Advancing Traffic Obfuscation Against Novel Fingerprinting Attacks

by

Diwen Xue

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2025

Doctoral Committee:

Associate Professor Roya Ensafi, Chair
Clinical Assistant Professor Sol Bermann
Professor Jedidiah Crandall
Professor J. Alex Halderman

Diwen Xue
diwenx@umich.edu
ORCID iD: 0000-0002-0616-4765

© Diwen Xue 2025

ACKNOWLEDGEMENTS

This dissertation is a testament to the guidance, patience, and unwavering support I've received from my advisor, Prof. Roya Ensafi. When I first entered graduate school, I was confused, immature, and uncertain about the path I wanted to pursue. Five years later, I've grown to genuinely love the work I do, deeply believing in its importance and the meaningful impact it can have on the world—all thanks to Roya's unwavering support and guidance. Throughout these years, Roya and I have shared our disagreements, challenges, good times, and difficult moments—as expected in any advisor-advisee relationship, especially as I'm not always the easiest person to work with. Yet, Roya's *patience* and *compassion* always carried us through these moments, and we made the promise never to let those differences undermine our friendship (Yes, I regard Roya not just as a mentor but also as one of my most supportive friends, a sentiment that deeply touched me when she once expressed the same feeling.) To this day, it remains a mystery to me what Roya saw in that fresh undergraduate she took on as one of her early PhD students six years ago, but I will forever be grateful and indebted for her 知遇之恩 (the life-changing kindness of someone who recognized and supported me when I needed it most). I strive to provide my future mentees with the same level of unwavering support that Roya has given me, and I will consider myself a worthy mentor if my students experience even half of the joy and fulfillment that I've found in my PhD journey with Roya.

I would like to thank my committee members—Prof. Sol Bermann, Prof. J. Alex Halderman, and Prof. Jedidiah R. Crandall—for providing their continued support and insightful feedback that have greatly strengthened this dissertation. In particular, I want to

thank Prof. Jedidiah R. Crandall, whose profound wisdom, stimulating discussions, and deep scientific curiosity have set an example for the kind of intellectual and researcher I aspire to become. I also extend my gratitude to all the other professors and staff at the University of Michigan for their support during my PhD journey. Outside the university, I owe special thanks to David Fifield, whose generosity in sharing his wisdom and whose ethical integrity have provided a role model for me and other emerging researchers in our field. Finally, I am deeply grateful to Prof. Joseph Bonneau and Prof. Suzanne McIntosh (may she rest in peace) for their early guidance and mentorship at New York University, which greatly influenced my decision to pursue a career in academia.

This dissertation was supported in part by a Rackham Predoctoral Fellowship and by grants from the U.S. National Science Foundation (CNS-2237552 and CNS-2141512).

Throughout this dissertation, I use the term "we" in recognition of the numerous collaborators and supporters whose contributions and insights were indispensable to this research.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii

LIST OF FIGURES viii

LIST OF TABLES xi

LIST OF APPENDICES xii

ABSTRACT xiii

CHAPTER

I. Introduction 1

1.1 Thesis 3
1.2 Contributions 4
1.2.1 Fingerprinting Obfuscated Tunneling Traffic at Scale . . 4
1.2.2 Protocol-Agnostic Traffic Fingerprinting Attacks 5
1.2.3 Timing-Based Fingerprinting against Traffic Obfuscation 6
1.2.4 The Discriminative Power of Congestion Control Behaviors 7
1.3 Ethical Considerations 7

II. Fingerprinting Obfuscated Tunneling Traffic at Scale 9

2.1 Introduction 10
2.2 Background & Related Work 13
2.3 Challenges in Real-world VPN Detection 17
2.4 Adversary Model and Deployment 18
2.5 Ethics, Privacy, and Responsible Disclosure 19
2.6 Identifying Fingerprintable Features 21
2.6.1 Opcode-based Fingerprinting 21
2.6.2 ACK-based Fingerprinting 23
2.6.3 Active Server Fingerprinting 24

2.6.4 Constructing *Filters* and *Probers* . . . 29
2.7 Fine-tuning for Deployment . . . 30
2.7.1 ACK Fingerprint Thresholds . . . 31
2.7.2 Choice of Observation Window N . . . 31
2.7.3 Effects of Packet Loss . . . 32
2.7.4 Server Churn for Asynchronous Probing . . . 33
2.7.5 Probe UDP and Obfuscated OpenVPN Servers . . . 35
2.8 Real-world Deployment Setup . . . 36
2.9 Evaluation & Findings . . . 38
2.9.1 Results for *control* VPN flows . . . 39
2.9.2 Results for all flows . . . 42
2.10 Discussion and Mitigations . . . 43
2.11 Conclusion of Chapter II . . . 45

III. Protocol-Agnostic Traffic Fingerprinting Attacks . . . 47

3.1 Introduction . . . 48
3.2 Background & Related Work . . . 52
3.2.1 Internet Censorship and Circumvention . . . 52
3.2.2 Attacks on Obfuscated Circumvention Tools . . . 54
3.3 Threat Model . . . 55
3.4 Encapsulated TLS Handshakes . . . 57
3.5 Ethical Considerations . . . 59
3.6 Similarity-Based Classification . . . 60
3.6.1 Data Collection . . . 61
3.6.2 Feature Extraction . . . 62
3.6.3 Classifiers . . . 63
3.6.4 Evaluation on plain TLS/non-TLS traffic . . . 66
3.7 Detecting Obfuscated Proxy Traffic . . . 67
3.7.1 Experiment Setup . . . 68
3.7.2 Detection Framework . . . 71
3.7.3 Results and Findings . . . 73
3.7.4 TLS 1.2 vs TLS 1.3 . . . 78
3.8 Discussion and Mitigation . . . 79
3.9 Conclusion of Chapter III . . . 83

IV. Timing-Based Fingerprinting against Traffic Obfuscation . . . 84

4.1 Introduction . . . 84
4.2 Background . . . 88
4.2.1 Internet Censorship . . . 88
4.2.2 Circumvention Arms Race . . . 89
4.2.3 Timing-based Fingerprinting Attacks . . . 90
4.3 Threat Model and Scope . . . 91
4.3.1 Scope . . . 92

4.4 Cross-layer RTT Difference . . . 94
4.4.1 Proxy Architecture and Round-Trip-Time . . . 94
4.4.2 Example: HTTPS vs. VLESS-TLS . . . 95
4.4.3 Efficacy and Assumptions . . . 97
4.5 Constructing a Cross-Layer RTT Exploit . . . 97
4.5.1 Estimating Cross-layer RTT Differences . . . 98
4.5.2 Distinguishing Δ_{direct} vs. Δ_{proxy} . . . 102
4.5.3 Implementation . . . 106
4.6 Evaluation . . . 107
4.6.1 Evaluating Cross-correlation Method for ARTT Estimation . . . 108
4.6.2 Evaluating the Susceptibility of Proxy Traffic and the selection of threshold T . . . 110
4.6.3 Evaluating RTT_{diff} as a Standalone Fingerprint . . . 111
4.7 Discussion . . . 117
4.7.1 Potential Mitigation Approaches . . . 118
4.7.2 Extending to Network-layer VPNs . . . 122
4.8 Conclusion of Chapter IV . . . 124

V. The Discriminative Power of Congestion Control Behaviors . . . 125

5.1 Introduction . . . 126
5.2 Threat Model . . . 128
5.3 Behavior Characterization . . . 129
5.3.1 Aggressive Sending Rates . . . 130
5.3.2 Differentiation from loss-reactive CCAs . . . 132
5.3.3 Generalizing across Network Conditions . . . 132
5.3.4 Active Throttling . . . 133
5.4 Evaluation . . . 134
5.4.1 Threshold-based Classifiers . . . 134
5.4.2 Results . . . 135
5.4.3 Limitations . . . 136
5.5 Conclusion of Chapter V . . . 137

VI. Conclusion and Future Vision . . . 138

APPENDICES . . . 141

A. Appendix for Fingerprinting Obfuscated Tunneling Traffic at Scale . . . 142

B. Appendix for Protocol-Agnostic Traffic Fingerprinting Attacks . . . 147

B.1 *XTLS-vision* and *obfs4* Padding Schemes . . . 147

C. Appendix for Timing-Based Fingerprinting against Traffic Obfuscation 151

C.1 Notation & Symbols 151
C.2 Dependence between RTT_{diff} observations 151
C.3 Distribution of Transport Delay from Proxy Locations 153
C.4 The Effect of Multiplexing on Estimating RTT_{diff} 154
C.5 Categorical False Positives with IMAP Traffic 154

D. The Discriminative Power of Congestion Control Behaviors 157

BIBLIOGRAPHY 159

LIST OF FIGURES

Figure

2.1 **OpenVPN Session Establishment (TLS mode).** 12
2.2 **Framework Deployment on *Merit*** Steps: (1) Client connects to VPN servers. (2) VPN connections, along with passing traffic, are being mirrored to the *Filter*. (3) *Filter* forwards server IP of suspected connections to the probing system. (4) Targets are sent to each dedicated *Probers*. (5) *Probers* send probes asynchronously. (6) Connections confirmed by probing are logged. 17
2.3 **OpenVPN Header in TCP and UDP modes.** (TLS only) 20
2.4 **XOR-Patch that leaves first byte un-reversed** 23
2.5 **OpenVPN TCP new packet processing routines** 26
2.6 **RST thresholds for OpenVPN and random endpoints.** 28
2.7 **ACK fingerprint DT from the ISP and VPN datasets.** 31
2.8 **Effects of (a) observation window and (b) average packet loss rate on *Filter*'s performance** 34
2.9 **OpenVPN server churns over time.** 36
2.10 ***Merit* evaluation results over days, excluding Control VPN flows.** . . . 39
3.1 **Channel-based Circumvention.** Users' traffic is encapsulated within a cover protocol in order to pass through censors without being directly exposed. Figure inspired by [74].◇ 48
3.2 **Comparison with previous attacks on circumvention tools.** Prior work targeted cover or proxy protocol's failure to achieve indistinguishability, while this paper exploits fingerprints of the encapsulated layers within, which are agnostic to proxy protocols.◇ 51
3.3 **TLS handshakes inside TLS-based proxy.** Messages transmitted during the handshake stage are well-specified and exhibit characteristics in their size, timing, and direction.◇ 56
3.4 ***Merit* Setup** for collecting plain TLS/non-TLS traffic (1–3:§ 3.6.1) as well as obfuscated proxy flows (4–5:§ 3.7.1.2).◇ 61
3.5 **Detection Framework in *Merit*.** For TLS flows, we remove the handshake of the cover TLS layer before feature extraction in order to only identify handshakes that are encapsulated.◇ 70

