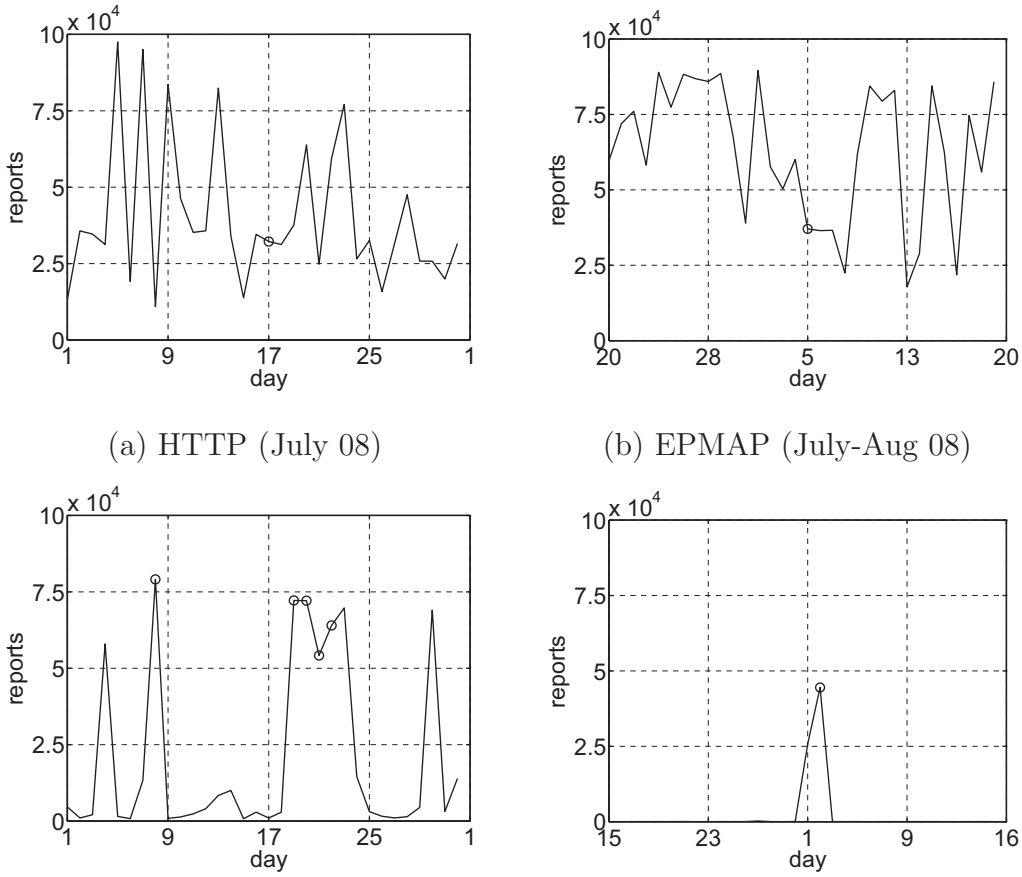
(a) total (b) fraction of BGP

Fig. 15. Progression of blacklisted IPs.

(e.g., destination IP) are obscured to protect the identity of subnets that submit their logs. Given that these reports represent information about unwanted traffic, they can be used to gain insight into how our scans are perceived by remote networks.

We examine ISC report summaries for several scans from Table VI. These summaries are compiled daily for each service (e.g., HTTP) and consist of the number of scanned targets, scanning hosts that targeted that service, and the ratio of packets that are TCP. We are particularly interested in the first metric as *all* reports related to our 24-hour scans should be contained in a single data point.

We downloaded summary data from ISC for one month surrounding a sample of our HTTP, EPMAP, DNS, and ECHO scans (i.e., 15 days prior and after). The result is plotted in Fig. 16, where the x-axis labels days in the 30-day window surrounding each scan and the highlighted points represent the days our scanner was actively probing that particular port. We happened to scan both HTTP in part (a) and EPMAP in part (b) on days when ISC experienced roughly a third of its peak number of daily reports (i.e., 27K compared to $80 - 90$K), which is nevertheless an huge number. The figure also shows that EPMAP clearly stands out as being scanned



(a) HTTP (July 08)

(b) EPMAP (July-Aug 08)

(c) DNS (May 08)

(d) ECHO (June-July 08)

Fig. 16. ISC reports with our scans marked.

with a consistently high amount of daily traffic.

In contrast, parts (c) and (d) for DNS and ECHO show that IRLscanner spiked report levels well above those of surrounding days. In fact, in the case of ECHO we produced an extremely anomalous event, raising the total from almost zero to 50K. Our activity on that port created concerns among network administrators that a new exploit was under way and/or a virus outbreak was in progress. All this eventually drew the attention of one of the traffic monitors at ISC, who wrote an explanatory blog post to calm down the ISC community.

Given the large amount of background noise from many scanning sources (whose

totals ISC also makes available) in parts (a) and (b) of Fig. 16, we conjecture that network administrators are more likely to react only to traffic that clearly stands out (i.e., makes its presence known by its high signal-to-noise ratio) rather than to scans on sensitive ports. This is confirmed by the fact that attack-reconnaissance port 135 generated the *least* number of complaints and that the ECHO port, which inherently represents little real threat to administrators due to the lack of hosts offering this service and heavy firewall filtering, produced an unusually strong blowback. This relationship where higher background scan traffic seems to imply fewer complaints might benefit researchers considering scans on sensitive/popular ports in the future.

3.5.3 Enumerating Contributors

It is well-known [11], [100] that the contributors to ISC and other firewall log correlation systems are vulnerable to losing their anonymity due to the nearly real-time public display of firewall reports with only the destination IP address omitted. Several techniques for correlating reports with targeted subnets (which is called *contributor enumeration*) have been proposed [11]; however, they require tens of billions of packets, allow for false positives, and consume multiple days during full enumeration.

Given our high-performance scanner that is capable of locally using multiple IP addresses, a much simpler attack preys on the source port, destination port, and source IP address reported in detailed ISC logs, which are displayed for all scanning hosts that ISC tracks. Probing each IP address in BGP set $\mathcal{B}$ with a unique combination of source/destination ports and source IP eliminates the possibility of false positives and the need to send any extra packets beyond those in $\mathcal{B}$. This can be accomplished for the current 2.1B hosts with 128 source IPs by simply rotating through all 64K source ports and roughly 250 destination ports, which can be hand-picked from the most-scanned lists to minimize the likelihood of raising suspicion. However,

by removing the source port from the public report, ISC can render this technique largely ineffective.

3.5.4 ACK Scans

To prepare their subnet's data for submission to ISC, many network administrators rely on firewall log analyzers such as psad [84] to separate scan traffic from innocuous packets dropped by the firewall. During our analysis, we discovered that many such tools ignore ACK packets, which suggests that network administrators often do not consider them to be particularly dangerous. To leverage this intuition, we propose a scan technique for cases where finding the majority of hosts in open set $\mathcal{O}$, while significantly reducing IDS detection, is beneficial (e.g., for rarely scanned ports).

The first phase of the technique is a simple ACK scan to every host in $\mathcal{B}$, which effectively discovers the subset of hosts that are not heavily protected by stateful firewalls. For every RST received, we verify that it has not been previously probed using a hash table and then immediately send it a SYN packet to establish whether the service is open or not. By only targeting hosts that previously responded, this type of scan reduces the SYN footprint by 94% for HTTP. We performed a single test measurement (HTTP_{AS} in Table VI), which discovered 31.7M of the 44M total responsive hosts, while requiring only 125M SYN packets to be sent. ISC data shows only $4,746$ reports for our IPs during HTTP_{AS} compared to $29,869$ reports collected for HTTP_2, which used the same T and m. This is significant as it amounts to an 84% decrease in the perceived intrusiveness of the scan.

3.5.5 DNS Lookups

We now turn to the last form of feedback we consider in this chapter. While whois lookups seem to be the predominate form of reconnaissance performed by remote

Table XII. DNS lookups on scanner source IPs

Scan	Reverse	Forward	Req/sec	Servers
$HTTP_2$	3.03M	85.4K	36	47.8K
$HTTP_3$	2.89M	80.1K	34	48.2K
$HTTP_6$	2.85M	66.3K	33	49.2K

network administrators and individuals when they detect a scan, many specialized tools augment IDS reports and firewalls logs with DNS lookups on offending IPs to provide more information on the scanning host to the user. While for large networks this functionality should be disabled (as it allows remote hosts to DoS the network by loading it up with billions of useless DNS lookups), many personal firewalls and small subnets implement some form of it.

We tested the frequency of these additional lookups by collecting all incoming requests for each scanning IP address to our locally controlled authoritative DNS server. To ensure that each request initiated by a remote entity contacted our name-server (rather than was answered from a cache), we set the DNS TTL to zero for both the reverse and forward lookups on scanner IPs/hostnames. After doing so, no RFC compliant nameserver should maintain our records in their cache.

The result of this collection process for three HTTP scans is contained in Table XII, which lists the number of reverse lookups for the IP addresses themselves and forward lookups on the names returned by those queries. We made sure that these IP addresses were not used for any other purpose but scanning and their names were not publicized beyond the project web-site. Therefore, forward lookups are almost certainly due to the common verification technique of determining the consistency between the forward and the reverse response. While the number of requests slightly declined for each subsequent scan, the last column shows that the number of unique

servers in each dataset had the opposite trend. The decline in lookup rates can be attributed to random noise, long-term caching at non-compliant DNS servers, and users growing tired of looking up our IPs.

It should be noted that performing DNS queries on scanner IPs potentially reveals the location (i.e., up to its local DNS server) of the IDS tool unless steps are taken to increase anonymity (e.g., using a well- publicized DNS forwarding service). The three scans in Table XII have identified $63,596$ unique DNS servers, out of which $35,296$ were present in each dataset. Further analysis of this data is deferred to future work.

3.6. Summary

In this chapter we developed a high-performance, Internet-wide service discovery tool called IRLscanner, whose main design objectives were to maximize politeness at remote networks, allow scanning rates that achieve coverage of the Internet in minutes/hours (rather than weeks/months), and significantly reduce administrator complaints. Using IRLscanner and 24-hour scan durations, we performed 20 Internet-wide experiments using 6 different protocols (i.e., DNS, HTTP, SMTP, EPMAP, ICMP and UDP ECHO), demonstrated the usefulness of ACK scans in detecting live hosts behind stateless firewalls, and undertook the first Internet-wide OS fingerprinting. In addition, we analyzed the feedback generated (e.g., complaints, IDS alarms) and suggested novel approaches for reducing the amount of blowback during similar studies.

CHAPTER IV

MODELING WINDOW-BASED IDS AND STEALTH SCANNING

4.1. Introduction

As the Internet has grown more hostile over time [78], [108], many networks now deploy Intrusion Detection Systems (IDS) [12], [105] to deal with the constant pressure of unsolicited traffic and attempts to exploit various vulnerabilities at end-hosts [78]. In its most general form, IDS monitors all inbound/outbound connections to detect such activities as *scanning* (e.g., attempts to find open services [3], [43], [78], [107], [117]), *intrusion* (e.g., malicious packets that exploit known vulnerabilities [68], [70], [97]), *anomalies* (e.g., new communication patterns indicating infection [31], [50], [106]), and *DoS attacks* (i.e., suspicious spikes in traffic/connection volume [51], [69]). In conjunction with firewalls, IDS can block offending hosts and raise alarms to alert administrators to potentially undesirable activity.

To maintain scalability [56], adapt over time, and keep state from growing to infinity, many existing IDS tools [12], [44], [74], [84], [105] utilize *window-based* processing of incoming traffic, which entails keeping per-flow statistics *only* for a limited period of time and applying IDS detection algorithms to the packets accumulated during this window. This makes the IDS detection process purely regenerative [89] and oblivious to any attacks that span multiple windows. One activity whose detection is particularly sensitive to the amount of state in each window is *horizontal scanning*, which consists of probing every Internet host on a given port to see if it is visible outside the firewall (repeating this process on multiple ports achieves vertical scanning as well).

To balance accuracy and false-positive rates, an IDS typically requires some minimum number of packets in the window before triggering an estimator or raising an alarm. As observed in [108], a worm could utilize so-called *stealthy* traffic patterns to prevent IDS from reaching this threshold, which makes such scans equally powerful against all underlying estimators. Our main interest lies in horizontal stealth scanning, where the main exposed technique [108] is to scan "very slowly," potentially dragging out the process over several months. However, it is unclear whether stealth scanning is possible at faster rates, in what particular order the IP space should be probed, and how likely existing IDS packages are to detect such approaches. To shed light on this issue, we model window rules of two popular IDS implementations (i.e., Snort [105] and Bro [12]), study the rates at which the existing scan techniques [3], [41], [59], [62], [82], [83], [108] become stealthy, and explore fundamental IDS limitations under stealth-optimal scan patterns.

While IDS avoidance in the literature commonly targets vulnerabilities of known implementations [45], [48], [77], [85], [101] or concealment of abnormal communication patterns [27], [110], [123], to our knowledge the inherent weaknesses of window-based IDS have not been analyzed before.

4.1.1 Formalization

We start the chapter by establishing stealth-scan objectives and their relationship to window-based IDS. We use the Flash-worm [108] model for the attacker and assume that it controls $m \geq 1$ source IPs (e.g., a botnet). In the first phase of the attack, the botnet scans the entire Internet for unprotected hosts. It then infects them in the second phase by attempting delivery of malicious payload only to vulnerable targets. The crux of this approach is the ability of the first phase to maximize penetration of IDS installations and remain undetected (i.e., stealthy). To understand whether this

is possible without knowing subnet boundaries or specifics of deployed tools, we first must expose rules for scanner operation and IDS window expiration.

We define a *scan pattern* X to be a combination of three algorithms – *permutation* (i.e., order of probed IPs), *split* (i.e., partitioning of targets between m zombies), and *schedule* (i.e., time instances when each IP is probed). During Internet-wide scans, the uniform permutation [62], [82], [83], [108] is generally considered better than IP-sequential [3], [41], [59] in terms of instantaneous load on target networks and stealthiness; however, it is currently unknown how likely IDS is to detect either one, what impact split/schedule have on stealth, and whether superior approaches exist.

Among the deployed IDS solutions [12], [44], [74], [84], [105], window expiration follows two main principles, which can be inferred from product source code and documentation. In a model we call IDS-A (e.g., Snort [105]), window expiration at subnet s occurs every Δ_s time units and resets the state of *all* scanning sources. In a model we call IDS-B (e.g., Bro [12]), the window of each source i resets Δ_s time units following the last target hit by i. Due to its selective tracking of sources that continuously scan, IDS-B performs much better than IDS-A at detecting slow scanners.

Equipped with the formalization above, we next introduce the concept of *stealth cover time* (SCT) T_X^s, which is the minimum scan duration T that allows a particular Internet-wide scan pattern X to avoid detection at s. A scanner is then called *stealth-optimal* (SO) if it simultaneously minimizes the SCT of all CIDR subnets under both IDS-A/B. To examine the specific performance improvements and whether developing a stealth-optimal algorithm is worth the effort, we first derive its expected gains over the existing patterns and assess whether they are significant.

4.1.2 Analysis

To gauge the relative stealth of different methods, define pattern X to be *k-faster* in s than Y if $T_X^s = T_Y^s/k$, i.e., its SCT is k times smaller than Y's. We start the analysis by deriving the probability that both IDS-A/B detect the commonly used permutations in scanning (i.e., IP-sequential and uniform) under pre-permutation [10], [77] and post-permutation [8], [35], [58], [59], [62], [83], [120], [119], [127] splits. Results show that the number of botnet IPs m makes no difference when used with a pre-permutation split; however, under a post-permutation split both methods become m-faster with m IPs in every subnet s.