3.6 **Results by configurations.** Data points are obtained using the parameter set of Chi-squared threshold δ and Mahalanobis threshold γ that yield the highest TPRs under a specific FPR constraint. ◇ 73
3.7 **Performance of classifiers on mixed TLS 1.2/1.3 traffic.** TPR is averaged from *vmess*, *shadowsocks*, *vless*, and *trojan*.◇ 78
3.8 **Round trip count and size of first burst after TLS handshakes.** *TLS-over-TLS* requires more round trips and a larger initial burst due to a second (encapsulated) handshake. Shaded areas highlight dissimilarities that padding and multiplexing cannot obfuscate. 81
4.1 **Threat model:** A censor positioned *on-path* relative to the client and proxy, receiving a copy of traffic. Figure inspired by [70]. 91
4.2 **Protocol layerings in Direct vs. Proxied Connections**. In proxied connections, transport sessions terminate at the proxy, whereas the application layer connection remains end-to-end. ◇ 92
4.3 **Sequence timing diagram for (a) direct HTTPS session vs. (b) TLS-based proxy (vless-over-tls) session.** For the proxied connection, we observe a much higher discrepancy in transport vs. application-layer delay as the sessions at two layers terminate at different endpoints (proxy and the web server, respectively). ◇ 93
4.4 RTT_{diff} **for varying server locations and varying censor position between the client and proxy.** RTT_{diff} (the red line) depends only on the relative distance between proxy and application server and is independent of the client's or censor's location. ◇ 96
4.5 **Cross-correlation for application-layer RTT Estimation.** (a) Bi-directional packet flow, packets ordered by SEQ/ACK; (b) Cross-correlating outgoing against a weighted distribution centered around incoming packets; (c) Correlation scores at different shifts. Shift corresponding the highest correlation is the estimated $ARTT$. 98
4.6 **Distributions of RTT differences for prior estimation.** T1-T0 is TRTT, T2-T0 is ARTT and T2-T1 is RTTdiff. Shaded area represents the separation of the distributions under different priors. ◇ 106
4.7 **Left:** Distributions of normalized correlation errors. **Right:** ARTT variance with different window sizes (median in red). 108
4.8 **Left:** Distribution of transport-layer delay from proxy locations. Blue lines represent the delay of all the flows generated and the red lines represent the delay of selected flows that had the maximum delay for each website visit. **Right:** Number of packets observed before detection for $T = 15ms$ and $T = 50ms$, fixing FPR=.01. 109
4.9 **Per-flow detection and performance of potential mitigations.** *mux=N* combines every N logical streams into one flow.◇ 118
4.10 **Sequence timing diagram illustrating cross-layer RTT discrepancies in network-layer VPN connections.** Both data-carrying and TCP control packets of the encapsulated flow exhibit cross-layer RTT discrepancies. ◇ 123
4.11 **Detection rates for network-layer VPNs**, with the Transport RTT Distributions for CrUX Top 1K domains. A request to *google.com* is used as the coexisting flow. ◇ 123

5.1 Testbed Setup. The middlebox can actively introduce loss or constrain bandwidth of traffic. It can also take advantage of the natural performance degradation on cross-border links. It can observe traffic before and after the middlebox device. . 128
5.2 Comparing Aggressive Proxy Tools with Traditional CCAs. The link has a bandwidth of 100 Mbps, an RTT of 60ms, and an uniform packet loss rate of 5%. 130
5.3 Differentiating Hysteria and Brutal from Loss-based CCAs based on Sending Rate. Sending rates over time is captured by box-plots. Hysteria and Brutal are configured at 50Mbps, 75Mbps, and 100Mbps on a link with 100Mbps bandwidth, 60ms RTT, and 5% uniform loss rate. 131
5.4 Differentiating Hysteria and Brutal across Varying Network Conditions. We quantify the average sending rate ratio of TCP-Cubic, TCP-BBR, Brutal, and Hysteria, respectively, with the estimated sending rate using the Mathis equation. 132
5.5 Differentiating Hysteria and Brutal with Active Throttling. The link has a bandwidth of 100Mbps, which is reduced to 50Mbps mid-connection, an RTT of 60ms, and no base loss rate. 134
5.6 Classification Decision Tree. The two-stage classifier distinguishes Brutal from loss-based and rate-based CCAs based on reaction to packet loss and bandwidth reduction mid-connection. 135
A.1 **Evaluation Process for Active Server Fingerprinting.** 143
A.2 **Decision tree derived from ISP and VPN datasets.** 144
B.1 **Performance of classifiers on plain TLS traffic** TPR and FPR for classifiers with varying detection thresholds $\delta \in 0.1, 0.15, ...1.0, \gamma \in 1, 1.25, ...5$. For each given FPR constraint, we select the parameter set that yield the highest TPR.◇ 148
B.2 **Analysis of server endpoints labeled as proxies** Top: CDF for the ratio of number of connections to the number of unique client IPs. Bottom: CDF for the number of unique SNIs.◇ 150
C.1 **Workflow.** The monitoring continues until either $\Lambda Y \geq \eta$ or the connection terminates. ◇ 153
C.2 **Comparing variability in RTT_{diff} for ISP and multiplexed proxy flows.** We use the ratio of the width of the confidence interval (confidence level = .95) to the mean as a normalized measure for variability.◇ 154
C.3 Interaction with IMAP servers for providers 1, 2, and 3. For all three services, actions such as fetching mailbox updates consistently result in delays that substantially exceed those at the transport layer. Note that it is hard to know precisely why IMAP leads to such delays without visibility into the functioning of the back-end servers. However, this behavior seems to be associated with only IMAP (see Table 4.3) as no other protocol flows within the ISP traffic qualified to consistently show a significant delay difference. 156
D.1 **Average sending ratio comparison of Brutal and Hysteria with BBR.** Both Hysteria and Brutal are similar to BBR for lower loss rates, whereas they significantly differ at higher loss rates ($> 20\%$). 158

LIST OF TABLES

Table

2.1 **Summary of Probes and the expected behaviors from an OpenVPN server.** . . . 26
2.2 **Set of thresholds for ACK Filtering.** . . . 30
2.3 **High-level evaluation statistics on *Merit*.** . . . 38
3.1 **TLS/non-TLS dataset for training and testing classifiers.** . . . 62
3.2 **Top Tri-grams with TLS labels.**◇ . . . 66
3.3 **Evaluation Results.** Top: performance of the detection framework, broken down by proxy configurations. Bottom: False Positive Rates estimated on ISP traffic. We do not include *vmess* in the Random Padding group, as its padding scheme is limited (0-63 bytes).◇ . . . 74
4.1 List of proxy protocols tested in evaluation. ◇ . . . 111
4.2 Evaluation results on proxy traffic. Results are shown for both per-flow detection rate / per-website detection rate. Threshold η selected for FPR = .01 . . . 114
4.3 Distribution of Detection Positives by Remote Port and SNI. Note: '*' in SNIs masks service names, not wildcards. . . . 116
5.1 Evaluation Results. Each tool was measured under 1260 different network condition combinations. The first stage labels flows as either loss-based or non-loss-based. The second stage then classifies non-loss-based flows further into rate-based or custom (e.g.,, Hysteria and TCP Brutal). . . . 136
A.1 **Recommendation Websites Used** . . . 145
A.2 **Evaluation results on *Merit*, breakdown by configuration.** Highlighted rows are "obfuscated" configurations. Variants marked with stars mean that the VPN provider does not disclose which obfuscation technique is used and we can only infer the variant type based on packet captures. Note Hide.me claims the *tls-crypt* option alone is enough to "obfuscate entire traffic" [93]. However, this option only encrypts control channel payloads but not the OpenVPN packer headers. . . . 146
B.1 **Multiplexing support for implementations examined in the paper.**◇ . . . 147
C.1 List of Symbols with Definitions . . . 152
C.2 Top server ASNs for flows exhibiting low transport-delay ($< 5ms$) from proxy locations.◇ . . . 153

LIST OF APPENDICES

Appendix

A. Appendix for Fingerprinting Obfuscated Tunneling Traffic at Scale 142

B. Appendix for Protocol-Agnostic Traffic Fingerprinting Attacks 147

C. Appendix for Timing-Based Fingerprinting against Traffic Obfuscation . . . 151

D. The Discriminative Power of Congestion Control Behaviors 157

ABSTRACT

The recent decades have witnessed a troubling escalation in network interference faced by users around the world, ranging from outright censorship and targeted bandwidth throttling to surreptitious traffic interception and injection of malicious content. Most of these interferences are fundamentally *targeted* in nature: adversaries selectively disrupt specific classes of traffic they deem undesirable, while ensuring the broader usability of the network remains intact. Such targeted interference relies fundamentally on accurate traffic classification methods, typically employing Deep Packet Inspection (DPI) devices to identify and single out specific types of traffic. Correspondingly, effective countermeasures against these interferences focus on thwarting classification efforts through *traffic obfuscation*. Over the years, a variety of obfuscation schemes have been proposed and deployed; however, the majority are ad-hoc, focusing only on packet payloads or protocol headers and failing to account for the increasingly sophisticated capabilities of modern network adversaries.