This suggests a simple technique that may significantly increase the stealthiness of a botnet. In this method, each infected host that does not reside behind a NAT uses ARP to sniff unused IPs on its subnet and alias them to its own NIC. Assuming the number of stolen IPs is j, the scan from this host becomes not only j-faster (i.e., can increase the speed by a factor of j for the same level of detection), but also much harder to map to the correct host without administrator access to ARP packets and MAC-layer addresses.

We next derive the SCT of both IP-sequential and uniform algorithms, showing that the latter is generally much stealthier than the former; however, we also identify cases when the SCT of the uniform permutation scales *quadratically* with subnet size and may exceed that of IP-sequential in sufficiently large networks. In fact, we show examples where IP-sequential is stealthier than uniform against IDS-A in all networks larger than /20 and against IDS-B in those larger than /21. Also, as expected, both permutations find IDS-B significantly harder to avoid and require a constant-factor slower scanning.

From this analysis, we additionally find that SO patterns not only similarly

benefit from ARP hijacking, but are also orders of magnitude faster than uniform under all practical conditions and can cover both IDS-A/B equally fast. In /16 subnets and default Bro settings, stealth-optimal scanning is 1,209-faster than uniform, which is equivalent, for example, to a reduction in scan duration T from 3.3 years to 1 day while keeping the detection probability the same. Assuming $T = 24$ hours and networks no larger than a /16, SO can avoid the open-source version of Snort [105] using just $m = 12$ IPs, Bro [12] using 24 IPs, and Bro TRW [12] using 455 IPs, assuming their respective default settings. With a 12K-node botnet, where each host hijacks just 10 local IPs using ARP, a stealth scanner can cover the Internet in one day and remain undetected in all /8 networks operating Snort/Bro/TRW with their default parameters.

This observation prompts us to examine whether it is possible to achieve stealth optimality in practice and at what cost/overhead.

4.1.3 Stealth Scanner

We next design a class of stealth-optimal algorithms and test them over the Internet. We first show that SO patterns must incorporate not only a new permutation, but also a different split and schedule. The general class of SO permutations contains 2^{n-1} unique elements, where $n = 2^{32}$ is the size of IP space, and requires 512 MB of seed state to be communicated to each scanning host. Given this overhead, one may consider these methods impractical and unthreatening for the Internet. However, we also show that an attacker can use a subset of all SO permutations and distribute state to each scanning host using just 12 bytes.

The overhead of actually generating these SO permutations consists of two arithmetic operations and two RAM lookups, whose combination runs at over 20M per second on commodity hardware. Both split and schedule are local modifications ap-

plied by individual source IPs to the main permutation delivered to them from the botmaster and incur very little additional cost. The new split algorithm ensures that each subnet s sees all m source IPs in a round-robin fashion, while the new schedule guarantees optimal scanning against IDS-B. Both consume almost no CPU unless scanning rates are exorbitantly high.

We test the developed stealth-optimal framework in three Internet-wide HTTP (port 80) scans using $T = 24$ hours and observe how SO patterns impact the generation of scanning reports at the SANS Internet Storm Center [94]. Our results show that almost 40% of the reports can be suppressed using SO patterns aimed at IDS-B rather than IDS-A. This not only suggests that IDS-B is actively deployed in the Internet, but also unveils the optimal parameters for reducing the detection footprint. These parameters agree with our analysis and support the proposed models.

We finish the chapter by proposing a simple, yet effective, model of IDS operation that dynamically changes Δ_s, which results in reduced effectiveness of SO scanners.

4.2. Related Work

Significant effort has been expended in the area of designing better IDS to detect malicious behavior, which can be broadly classified into three thrusts. The first is *signature-based* detection [46], [79], [90], which checks incoming packets against a database of known exploits. The second, *anomaly-based* detection [31], [50], [106], relies on deviation in network traffic from an established normal pattern. The third approach we call *pattern-based* detection [43], [95], [113], which depends on inherent qualities of scanners (e.g., excessive failed connections).

The area of IDS avoidance, which is the focus of this chapter, can be partitioned along the same three dimensions. The most common directions for evading IDS

involves sending malicious packets that do not match the signature database [48], [77], [85], [101]. Public tools such as nmap [77] rely on incorrect reconstruction of the packet by IDS (e.g., IP-level fragmentation [85], incorrect checksums, TTL tricks [101]), as well as the ability of the attacker to hide its identity and/or packet contents (e.g., source-address spoofing, confusing IP options and flags [85], [101], and polymorphic worms [48] that modify the payload of every packet).

The second IDS-avoidance approach relies on concealing abnormal communications to bypass anomaly detectors [27], [110], [123]. Attackers can mimic the normal traffic of exploited applications (e.g., matching sending rate [27] and pattern of packets sent [110]) or modify scan rates [123] to avoid appearing like a propagating worm.

The last direction, which is the topic of this chapter, works against pattern-based detectors by leveraging the specifics of IDS algorithms and designing scan patterns that never reach a detection threshold. We are aware of only one effort in this area, in which [45] alternates between known alive hosts and unexplored space to manipulate the TRW [43] detection algorithm.

4.3. Formalizing Scanning

In this section, we outline the goals of a large-scale scanner, introduce three fundamental elements of a scan that determine its performance, and set forth assumptions on the various types of IDS. We then discuss stealth-optimal scans and their properties.

4.3.1 Scan Objectives

Assume $\mathcal{F} = \{0, 1, \ldots, n\}$ is the IPv4 address space, where $n = 2^{32}$, and $\mathcal{S}$ is the set of all CIDR networks. As discussed in [108], one of the most effective penetration

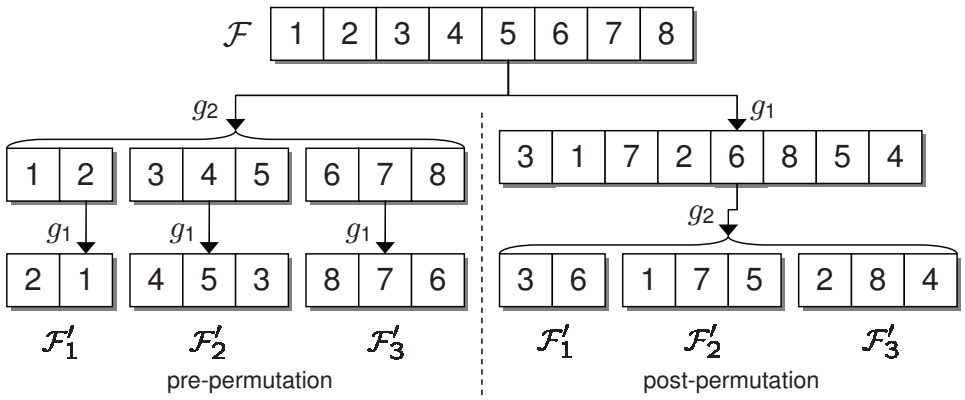


Fig. 17. Illustration of permutation/split ($m = 3$).

models used by an attacker (i.e., the Flash worm) relies on a two-phase scan/infect approach. The first phase scans $\mathcal{F}$ using m source IPs in some set $\mathcal{M}$ (e.g., a subset of the attacker's botnet) to build a list of vulnerable targets $\mathcal{V}$. The second phase uses zombie hosts in another set $\mathcal{M}'$ to attempt infection of $\mathcal{V}$ using a new exploit (either simultaneously or at some later time). Sets $\mathcal{M}$ and $\mathcal{M}'$ may overlap if exposure during the first phase does not reduce the infection performance of each IP during the second phase.

As there is no need for newly infected hosts to scan the entire Internet, they perform a quick scan of the local network (e.g., the corresponding BGP prefix) and then stop. Due to the short duration of the infection phase (hours rather than weeks) and limited local scanning, this attack is difficult to stop once it starts and infections are hard to detect after phase two is over.

For a given budget m and fixed scan duration T, we assume the attacker's goal is to minimize its detection probability at each CIDR subnet s (i.e., maximize its stealthiness) during the first phase of the attack. The problem of delivering malicious payload is implementation/exploit-dependent and outside the scope of this chapter. Due to the static nature of set $\mathcal{M}$, we are also not concerned with sub-allocating the

scan space dynamically to each newly infected host as commonly studied in worm propagation [62].

4.3.2 Scan Patterns

Any Internet-wide *scan pattern* can be decomposed into three principle elements – permutation, split, and schedule. The existing literature [8], [10], [35], [68], [70], [82], [83], [97] has glanced over the first two elements, but without any formalization or analysis. Given a list of items $\mathcal{F}$, a *permutation* is a one-to-one mapping function $g_1 : \mathcal{F} \to \{1, 2, \ldots, |\mathcal{F}|\}$ that simply shuffles the elements in $\mathcal{F}$. We often denote the permuted sequence by $\mathcal{F}' = g_1(\mathcal{F})$. Permuting the IP space is highly beneficial because it reduces the instantaneous load on target networks, increases delays between packets entering IDS, and generally lowers the detection probability. It can also control randomness and correlation among the destinations within each s.

We define a *split* as a many-to-one function $g_2 : \mathcal{F} \to \mathcal{M}$ that assigns the elements of list $\mathcal{F}$ to scanner IPs. One can view this as a partition of $\mathcal{F}$ into non-overlapping lists $\mathcal{F}_1, \ldots, \mathcal{F}_m$, where $\mathcal{F}_i$ is given to host $i \in \mathcal{M}$. If each of $\mathcal{F}_i$ is an ordered subset of $\mathcal{F}$, we call this arrangement a *block-split*. In the context of the Internet, a *pre-permutation* scanner [10], [77] first applies partitioning g_2 to $\mathcal{F}$ and then permutes each $\mathcal{F}_i$ using some algorithm g_1 to produce the final assignment $\mathcal{F}'_i = g_1(\mathcal{F}_i)$ of source i. A *post-permutation* scanner [8], [35], [58], [59], [62], [83], [119], [120], [127] first applies permutation g_1 to $\mathcal{F}$ and then partitions list $\mathcal{F}'$ using g_2 into $\mathcal{F}'_1, \ldots, \mathcal{F}'_m$. This is schematically shown in Fig. 17, where the pre-permutation scanner (left side) uses a block-split, while the post-permutation one (right side) does not.

The final issue is to determine how each host i probes its target set $\mathcal{F}'_i$ so as to complete the scan by a certain time T. To allow i to periodically send packets faster or slower than its average rate $r_i = |\mathcal{F}'_i|/T$, define a *schedule* to be a many-

to-one function $g_3 : \mathcal{F}'_i \to [0, T]$ that decides the exact time instances at which i hits each of its assigned targets. While all existing scanners draw elements from $\mathcal{F}'_i$ with a constant inter-probe delay $1/r_i$, additional (bursty) patterns will be discussed shortly.

4.3.3 Window-based IDS

To understand the relationship between detectability of a scan and its probing rate r, one first requires a model of IDS. In what follows, we formalize two window-based detection rules that are loosely based on popular IDS packages [12], [44], [74], [105] and firewall-log analyzers [84]. Since scalability [56] generally requires that IDS expire state and operate in windows of finite size, other high-performance IDS designs are also likely to fall under one of the two categories studied here.

Our first model, which we call IDS-A, stems from the rules of Snort [105] and its commercial implementations [44], [74]. For each source IP $i \in \mathcal{M}$ sending packets into a given subnet $s \in \mathcal{S}$ protected by an IDS, define $C_i^s(t)$ to be the count of unique targets seen by the IDS from i in the interval $[0, t]$. Since keeping infinite history of hosts contacted by i incurs substantial RAM/CPU overhead and fails to properly discount outdated information, IDS-A periodically resets i's state as illustrated in Fig. 18. Here, random process $C_i^s(t)$ increases by 1 for each new target hit by i, returns to state 0 every Δ_s time units, and absorbs in some pre-defined threshold state $a_s \geq 1$ that triggers an IDS alarm or some internal estimation algorithm (e.g., TRW [43], CBCRL [95]), which we assume *always* detects the scanner once invoked.

Our second model, which we call IDS-B, is derived from the techniques used by Bro [12] and certain firewall-log analyzers [84]. In this method, $C_i^s(t)$ represents the number of unique unresponsive targets hit by i in the interval $[0, t]$. Unlike IDS-A, this model expires i's state *only* if it does not probe any new unresponsive targets for Δ_s time units. Assuming the worst-case scenario where none of the targets respond,

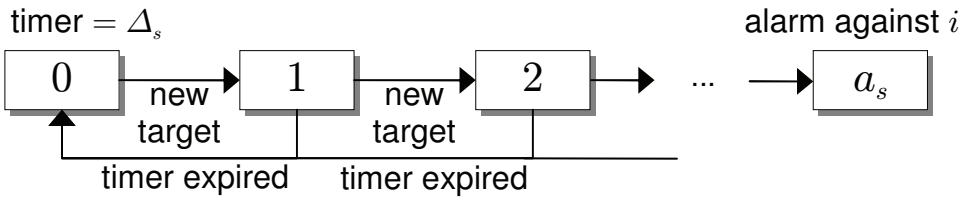


Fig. 18. Process $C_i^s(t)$ of IDS-A.

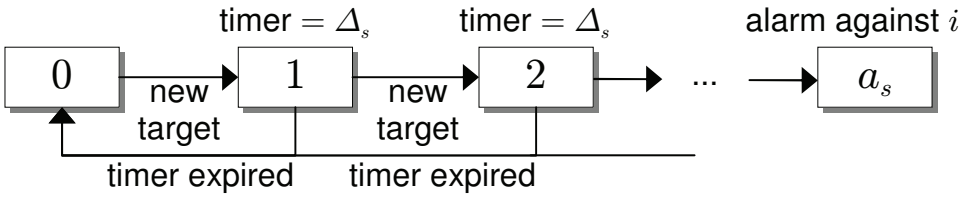


Fig. 19. Process $C_i^s(t)$ of IDS-B.

this logic can be described by Fig. 19, where the expiration timer of i resets to Δ_s upon each state transition.

For the same parameter set, IDS-B is stricter than IDS-A in the sense that any scanner detected by the latter is always detected by the former. Similarly, a scanner avoiding IDS-B always avoids IDS-A. However, IDS-B achieves this improvement at the expense of maintaining a separate timer for each i and stochastically higher overhead (i.e., longer lists of seen targets) in steady-state. Default parameters (Δ_s, a_s) of deployed open-source and commercial IDS-A/B are summarized in Table XIII.

4.3.4 Stealth

We are now ready to formalize the detectability of a scan and its stealthiness. Let $\mathcal{I} \subseteq \mathcal{S}$ be the set of all IDS-equipped networks, where each element of $\mathcal{I}$ is a full CIDR block (often written in the $/x$ notation). Then, we have the following classification.

Definition 1. *A network $s \in \mathcal{I}$ is called* size-trivial *if $m(a_s - 1) \geq |s|$,* unavoidable *if $a_s = 1$, and* normal *otherwise.*

Size-trivial subnets can be covered with fewer than a_s packets per source IP, which means they pose no threat of detection if the scanner can probe them while perfectly load-balancing between its IPs in $\mathcal{M}$. In contrast, unavoidable networks raise an alarm on the very first probe (e.g., darknets, personal firewalls) and thus cannot be avoided in practice by any scanner. Define $\mathcal{I}_{ST}, \mathcal{I}_U, \mathcal{I}_N$ to be pair-wise non-overlapping sets of respectively size-trivial, unavoidable, and normal networks in $\mathcal{I}$.

Define $r = n/T$ to be the scanning rate. Then, for each source IP $i \in \mathcal{M}$, let

$$\tau_i^s = \inf\{t > 0 : C_i^s(t) = a_s | C_i^s(0) = 1\} \tag{2}$$

be the amount of time it takes s to detect i (i.e., the hitting time of $C_i^s(t)$ onto state a_s after the IDS sees the first packet from i). Let $A_i^s(r)$ be an indicator variable of detection event $\tau_i^s < T$ and $A^s(r) = \sum_{i \in \mathcal{M}} A_i^s(r)$ be the number of source IPs detected by subnet $s \in \mathcal{I}$ in $[0, T]$. Then, $\rho^s(r) = P(A^s(r) \geq 1)$ is the probability that network s detects the scan at rate r.

Assume X is a pattern that scans all IPs in $\mathcal{F}$. Then, define the *stealth-cover time* (SCT) T_X^s of a normal subnet $s \in \mathcal{I}_N$ to be the minimum scan duration T that allows X to avoid detection at s. Recalling that $r = n/T$, observe that $T_X^s = \inf\{t \geq 0 : \rho^s(n/t) = 0\}$. Note that the concept of SCT applies only to normal subnets since size-trivial networks can be scanned without detection in $T_X^s = 0$ and unavoidable networks require $T_X^s = \infty$, neither of which is helpful in establishing the performance of scanning algorithms.

Definition 2. *A scan pattern X is called* k-faster *in $s \in \mathcal{I}_N$ than Y if it exhibits k*

Table XIII. Parameters of common IDS

Type	Name	Δ_s (sec)	a_s
IDS-A	Snort [105]	60	5
	Juniper [44]	120	50
	NIKSUN [74]	300	200
IDS-B	Bro [12]	600	20
	Bro TRW [43]	1800	4
	Psad [84]	3600	5

times smaller SCT, i.e., $T_X^s = T_Y^s/k$. *It is called* IP-scalable *if it is m-faster in all* $s \in \mathcal{I}_N$ *with m source IPs than with one.*

It is usually safe to assume that the scanner remains oblivious to individual IDS parameters (Δ_s, a_s) and CIDR subnet boundaries in set $\mathcal{I}$. However, from the analysis of common IDS implementations (e.g., Bro-TRW [43] requires at least 4 samples for its estimator), one may possess a uniform lower bound β on parameter a_s. In that case, we call a scanner β-*aware* if $2 \leq \beta \leq a_s$ holds simultaneously for all normal subnets $s \in \mathcal{I}_N$ and no larger bound is known. If $\beta = 2$, we call the algorithm *unaware* since it benefits from no additional knowledge.

Definition 3. *A* β*-aware scan pattern X is called* stealth-optimal *(SO) if for both IDS-A/B it 1) achieves* $\rho^s(r) = 0$ *in all size-trivial networks; and 2) minimizes the SCT of all normal subnets, i.e.,*

$$\forall s \in \mathcal{I}_N : T_X^s = \min_Y T_Y^s \tag{3}$$

4.3.5 Existence

To understand optimal patterns, we next derive a lower bound on $\min_Y T_Y^s$ in (3) and show that there exists a *local* (i.e., as seen by each s) arrival pattern of packets that achieves it under *both* IDS-A/B. Later in the chapter, we develop a scanner that implements this pattern *globally* (i.e., simultaneously in all CIDR subnets).

Theorem 3. *The SCT of $s \in \mathcal{I}_N$ is lower-bounded by*

$$\min_Y T_Y^s \geq \frac{|s|\Delta_s}{m(\beta-1)}. \tag{4}$$

Proof. For a given scan duration T, the average number of probes sent from source IP i to s per Δ_s-interval is $b_i = |\mathcal{F}_i'(s)|\Delta_s/T$, where $\mathcal{F}_i'(s)$ is the set of addresses in s assigned to i. From the pigeonhole principle, observe that if b_i is larger than $\beta-1$, then there will be at least one Δ_s-interval with β targets. Since the scanner does not know the actual a_s, it must assume that detection is avoided if and only if $b_i \leq \beta-1$ for all i, i.e.,

$$\max_{i\in\mathcal{M}} \frac{|\mathcal{F}_i'(s)|\Delta_s}{T} \leq \beta-1, \tag{5}$$

where $\sum_{i\in\mathcal{M}} |\mathcal{F}_i'(s)| = |s|$. Therefore, the SCT of any method Y must be bounded

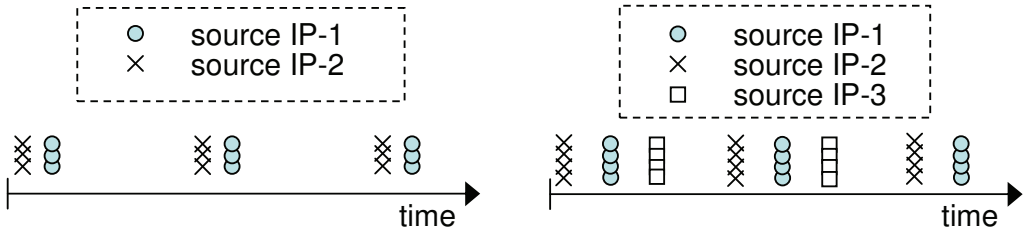
$$T_Y^s \geq \frac{\max_{i\in\mathcal{M}} |\mathcal{F}_i'(s)|\Delta_s}{\beta-1} \geq \frac{|s|\Delta_s}{m(\beta-1)}, \tag{6}$$

which leads to the desired result in (4). □

To show that SO patterns exist locally, suppose each source i shapes its traffic to s into bursts of $\beta-1$ packets separated by an intra-IP gap

$$\delta_{intra}^s = \frac{Tm(\beta-1)}{|s|}. \tag{7}$$

As illustrated in Fig. 20(a), this pattern initially raises target count $C_i^s(t)$ to



(a) SO ($\beta = 4$) (b) SSO ($\beta = 4$)

Fig. 20. Stealthy β-aware probing seen by s.

$\beta - 1$ and then follows it up with the proportionally-stretched gap in (7). Detection is avoided for IDS-B if and only if $\delta^s_{intra} \geq \Delta_s$. As discussed earlier, IDS-B is stricter than IDS-A, which means that the scanner also automatically avoids IDS-A. Combining the two cases and solving $\delta^s_{intra} \geq \Delta_s$ for T, this pattern exhibits the same SCT for both types of IDS

$$T^s_O = \frac{|s|\Delta_s}{m(\beta - 1)}, \tag{8}$$

which is optimal as it equals the lower bound in (4). While the existence of global SO patterns may not be immediately obvious, they will be shown later in the chapter.

Examining (8), notice that the optimal SCT is a linear function of subnet size $|s|$ and all IDS parameters, unlike the uniform permutation (studied in the next section), whose SCT sometimes scales as $|s|^2$. Furthermore, SO patterns are not only IP-scalable, but also $(\beta - 1)$-faster than any unaware pattern.

4.3.6 Improvements

One drawback to the SO pattern is its inability to control the distance between probes from different IPs, which for large m (e.g., hundreds or thousands) may lead to non-trivial spikes in flow and packet intensity at individual subnets. Since β is usually small (i.e., no larger than 4 from Table XIII), normal intra-burst spikes are of much

less concern. To prevent IP-burstiness from raising suspicion, triggering DoS filters, and potentially losing packets due to congestion, we next define a more stealthy class of scan algorithms.

Definition 4. *SO patterns that minimize the burstiness of arriving traffic at each* $s \in \mathcal{S}$ *are called* smooth and stealth-optimal *(SSO).*

As shown in Fig. 20(b), SSO maximizes the inter-IP delay between adjacent bursts and keeps it constant at

$$\delta^s_{inter} = \frac{T(\beta - 1)}{|s|}. \tag{9}$$

Since size-trivial networks can never detect SO scanners, SSO can additionally reduce burstiness within such networks by spacing its intra-IP probes as far as possible (i.e., utilize unaware probing for $s \in \mathcal{I}_{ST}$).

Finally, the last desirable feature of a scanner is random sampling of the internal space within each subnet s. To properly capture this, define for any list $\mathcal{L}$ operator ${}_b\mathcal{L}$ to retain the lower b bits of its elements. Then, for all valid permutations, the items in ${}_b\mathcal{F}'$ are distributed uniformly in $[0, 2^b - 1]$; however, this sequence may be strongly correlated and/or almost deterministic, which is highly undesirable as it attracts the attention of administrators and tools trained to react to obvious scanning patterns.

Definition 5. *SSO patterns for which* ${}_b\mathcal{F}'$ *approximates an iid sequence of uniform variables in* $[0, 2^b - 1]$ *for all* $1 \leq b \leq 32$ *are called* uncorrelated, smooth, and stealth-optimal *(USSO).*

4.4. Analysis of Existing Methods

Our goal in this section is to analyze two popular methods for scanning the Internet – IP-sequential [3], [41], [59] and uniform [62], [82], [83], [108] – and compare them

to stealth scanners defined in the previous section. We not only derive the detection probability $\rho^s(r)$ for both IDS-A/B, but also develop a unifying modeling framework that covers both pre/post-permutation splits.

4.4.1 IP-sequential

Our first studied method, which we call *IP-sequential*, does not permute the IP space (i.e., $\mathcal{F}' = \mathcal{F}$), uses a block-split that partitions $\mathcal{F}$ into m equal-size chunks, and sends packets from each i with constant spacing $\delta = 1/r_i = Tm/n$. Note that both pre/post permutation splits are equivalent for this method and each subnet s (smaller in size than n/m and not falling on the boundary between adjacent source IPs) is scanned by a single $i \in \mathcal{M}$ assigned to it.

The IP-sequential permutation is guaranteed to avoid IDS-A if and only if each source allows no more than $\beta-1$ inter-packet gaps within any interval $[t, t+\Delta_s)$, which is equivalent to $\delta(\beta - 1) \geq \Delta_s$. For IDS-B, this condition is much more conservative since none of the inter-packet delays δ can be smaller than Δ_s. Combining the two cases, we have the IP-sequential SCT as

$$T_Q^s = \frac{\Delta_s n}{m\zeta}, \quad \text{where} \quad \zeta = \begin{cases} \beta - 1 & \text{IDS-A} \\ 1 & \text{IDS-B} \end{cases}. \tag{10}$$

Notice from (10) that sequential scanning is IP-scalable and $(\beta-1)$-faster against IDS-A than IDS-B. However, unlike SO, this pattern does not automatically avoid all size-trivial networks. Only subnets with $|s| < a_s$ fall into this category and m cannot be used to expand it. In terms of SCT performance, sequential is $n(\beta-1)/\zeta|s|$ times slower than SO in each s. Given a /16 subnet with Bro's default $\beta = 4$, SO is $65,536$-faster than IP-sequential against IDS-A and $196,608$-faster against IDS-B.

In terms of probing rates, IP-sequential scans each s at

$$\max\Big(\frac{n}{mT}, \frac{|s|}{T}\Big) \tag{11}$$

packets per second (pps). Depending on the scan duration T, this rate may become quite noticeable in comparison to the background traffic and may lead to easy detection. For $T = 24$ hours, the first term of (11) is $49.7/m$ Kpps, regardless of the target subnet size. For the same T, the SO pattern's rate is $\max(n/m|s|, 1)$ times smaller at s. For $m = 10$, this ratio is $6,553$ for /16 networks (i.e., 0.76 pps) and 1.67M for /24 subnets (i.e., one packet every 337 seconds). However, if both the botnet and target network s are large (i.e., $m|s| \approx n$), the scan rate of IP-sequential might not be too far from optimal, which is possibly one of the reasons for its widespread use in the Internet [3].

4.4.2 Uniform Pattern

The main drawback to the sequential permutation is that it does not explore other subnets before hitting the same s with repeat packets. Uniform scanning improves upon this basic algorithm by spreading packets between random subsets of the Internet. We call a permutation function g_1 on list $\mathcal{F}$ *uniform* if the probability that each $i \in \mathcal{F}$ moves into position $j \in [1, |\mathcal{F}|]$ is $1/|\mathcal{F}|$. All existing uniform scanners use block-split and constant inter-packet delays $\delta = Tm/n$.

Consider a particular subnet s with $|s|$ IPs that need to be scanned in $[0, T]$. The uniform permutation randomly scatters these $|s|$ targets throughout a discrete set $\mathcal{D}$, which equals $\mathcal{F}'_i$ for pre-permutation, where i is the host scanning s, and $\mathcal{F}'$ for post-permutation. This is illustrated in Fig. 21, where the IPs in s are marked

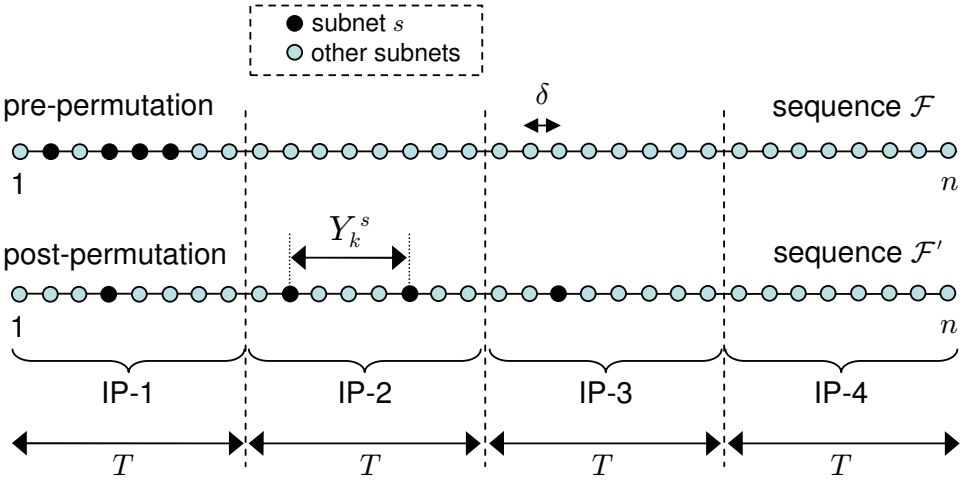


Fig. 21. Uniform model ($m = 4, |s| = 4$).

with black dots and all the remaining IPs are gray. Defining

$$\omega = \begin{cases} 1 & \text{pre-permutation} \\ m & \text{post-permutation} \end{cases} \tag{12}$$

the size of set $\mathcal{D}$ is $|\mathcal{D}| = \omega n/m$.

Assuming $n \gg 1$, the shuffle can be viewed as occurring in *time* rather than inside a discrete set $\mathcal{D}$. This transformation simplifies understanding of the derivations below and does not impact any IDS detection probabilities. Specifically, imagine that source IPs scan the Internet *sequentially* (rather than concurrently) as shown at the bottom of Fig. 21. Then, the time instances when s sees probes from $\mathcal{M}$ can be viewed as uniformly random in the time interval $[0, \omega T]$.

4.4.3 Uniform Detection Probability

We start by analyzing how the uniform pattern delivers packets to individual networks and develop a simple model for the detection probability in IDS-A. We later extend this result to IDS-B.

Theorem 4. *For $T \gg \Delta_s$, the probability that a normal subnet $s \in \mathcal{I}_N$ with IDS-A detects a uniform scanner is*

$$\rho_A^s(r) \approx 1 - \Big(\sum_{j=0}^{a_s-1} \binom{|s|}{j} q^j (1-q)^{|s|-j}\Big)^{1/q} \tag{13}$$

where $q = \Delta_s/\omega T$.