In this dissertation, I argue the following thesis: *Obfuscating headers and payloads alone cannot protect vulnerable traffic from determined adversaries, as non-content fingerprints in traffic shape, timing, and flow-level behaviors provide sufficient discriminative power; thus, next-generation obfuscation schemes must systematically mask these additional dimensions to provide meaningful resistance to traffic analysis.* To validate this statement, I develop a range of empirical fingerprinting methods and evaluate them under realistic conditions. First, in Chapter II, I investigate the fingerprintability of tunneling traffic at scale, demonstrating that most currently deployed "obfuscated" VPNs can be reliably detected in practical scenarios. Next, in Chapter III, I introduce a protocol-agnostic fingerprinting approach, highlighting a fundamental vulnerability shared across all obfus-

cated proxy protocols. In Chapter IV, I explore traffic features that are often overlooked in obfuscation designs—specifically packet inter-arrival times—showing that packet timing alone possesses considerable discriminative power, unaffected even by aggressive padding techniques deployed in state-of-the-art obfuscation schemes. Finally, in Chapter V, I reveal how the congestion control behavior of traffic flows can inadvertently leak information to traffic analysis and therefore have security implications. My findings collectively reveal critical gaps in current obfuscation methods, highlighting the urgent need to develop more systematic, principled, and robust approaches to traffic obfuscation.

CHAPTER I

Introduction

The ideal of an open Internet has given way to networks where deliberate interference is becoming routine. News reports, personal anecdotes, and measurement studies collectively suggest an increasing trend in network interference on a global scale—such as censorship, throttling, interception, and malicious injection—as governments, authorities, and private actors increasingly seek greater control over the flow of information deemed undesirable within and across their jurisdictions [164, 122, 108, 158]. Recent examples include targeted throttling of social media platforms during political unrest [226], pervasive censorship of foreign content by national firewalls [224], and even interception of encrypted communication services [173]. As these interferences grow in both frequency and sophistication, their impacts extend beyond the inconvenience of individual users to substantial economic and social consequences.

Most network interference is fundamentally *targeted* in nature [182]: adversaries seek to disrupt or degrade specific classes of traffic while maintaining overall network usability and connectivity. For instance, nation-state firewalls target VPN and proxy protocols used for censorship circumvention, while commercial ISPs selectively throttle bandwidth-heavy services like video streaming to manage network congestion or enforce data usage policies. The targeted nature of these interferences compels adversaries to deploy traffic classifiers capable of distinguishing between the traffic they intend to manipulate and other traffic they

consider benign. Accurate classification, or *traffic fingerprinting*, thus becomes a critical enabler of targeted network interference.

This fundamental ability to differentiate target traffic from benign network traffic is precisely the focus of this dissertation. To provide an effective countermeasure against targeted interference, the traffic class under threat must avoid recognition or identification—a goal broadly termed as *traffic obfuscation*. Traffic obfuscation schemes typically pursue one of two objectives: either concealing the identity of traffic by preventing protocol or service fingerprinting (i.e., avoid looking like a protocol that is forbidden) or blending into general background traffic to make it indistinguishable. Achieving the latter objective inherently implies the former, under the assumption that targeted traffic constitutes a minority on the network. While the bulk of this dissertation addresses protocol obfuscation (the former), I will also articulate a vision for future research that moves toward achieving indistinguishability.

Existing literature on traffic obfuscation and fingerprinting, however, suffers from significant limitations. On the obfuscation side, many deployed tools lack rigorous, principled security analyses [182]. They adopt ad-hoc methods primarily targeting packet payload obfuscation (e.g., by fully encrypting the packet, including protocol headers [216]) while often ignoring non-content traffic features. Protocols such as obfs2 and obfs3 explicitly state that they do not consider "sophisticated adversaries" such as "non-content protocol fingerprints, like packet size or timing" [139]. Yet, with rapid advances and commoditization of networking technologies such as deep packet inspection (DPI), the threat models underlying the design of these obfuscation schemes may no longer accurately reflect real-world adversarial capabilities. With firewalls and DPI devices being more capable than ever, it is increasingly imprudent to exclude non-content-based fingerprinting techniques from the threat models of obfuscation protocols.

Similarly, on the fingerprinting side, the most significant limitation is a lack of practical realism in evaluating exploits. Demonstrating theoretical vulnerabilities alone is insufficient

to motivate the development of more robust, next-generation obfuscation techniques. Effective fingerprinting research must provide realistic exploit scenarios that account for practical constraints and capabilities of real-world adversaries, such as operational ISPs [227]. Past studies employing advanced deep learning models for fingerprinting obfuscated traffic often neglect the complexities and operational constraints encountered by adversaries in practical settings, failing to provide convincing evidence to support the feasibility of such fingerprinting attacks and weakening the motivation to pursue more principled obfuscation approaches.

1.1 Thesis

Thesis Statement: *Adopting the perspective of a modern network intermediary reveals that encrypting protocol payloads alone cannot effectively obfuscate traffic—protocol-agnostic fingerprints in traffic shape, timing, and flow-level behavior let adversaries discriminate flows with high precision, exposing an attack surface beyond payload semantics; therefore, next-generation traffic-obfuscation must systematically mask these behavioral dimensions to provide meaningful resistance to traffic analysis.*

I hypothesize that modern network intermediaries, such as firewalls and deep packet inspection (DPI) devices, now possess sufficient capability to effectively exploit non-content traffic features—even against most obfuscation schemes currently deployed in practice. Furthermore, I argue that these fingerprinting attacks are immediately practical and deployable, operating within the realistic constraints faced by network intermediaries, such as limited computational power, memory resources, and most critically, their sensitivity to collateral damage. To substantiate this hypothesis, my research develops empirical fingerprinting methods, rigorously evaluating them under realistic conditions to expose existing gaps in traffic obfuscation. In doing so, I constructively validate my thesis, resulting specific contributions as outlined below.

1.2 Contributions

1.2.1 Fingerprinting Obfuscated Tunneling Traffic at Scale

In this dissertation, I first **investigate the fingerprintability of tunneling traffic (e.g.,, VPN, proxies) [227] (Chapter II)**. Motivated by the growing adoption of VPNs and the corresponding rise in adversarial efforts by ISPs and governments to curb VPN usage, there has been a marked demand for an emerging class of services called "obfuscated" VPN, especially from users in countries with heavy censorship or laws against personal VPN usage. My research develops and evaluates practical fingerprinting techniques against these services to evaluate the effectiveness of their obfuscation. Leveraging non-content traffic features, I identify distinctive protocol fingerprints based on packet patterns, size distributions, and active server behaviors under probing. To operationalize these findings, I design a two-phase fingerprinting framework inspired by prior measurement studies, combining passive monitoring with active probing.

Partnering with Merit, a regional ISP serving over a million users, I deploy and evaluate this detection framework in a realistic, large-scale network setting. The framework efficiently processes traffic at line speed, successfully identifying more than 85% of tunneling traffic with an extremely low rate of false positives. Remarkably, even commercial VPN providers offering supposedly "obfuscated" VPN services—marketed as resistant to detection—are vulnerable to such framework, with successful detection of 34 out of 41 tested obfuscated configurations. This outcome demonstrates detecting and blocking of tunneling traffic, even with most current obfuscation methods, is straightforward and within the reach of any ISP or network operator, as well as nation-state adversaries. This Chapter highlights the critical need for VPN providers to adopt more principled obfuscation techniques. It also contributes short-term defensive strategies and outlines directions for more sustainable and robust obfuscation measures.

1.2.2 Protocol-Agnostic Traffic Fingerprinting Attacks

Building upon the insights from the previous chapter, this dissertation next explores a broader and more powerful method for detecting obfuscated tunneling traffic through a **protocol-agnostic fingerprinting approach [223] (Chapter III)**. Unlike previous work that primarily targets vulnerabilities specific to individual tunneling protocols, I introduce and evaluate a generalized technique based on the fundamental characteristic shared by nearly all proxy and tunneling systems—nested protocol stacks. Specifically, I investigate *encapsulated TLS handshakes*, a unique fingerprint arising from TLS handshakes nested within encrypted or obfuscated layers.

Through a comprehensive evaluation framework, again deployed in collaboration with Merit, I demonstrate the practical feasibility of detecting obfuscated proxy traffic using encapsulated TLS handshakes. Testing 23 popular obfuscated proxy protocols—including widely used tools like shadowsocks, vmess, trojan, and vless [169, 184, 24, 219]—I find that all exhibit significant vulnerability to this fingerprinting method, achieving detection rates consistently above 70%. Notably, these high detection rates persist even when protocols incorporate aggressive obfuscation techniques such as random padding and multiple layers of encapsulation.

The results from this Chapter challenged a widely held assumption in the obfuscation community [88]—that the proliferation of obfuscation protocols inherently protects against traffic fingerprinting by imposing resource-intensive analysis on adversaries. This strategy, often summarized as "letting a thousand flowers bloom", presumes each protocol requires unique, manual fingerprinting efforts by adversaries. However, results from this chapter demonstrate a fundamental vulnerability shared across obfuscation methods: common patterns within inner, encapsulated protocol layers that can be uniformly detected. Consequently, simply introducing new obfuscation schemes without addressing these shared fingerprints may offer only temporary and limited protection, further motivating a fundamental rethinking of our approaches to designing obfuscation schemes.

1.2.3 Timing-Based Fingerprinting against Traffic Obfuscation

Expanding upon the insights gained from previous chapters, this dissertation next **investigates traffic features often overlooked by existing obfuscation designs—such as packet timing [229] (Chapter IV)**. Through empirical evaluations using controlled testbeds and real-world backbone traffic, I demonstrate that packet timing alone provides surprisingly strong discriminative power for detecting obfuscated proxy traffic. Unlike previous fingerprinting methods relying primarily on packet payload or size distributions, this timing-based fingerprint remains effective even against sophisticated obfuscation schemes that fully encrypt payload and employ aggressive random padding.

Through empirical evaluations, I demonstrate that this timing-based fingerprint can reliably distinguish proxy traffic from direct client-server communications across a variety of proxy protocols and network conditions. Crucially, the efficacy of this fingerprinting attack is unaffected by geographical locations of the clients or proxy servers. In addition to being of interest on its own, the results from this chapter highlights the deficiencies in existing obfuscation approaches: over the years, researchers have long recognized that distributions of packet sizes hold significant classification power for detecting circumvention traffic [197, 111], and censors have indeed exploited packet sizes in practice [135, 216]. As such, many obfuscated proxy protocols have incorporated (random) padding schemes directly into their designs [26, 219]. While padding provides a straightforward defense to obfuscate patterns over packet sizes, it leave packet timings unprotected, a source of information often overlooked by designers of obfuscation schemes. What this chapter showcases, however, is one example of what fingerprinting attacks can accomplish in detecting obfuscated proxy traffic using *only* features derived from packet timings, therefore calling attention to the circumvention community to consider packet timing more rigorously in future obfuscation designs.

1.2.4 The Discriminative Power of Congestion Control Behaviors

In the final chapter of this dissertation, I explore the **fingerprintability introduced by custom congestion control algorithms (CCAs) used by obfuscated proxies under adverse network conditions [200] (Chapter V)**. Motivated by the performance challenges faced by such proxies traversing congested, high-loss networks—such as cross-border links—I investigate how custom CCAs, like those implemented by proxy tools Hysteria and TCP-Brutal [112, 114], deviate significantly from standard protocol behaviors. I empirically characterize the aggressive sending patterns of these custom CCAs and demonstrate their distinguishability from standard TCP and QUIC implementations. Using straightforward threshold-based classifiers, I demonstrate the practicality of detecting traffic with non-standard congestion control behaviors across a wide range of network conditions. The findings of this chapter highlight a critical tension between optimizing proxy performance and maintaining the essential indistinguishability required for effective obfuscation. The study strongly advocates for adhering to standard congestion control protocols, such as BBR, that achieve comparable performance without compromising the fundamental obfuscation principles.

1.3 Ethical Considerations

The empirical study of traffic fingerprinting and obfuscation methods inherently involves handling sensitive network traffic data, posing important privacy and ethical concerns. Although my university's Institutional Review Board (IRB) determined that my research falls outside its regulatory scope, I nevertheless took careful procedural and methodological precautions to mitigate potential risks. All empirical evaluations involving real user traffic were conducted in collaboration with Merit [124], an ISP experienced in handling privacy-sensitive projects. Throughout the studies, I ensured that the network traffic analyzed consisted solely of mirrored copies, thereby avoiding any disruption or interference with

actual user connectivity. In each chapter below, I will also discuss additional ethical considerations specific to that chapter.

A broader ethical dimension involves framing traffic fingerprinting and obfuscation within the context of *access control*. In general, **whether obfuscation represents a defensive action or fingerprinting constitutes an attack is not purely technical but depends heavily on normative ethical considerations**. Specifically, the ethical standing of either approach hinges on the legitimacy of the access control they respectively seek to evade or enforce. Recognizing the complexity and importance of these ethical questions, this dissertation explicitly adopts a simplified, pragmatic framing: traffic obfuscation is treated as a defense against fingerprinting attacks. However, **this naive framing is employed solely for clarity and convenience in technical discussions, and does not represent a definitive ethical position**. A comprehensive exploration of these normative ethical questions lies beyond the scope of this dissertation and is deferred to future, dedicated studies.

CHAPTER II

Fingerprinting Obfuscated Tunneling Traffic at Scale[1]

In this chapter, I investigate the fingerprintability of tunneling traffic. The adoption of tunneling tools such as VPNs has seen steady growth over the past decade due to increased public awareness of privacy and surveillance threats. In response, certain governments are attempting to restrict VPN access by identifying connections using "dual use" DPI technology. To investigate the potential for VPN blocking, I develop mechanisms for accurately fingerprinting connections using OpenVPN, the most popular protocol for commercial VPN services. I identify three fingerprints based on protocol features such as byte pattern, packet size, and server response. Playing the role of an attacker who controls the network, I design a two-phase framework that performs passive fingerprinting and active probing in sequence. I evaluated the framework in partnership with a million-user ISP, and was able to identify over 85% of OpenVPN flows with only negligible false positives, suggesting that OpenVPN-based services can be effectively blocked with little collateral damage. Although some commercial VPNs implement countermeasures to avoid detection, the framework successfully identified connections to 34 out of 41 "obfuscated" VPN configurations. Finally, at the end of this chapter, I discuss the implications of the VPN fingerprintability for different threat models and propose short-term defenses. In the longer term, the results from

[1]This chapter is based on: Diwen Xue, Reethika Ramesh, Arham Jain, Michalis Kallitsis, J. Alex Halderman, Jedidiah R. Crandall, Roya Ensafi. 2022. **OpenVPN is Open to VPN Fingerprinting** In USENIX Security Symposium 2022 (USENIX'22) [227] This work was made possible by the National Science Foundation under Grant No.1518888, 1823192, 2007741, 2042795, 2120400.

this chapter urge commercial VPN providers to be more transparent about their obfuscation approaches and to adopt more principled obfuscation countermeasures.

2.1 Introduction

ISPs, advertisers, and national governments are increasingly disrupting, manipulating, and monitoring Internet traffic [206, 115, 172, 86, 128]. As a result, virtual private network (VPN) adoption has been growing rapidly, not only among activists and journalists with heightened threat models but also among average users, who employ VPNs for reasons ranging from protecting their privacy on untrusted networks to circumventing censorship. As a recent example, with the passage of Hong Kong's new national security law, popular VPN providers observed a 120-fold surge in downloads due to fears of escalating surveillance and censorship [190].

In response to the growing popularity of VPNs, numerous ISPs and governments are now seeking to track or block VPN traffic in order to maintain visibility and control over the traffic within their jurisdictions. Binxing Fang, the designer of the Great Firewall of China (GFW) said there is an "eternal war" between the Firewall and VPNs, and the country has ordered ISPs to report and block personal VPN usage [189, 188]. More recently, Russia and India have proposed to block VPN services in their countries, both labeling VPNs a national cybersecurity threat [163, 187]. Commercial ISPs are also motivated to track VPN connections. For example, in early 2021, a large ISP in South Africa, Rain, Ltd., started throttling VPN connections by over 90 percent in order to enforce quality-of-service restrictions in their data plans [192].

ISPs and censors are known to employ a variety of simple anti-VPN techniques, such as tracking connections based on IP reputation, blocking VPN provider (provider from hereon) websites, and enacting laws or terms of service forbidding VPN usage [168, 179, 188]. Yet, these methods are not robust; motivated users find ways to access VPN services in spite of them. However, even less-powerful ISPs and censors now have access to technologies

such as carrier-grade deep packet inspection (DPI) with which they can implement more sophisticated modes of detection based on protocol semantics [174, 157].

In this chapter, we explore the implications of DPI for VPN detection and blocking by studying the fingerprintability of OpenVPN (the most popular protocol for commercial VPN services [56]) from the perspective of an adversarial ISP. We seek to answer two research questions: *(1) can ISPs and governments identify traffic flows as OpenVPN connections in real time*? and *(2) can they do so at-scale without incurring significant collateral damage from false positives?* Answering these questions requires more than just identifying fingerprinting vulnerabilities; although challenging, we need to demonstrate practical exploits under the constraints of how ISPs and nation-state censors operate in the real world.

We build a detection framework that is inspired by the architecture of the Great Firewall [30, 209, 65], consisting of *Filter* and *Prober* components. A *Filter* performs passive filtering over passing network traffic in real time, exploiting protocol quirks we identified in OpenVPN's handshake stage. After a flow is flagged by a *Filter*, the destination address is passed to a *Prober* that performs active probing as confirmation. By sending probes carefully designed to elicit protocol-specific behaviors, the *Prober* is able to identify an OpenVPN server using side channels even if the server enables OpenVPN's optional defense against active probing. Our two-phase framework is capable of processing ISP-scale traffic at line-speed with an extremely low false positive rate.

In addition to core or "vanilla" OpenVPN, we also include commercial "obfuscated" VPN services in this study. In response to increasing interference from ISPs and censors, obfuscated VPN services have started to gain traction, especially from users in countries with heavy censorship or laws against the personal usage of VPNs. Obfuscated VPN services, whose operators often tout them as "invisible" and "unblockable" [180, 175, 45], typically use OpenVPN with an additional obfuscation layer to avoid detection [194, 36].

Partnering with *Merit* (a mid-size regional ISP that serves a population of 1 million users), we deploy our framework at a monitor server that observes 20 Gbps of ingress and