Proof. From the discussion in Section 4.4.2 and Fig. 21, each address from s has the same probability $q = \Delta_s/\omega T$ of falling into a given bin of size Δ_s. Ignoring the last potentially incomplete bin of each user (which we can do since $\Delta_s \ll T$), the number of probes sent to s in bin $[j\Delta_s, (j+1)\Delta_s) \subseteq [0, \omega T]$ is a binomial variable $W_j^s \sim B(|s|, q)$. Define $\phi_{bin}^s = P(W_j^s \geq a_s)$ to be the probability that s detects the scan in a given bin j. Since $\sum_{j=1}^{1/q} W_j^s = |s|$, the variables from different bins are dependent; however, for large T/Δ_s, one can treat them as approximately iid, which leads to

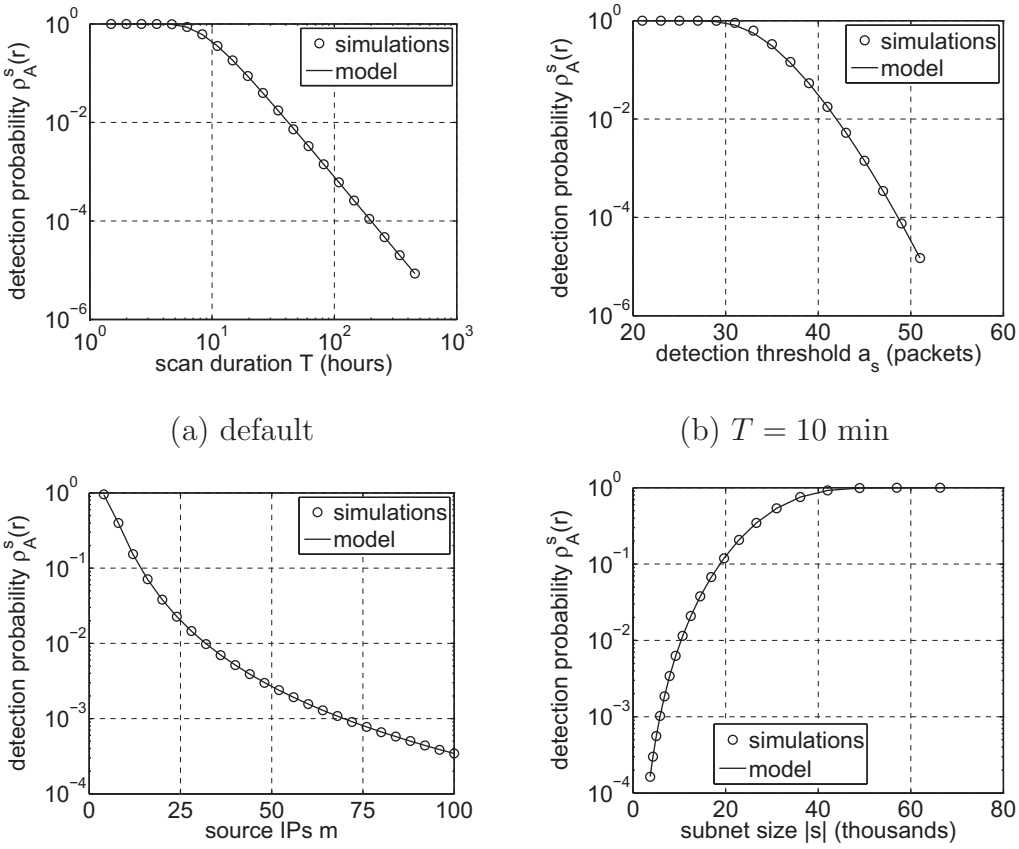
$$\rho_A^s(r) \approx 1 - (1 - \phi_{bin}^s)^{1/q}. \tag{14}$$

Substituting the CDF of W_j^s in (14), we get (13). □

Fig. 22 compares simulations to (13) as four of the main parameters of the model change. Numerical results indicate that (13) is accurate to within 1% as long as $T \geq 100\Delta_s$. Part (b) shows one example where $T = 10\Delta_s$ is insufficiently large, which results in some discrepancy for values of $a_s \in [30, 35]$.

From the analysis of (13), observe that $\rho^s(r)$ is a function of product ωT, which automatically means that uniform scanning is IP-scalable against IDS-A if and only if it uses post-permutation split. Otherwise, the detection probability stays constant regardless of m and the scanner ends up wasting IPs without improving its stealthiness.

We now turn our attention to IDS-B and its detection probability. Our first step



(a) default

(b) $T = 10$ min

(c) $T = 4,750$ sec

(d) $T = 250$ days

Fig. 22. Comparison of post-permutation IDS-A model (13) to simulations (default parameters $|s| = 2^8, \Delta_s = 60$ sec, $a_s = 4$, and $m = 1$).

is to understand inter-probe delays $\{Y_k^s\}_k$ seen by s from $\mathcal{M}$ in our continuous model in Fig. 21.

Theorem 5. *Inter-probe delays $Y_1^s, \ldots, Y_{|s|-1}^s$ are identically distributed random variables with $E[Y_k^s] = \omega T/|s|$ and the following CDF tail*

$$P(Y_k^s \geq y) = \left(1 - \frac{y}{\omega T}\right)^{|s|}, \quad 0 \leq y \leq \omega T. \tag{15}$$

Proof. First, notice that the uniform permutation is equivalent to randomly distributing $|s|$ points on the ring of length ωT. Since there are $|s|$ inter-probe gaps on the ring, their mean is simply $E[Y_k^s] = \omega T/|s|$. Second, the probability that a given address from s falls in the interval $[t, t+y) \subseteq [0, \omega T]$ is $y/\omega T$. Then, the probability that none of the addresses from s land into $[t, t+y)$ is $P(Y_k^s \geq y) = (1 - y/\omega T)^{|s|}$. □

We omit simulations showing that (15) is very accurate. Instead, we define $\chi_s = P(Y_k^s < \Delta_s)$ and proceed to the next result.

Theorem 6. *For $(|s| - a_s)(1 - \chi_s)/m \to \infty$, the probability that IDS-B at a normal subnet $s \in \mathcal{I}_N$ detects a uniform scanner is asymptotically*

$$\rho_B^s(r) \approx 1 - e^{-(|s|-a_s+1)(1-\chi_s)\chi_s^{a_s-1}}. \tag{16}$$

Proof. Define J_k^s to be an indicator variable of event $Y_k^s < \Delta_s$. Then, $P(J_k^s = 1) = 1 - P(J_k^s = 0) = \chi_s$. Since IDS-B needs $a_s - 1$ consecutive 1s in set $\{J_k^s\}_k$ to arrive into state a_s, define

$$X_k^s = \begin{cases} 1 & J_k^s = J_{k+1}^s = \ldots = J_{k+a_s-2}^s = 1 \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

to be an indicator of a detection event occurring at time $k + a_s - 2$. Denoting by $l = |s| - a_s + 1$ the size of set $\{X_k^s\}_k$, we have that $X^s = \sum_{k=1}^{l} X_k^s$ is the total number

of detections in $[0, \omega T]$ and $\rho^s(r) = P(X^s \geq 1)$.

Before deriving this probability, note that we need to analyze only those consecutive runs of 1s in sequence $\{J_k^s\}_k$ that follow a 0 and start no later than position l. Indeed, supposing that this set contains Z zeroes, X^s is non-zero if and only if any of the Z runs of 1s that immediately follow a zero has length at least $a_s - 1$. All other runs provide redundant information and can be removed from consideration.

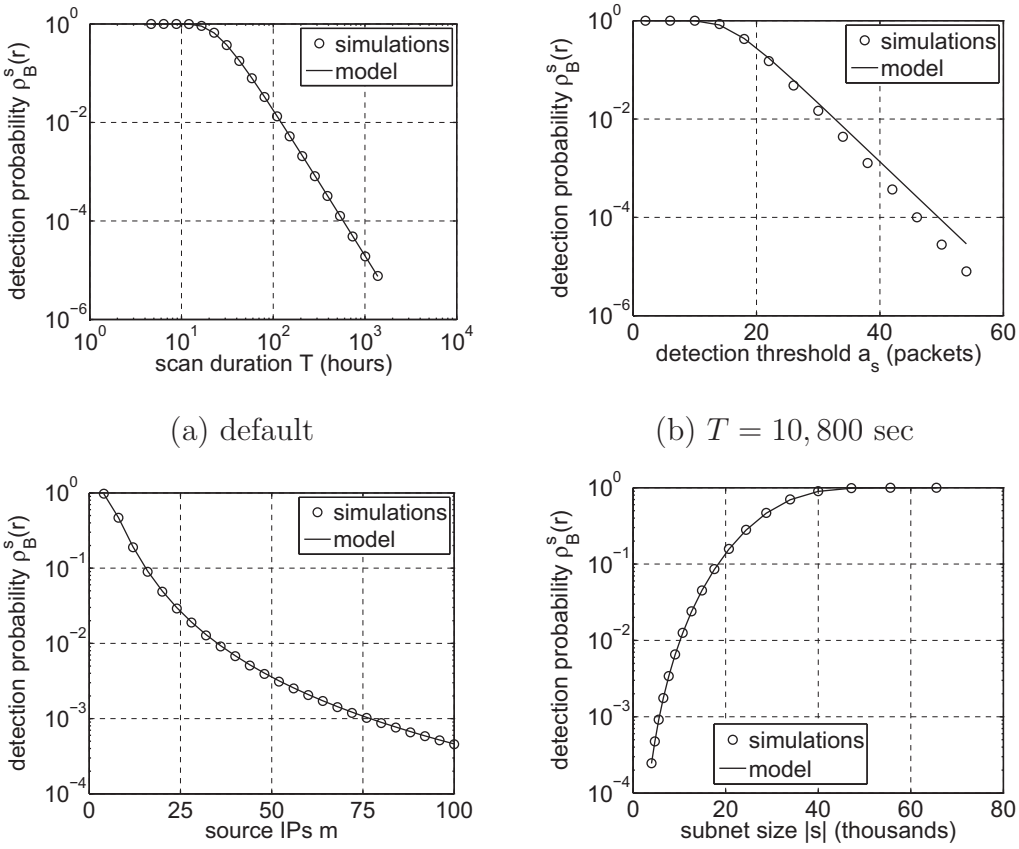
Define V_j to be the value of X_k^s following the j-th zero in set $\{J_k^s\}_{k=1}^l$. We then obtain

$$X^s = \sum_{j=1}^{Z} V_j. \tag{18}$$

From the Chen-Stein theorem [4] and treating set $\{J_k^s\}_k$ as approximately iid, variable X^s converges to the Poisson distribution with rate $\lambda = E[X^s] = E[Z]E[V_1^s]$ as $E[Z] \to \infty$. Noticing that $E[Z] = l(1 - \chi_s)$ and $E[V_1^s] = \chi_s^{a_s - 1}$, we get $\lambda = l(1 - \chi_s)\chi_s^{a_s - 1}$, which immediately leads to $\rho^s(r) \approx 1 - e^{-\lambda}$ in (16).

We should make three observations about this derivation. First, for small $|s|$ and large a_s, the dependency in set $\{J_k^s\}_k$ may be strong enough for $\rho^s(r)$ to disagree with the model (which arises because $\sum_k Y_k^s \leq \omega T$ and set $\{Y_k^s\}_k$ is not iid); however, in the limit (16) is exact. Second, we have replaced Z with its expectation in the Chen-Stein method; however, simple but tedious math shows that $E[e^{aZ}]$ for binomial Z behaves almost the same as $e^{aE[Z]}$ as $E[Z] \to \infty$. Finally, although each delay Y_k^s may span several source IPs, condition $(|s| - a_s)(1 - \chi_s)/m \to \infty$ ensures that *each* IP gets enough 0s in $\{J_k^s\}_k$ to invoke the Chen-Stein theorem and keeps the overall result asymptotically accurate. □

Fig. 23 compares simulations to (16) under the same default conditions as in Fig. 22. Results show that T, m, and $|s|$ do not influence the accuracy of the model



(a) default

(b) $T = 10,800$ sec

(c) $T = 12,500$ sec

(d) $T = 700$ days

Fig. 23. Comparison of post-permutation IDS-B model (16) to simulations (default parameters $|s| = 2^8, \Delta_s = 60$ sec, $a_s = 4$, and $m = 1$).

if threshold a_s is small compared to $|s|$ (i.e., the error is below 0.1% for $a_s = 4$ and subnet sizes as small as 2^8). However, significantly larger a_s create too much dependency among consecutive delays $\{Y_k^s\}_k$ leading up to detection and result in a more serious mismatch with the model (shown in part (b) of the figure). Increasing $|s|$ or lowering a_s fixes the problem.

As with IDS-A, uniform scanners are IP-scalable against IDS-B if and only if they use post-permutation split, which can be inferred from the ωT term in (15). Since our analysis shows that pre-permutation carries no benefit, we omit its further discussion in the rest of the chapter.

4.4.4 Uniform Cover Time

We next examine the time needed for the uniform permutation to cover a particular subnet. In order to determine this metric, we first relax the definition of SCT since uniform scanners can never achieve $\rho^s(r) = 0$ with finite T. For a pattern X, define the ϵ-SCT $T_X^s(\epsilon)$ of a normal subnet $s \in \mathcal{I}_N$ to be the minimum duration T in which X can reduce the detection probability at s below ϵ, i.e., $T_X^s(\epsilon) = \inf\{t \geq 0 : \rho^s(n/t) \leq \epsilon\}$. We similarly relax the definition of k-faster and IP-scalable to operate in terms of ϵ-SCT instead of SCT.

This leads to the following approximation.

Theorem 7. *Define $c = 1/(\beta - 1)$. Then, for $\epsilon \to 0$ and $|s| \gg \beta$, the ϵ-SCT of a β-aware uniform permutation is asymptotically*

$$T_U^s(\epsilon) \approx \frac{\alpha|s|\Delta_s}{\omega} \begin{cases} e^{\eta_1}(\beta!)^{-c} & \textit{IDS-A} \\ e^{\eta_2}\eta_3^{-1} & \textit{IDS-B} \end{cases} \tag{19}$$

where

$$\alpha = \Big(\frac{|s|}{-\log(1-\epsilon)}\Big)^c, \qquad \eta_1 = W(-c(\beta!)^c/\alpha), \tag{20}$$

$$\eta_2 = W(-c/\alpha), \qquad \eta_3 = \sum_{j=0}^{\infty} \frac{(\alpha e^{\eta_2})^{-j}}{j+1}, \tag{21}$$

and $W(.)$ *is Lambert's function.*

Proof. Since $\rho^s(r) = \epsilon$ is asymptotically small, one can make a number of approximations that greatly simplify inversion of (13) and (16). For small x, we use Taylor expansions $(1-x)^y \approx e^{-xy}$, $1 - e^{-x} \approx x$, and $\log(1-x) \approx -x$. We also neglect β in comparison to $|s|$, i.e., $|s| - \beta \approx |s|$.

Without a-priori knowledge of a_s, a uniform scanner must assume that counter $C_i^s(t)$ reaching β triggers detection for both IDS-A/B. This means (13) and (16) must undergo inversion with a_s replaced by β. For IDS-A and constant $|s|$, observe that $\epsilon \to 0$ implies $q \to 0$ and the leading term of ϕ_{bin}^s is

$$\phi_{bin}^s \approx \binom{|s|}{\beta} q^\beta (1-q)^{|s|} \approx \binom{|s|}{\beta} e^{\beta \log q - |s|q}. \tag{22}$$

Recalling that $\rho_A^s(r) \approx 1 - (1 - \phi_{bin}^s)^{1/q}$, we have

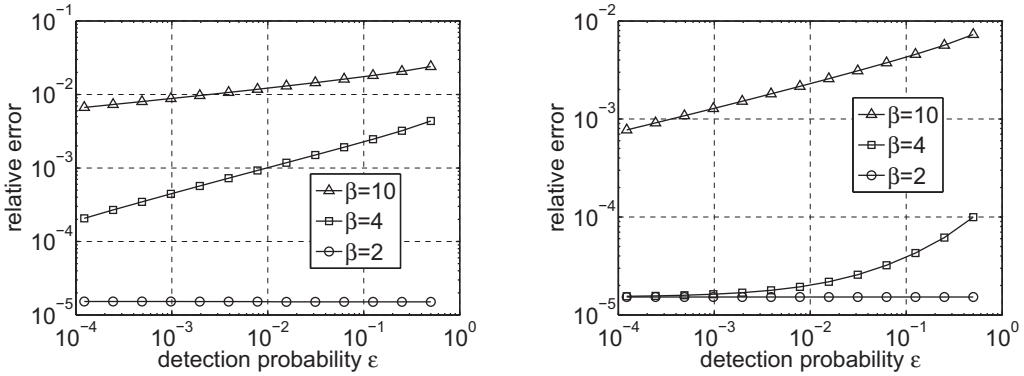
$$\log(1-\epsilon) \approx \frac{\log(1-\phi_{bin}^s)}{q} \approx \frac{-\phi_{bin}^s}{q}. \tag{23}$$

Using (22) in (23) and taking log of both sides, we get

$$\log\Big(\frac{-\beta! \log(1-\epsilon)}{|s|^\beta}\Big) \approx (\beta - 1)\log q - |s|q. \tag{24}$$

This equation is of the general form $ay + b\log y = c$, where $y = q$, whose solution using Lambert's $W(.)$ function is given by

$$y = \exp\Big[-W\Big(\frac{ae^{c/b}}{b}\Big) + \frac{c}{b}\Big]. \tag{25}$$



(a) IDS-A (b) IDS-B

Fig. 24. Relative error between the binary-search SCT and its closed-form approximations ($|s| = 2^{16}, \Delta_s = 60$ sec, $m = 1$).