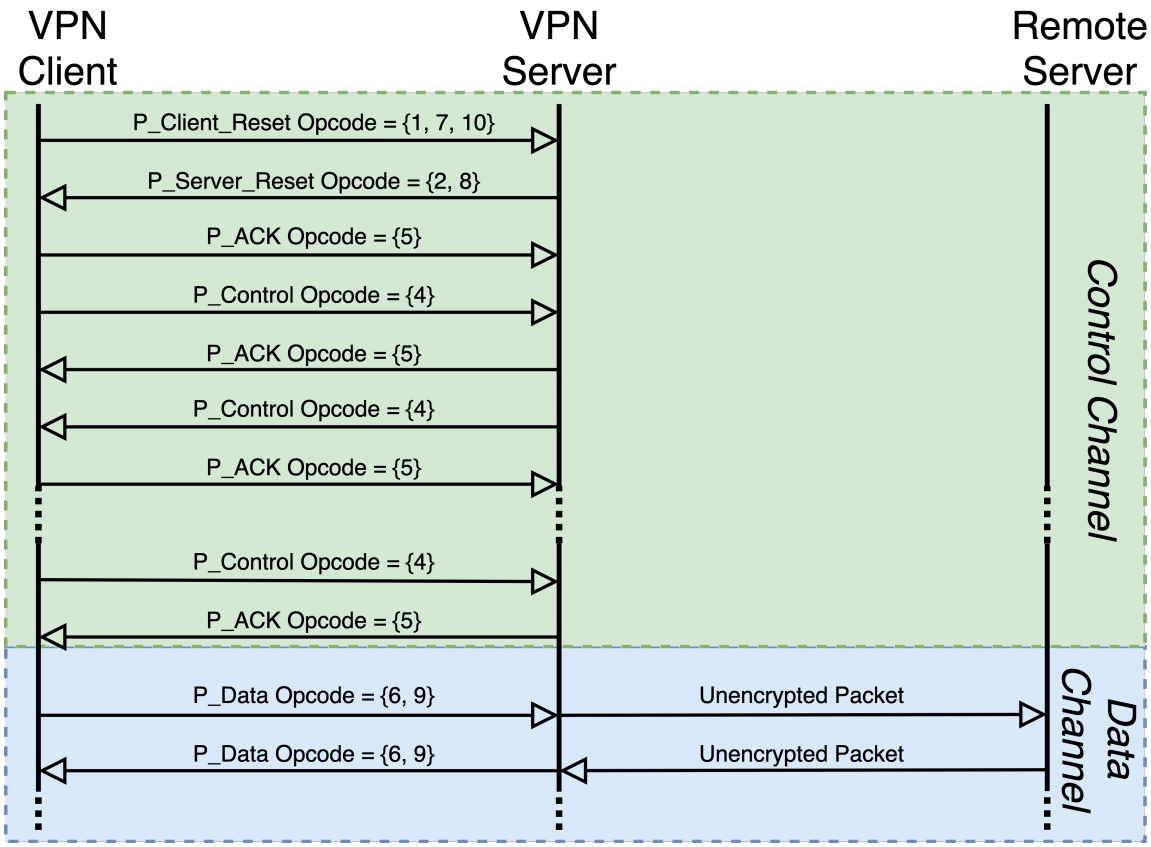


Figure 2.1: **OpenVPN Session Establishment (TLS mode).**

egress traffic mirrored from a major *Merit* point-of-presence. (Refer to § 2.5 for ethical considerations.) We use PF_RING [148] in zero-copy mode for fast packet processing by parallelized *Filters*. In our tests, we are able to identify 1718 out of 2000 flows originating from a control client machine residing within the network, corresponding to 39 out of 40 unique "vanilla" OpenVPN configurations.

More strikingly, we also successfully identify over two-thirds of obfuscated OpenVPN flows. Eight out of the top 10 providers offer obfuscated services, yet *all* of them are flagged by our *Filter*. Despite providers' lofty unobservability claims (such as "... even your Internet provider can't tell that you're using a VPN" [175]), we find most implementations of obfuscated services resemble OpenVPN masked with the simple XOR-Patch [143], which is easily fingerprintable. Lack of random padding at the obfuscation layer and co-location with vanilla OpenVPN servers also make the obfuscated services more vulnerable to detection.

In a typical day, our single-server setup analyzes 15 TB of traffic and 2 billion flows.

Over an eight-day evaluation, our framework flagged 3,638 flows as OpenVPN connections. Among these, we are able to find evidence that supports our detection results for 3,245 flows, suggesting an upper-bound false-positive rate three orders of magnitude lower than previous ML-based approaches [125, 37, 84].

We conclude that tracking and blocking the use of OpenVPN, even with most current obfuscation methods, is straightforward and within the reach of any ISP or network operator, as well as nation-state adversaries. Unlike circumvention tools such as Tor or Refraction Networking [60, 218], which employ sophisticated strategies to avoid detection, robust obfuscation techniques have been conspicuously absent from OpenVPN and the broader VPN ecosystem. For average users, this means that they may face blocking or throttling from ISPs, but for high-profile, sensitive users, this fingerprintability may lead to follow-up attacks that aim to compromise the security of OpenVPN tunnels [177, 151]. We warn users with heightened threat models *not* to expect that their VPN usage will be unobservable, even when connected to obfuscated services. While we propose several short-term defenses for the fingerprinting exploits described in this chapter, we fear that, in the long term, a cat-and-mouse game similar to the one between the Great Firewall and Tor is imminent in the VPN ecosystem as well. We implore VPN developers and providers to develop, standardize, and adopt robust, well-validated obfuscation strategies and to adapt them as the threats posed by adversaries continue to evolve.

2.2 Background & Related Work

VPN tools create private networks across the public Internet through encrypted tunneling. Although many VPN protocols are being used, such as IPSec and WireGuard, OpenVPN remains the most supported and trusted protocol among commercial VPN providers [56]. Due to its versatility and open-source nature, OpenVPN has been used as the underlying protocol in numerous VPN products, which often advertise the protocol

for its proven security [194]. In addition, OpenVPN's popularity continues to rise with the trend of users choosing to self-host open-source VPN tools [193].

OpenVPN Protocol. OpenVPN was first released in 2002 with the aim of creating a tunneling protocol focusing on security, while also being free and fast over the standard TCP and UDP [141]. When the OpenVPN tunnel is active, raw IP packets being sent to or from the tunnel to the final destination are encapsulated inside OpenVPN packets. To achieve secure communication, OpenVPN leverages the OpenSSL library as its cryptographic layer. Two methods for authentication and key exchange are provided to establish trust with peers: either pre-shared static key(s) or TLS-based negotiations. The latter has been adopted by the majority of commercial VPN services. Two separate channels are used for key exchange and data transfer, both sharing a single multiplexed TCP/UDP stream. In the control channel, the client and server engage in a TLS-style exchange of key materials. As TLS is designed to operate over a reliable transport, OpenVPN provides its control channel with a sequential, reliable layer based on an explicit acknowledgement and re-transmission mechanism. The negotiated key from the control channel will be used to encrypt packets transferred in the data channel, which does not provide any reliability guarantee. Figure 2.1 presents a typical initialization sequence of OpenVPN packets leading to a fully encrypted data channel.

Tor, Proxy, and VPN Detection. The ongoing arms-race between the GFW and Tor has been extensively studied and is most representative of the conflict between censorship & surveillance and circumvention tools [65, 209, 182, 68, 61, 181]. Censors started by blocking Tor's website and public relays, which Tor responded to by deploying website mirrors and private, unpublished bridges. Next, censors moved to blocking with DPI by fingerprinting Tor's TLS handshake, e.g., cipher suites. Tor used Pluggable Transports (PT) obfuscators, such as Obfsproxy and meek [150], to mask the handshake. In response, censors deployed active probing to complement DPI-based fingerprinting to detect Tor and certain obfuscators.

There is limited previous work focusing on VPN traffic detection. Hoogstraaten [96] explored server-side VPN detection methods, ranging from using existing information databases (e.g., WHOIS, rDNS) to fingerprinting TCP options (e.g., advertised MSS). Webb et al. [207] proposed detecting proxies and VPNs based on traffic timing and latency. Their approach relied on the hypothesis that when a service is accessed through a proxy, the RTT measurement will be different from the RTT of a direct connection. Another class of previous work uses computational and machine learning models to passively detect VPN traffic [89, 201, 119, 125, 37, 84, 85], leveraging flow-level statistics such as connection duration and packet interval. Most of this work uses the same synthetic `ISCXVPN2016` dataset [89]—which contains a balanced mixture of VPN and non-VPN traffic—to train and test a variety of machine learning and neural network classifiers in an offline, lab-setting. In contrast, our work primarily focuses on whether ISP-level adversaries can identify OpenVPN flows in near real time, and whether they can do so at scale, under practical constraints, and with minimal collateral damage. For this reason, we omit a full analysis of ML-based work, and only compare them with our approach in terms of false positives (falsely blocking legitimate traffic).

Obfuscated (Open)VPN. Various traffic obfuscation techniques have been examined in previous work. Wang et al. examined the detectability of Obfsproxy, FTE, and meek [198]. Using attacks based on protocol semantics, packet entropy, and timing-related features, they concluded that a determined censor could detect all three obfuscators reliably. Houmansadr et al. demonstrated that popular mimicry-based obfuscation tools failed to achieve unobservability because seamlessly simulating another protocol is extremely challenging [97]. Previous studies have suggested censors can use active probing to detect proxies that obfuscate traffic [65, 30, 209]. In response, "probe-resistant" proxies were developed, which remain silent when being probed by an unauthenticated adversary. However, researchers have demonstrated that carefully designed probes could still identify these proxies [80].

There is a marked demand for an emerging class of services called "stealth" or "obfuscated" VPN, especially from users in countries with heavy censorship or laws against personal VPN usage [191, 188]. Most obfuscated VPN services use OpenVPN as the underlying protocol for security and routing, with an obfuscation layer overlaid to avoid detection [194, 36] [2]. OpenVPN's core developers prefer that obfuscation remains a separate project operating alongside the vanilla/core protocol, as they "do not want to play the cat-and-mouse game [as Tor]" [142]. The absence of a standardized obfuscation solution has led to a plethora of obfuscators implemented by different VPN providers, who often claim that their obfuscated services can remain undetected by ISPs and censors alike. For example, TorGuard introduces their obfuscated VPN service as "Engineered from the ground up to be impossible to detect" [180]. BolehVPN claims that their VPN obfuscation "...keeps you out of trouble, even in China" [45]. Common obfuscation strategies adopted by commercial VPNs include employing XOR-based scramblers, wrapping OpenVPN inside encrypted tunnels, or using proprietary protocols.

OpenVPN XOR Patch: Originally developed by Clayface as a patch for vanilla OpenVPN, the XOR patch scrambles a packet by either xor-ing bytes with a pre-shared key, reversing the order of the bytes, xor-ing each byte with its position, or a combination of these steps [143]. Notably, OpenVPN developers discourage its use due to the lack of code audit [183].

OpenVPN over Encrypted Tunnels: Some VPN services wrap OpenVPN traffic inside encrypted tunnels to prevent DPI fingerprinting. Some of the adopted obfuscation tunnels are Obfsproxy (obfs{2/3/4}), Stunnel, Websocket Tunnel, and encrypted proxies (shadowsocks, V2Ray).

Proprietary Protocols: A few VPN providers have developed proprietary obfuscated protocols, some of which are built on top of OpenVPN with a proprietary obfuscation layer added, such as VyprVPN or Astrill [194, 36].

[2] There are discussions on obfuscating WireGuard [212, 213], but to the best of our knowledge, they have yet to be deployed by any commercial VPNs

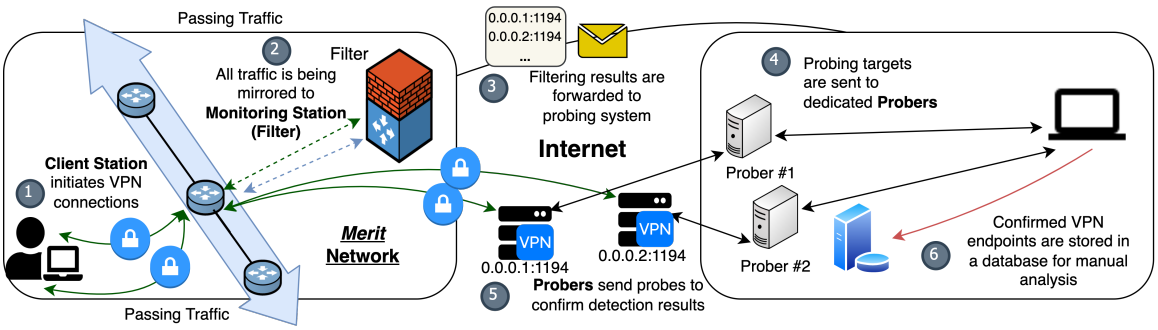


Figure 2.2: **Framework Deployment on *Merit*** Steps: (1) Client connects to VPN servers. (2) VPN connections, along with passing traffic, are being mirrored to the *Filter*. (3) *Filter* forwards server IP of suspected connections to the probing system. (4) Targets are sent to each dedicated *Probers*. (5) *Probers* send probes asynchronously. (6) Connections confirmed by probing are logged.

To the best of our knowledge, we are the first to explore the fingerprintability of commercial and/or obfuscated OpenVPN services on real traffic. Our unique study highlights the practicality of such fingerprinting, which has profound real-world security implications on end-users expecting certain privacy and anonymity guarantees from using these services.

2.3 Challenges in Real-world VPN Detection

Effective investigation of fingerprintability requires incorporating perspectives of how ISPs and censors operate in practice. It is not enough to simply identify fingerprinting vulnerabilities, we need to demonstrate realistic exploits to illustrate the practicality of exploiting the vulnerability, while taking into consideration the ISP and censors' capabilities and constraints [182]. For instance, previous academic works considered using flow-level features to train ML classifiers for VPN detection [89, 201, 119, 125, 37, 84]. Yet, it remains unclear how practical these detection approaches are for ISPs and censors, and we know of no rigorous studies that examine real-world deployment of an ML-based censorship system [182]. Furthermore, previous works test on the `ISCXVPN2016` dataset [89] with balanced OpenVPN and non-VPN traffic. However, we note that due to the low base rate

of VPN traffic in the wild, even the best-performing ML system has false positive rates that can be economically impractical for real-world censors sensitive to collateral damage [198].

However, investigations adopting the viewpoint of ISPs and censors can be challenging. First, such investigation requires collaboration with real-world ISPs and access to their network traffic. We need to install monitors inside an ISP's network, while ensuring our analysis will not affect ISP's normal routing operations. Furthermore, analyzing traffic from real users raises ethical concerns. Processing raw network data may violate the privacy of users, in particular VPN users who often have a heightened threat model. Finally, deploying a system that performs ad-hoc traffic analysis in real time poses significant engineering challenges. We need to ensure the entire analysis framework (including processing and logging) keeps pace with the packet arrival rate and take into consideration the effect of potential asymmetric routing or packet loss on the analysis and results.

2.4 Adversary Model and Deployment

We assume a realistic censor (ISP) capability model based on knowledge from previous measurement studies on the arms race between censors and circumventors [30, 209, 182, 65]. We outline a censor-controlled on-path filter that passively observes and examines passing network traffic. The filter is stateful, but has limited resources and can maintain a limited amount of per-connection states for a short time. The filter is also constrained by long-term data storage and computational resources. In addition to filters installed inside the monitored networks, we assume the censor also operates measurement machines that can send protocol-specific probes to further confirm the detection result. Such two-phase systems have already been adopted by real-world censors such as the GFW against Tor and Shadowsocks [209, 30]. Finally, we expect the censor is familiar with the protocol of interest and has access to the different obfuscators deployed by VPN providers (e.g.,, as a paid customer). We emphasize that this threat model corresponds to censor's capabilities as observed in practice *today*, rather than future capabilities.

To investigate the fingerprintability of OpenVPN and existing obfuscated solutions, we set up a two-phase detection framework in order to answer our key questions: 1) whether real-world censors are *capable* of performing such detection, and 2) whether it is economical to do this *at scale*. Figure 2.2 shows an overview of our framework deployment. Partnering with *Merit*, we instantiate a *Filter* on a Monitoring Station overseeing mirrored traffic from a router that handles 20% of the ISP's traffic. The *Filter* performs passive fingerprinting over raw packets, exploiting traffic features unique to OpenVPN. IP and port information of flows flagged by the *Filter* are forwarded to a probing system and then distributed to dedicated *Probers*. The *Probers* send a set of pre-defined probes specifically designed to fingerprint an OpenVPN server. Finally, probed servers that are confirmed as OpenVPN are logged for manual analysis. Such a two-phase framework resembles how real-world censors operate: lightweight filtering followed up by more expensive, but also more accurate, active probing. This framework is capable of processing massive traffic in real-time while also preventing excessive collateral damage.

2.5 Ethics, Privacy, and Responsible Disclosure

Raw network traffic that contains real users' data is highly sensitive, and this is especially true for traffic related to privacy-oriented services such as VPNs. Here we describe how we consider the security and privacy risks and ethical issues raised by our work, and we detail the procedural and technical steps we take to mitigate the risks.

Foremost among the ethical concerns associated with this work is our *Filter* deployment inside *Merit*'s network to analyze user traffic. *Merit*, which has extensive previous experience collaborating with universities and has well-defined ethics and privacy rules to govern such projects, supervised the deployment. We also cleared our research plan with our university legal counsel and IRB. Although the IRB determined that the work is not regulated, we take extensive measures to minimize potential risks for end-users.

Our framework is fine-tuned on both real and lab-generated traffic data, and it is evaluated

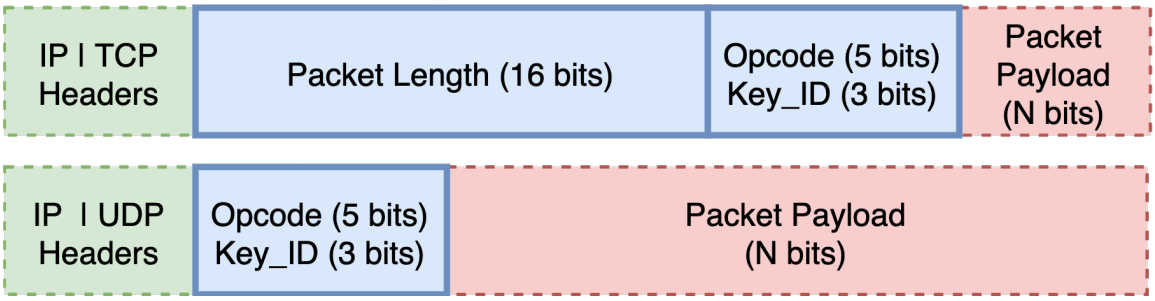


Figure 2.3: **OpenVPN Header in TCP and UDP modes.** (TLS only)

on live ISP traffic. For controlled fine-tuning, a small traffic snapshot (the ISP Dataset in section 2.7) was used to calibrate parameters, e.g.,, the size of observation window. The traffic snapshot, sampling 1/30 of all flows for 45 minutes on July 28, 2021, was generated and analyzed entirely on *Merit* systems, with security mechanisms limiting access to select members of the team. As with the design described in Section 2.6, *Filter* analyzed only the first payload byte, completely ignoring the remainder of the payload, and it recorded only the observed degree of variation. The raw snapshot was never inspected by humans and was deleted after the fine-tuning concluded.

For deployment and evaluation on live ISP traffic, the *Filter* architecture is designed to minimize risks of disrupting or modifying user traffic. The Monitoring Station only receives a copy of the traffic, so even if our software were to malfunction, network service would be unaffected. In addition, to reduce privacy risks, the *Filter* collects only the minimum information necessary for the subsequent probing operation. It records only the server IP addresses and ports of matching connections, which are bucketed into 5-minute internals to inhibit time correlation. These logs are stored and analyzed on a server that is securely maintained by *Merit* and is accessible only to a few members of our research team on a least-privilege basis. *Merit* reviewed our source code prior to deploying it on their network. During deployment and evaluation, no packet payloads or client IP addresses are ever recorded to disk or inspected by humans.

Based on the *Filter* log, the *Probers* send probes to candidate VPN servers. To minimize the risk of disrupting server operations, we design the probes to be non-invasive and make

information available to assist operators in debugging any problems we inadvertently cause. Each server receives only 2–10 innocuous connection attempts, similar to those commonly used in Internet measurement tools like Nmap. The probes originate from two dedicated machines that we provisioned with web pages that explain the nature of the experiment and provide our contact information. We did not receive any inquiries, complaints, or problem reports. Since the server IP addresses themselves may sometimes be non-public, we only report aggregate statistics (e.g.,, the false positive rate) and will not publish any of the addresses that we collect. Any data requests will be referred to *Merit*.

As with all attack-oriented research, there is a risk that our work developing VPN fingerprinting techniques will be adopted by real attackers. To minimize this risk, we are in the process of responsibly disclosing our findings to the VPN operators whose obfuscated servers we successfully identified in our evaluation. We believe that the security of the VPN ecosystem is best advanced by having these problems surfaced by responsible researchers. Our work will help accurately inform users about the VPN services they rely on, and we hope it will enable more robust countermeasures to be developed and deployed.

2.6 Identifying Fingerprintable Features

In this section, we identify three features that fingerprint OpenVPN, exploiting byte pattern, packet length, and server behaviors, respectively.