Applying this result to (24) and recalling that $q = \Delta_s/\omega T$, we arrive at the first line of (19).

For IDS-B, observe that (16) can be written as

$$-\log(1-\epsilon) \approx |s|(1-\chi_s)\chi_s^{\beta-1}. \tag{26}$$

Since $\chi_s \to 0$, we have $\log(1-\chi_s) \approx -\chi_s$ and

$$\log\Big(\frac{-\log(1-\epsilon)}{|s|}\Big) \approx -\chi_s + (\beta-1)\log\chi_s, \tag{27}$$

which again has shape $ay + b\log y = c$ for $y = \chi_s$. Solving (27), we get $\chi_s = e^{-\eta_2}/\alpha$. Expanding $\chi_s = 1-(1-q)^{|s|}$ and applying log to both sides, we have

$$\frac{\log(1-\frac{e^{-\eta_2}}{\alpha})}{|s|} \approx \log(1-q) \approx -q = \frac{-\Delta}{\omega T}. \tag{28}$$

Substituting $-\log(1-z) = z(1+z/2+z^2/3+\ldots)$ with $z = e^{-\eta_2}/\alpha$ into (28), we end up with the second line of (19). □

Fig. 24 shows the relative error between approximations (19) and the corre-

sponding ϵ-SCT found using binary search on models (13), (16) as $\epsilon \to 0$. For $\beta = 2$, the latter is so close to the former that their relative difference is initially less than 10^{-5}, which falls below Matlab's default precision for binary search and explains why it does not improve as $\epsilon \to 0$. The other two curves in each subfigure show monotonic decay as a function of ϵ, with the IDS-B model generally agreeing better with the original than IDS-A. This arises from the extremely crude approximation in Theorem 7 to the binomial distribution in (13). For larger β, the error is generally more pronounced and decays slower since the magnitude of the omitted terms is higher; however, in all cases in the figure it stays below 2.4% (including $\epsilon = 0.5$).

4.4.5 Discussion

We finish this section by analyzing the relative performance of the various algorithms. As $\epsilon \to 0$, the numerous constants in (19) disappear. Specifically, α becomes large and $\eta_1 \to 0, \eta_2 \to 0, \eta_3 \to 1$, which leads to

$$T_U^s(\epsilon) \approx \frac{|s|^{1+c}\Delta_s}{\omega\gamma\epsilon^c}, \quad \text{where } \gamma = \begin{cases} (\beta!)^c & \text{IDS-A} \\ 1 & \text{IDS-B} \end{cases}.$$

First, observe that uniform is SCT-slower against IDS-B by a factor of $(\beta!)^c$ than against IDS-A. This term is always no smaller than 2 and is approximately $(\beta/e)^{1+c}$ for $\beta \gg 1$. While for IP-sequential this ratio is always $\beta - 1$ and for SO it is 1, the uniform permutation splits these two extremes somewhere in the middle as $\beta \to \infty$.

Second, notice that $T_U^s(\epsilon)$ is proportional to $|s|^{1+c}$, which may scale quite aggressively as $|s|$ becomes large (e.g., quadratically for $\beta = 2$). Because of this, uniform becomes SCT-slower than IP-sequential for any s with $|s| > n_0$, where

$$n_0 = \left(\frac{n\gamma\epsilon^c}{\zeta}\right)^{\frac{\beta-1}{\beta}}, \tag{29}$$

which has not been previously documented and is quite counter-intuitive.

For $\beta = 2$, this translates into $n_0 = \sqrt{\gamma n \epsilon}$. Assuming the desired detection probability $\epsilon = 10^{-3}$ (i.e., on average, one in $1,000$ subnets detects the scan), IP-sequential is faster against IDS-A on any network with more than $2,930$ IPs and against IDS-B with more than $2,072$ IPs (i.e., these roughly map to /20 and /21 subnets). However, as β increases, (29) quickly rises as well. For $\beta = 4$, the corresponding thresholds are 14.1M (IDS-A) and 9.9M (IDS-B), which are large enough (i.e., /8 or bigger) to conclude that uniform is superior to IP-sequential in all but a handful of cases. Its average probing rate $|s|/T$ of each s is also much better than IP-sequential's.

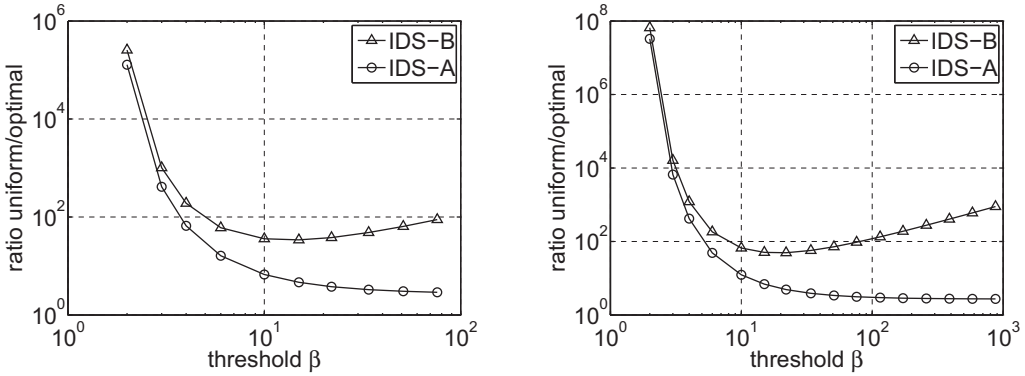
Third, even though for some scan patterns two sets of IDS-A parameters are equivalent if ratio $\Delta_s/(a_s - 1)$ (i.e., the average allowed gap between packets) remains constant, this is not the case against the uniform permutation. Lowering Δ_s while keeping the ratio constant actually *increases* the uniform cover time and makes IDS-A perform better at detecting the scanner. Thus, for example, combination $(15, 2)$ is much stricter than Snort's default $(60, 5)$ even though both allow on average 1 scan packet per 15-second interval.

Our final observation is that the stealth-optimal pattern is

$$\pi(\epsilon) = \frac{T_U^s(\epsilon)}{T_O^s} = \frac{|s|^c(\beta - 1)}{\gamma \epsilon^c} \tag{30}$$

times SCT-faster than uniform. This ratio is plotted in Fig. 25 for two subnet sizes. In both subfigures, (30) for IDS-A starts at $|s|/2\epsilon$ for $\beta = 2$ and converges toward e as $\beta \to \infty$. For IDS-B, it starts at double the IDS-A value and never drops below its global minimum $\pi_0 = e \log(|s|/\epsilon)$ achieved at $\beta_0 = \pi_0/e + 1$. This shows that regardless of β, the SO pattern is at least π_0-faster against IDS-B than uniform. For the examples in the figure, this is 33.8 and 48.9, respectively.

In summary, this section has shown that the uniform pattern performs signif-



(a) $|s| = 2^8$ (b) $|s| = 2^{16}$

Fig. 25. Ratio $\pi(\epsilon)$ for $\epsilon = 10^{-3}$.

icantly worse than might have been expected, especially when β is small. For the default $\beta = 4$ and $\epsilon = 10^{-3}$ used in this section, SO scanners in /16 subnets are 419-faster than uniform when facing IDS-A and 1,209-faster when facing IDS-B, which leads to our next topic of how to leverage these findings in practice and achieve SCT-optimality globally.

4.5. Stealth Scanning

To show that a stealth-optimal scanner is possible and test it in practice, this section develops permutation, split, and scheduling algorithms for achieving USSO simultaneously in all CIDR subnets. To keep track of the various pieces and how they map to our earlier definitions, assume $SO(\beta, m)$ is the set of stealth-optimal patterns for some $\beta \geq 2$ and $m \geq 1$. We define sets $SSO(\beta, m)$ and $USSO(\beta, m)$ similarly.

4.5.1 Permutation

Our first goal is to tackle the simplest case $SO(2, 1)$. We start by formulating a condition that is simpler to satisfy, show its equivalence to $SO(2, 1)$, and then develop

methods that achieve it.

Definition 6. *For $s \in \mathcal{S}$, any permutation that returns to s with a period of $n/|s|$ is called* IP-wide at s. *A permutation that achieves this for all s is called* globally IP-wide *(GIW).*

It is not difficult to show that a permutation covers every s in time $|s|\Delta_s$ if and only if it is GIW. Since this SCT is optimal for $\beta = 2$ and $m = 1$, we obtain that set $GIW = SO(2, 1)$. To implement GIW, visualize permutation $\mathcal{F}'$ as a binary tree of depth 32, where target IPs reside in leaves and all edges are labeled with 0/1 bits. A permutation can be viewed as a process that traverses the tree and accumulates individual bits along the edges into the next IP. Decisions to move left (L) or right (R) at internal gateways v depend on their state θ_v, which must be altered each time to ensure that all generated IPs are unique. If the state of each visited node is flipped during traversal, we call this structure an *alternating gateway tree* (AGT). Fig. 26(a) shows the bottom four levels of some random AGT whose next generated IP address ends with bits 011 and the other after that with 101.

Since balanced binary trees have well-defined rules for calculating the offset of each internal node, AGTs do not require storing child pointers. Thus, their RAM overhead is $(n - 1)/8 = 512$ MB needed to store tuple $(\theta_1, \ldots, \theta_{n-1})$ and their computational complexity is 26 memory reads/writes (i.e., 52 total) per generated IP (assuming depth-31 traversal and 64-bit lookups that yield the first 5 levels of the tree in one RAM access).

Theorem 8. *A permutation is $SO(2, 1)$ if and only if it can be realized by an AGT. Furthermore, $|SO(2, 1)| = 2^{n-1}$.*

Proof. For the first part, induction on the depth of each subnet in AGT shows that it is equivalent to GIW, which in turn is equivalent to $SO(2, 1)$. For the second part,

notice that the number of unique AGT permutations is simply the number of unique seeds $(\theta_1, \ldots, \theta_{n-1})$, which trivially equals 2^{n-1}. □

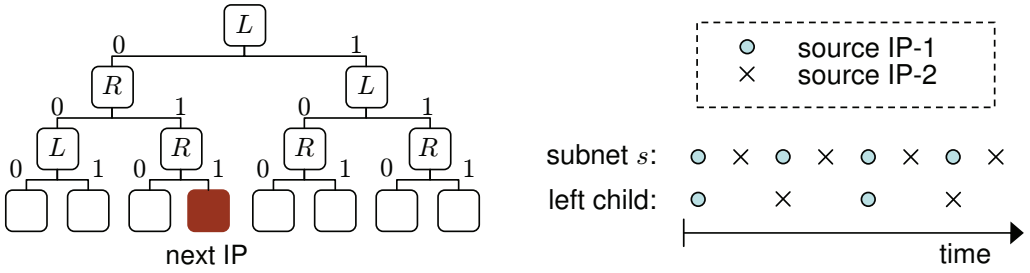
Note that 2^{n-1} is smaller than the total number of permutations of $\mathcal{F}$ (i.e., $n!$), but nevertheless enormous (i.e., $10^{1,292,913,986}$). In practice, one does not require this much diversity in their ability to scan the IP space and other algorithms with fewer unique permutations are quite sufficient. One reason to seek alternatives is that AGT requires huge overhead during checkpointing and transmission of state in distributed implementations. Another reason is that AGT's CPU complexity is quite high and leaves room for improvement, which we achieve next.

Recalling that ${}_bx$ is the lower b bits of x, define ${}_b\bar{x}$ to be ${}_bx$ with its bits reversed and consider our next result.

Theorem 9. *Given a sequence of integers $\{x_k\}_{k=1}^n$, suppose $\{{}_bx_k\}_k$ have full periods for all $b = 1, 2, \ldots, 32$. Then, permutation $\{{}_{32}\bar{x}_k\}_k$ is GIW.*

Proof. Assume that s has depth b in the AGT (i.e., $n/|s| = 2^b$) and observe that GIW patterns must visit all remaining $2^b - 1$ subnets at depth b before returning to s. In practice, this means that the permutation must exhibit a full period in the upper b bits. Since this holds for all s, the full period must be maintained at all depths $1 \leq b \leq 32$. Reversing the bits in each IP, we replace this condition with a much simpler one – the full period must hold in the lower b bits, which is equivalent to the statement of the theorem. □

While complex dependency between the elements of $\{x_k\}_k$ is possible, we limit ourselves to Markovian sequences $x_k = h(x_{k-1})$. This keeps scanner overhead minimal – both state x_k and seed x_0 consist of one integer and the CPU complexity is that of computing $h(.)$ and reversing the bits. To maximize the speed of generating the



(a) AGT (b) $SSO(2,2)$

Fig. 26. Illustration of AGT and $SSO(2,2)$.

permutation, one natural choice is to consider only linear $h(.)$, which leads us to the concept of an LCG (linear congruential generator). Recall that LCGs are recurrences $x_k = ax_{k-1} + c$, whose well-known properties yield the following.

Theorem 10. *An LCG $\{x_k\}_k$ satisfies Theorem 9 if and only if $a-1$ is divisible by 4 and c is odd.*

We call the resulting permutation *Reversed LCG* (RLCG) and note that $RLCG \subseteq GIW$. To efficiently reverse the bits, one can use a 2-byte hash table that flips the order of bits in 16-bit integers. Therefore, any 32-bit IP can be processed in two memory lookups (i.e., 26 times faster than AGT); however, the CPU cache often makes this operation run even faster in practice.

4.5.2 Split

We now augment GIW with a novel split algorithm that puts it into set $SSO(2,m)$. While block-split has been frequently used in related work, its ability to regulate the burstiness of different source IPs at each s is lacking. Indeed, block-split can be visualized as m AGTs assigned to m scanner hosts, where each AGT starts with a different seed. However, synchronizing the order and schedule of m trees to ensure perfectly non-bursty operation at each s is a challenging problem.

To overcome this setback, we introduce a new partitioning scheme called *round-robin* (RR) that assigns the j-th element of $\mathcal{F}'$ to host $j \bmod m$. The corresponding scheduler in each host $i \in \mathcal{M}$ needs to know the main seed x_0, its own position i, and the total number of scanner IPs m. It then generates the entire sequence $\{z_k\}_k$ locally and hits target z_{i+jm} at time $(i + jm)T/n$ for $j = 0, 1, \ldots, n/m$. Note that perfect synchronization of start times is unnecessary as inter-IP delays at each s are quite large (e.g., 337 seconds at /24 and 1.31 seconds at /16 for $T = 24$ hours).

Since there is only one AGT and inter-packet delays are fixed, RR-split ensures that any GIW permutation that visits each $s \notin \mathcal{I}_{ST}$ with all m IPs is $SSO(2, m)$. We next examine what values of $m \geq 2$ guarantee this condition.

Theorem 11. *RR-split with any GIW permutation scans each s with* $\min(|s|, m_s)$ *sources, where*

$$m_s = \frac{m}{\gcd(\frac{n}{|s|}, m)} \tag{31}$$

and $\gcd(a, b)$ *is the greatest common divisor of* (a, b).

Proof. Examine permutation $\{z_k\}_k$ and assume it is GIW. For a given subnet s, observe that its IPs appear in this list with a period $n/|s|$, which follows from the definition of GIW. Assuming $w_j \in \mathcal{M}$ is the j-th IP that hits s, we have

$$w_j = \left(w_{j-1} + \frac{n}{|s|}\right) \bmod m, \quad j = 1, 2, \ldots \tag{32}$$

This recurrence is an additive-only LCG whose period [9] is given by (31), which means that the number of sources scanning s is the smaller of its size and m_s. □

Defining $\mathbb{N}$ to be the set of natural numbers, this observation leads to the following.

Theorem 12. *RR-split with any GIW permutation is SSO for odd m, i.e., $\forall k \in \mathbb{N}$: $GIW/RR \subseteq SSO(2, 2k-1)$. Furthermore, set $SSO(\beta, 2k)$ is empty for any $k, \beta \in \mathbb{N}$.*

Proof. Since odd m produces $m_s = m$ and even m leads to $m_s \leq m/2$, the first part of this theorem follows from Theorem 11 and the discussion immediately preceding it.

The second part we prove by contradiction using $\beta = m = 2$ (generalization to larger values is straightforward and omitted for brevity). Suppose s sees an equally spaced $SSO(2, 2)$ sequence of packets from two alternating source IPs as shown on top of Fig. 26(b). Without loss of generality, assume the first packet that arrives to s is destined to its left child, whose $SSO(2, 2)$ pattern is also drawn in the figure right below that of s. Now notice that this combination of patterns is mutually exclusive, which means that $SSO(2, 2)$ cannot be achieved globally by *any* algorithm. □

While GIW permutations coupled with even m and block-split keep the method in set $SO(2, m)$, such choices of m are quite disastrous with RR-split as they at least double the pattern's SCT and make it no longer optimally stealthy. Selecting m as a power of 2 is the worst choice of all such options as it produces $m_s = 1$ for all s smaller than n/m.

4.5.3 Schedule

We now build upon GIW/RR to achieve $SSO(\beta, 2k-1)$ for any $\beta \geq 2$ and $k \in \mathbb{N}$. We first explain this algorithm using the AGT and then transform it to the RLCG. Define $d = 32 - \lfloor \log_2(m(\beta - 1)) \rfloor$ to be the depth at which subnets become size-trivial. The main challenge in achieving $SSO(\beta, m)$ is to send $\beta - 1$ back-to-back probes to each s above level d, but then spread out and use unaware probing below d. Using AGT, this can be accomplished by traversing the tree $\beta - 2$ times and flipping gateways

Algorithm 2 $SSO(\beta, m)$ at each source IP

```
1: d = 32 − ⌊log2(m(β − 1))⌋                          ▷ Size-trivial depth
2: start = rand()                                      ▷ Initial seed
3: totalB = 0                                          ▷ Total bursts generated
4: while start != EOS do
5:     for j = 0 to β − 2 do
6:         lcg[j].Init(start)                          ▷ Set the seed
7:         lcg[j].Skip(j2^d)                           ▷ Jump forward
8:     end for
9:     for k = 1 to 2^d do                             ▷ Iterate through 2^d bursts
10:        ip = totalB mod m                           ▷ Assigned source IP
11:        totalB++                                    ▷ Next burst
12:        for j = 0 to β − 2 do
13:            x = lcg[j].Next()                       ▷ Advance LCG
14:            if (x != EOS) AND (ip is ours) then
15:                y = ReverseBits(x)
16:                if y is valid then
17:                    probe(y)                        ▷ Hit destination
18:                end if
19:            end if
20:        end for
21:        Sleep(T(β − 1)/n)                           ▷ Wait for next burst
22:    end for
23:    start = lcg[β − 2].Current()                    ▷ Get current state
24: end while
```

only at depth no smaller than d. The last $(\beta - 1)$-st traversal flips all 32 gateways along the path to ensure that the next burst proceeds according to GIW.

The above algorithm can be implemented using $\beta - 1$ RLCGs maintained by each source IP. Specifically, assume that the main RLCG is in position k in its permutation $\{z_k\}_k$ and that the scanner needs to generate the next $\beta - 1$ targets $y_0, \ldots, y_{\beta-2}$ in a burst. The first target y_0 is simply z_k. Since the remaining $\beta - 2$ destinations do not change the top d levels of the tree, they can be found in the permutation where $\{z_k\}_k$ returns to the same subnet at level d. This is equivalent to skipping forward by 2^d elements each time. This leads to

$$y_j = z_{k+j2^d}, \quad j = 0, 1, \ldots, \beta - 2. \tag{33}$$

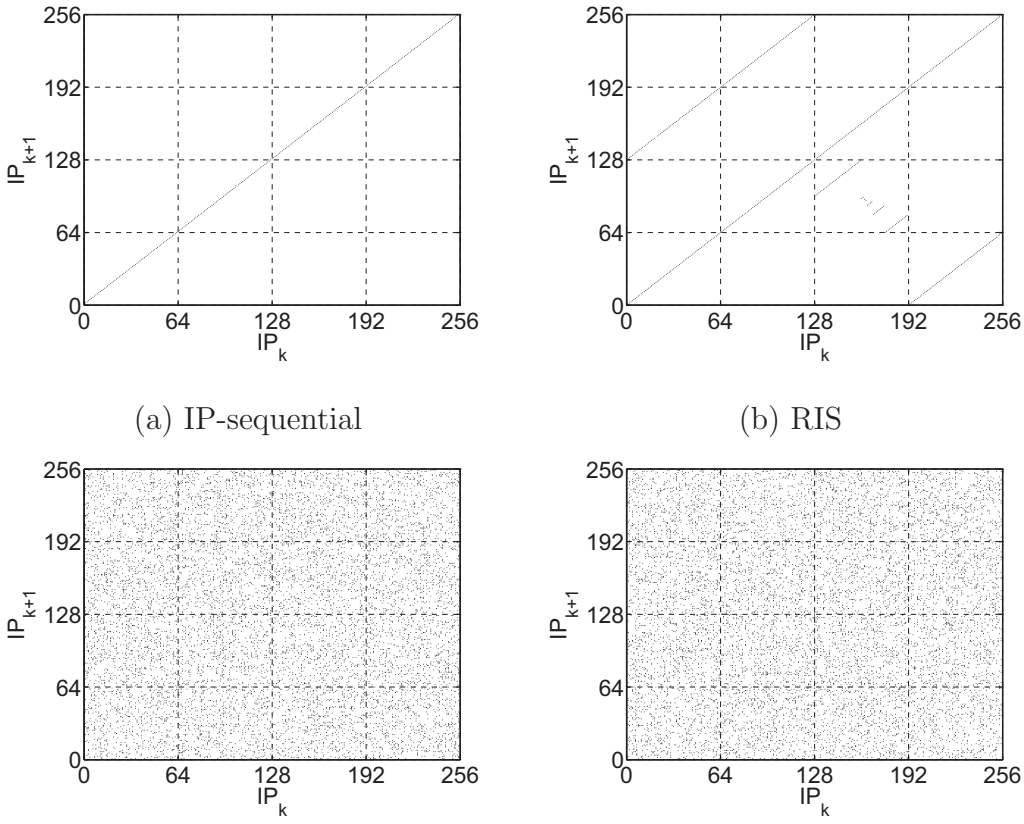
To avoid having to re-generate the entire sequence for each burst, the scanner

operates with $\beta - 1$ LCGs, each pointing to a different part of the original sequence as shown in Lines 6-7 of Algorithm 2. After an LCG wraps back to the original seed, it returns a special EOS (end of sequence) IP address. Algorithm 2 applies RR-split at the *burst* rather than packet level in Lines 9-20 and the LCGs are advanced for every packet in a burst (transmitted or not) in Line 13. Our last note is that target y in Line 16 may be invalid if it falls outside the scanned space (e.g., IANA-allocated IP blocks).

4.5.4 Correlation

Our last step is to check RLCG for correlation and determine whether its targets are sufficiently randomized. With a properly designed LCG, sequence $\{x_k\}_k$ is uniformly random and uncorrelated, with the exception of having full periods in all lower b bits. It then follows that its bit-reversed version $\{z_k\}_k$ is also uniformly random and uncorrelated, with the exception of having full periods in all upper b bits. Thus, it could be argued that among all GIW sequences, RLCG is as uncorrelated as one can expect to achieve. One common test for correlation in random number generators is to examine adjacent pairs (z_k, z_{k+1}) of elements in $\mathcal{B}(b) = {}_b\mathcal{F}'$ and plot them on a 2D plane. Generally, the closer the number of unique points on the plot, which we call *diversity*, to 2^{2b} and the quicker this is achieved, the better the sequence. For example, IP-sequential's set $\mathcal{B}(b)$ consists of $n/2^b$ repeated patterns of 2^b numbers each, i.e., $\{0, 1, 2, \ldots, 0, 1, 2, \ldots\}$. The corresponding IP-sequential plot is a straight line with diversity 2^b as shown in Fig. 27(a).

As this chapter was being written, we became aware of another such pattern [35], which we call *reverse IP-sequential* (RIS). The recurrence of this approach is $x_k = x_{k-1} + 1$ and the permutation is given by $z_k = ({}_{32}\bar{x}_k)\ XOR\ Q$, where Q is some constant. While this pattern is GIW, its intra/inter-subnet targets are far from random.



(a) IP-sequential

(b) RIS

(c) uniform

(d) RLCG

Fig. 27. Correlation in $\mathcal{B}(8)$.

Its set $\mathcal{B}(b)$ consists of 2^b runs of $n/2^b$ constants, i.e., $\{x_1, x_1, x_1, \ldots, x_2, x_2, x_2, \ldots\}$, whose diversity is 2^{b+1}. In practice, this means that the scanner hits the same target in all $/(32-b)$ networks before moving on to another address. The corresponding correlation plot is shown in Fig. 27(b).

In contrast, the uniform permutation in part (c) of the figure exhibits a much more random pattern and diversity $9,231$ in the first 10K elements of $\mathcal{B}(b)$. We do not plot more points since it clutters the graph, but note that diversity rises to $64,841$ in the first 300K elements of $\mathcal{B}(b)$. Part (d) of the figure shows that RLCG with carefully chosen parameters (e.g., $a = 214,013$ and $c = 2,531,011$) also covers the entire 2D plane without much bias and numerical results place its diversity at $9,235$ among the same 10K points (for the larger sample, it is $64,861$). Additional tests (omitted for brevity) using the distribution of $z_{k+1} - z_k$, the autocorrelation function, and larger b confirm that RLCG satisfies the randomness definition of USSO.

4.5.5 Vertical Scans

USSO patterns can be extended to probe multiple ports at little additional cost. Assume that an attacker needs to scan the entire Internet on K unique ports. We use the general defense mechanism of Snort to outline how this could be done. One component of Snort, discussed earlier in the chapter, monitors horizontal scans by counting the number of local IPs contacted by each remote host. Another (independent) component counts how many ports on each internal IP have been hit from external sources in the same window Δ_s. Once this counter reaches some threshold P, Snort detects a vertical port-scan.

To bypass both components simultaneously, the idea is to modify USSO to hit each target y_j in (33) on exactly $P-1$ unique ports. If a given scan pattern requires time T in one horizontal scan to avoid detection, the hybrid method above achieves

the same detection footprint and covers K ports in $\lceil KT/(P-1) \rceil$ time units, which is a significant improvement over performing K full scans.

4.6. Experiments

In this section, we build a scanner based on USSO patterns and perform several Internet-wide scans to study detection rates at remote networks as β changes.

4.6.1 Methodology

Testing open-source Snort [105] and Bro [12] in our lab has confirmed the validity of the IDS model proposed in this chapter and showed that stealth-optimal scanning indeed bypassed both systems as long as target counter $C_i^s(t)$ was never allowed to reach its threshold a_s. We have also validated that β-aware scanning, first introduced in this chapter, performed significantly better against Bro than unaware scanning. While these results are encouraging, it is unclear whether β-aware patterns are stealthier in the actual Internet and how much IDS-B has been deployed. To answer these questions, one requires a fast Internet scanner and ability to verify detection rates at remote networks.

To address the first issue, we designed a high-performance scanner that uses raw IP packets (TCP SYN, ICMP, or UDP) to cover the Internet using USSO. While details of our implementation are presented in the last chapter, it should be noted that to reduce synchronization complexity and decrease cost we alias the m scanning IPs to one host and probe only the BGP-routable space (i.e., 2.11B destinations). Using $T = 24$ hours and TCP SYN packets, this amounts to approximately 16 Mbps of traffic (including MAC-layer overhead).

The second issue, however, is more challenging since verification of detection by

remote systems is complicated by our lack of knowledge about the location of IDS, the size and boundaries of the networks they protect, customization of parameters Δ_s and a_s, and most importantly lack of access to remote alert logs. Fortunately, certain network administrators participate in online collaborative systems [71], [94], whereby they submit firewall and IDS logs for aggregation and public consumption in an effort to reduce the time necessary to detect distributed scans or large-scale worm attacks.

For the SANS Internet Storm Center (ISC) [94] that we focus on in this chapter, network administrators preprocess their logs and submit packet headers that have been deemed suspicious by either their IDS or firewall log analyzer [84]. Reports compiled against our IPs and posted by ISC allow us to determine the exact number of alarms/reports raised against a particular scan.

4.6.2 Results

We performed three tests in August 2009, each employing USSO and consisting of a full Internet-wide HTTP SYN scan. We chose TCP as attackers are more likely to target TCP services in the Internet and because of the evidence in [25], [35] suggesting that administrators are more sensitive to TCP than other protocols (in fact, [35] notes that TCP scans are 30 times more likely to receive complaints than ICMP scans). We use $T = 24$ hours, which is 90 times faster than any prior Internet-wide TCP scan in the literature [10], and $m = 61$ IPs available in our subnet.

The first scan (i.e., HTTP_1) establishes a baseline for comparison and uses the unaware pattern (i.e., $\beta = 2$). From Table XIV, observe that HTTP_1 attracts a combined total of 29K reports for all m IP addresses. Since all three scans run at the same average rate per IP, they should result in the same amount of ISC activity, *unless* IDS-B devices exist among ISC contributors. Since the lowest known detection

Table XIV. ISC reports

Scan	T	m	β	Reports	Hosts Found
$HTTP_1$	24h	61	2	29,869	44.3M
$HTTP_2$	24h	61	4	18,470	44.0M
$HTTP_3$	24h	61	5	23,969	44.5M

threshold for IDS-B is 4 (see Table XIII), we skip $\beta = 3$ and directly test in $HTTP_2$ the impact of $\beta = 4$. This leads to a 38% reduction in the number of reports, which confirms that IDS-B does in fact exist in the Internet and is responsible for a large fraction of overall detection.

We finish our evaluation by increasing β to 5 in $HTTP_3$, which results in 5.5K additional alarms compared to $HTTP_2$. The only explanation for this increase is IDS-B with $a_s = 4$, which gets tripped by our scanner with $\beta = 5$ but not $\beta = 4$. It can then be concluded that almost half of 11.4K IDS-B reports in the table are generated from networks with $a_s = 4$, coincidentally the default value in Bro. The remaining 5.9K IDS-B alarms in the table come from networks with $a_s \geq 5$, whose less-prevalent existence was anticipated given the values in Table XIII.