2.6.1 Opcode-based Fingerprinting

As shown in Figure 2.3, each OpenVPN packet has a header of 24 bits in TCP mode or 8 bits in UDP mode, which is not part of the encrypted payload. Each OpenVPN header starts with an opcode that specifies the message type of the current packet and a key ID that refers to a (new) TLS session. The opcode field can take over 10 defined values, corresponding to message types transmitted during different communication stages. A typical OpenVPN session starts with the client sending a `Client Reset` packet. The server then responds

Algorithm 1 Opcode Fingerprinting Logic

Require: $N \geq 0$
  $OCSet \leftarrow \{\}$, $CR \leftarrow Opcode0$, $SR \leftarrow Opcode1$
  $i \leftarrow 2$
  while $i \neq N$ & $i < |Opcode|$ **do**
    if $Opcodei \in CR,SR$ & $|OCSet| \geq 4$ **then**
      Return False
    end if
    $OCSet$ += $Opcodei$
    $i \leftarrow i$ 1
  end while
  Return $i == N$ & $4 \leq |OCSet| \leq 10$
  #At least 4 different Opcodes needed to complete handshake. In total 10 Opcodes defined by the protocol.

with a `Server Reset` packet, and a TLS handshake follows. OpenVPN packets that carry TLS ciphertexts have `P_Control` as their message type. Since OpenVPN can run over UDP but has to provide a reliable channel for TLS, each `P_Control` packet is explicitly acknowledged by `P_ACK` packets. Finally, actual payloads are transmitted as `P_Data` packets. Figure 2.1 illustrates this packet exchange with opcode annotations.

A packet field taking a fixed number of values can be easy to fingerprint and has been exploited before against other protocols [30]. We fingerprint OpenVPN's handshake sequence by analyzing each opcode byte for the first N packets of a flow (the threshold N is explored in Section 2.7.2). Algorithm 1 shows the process of opcode fingerprinting, with `Opcode` referring to the sequence of N opcode values found in the first N packets of a given flow. Briefly, the filter flags a flow if the number of different opcodes observed accords with the protocol *and* the `Client` and `Server Resets` are not seen once the handshake is completed.

Previous work and existing open-source DPIs [133, 231, 118, 144] considered statically matching opcode values and packet sizes based on the protocol specification. In contrast, we propose to dynamically capture the variation in opcode values that reflects the establishment of OpenVPN sessions. Notably, our heuristics do not require exact matching of opcode values or packet length (e.g.,, do not require the third byte of the first packet to be 0x38),

```
+int buffer_reverse (struct buffer *buf) {
+  int len = BLEN(buf);
+  if (  len > 2  ) {
+    int i;
+    uint8_t *b_start = BPTR (buf) + 1;
+    uint8_t *b_end   = BPTR (buf) + (len - 1);
     .....
```

Figure 2.4: **XOR-Patch that leaves first byte un-reversed**

thereby ensuring it works effectively against XOR-obfuscated flows. The XOR obfuscation masks packet payloads to ensure that the opcode bytes are altered. Notably, according to the specification [143], when it reverses the packet as one of the obfuscation steps, it excludes the first character of the buffer (where the opcode byte is located) from reversal, as shown in Figure 2.4. As such, the opcode byte is always XOR-ed with the same byte of the XOR key, and the same opcodes would be mapped to the same value after obfuscation. This behavior is preserved when Tunnelblick (a popular OpenVPN client on macOS) adopts the patch [183], and has been used in multiple mobile apps [232]. By considering only the number of unique opcodes seen so far, our heuristics are more flexible and target various XOR-based obfuscations of OpenVPN.

2.6.2 ACK-based Fingerprinting

OpenVPN engages in a TLS-style handshake with its peer over the control channel. Since TLS is designed to operate over a reliable layer, OpenVPN implements an explicit acknowledgement and re-transmission mechanism for its control channel messages. Specifically, incoming `P_Control` packets are acknowledged by `P_ACK` packets, which do not carry any TLS payloads and are uniform in size (Note these ACK packets are carried over by TCP as payload and are not the same as TCP ACK flags). Moreover, these ACK packets are seen mostly only in the early stage of a flow, during the handshake phase, and are not used in the actual data transfer channel, which can run over an unreliable layer.

To our knowledge, we are the first to devise fingerprinting attempts based on the distinct protocol-layer ACKs against OpenVPN. Previously, the unique timing pattern in meek's TCP-level ACK traffic has rendered the obfuscation tool vulnerable to detection [198]. For OpenVPN, the presence of explicit ACK packets, uniform in size and only seen in some parts of a session, provides another fingerprintable feature. Specifically, we first identify a likely ACK packet of a session by locating an initial packet exchange sequence of C->S (Client-Reset), S->C (Server-Reset), C->S (ACK), C->S (Control), as illustrated in Figure 2.1. For vanilla OpenVPN and XOR-based obfuscation, the first ACK packet usually appears as the third (data) packet transmitted in a session. For tunnels or obfuscators that have their own handshake or key exchange process (e.g.,, Stunnel, SSH tunnel, or Obfsproxy), this counting is offset by the number of tunnel handshake packets. Next, we group packets into 10-packet bins, and we derive the ACK fingerprint for each flow by counting the number of packets in each bin that have the same size as the identified ACK packet. For OpenVPN flows, we expect to observe a high number of ACK packets in early bins and an absence of them in later bins. (Later in the session, Control and ACK packets can be exchanged again to transfer random key materials, but it is not expected to be observed within our observation window N.) This approach proves effective to fingerprint vanilla OpenVPN as well as obfuscated services running over encrypted tunnels that lack random padding. We quantify exact fingerprinting thresholds in Section 2.7.1.

2.6.3 Active Server Fingerprinting

We explore the feasibility of identifying an OpenVPN server through active probing. Typically, OpenVPN servers respond to a client reset with an explicit server reset, thereby giving away their identity. However, most commercial providers now have adopted *tls-auth* or *tls-crypt* options [176]. These options add an additional HMAC signature—signed by a pre-shared key—to every control channel packet for integrity verification, including the initial reset packets. With either of these options enabled, an OpenVPN server would not

respond to an unauthenticated client reset with a server reset, but would instead drop such packets without further processing. The presence of such HMAC mechanism increases the complexity of doing active probing: it effectively makes OpenVPN servers "probe-resistant" [80] by remaining silent when probed by an unauthenticated client.

In fact, similar HMAC mechanisms are used by more popular "probe-resistant" proxies, such as obfs4 [140]. However, unlike obfs4 which waits for a server-specific random delay before dropping an unauthenticated connection, OpenVPN always *immediately* closes the connection if a valid HMAC cannot be located. We design our probes to leverage this protocol-specific behavior, and as a result, we manage to fingerprint OpenVPN servers even if they do not respond throughout our probing cycles. The key concept is that although the application may not respond to probing, an attacker may still be able to fingerprint application-specific thresholds at the TCP level, such as timeouts or RST thresholds, as demonstrated by Frolov et al. [80].

We use two datasets in this section to help with designing probes. **ZMap Set**: to construct a realistic non-VPN endpoints dataset, we use ZMap to scan each of the 65,535 TCP ports over the entire IPv4 space, limiting results for each port to 200 endpoints (with the specific port open), resulting in over 13 million endpoints. **Censys Set**: We query the `Censys.io` [62] database for hosts with TCP port 1194/OpenVPN open. Next, we probe each endpoint with a typical OpenVPN `Client Reset` and group endpoints that respond with explicit `Server Resets`. This results in 180,858 hosts known to be OpenVPN endpoints (with "tls-auth" disabled).

2.6.3.1 Base Probes

We design probes exploiting a behavior associated with how OpenVPN packetizes TCP streams. When OpenVPN operates over TCP, it needs to split the continuous stream into discrete OpenVPN packets. Figure 2.5 presents a high-level abstraction of this process. The most relevant parts are: a buffer is allocated in memory to reassemble fragments of

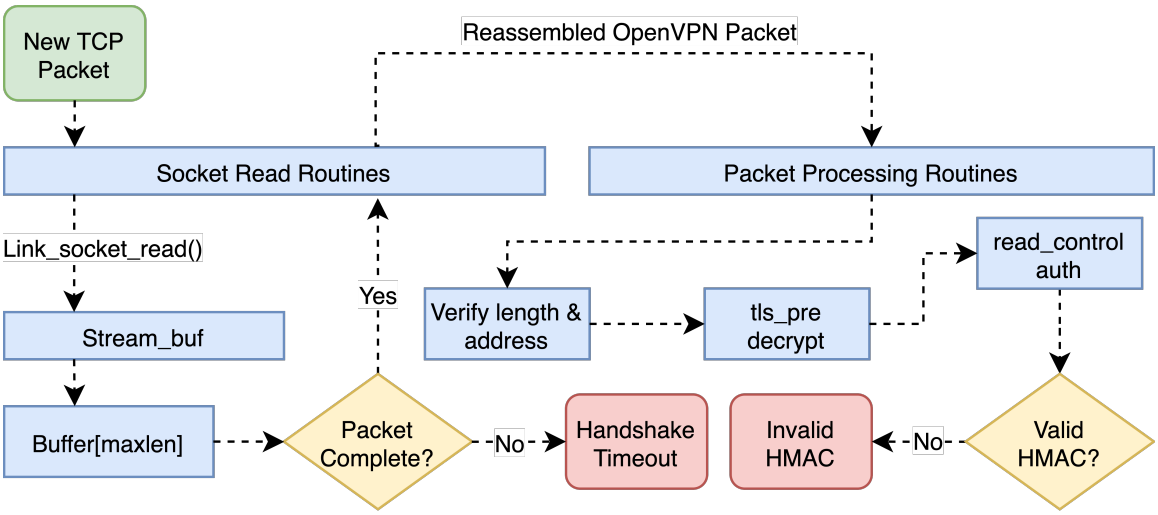


Figure 2.5: **OpenVPN TCP new packet processing routines**

ProbeName	Probe Content	Expected Behavior
BaseProbe 1	x00x0ex38.{8}x00x00x00x00x00	Explicit ServerReset or Short Close
BaseProbe 2	x00x0ex38.{8}x00x00x00x00	Long Close
TCP Generic	x0dx0ax0dx0a	Short Close
One Zero	x00	Long Close
Two Zero	x00x00	Short Close
Epmd	x00x01x6e	Short Close
SSH	SSH-2.0-OpenSSH_8.1/r/n	Short Close
HTTP-GET	GET/HTTP/1.0 /r /n /r /n	Short Close
TLS	Typical Client Hello by Chromium	Short Close
2K-Random	Random 2000 Bytes	Short Close & RST