It should also be noted that the majority of the 18K reports in $HTTP_2$ are likely from unavoidable networks and those with extreme combinations (Δ_s, a_s), such as those of Psad and TRW in Table XIII. The total number of email complaints we received over these three scans was 11, none of which were generated by automated tools. In fact, most of these came from end-users relaying messages from their personal firewalls (such as ZoneAlarm and Norton).

All three scans find approximately the same number of alive web servers, which shows that parameter β in the range $2 - 5$ does not have much impact on the ability of a scanner to find open services in the current Internet.

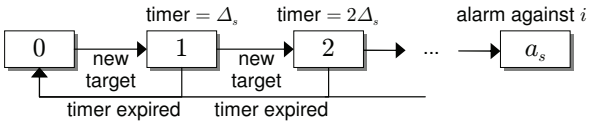


Fig. 28. Process $C_i^s(t)$ of IDS-C.

Table XV. Comparison of RAM Usage

	Juniper			Bro		
Method	a_s	Δ_s	RAM	a_s	Δ_s	RAM
IDS-A	50	120s	247 KB	20	600s	514 KB
IDS-B	50	120s	305 KB	20	600s	649 KB
IDS-C	50	2.4s	78 KB	20	31.5s	202 KB

4.7. Defense

Observe that SO leverages the fact that it costs nothing for the scanner to raise the counter $C_i^s(t)$ to $a_s - 1$ and then wait for Δ_s units before sending another burst. This allows it to scan both IDS-A/B at an average rate of $(a_s - 1)/\Delta_s$ packets per second (pps), which contributes to its optimal stealth cover time across the entire Internet (i.e., full scans finish quicker).

To discourage such exploits, we propose a new model called IDS-C that linearly increases the timeout duration based on the current state of counter $C_i^s(t)$, as shown in Fig. 28. This modification allows IDS-C to detect *all* scanners with long-term rates faster than $1/\Delta_s$ pps, even if they utilize SO. In practice, this means that IDS-C can lower its interval Δ_s by a factor of $a_s - 1$ while offering the same stealth protection against SO as IDS-A/B. This in turn leads to smaller RAM overhead since entries in the table expire quicker and unique-target lists are shorter.

To verify these conclusions, we perform a simulation using CAIDA's 1-hour trace

with 9.6M flows and 117M packets from an OC48 link [19]. We record all transitions made by the three types of IDS in response to incoming traffic to a popular /16 target subnet, as well as the steady-state amount of RAM being used by each algorithm. For this comparison, we utilize Juniper's and Bro's default IDS settings from Table XIII as examples of low and high alarm rates (i.e., 3/hour and 50/hour, respectively). Table XV shows that the typical savings in terms of RAM for IDS-C in comparison to the other two methods amounts to $60 - 75\%$. Since the list of targets tracked by IDS for each source IP is smaller, the CPU overhead needed to verify that incoming packets belong to an existing target is also lower.

4.8. Implications

This chapter investigated a potentially sensitive issue of avoiding IDS and dissected the algorithms of currently deployed window-based tools. While our interest is purely to propose a novel model of IDS operation and understand its fundamental limitations, one concern might be that attackers could benefit from stealth-optimal scan patterns exposed in this work and thus could inflict certain damage that would not otherwise be possible.

However, we do not believe this to be the case. First, as hackers must constantly remain two steps ahead of the security community to be able to exploit the implemented defenses, our results are not necessarily novel or useful to them. Second, scanning by itself does not compromise hosts; instead, intrusion using malicious payload (e.g., delivered through unsolicited packets or email) does. As a result, many networks should remain well protected despite the findings of this chapter. Third, botnets afford hackers such a diverse pool of IPs that they often do not care to remain stealthy and rely on the most basic sequential probing [3], which apparently is

sufficient for their purposes.

This chapter should be viewed as analyzing the worst-case scenario of a Flash worm and its coordinated delivery of new exploits from a large set of IPs. The main challenge in defending against this kind of attack lies in the impossibility of lowering IDS thresholds and triggering their estimators sooner. While setting the threshold to 1 packet detects all scanners, it also raises the false-positive rate beyond the level that typical administrators can manage [112]. Our findings therefore emphasize the importance of deploying algorithms that provide higher accuracy, produce lower false-positive rates, and require fewer packets. The reality of the situation, however, is that Snort and its derivatives continue to account for an overwhelming majority of the IDS market (contrary to what might be inferred from Table XIV, which is a limited sample of Internet IDS). Algorithms such as Bro TRW [43] not only improve Snort by using the IDS-B model, but also by applying a much better estimator to observed traffic.

4.9. Summary

This chapter introduced a novel formalization of scanner algorithms and IDS detection rules related to horizontal scanning. We thoroughly investigated the detection probability of previous scan patterns and brought awareness to the existence of low-overhead algorithms for stealth-optimal scanning, which can remain undetected at much faster rates compared to the known approaches. We also suggested a simple, yet effective, technique for making windows-based IDS expiration rules robust against stealth-optimal scanners. This method is versatile and can be applied to any existing or future implementation, regardless of the actual detection algorithm.

CHAPTER V

SUMMARY AND FUTURE WORK

5.1. Summary

This dissertation was motivated by a need for efficient algorithms capable of performing Internet-wide measurement studies using existing or inexpensive resources. We next summarize our contributions in this area, considering each chapter of the dissertation in turn.

5.1.1 Turbo King

Distance estimation and topological proximity in the Internet have recently emerged as important problems for many distributed applications [1], [24], [28], [49], [72], [76], [98], [99], [111]. Besides deploying tracers and using virtual coordinates, distance is often estimated using end-to-end methods such as King [32] that rely on the existing DNS infrastructure. However, the question of accuracy in such end-to-end estimation and its ability to produce a large-scale map of Internet delays had never been examined. We tackled this problem initially by showing that King produces biased latency estimates given common DNS deployments of geographically diverse authoritative servers and forwarders, requires significant cache pollution at remote servers, and employs large traffic overhead for Internet-wide measurements. To overcome these drawbacks while still using the ubiquitous DNS infrastructure, we proposed the Turbo King latency estimation framework that obtains end-to-end samples without suffering from the bias endured by King. Turbo King also reduces cache pollution of remote servers by several orders of magnitude and consumes half the bandwidth

required by King. We performed several experiments to validate our claims about King and to demonstrate that T-King is more accurate than prior methods.

5.1.2 IRLscanner

Motivated by our need with Turbo King to discover many remote nameservers, recent interest in the literature [10], [25], [35], [83], and the many apparent obstacles, we next tackled the problem of performing Internet-wide service discovery measurements. Given the design objectives of maximizing politeness at remote networks, allowing Internet-wide scans that complete in hours on commodity hardware, and allowing for accurate extrapolations in partial scans, we developed a novel permutation/split algorithm and several other features that culminated in IRLscanner. To verify its effectiveness, we used IRLscannerand 24-hour scan durations to perform 20 Internet-wide experiments spanning ICMP, UDP (i.e., DNS, ECHO), and TCP (i.e., HTTP, SMTP, EPMAP) and targeting both very popular ports (i.e., DNS, HTTP) and those used by hackers and scammers (i.e., UDP ECHO, SMTP, EPMAP). In addition, we performed the first Internet-wide OS fingerprinting of HTTP servers, presented an alternative method for determining server uptime, and used ACK scans to stealthily detect live hosts behind stateless firewalls. We concluded this chapter by analyzing the various forms of feedback received during our measurements (e.g., email complaints, IDS alarms, DNS lookups) in an effort to inform researchers interested in similar studies.

5.1.3 Modeling Window-based IDS and Stealth Scanning

Our interest in horizontal scanning derived from working on IRLscanner and a desire to understand its effect on remote networks led us to explore IDS, which have become ubiquitous in the defense against virus outbreaks, malicious exploits of OS vulnerabil-

ities, and botnet proliferation. While an IDS is often used by network administrators to detect reconnaissance scans preceding attempted penetration, it was previously unknown how likely an IDS was to detect a given Internet-wide scan pattern and whether fast stealth techniques existed that could largely remain undetected at Internet-scale. We tackled this problem first by proposing a simple analytical model for the window-expiration rules of popular IDS tools (i.e., Snort and Bro), then used a variation of the Chen-Stein theorem to derive their respective detection probabilities for existing scan techniques (i.e., uniform and sequential). When finally showed through both analysis and several Internet-wide experiments that stealth-optimal patterns exist and are effective, then proposed a simple modification to existing algorithms that proved both highly effective and provided a more efficient use of resources under real-world traffic.

5.2. Future Work

As demonstrated by both experiments and analysis, the work presented in this dissertation allows for the possibility of significant additional research. We consider three different areas of future work in turn.

5.2.1 Turbo King

Future work includes running T-King in active mode to generate the first Internet-wide all-to-all map of distances between BGP networks, which given the Internet's current size would require more than 40B measurements. Given this data, it also involves evaluating the plethora of distance estimators proposed in the literature, eventually releasing the distance map to other researchers for their use as well. Further, while a static map of distances would be useful, deployed applications require updated latencies that capture the current state of the ever-changing Internet. Future

work also includes deploying a system that provides approximate real-time estimates of distance, then using the resulting data would to study changes in latency over time in the Internet. This work then culminates in leveraging the knowledge gained by analyzing current techniques and our experience with measuring latencies to create a system that combines both theoretical approaches and actual latency estimates in the Internet.

5.2.2 IRLscanner

Future work involves more in-depth analysis of scan data from the 20 Internet-wide scans already performed. Further development of IRLscanner through exploring methods for reducing $\mathcal{B}$ to avoid scanning unproductive networks would allow for shorter durations, which would result in a more accurate snapshot of service availability. Future work also includes expanding RLCG/RR to provide optimal spacing for multiple destination ports (i.e., in hybrid vertical/horizontal scanning), which would allow us to enumerate and maintain an updated count of the number of hosts offering *every* available service. Finally, based on the promise shown by our initial OS fingerprinting, future work includes enhancing and developing new methods for operating systems and individual service fingerprinting.

5.2.3 Modeling Window-based IDS and Stealth Scanning

Future work involves detection of combined horizontal-vertical scan patterns, which can be used to confuse current IDS and avoid detection. It also includes developing more robust techniques for detecting distributed scans originating from multiple source IP addresses, which is critical to making stealth-optimal scans infeasible. Finally, future work involves a deeper analysis of IDS-C, development of more advanced techniques to thwart steal-optimal scans, and comparison of IDS RAM overhead un-

der various types of traffic to understand the practical limitations of window sizes in current IDS.

REFERENCES

[1] A. Adams, J. Mahdavi, M. Mathis, and V. Paxson, "Creating a scalable architecture for internet measurement," presented at the ISOC Internet Summit (INET), Jul. 1998.

[2] M. Allman, W. M. Eddy, and S. Ostermann, "Estimating loss rates with TCP," *ACM Performance Evaluation Review*, vol. 31, pp. 12–24, 2003.

[3] M. Allman, V. Paxson, and J. Terrell, "A brief history of scanning," in *Proc. ACM IMC*, Oct. 2007, pp. 77–82.

[4] R. Arratia, L. Goldstein, and L. Gordon, "Two moments suffice for poisson approximations: The Chen-Stein method," *The Annals of Probability*, vol. 17, no. 1, pp. 9–25, Jan. 1989.

[5] A. Baggio and M. van Steen, "Distributed redirection for the world-wide web," *Computer Networks*, vol. 49, no. 6, pp. 743–765, Dec. 2005.

[6] H. Ballani, P. Francis, and S. Ratnasamy, "A measurement-based deployment proposal for IP anycast," in *Proc. ACM IMC*, Oct. 2006, pp. 231–244.

[7] S. Banerjee, C. Kommareddy, and B. Bhattacharjee, "Scalable peer finding on the internet," in *Proc. IEEE GLOBECOM*, Nov. 2002, pp. 2205–2209.

[8] G. Bartlett, J. Heidemann, and C. Papadopoulos, "Understanding passive and active service discovery," in *Proc. ACM IMC*, Oct. 2007, pp. 57–70.

[9] H. Bauke and S. Mertens, "Random numbers for large-scale distributed monte carlo simulations," *Phys. Rev. E*, vol. 75, no. 6, p. 066701, 2007.

[10] D. Benoit and A. Trudel, "World's first web census," *Intl. Journal of Web Information Systems*, vol. 3, no. 4, pp. 378–389, 2007.

[11] J. Bethencourt, J. Franklin, and M. Vernon, "Mapping internet sensors with probe response attacks," in *Proc. USENIX Security*, Jul. 2005, pp. 193–208.

[12] Bro IDS. [Online]. Available: http://bro-ids.org/. (Accessed in Jul. 2009).

[13] H. Burch and B. Cheswick, "Mapping the internet," *IEEE Computer*, vol. 32, no. 4, pp. 97–98,102, 1999.

[14] M. Casado, T. Garfinkel, W. Cui, V. Paxson, and S. Savage, "Opportunistic measurement: Extracting insight from spurious traffic," presented at the 4th ACM Workshop on Hot Topics in Networks (Hotnets-IV), Nov. 2005.

[15] C. Chambers, W. Feng, and W. Feng, "A geographic redirection service for on-line games," in *Proc. ACM Multimedia*, Nov. 2003, pp. 227–230.

[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] Z. Chen and C. Ji, "Optimal worm-scanning method using vulnerable-host distributions," *ACM IJSN*, vol. 2, no. 1/2, pp. 71–80, Mar. 2007.

[18] B. Cheswick, H. Burch, and S. Branigan, "Mapping and visualizing the internet," in *Proc. USENIX Annual Technical Conference*, Jun. 2000, pp. 1–12.

[19] Cooperative Association for Internet Data Analysis (CAIDA). [Online]. Available: http://www.caida.org/. (Accessed in Oct. 2009).

[20] M. Costa, M. Castro, A. Rowstron, and P. Key, "PIC: Practical internet coordinates for distance estimation," in *Proc. IEEE ICDCS*, Mar. 2004, pp. 178–187.

[21] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris, "Practical, distributed network coordinates," *ACM SIGCOMM Comp. Comm. Rev.*, vol. 34, no. 1, pp. 113–118, Jan. 2004.

[22] Team Cymru. [Online]. Available: http://www.team-cymru.org/Services/Bogons/. (Accessed in Sep. 2009).

[23] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," in *Proc. USENIX NSDI*, Mar. 2004, pp. 85–98.

[24] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proc. ACM SIGCOMM*, Aug. 2004, pp. 15–26.

[25] D. Dagon, N. Provos, C. P. Lee, and W. Lee, "Corrupted DNS resolution paths: The rise of a malicious resolution authority," presented at the 15th Network and Distributed System Security Symposium (NDSS), Feb. 2008.

[26] L. Deri and F. Fusco, "Exploiting commodity multicore systems for network traffic analysis," July 2009. [Online]. Available: http://ethereal.ntop.org/MulticorePacketCapture.pdf.

[27] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee, "Polymorphic blending attacks," in *Proc. USENIX Security*, Jul. 2006, pp. 241–256.

[28] 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 Trans. Networking*, vol. 9, no. 5, pp. 525–540, Oct. 2001.

[29] L. Garcés-Erice, K. Ross, E. Biersack, P. Felber, and G. Urvoy-Keller, "Topology-centric look-up service," in *Proc. NGC*, Sep. 2003, pp. 58–69.

[30] R. Govindan and H. Tangmunarunkit, "Heuristics for internet map discovery," in *Proc. IEEE INFOCOM*, Mar. 2000, pp. 1371–1380.

[31] Y. Gu, A. McCullum, and D. Towsley, "Detecting anomalies in network traffic using maximum entropy estimation," in *Proc. ACM/USENIX IMC*, Oct. 2005,

pp. 345–350.

[32] 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.

[33] J. Guyton and M. Schwartz, "Locating nearby copies of replicated internet servers," in *Proc. ACM SIGCOMM*, Aug. 1995, pp. 288–298.

[34] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang, "Accessing the deep web," *Comm. of the ACM*, vol. 50, no. 5, pp. 94–101, 2007.

[35] 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.

[36] S. Hotz, "Routing information organization to support scalable interdomain routing with heterogeneous path requirements," Ph.D. dissertation, University of Southern California, Oct. 1994.

[37] G. Huston. [Online]. Available: http://bgp.potaroo.net. (Accessed in Jul. 2007).

[38] IANA, "Special-use ipv4 addresses," *IETF RFC 3330*, Sep. 2002.

[39] IANA, "IPv4 global unicast address assignments," Sep. 2009. [Online]. Available: http://www.iana.org/assignments/ipv4-address-space/.

[40] Internet Assigned Numbers Authority (IANA). [Online]. Available: http://www.iana.org. (Accessed in Jun. 2007).

[41] B. Irwin and J. P. van Riel, "Using inetVis to evaluate snort and bro scan detection on a network telescope," in *Proc. IEEE VizSEC*, Oct. 2007, pp. 255–273.

[42] ISC Internet Domain Survey. [Online]. Available: http://www.isc.org/index.pl?/ops/ds/. (Accessed in Jun. 2007).

[43] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Proc. IEEE S&P*, May 2004, pp. 211–225.

[44] Juniper IDP. [Online]. Available: http://www.juniper.net/. (Accessed in Jul. 2009).

[45] M. G. Kang, J. Caballero, and D. Song, "Distributed evasive scan techniques and countermeasures," in *Proc. DIMVA*, Jul. 2007, pp. 157–174.

[46] H.-A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," in *Proc. USENIX Security*, Aug. 2004, pp. 271–286.

[47] D. Knuth, *The Art of Computer Programming, Vol. II*, 2nd ed. Reading, Massachusetts: Addison-Wesley, 1981.

[48] O. M. Kolesnikov, D. Dagon, and W. Lee, "Advanced polymorphic worms: Evading IDS by blending in with normal traffic," Georgia Tech, Tech. Rep. GIT-CC-04-13, 2004.

[49] C. Kommareddy, N. Shankar, and B. Bhattacharjee, "Finding close friends on the internet," in *Proc. IEEE ICNP*, Nov. 2001, pp. 301–309.

[50] A. Lakhina, M. Crovella, and C. Diot, "Characterization of network-wide anomalies in traffic flows," in *Proc. ACM/USENIX IMC*, Oct. 2004, pp. 201–206.

[51] F. Lau, S. Rubin, M. Smith, and L. Trajkovic, "Distributed denial of service attacks," in *Proc. IEEE SMC*, Oct. 2000, pp. 2275–2280.

[52] S. Lawrence and C. L. Giles, "Accessibility of information on the web," *Intelligence*, vol. 11, no. 1, pp. 32–39, 2000.

[53] J. Ledlie, P. Gardner, and M. Seltzer, "Network coordinates in the wild," in *Proc. USENIX NSDI*, Apr. 2007, pp. 299–311.

[54] L. Lehman and S. Lerman, "PCoord: Network position estimation using peer-to-peer measurements," in *Proc. IEEE NCA*, Sep. 2004, pp. 15–24.

[55] D. H. Lehmer, "Mathematical methods in large-scale computing units," in *Proc. Large-Scale Digital Calculating Machinery*, 1949, pp. 141–146.

[56] K. Levchenko, R. Paturi, and G. Varghese, "On the difficulty of scalably detecting network attacks," in *Proc. ACM CCS*, Oct. 2004, pp. 12–20.

[57] J. Li, "Routing tradeoffs in dynamic peer-to-peer networks," Ph.D. dissertation, Massachusetts Institute of Technology, 2005.

[58] Y. Li, Z. Chen, and C. Chen, "Understanding divide-conquer-scanning worms," in *Proc. IEEE IPCCC*, Dec. 2008, pp. 51–58.

[59] 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.

[60] 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.

[61] E. K. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft, "On the accuracy of embeddings for internet coordinate systems," in *Proc. ACM IMC*, Oct. 2005, pp. 125–138.

[62] P. K. Manna, S. Chen, and S. Ranka, "Exact modeling of propagation for permutation-scanning worms," in *Proc. IEEE INFOCOM*, Apr. 2008, pp. 1696–1704.

[63] S. McClure, J. Scambray, and G. Kurtz, *Hacking Exposed: Network Security Secrets & Solutions*, 5th ed. Emeryville, California: McGraw-Hill/Osborne,

2005.

[64] The Measurement Factory DNS Surveys. [Online]. Available: http://dns.measurement-factory.com/surveys/. (Accessed in Jun. 2007).

[65] X. Meng, Z. Xu, B. Zhang, G. Huston, S. Lu, and L. Zhang, "IPv4 address allocation and the BGP routing table evolution," *ACM SIGCOMM Comp. Comm. Rev.*, vol. 35, no. 1, pp. 71–80, 2005.

[66] P. Mockapetris, "Domain names – concepts and facilities," *IETF RFC 1034*, Nov. 1987.

[67] P. Mockapetris, "Domain names – implementation and specification," *IETF RFC 1035*, Nov. 1987.

[68] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the slammer worm," *IEEE S&P*, vol. 1, no. 4, pp. 33–39, 2003.

[69] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring internet denial-of-service activity," *ACM TOCS*, vol. 24, no. 2, pp. 115–139, 2006.

[70] D. Moore, C. Shannon, and J. Brown, "Code-red: A case study on the spread and victims of an internet worm," in *Proc. IMW*, Nov. 2002, pp. 273–284.

[71] myNetWatchman – Network Intrusion Detection and Reporting. [Online]. Available: http://www.mynetwatchman.com/. (Accessed in Sep. 2009).

[72] National Laboratory for Applied Network Research (NLANR). [Online]. Available: http://moat.nlanr.net/. (Accessed in Jun. 2007).

[73] Netcraft Web Server Survey. [Online]. Available: http://news.netcraft.com/. (Accessed in Jan. 2010).

[74] NetVCR. [Online]. Available: http://www.niksun.com/. (Accessed in Sep. 2009).

[75] T. S. E. Ng and H. Zhang, "A network positioning system for the internet," in *Proc. USENIX*, Jun. 2004, pp. 141–154.

[76] T. S. E. Ng and H. Zhang, "Predicting internet network distance with coordinates-based approaches," in *Proc. IEEE INFOCOM*, Jun. 2002, pp. 170–179.

[77] Nmap. [Online]. Available: http://nmap.org/. (Accessed in Jan. 2010).

[78] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson, "Characteristics of internet background radiation," in *Proc. ACM IMC*, Oct. 2004, pp. 27–40.

[79] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23–24, pp. 2435–2463, Dec. 1999.

[80] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti, "Lighthouses for scalable distributed location," in *Proc. IPTPS*, Feb. 2003, pp. 278–291.

[81] PingER Project at Stanford. [Online]. Available: http://www-iepm.slac.stanford.edu/pinger/. (Accessed in Jun. 2007).

[82] N. Provos and P. Honeyman, "ScanSSH - scanning the internet for SSH servers," in *Proc. USENIX LISA*, Dec. 2001, pp. 25–30.

[83] 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.

[84] Psad: Intrusion Detection and Log Analysis with Iptables. [Online]. Available: http://www.cipherdyne.org/psad/. (Accessed in Sep. 2009).

[85] T. H. Ptacek and T. N. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," Secure Networks, Inc., Tech. Rep., Jan. 1998.

[86] M. Rabinovich, S. Triukose, Z. Wen, and L. Wang, "DipZoom: The internet measurements marketplace," in *Proc. IEEE INFOCOM*, Apr. 2006, pp. 3083–3088.

[87] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proc. IEEE INFOCOM*, Jun. 2002, pp. 1190–1199.

[88] S. Ren, L. Guo, and X. Zhang, "ASAP: An AS-aware peer-relay protocol for high quality VoIP," in *Proc. IEEE ICDCS*, Jul. 2006, pp. 70–79.

[89] S. Resnick, *Adventures in Stochastic Processes*. Boston, Massachusetts: Birkhäuser, 2002.

[90] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proc. USENIX LISA*, Nov. 1999, pp. 229–238.

[91] I. Rose, R. Murty, P. Pietzuch, J. Ledlie, M. Roussopoulos, and M. Welsh, "Cobra: Content-based filtering and aggregation of blogs and RSS feeds," in *Proc. USENIX NSDI*, Apr. 2007, pp. 29–42.

[92] Route-Views. [Online]. Available: http://www.routeviews.org. (Accessed in Jun. 2007).

[93] Route Views Project. [Online]. Available: http://www.routeviews.org/. (Accessed in Sep. 2009).

[94] SANS Internet Storm Center – Cooperative Network Security Community. [Online]. Available: http://isc.sans.org/. (Accessed in Sep. 2009).

[95] S. Schechter, J. Jung, and A. W. Berger, "Fast detection of scanning worm infections," in *Proc. RAID*, Sep. 2004, pp. 59–81.

[96] H. Shang and C. E. Wills, "Piggybacking related domain names to improve DNS performance," *Computer Networks*, vol. 50, no. 11, pp. 1733–1748, Aug. 2006.

[97] C. Shannon and D. Moore, "The spread of the witty worm," *IEEE S&P*, vol. 2, no. 4, pp. 46–50, 2004.

[98] Y. Shavitt and T. Tankel, "Big-bang simulation for embedding network distances in euclidean space," *IEEE/ACM Trans. Netw.*, vol. 12, no. 6, pp. 993–1006, Dec. 2004.

[99] Y. Shavitt and T. Tankel, "On the curvature of the internet and its usage for overlay construction and distance estimation," in *Proc. IEEE INFOCOM*, Mar. 2004, pp. 385–395.

[100] Y. Shinoda, K. Ikai, and M. Itoh, "Vulnerabilities of passive internet threat monitors," in *Proc. USENIX Security*, Jul. 2005, pp. 209–224.

[101] S. Siddharth, "Evading NIDS, revisited," Dec. 2005. [Online]. Available: http://www.securityfocus.com/infocus/1852.

[102] Skitter Project at CAIDA. [Online]. Available: http://www.caida.org/tools/measurement/skitter/. (Accessed in Jun. 2007).

[103] M. Smith and D. Loguinov, "Enabling high-performance internet-wide measurements on windows," in *Proc. PAM*, Apr. 2010, pp. 121–130.

[104] Snacktime: A Perl Solution for Remote OS Fingerprinting. [Online]. Available: http://www.planb-security.net/wp/snacktime.html. (Accessed in Jan. 2010).

[105] Snort IDS. [Online]. Available: http://www.snort.org/. (Accessed in Sep. 2009).

[106] A. Soule, K. Salamatian, and N. Taft, "Combining filtering and statistical methods for anomaly detection," in *Proc. ACM/USENIX IMC*, Oct. 2005, pp. 331–344.

[107] S. Staniford, J. A. Hoagland, and J. M. McAlerney, "Practical automated detection of stealthy portscans," *Computer Security*, vol. 10, no. 1–2, pp. 105–136, 2002.

[108] S. Staniford, V. Paxson, and N. Weaver, "How to 0wn the internet in your spare time," in *Proc. USENIX Security*, Aug. 2002, pp. 149–167.

[109] M. Szymaniak, G. Pierre, and M. van Steen, "Scalable cooperative latency estimation," in *Proc. IEEE ICPADS*, Jul. 2004, pp. 367–376.

[110] K. Tan, K. Killourhy, and R. Maxion, "Undermining an anomaly-based intrusion detection system using common exploits," in *Proc. Recent Advances in Intrusion Detection*, 2002, pp. 54–73.

[111] L. Tang and M. Crovella, "Virtual landmarks for the internet," in *Proc. ACM IMC*, Oct. 2003, pp. 143–152.

[112] G. C. Tjhai, M. Papadaki, S. Furnell, and N. L. Clarke, "Investigating the problem of IDS false alarms: An experimental study using snort." in *Proc. IFIP SEC*, Sep. 2008, pp. 253–267.

[113] J. Twycross and M. M. Williamson, "Implementing and testing a virus throttle," in *Proc. USENIX Security*, Aug. 2003, pp. 285–294.

[114] R. Vaughn and G. Evron, "DNS amplification attacks," ISOTF, Tech. Rep., Mar. 2006.

[115] F. Veysset, O. Courtay, and O. Heen, "New tool and technique for remote operating system fingerprinting," Intranode Software Technologies, Tech. Rep.,

Apr. 2002.

[116] X. Wang, Z. Yao, and D. Loguinov, "Residual-based measurement of peer and link lifetimes in gnutella networks," in *Proc. IEEE INFOCOM*, May 2007, pp. 391–399.

[117] N. Weaver, S. Staniford, and V. Paxson, "Very fast containment of scanning worms," in *Proc. USENIX Security*, Aug. 2004, pp. 29–44.

[118] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: A lightweight network location service without virtual coordinates," in *Proc. ACM SIGCOMM*, Aug. 2005, pp. 85–96.

[119] J. Wu, S. Vangala, L. Gao, and K. Kwiat, "An efficient architecture and algorithm for detecting worms with various scan techniques," in *Proc. NDSS*, Feb. 2004, pp. 143–156.

[120] J. Xia, S. Vangala, J. Wu, L. Gao, and K. Kwiat, "Effective worm detection for various scan techniques," *Computer Security*, vol. 14, no. 4, pp. 359–387, Jul. 2006.

[121] Z. Yao, D. Leonard, X. Wang, and D. Loguinov, "Modeling heterogeneous user churn and local resilience of unstructured p2p networks," in *Proc. IEEE ICNP*, Nov. 2006, pp. 32–41.

[122] 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.

[123] W. Yu, X. Wang, P. Calyam, D. Xuan, and W. Zhao, "On detecting camouflaging worm," in *Proc. ACSAC*, Dec. 2006, pp. 235–244.

[124] B. Zhang, T. Ng, A. Nandi, R. Riedi, P. Druschel, and G. Wang, "Measurement-

based analysis, modeling, and synthesis of the internet delay space," in *Proc. ACM IMC*, Oct. 2006, pp. 85–98.

[125] J. Zhang, P. Porras, and J. Ullrich, "Highly predictive blacklisting," in *Proc. USENIX Security*, Jul. 2008, pp. 107–122.

[126] R. Zhang, Y. C. Hu, X. Lin, and S. Fahmy, "A hierarchical approach to internet distance prediction," in *Proc. IEEE ICDCS*, Jul. 2006, pp. 73–81.

[127] C. C. Zou, D. Towsley, and W. Gong, "On the performance of internet worm scanning strategies," *Elsevier Performance Evaluation*, vol. 63, no. 7, pp. 700–723, Jul. 2006.

[128] C. C. Zou, D. Towsley, W. Gong, and S. Cai, "Routing worm: A fast, selective attack worm based on ip address information," in *Proc. IEEE PADS*, Jun. 2005, pp. 199–206.

VITA

Derek Anthony Leonard received the B.A. degree (with distinction) in computer science and mathematics from Hendrix College, Conway, AR, in 2002. He joined the Department of Computer Science and Engineering at Texas A&M University in 2002, graduating with a Ph.D. in Computer Science in 2010. His research interests include large-scale measurements of the Internet, experimental analysis, and peer-to-peer networks.

He can be reached at the Department of Computer Science and Engineering, Texas A&M University, TAMU 3112, College Station, TX 77843-3112. His email is `dleonard@cse.tamu.edu`.