Table 2.1: **Summary of Probes and the expected behaviors from an OpenVPN server.**

OpenVPN packets encapsulated in TCP streams. The length N for the next OpenVPN packet is extracted from the first two bytes of the header (see Figure 2.3), and the routine keeps reading N additional bytes before it returns the reassembled packet to the caller. This means that an OpenVPN packet will not be parsed and checked for syntax and encryption errors until all its parts arrive at the server. Based on this behavior, we design two sequential probes to trigger an OpenVPN server into different code paths—which result in different connection timeouts—and measure the time elapsed before the server responds or terminates the connection. As shown in Table 2.1, *Base Probe 1* carries a typical 16-byte OpenVPN `Client Reset`, while *Base Probe 2* has the same payload with the last byte stripped off. The assumption is since our two probes only differ in one byte, most non-OpenVPN servers

will respond to our probes in a similar way. However, for an OpenVPN server with HMAC enabled, the connection sending the first probe will be dropped immediately because the OpenVPN packet is reassembled and a valid HMAC cannot be located. The second probe will not receive an immediate response, as the server will wait for an additional byte to arrive for reassembly. The connection will stay idle until a server specific handshake timeout has passed, after which the connection will be dropped. As such, the first probe will be dropped at the decryption routine, while the second probe will be dropped at the packet reassembly routine, both labeled red in Figure 2.5.

2.6.3.2 Additional Probes

The two probes, although useful, are limited and there may be other protocols with behaviors similar to OpenVPN. After using both to probe the *ZMap Set*, we still identify a handful of services that respond similarly to OpenVPN servers, such as Microsoft WBT Server (3389), Microsoft Message Queuing (1801), and Erlang Port Mapper Daemon (4369).

We design additional probes based on the fact that OpenVPN validates packet length and will drop connections sending invalid length without waiting for the next packet to be reassembled. Here, packet length refers to the length declared by the first two bytes of an OpenVPN header (see Figure 2.3), rather than the TCP packet length. A "valid" length is in the range of [1, *max_len*], where *max_len* is derived from the server's MTU configurations. For instance, default TUN MTU of 1500 bytes, combined with overheads (crypto IV, packet length, *etc*), results in a *max_len* of 1627 bytes. In this case, probes whose first two bytes have a decimal value greater than 1627 (0x06,0x5B) will be dropped immediately.

We also design probes leveraging the way a Linux server closes a TCP connection. When a TCP connection terminates, the operating systems at both ends typically complete a FIN 4-way handshake. However, previous work has found that if a connection is closed with unread bytes in buffer, Linux will send a RST packet [80]. A server's "RST Threshold"

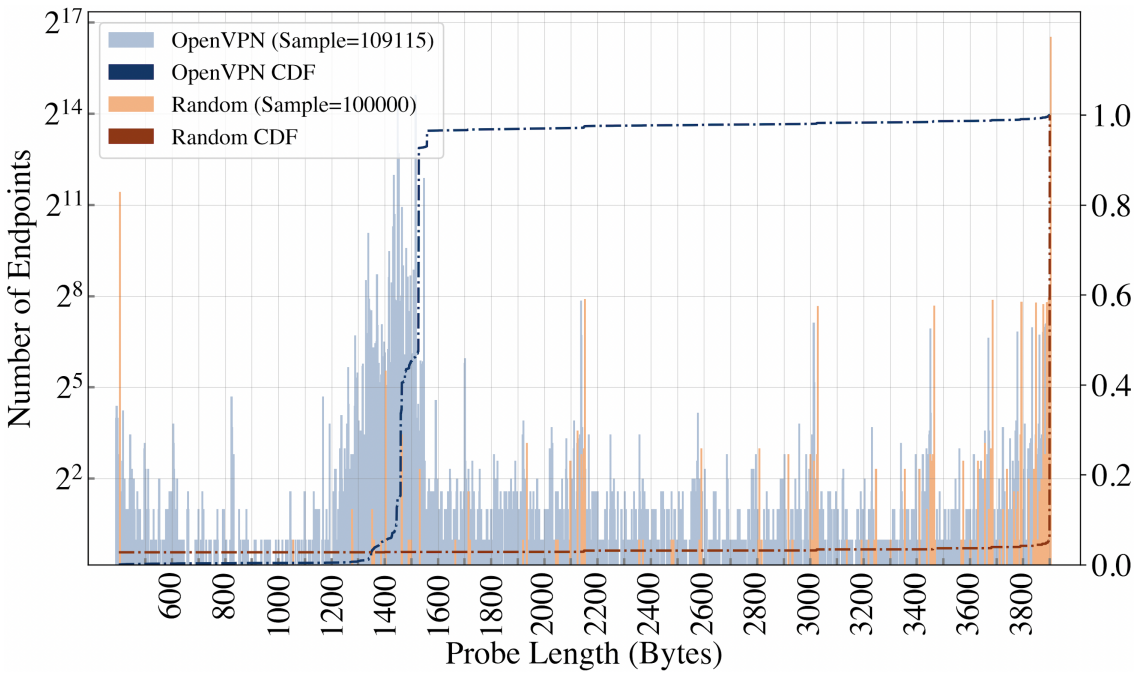


Figure 2.6: **RST thresholds for OpenVPN and random endpoints.**

is defined as the minimum number of bytes needed to send to the server to trigger a RST. We determine the RST threshold distribution for both *ZMap Set* and *Censys Set*. As shown in Figure 2.6, the vast majority of OpenVPN servers have a RST threshold around 1550-1660 bytes, corresponding to buffers allocated with typical MTU configurations. In contrast, over 97% of random ZMap endpoints have a RST threshold less than 500 or greater than 4000. We therefore construct an additional probe with 2,000 random bytes, which we expect over 98% of legitimate OpenVPN servers and less than 3% of random servers to respond to with RST packets.

Complication from Port Sharing OpenVPN provides native support for another application to share the same port. This is accomplished by checking whether the first incoming packet has a valid OpenVPN-conforming length field. If not, the OpenVPN server will forward the packet to the other service sharing the port. This means that most of our additional probes will be forwarded to and responded by the other application due to invalid packet length. To account for this, we observe that when an OpenVPN shares a port, it is usually shared with a HTTP, TLS, or SSH service. Thus, we send probes targeting these three

protocols after our base probes, and we stop further probing if we get an explicit response for any of these probes.

It is worth noting that the majority of "typical" HTTP, TLS, and SSH servers have already been filtered out by our base probes, so endpoints that respond at this stage are likely sharing the port with another service, thus warranting manual analysis (e.g.,, checking TLS certificate). While these three services are what we commonly observed, there may be instances where other services are running along with OpenVPN. This could lead to false negatives.

Table 2.1 lists all probes and the expected behaviors from an OpenVPN server. An evaluation process is shown in Appendix Figure A.1.

2.6.4 Constructing *Filters* and *Probers*

Our *Filter* performs both opcode and ACK-based fingerprinting, flagging a flow if at least one fingerprint matches. This is because the opcode and ACK fingerprints are designed to be complementary: both are effective against vanilla OpenVPN and they each target a specific subset of obfuscations. The former works against XOR-based obfuscations that work like Vigenère ciphers, i.e. they always encrypt the same plaintext opcodes at the same position to the same ciphertext bytes. The latter targets tunneling-based obfuscation that lacks random padding and preserves the 1:1 correspondence between the original and obfuscated packet streams. Combining the two features maximizes our fingerprinting coverage, as we discovered that even within the same provider, obfuscating strategies can vary a lot (§ 2.9). Table A.2 in Appendix shows the effectiveness of each feature against each commercial VPN service we tested. Following *Filter*'s result, the *Prober* performs the active probing scheme to further lower potential false positives.

We implement the *Filter* in Zeek [231], an open-source network monitoring tool. We note that the evaluation processes for opcode and ACK-based fingerprinting are quite simple: both only require several dozen integer comparisons (limited by the observation

Bin Number	Threshold
1	$1 \leq Bin1 \leq 3$
2	$2 \leq Bin2 \leq 5$
For i in range $3,5$	$Bini \leq 5$
For i in range $6,N10$	$Bini \leq 1$

Table 2.2: **Set of thresholds for ACK Filtering.**

window) while maintaining a small number of per-flow states. We implement the *Prober* in Nim [138]. We believe that both components can be easily deployed by any ISP or censor.

2.7 Fine-tuning for Deployment

So far, we have described features that render OpenVPN vulnerable to fingerprinting. We still need to quantify detection thresholds (e.g., ACK fingerprints) for implementation. Furthermore, there are metrics that can affect the system performance, such as packet loss or observation window choice. We seek to fine-tune our system by quantifying these parameters.

We use two datasets here. **ISP Dataset**: we collected a snapshot of network traffic going through a server installed within *Merit*. Over 45 minutes on July 28, 2021, we sampled 1/30 of all flows passing through the server, resulting in 461 GB of traffic that corresponds to 221,534 flows with full packet payloads. Refer to § 2.5 for details on how this data was handled to limit privacy risks. **VPN Dataset**: we collected traces from 20 commercial VPN providers as well as 2 self-hosted OpenVPN services (Streisand, OpenVPN Access Server) following the automated process described in Section 2.8. Note the 20 VPN providers do not overlap with the providers used in evaluation. For each provider, we repeated the trace collection process 50 times each in TCP and UDP mode, resulting in a 7.65 GB dataset comprised of 2,200 vanilla OpenVPN traces.

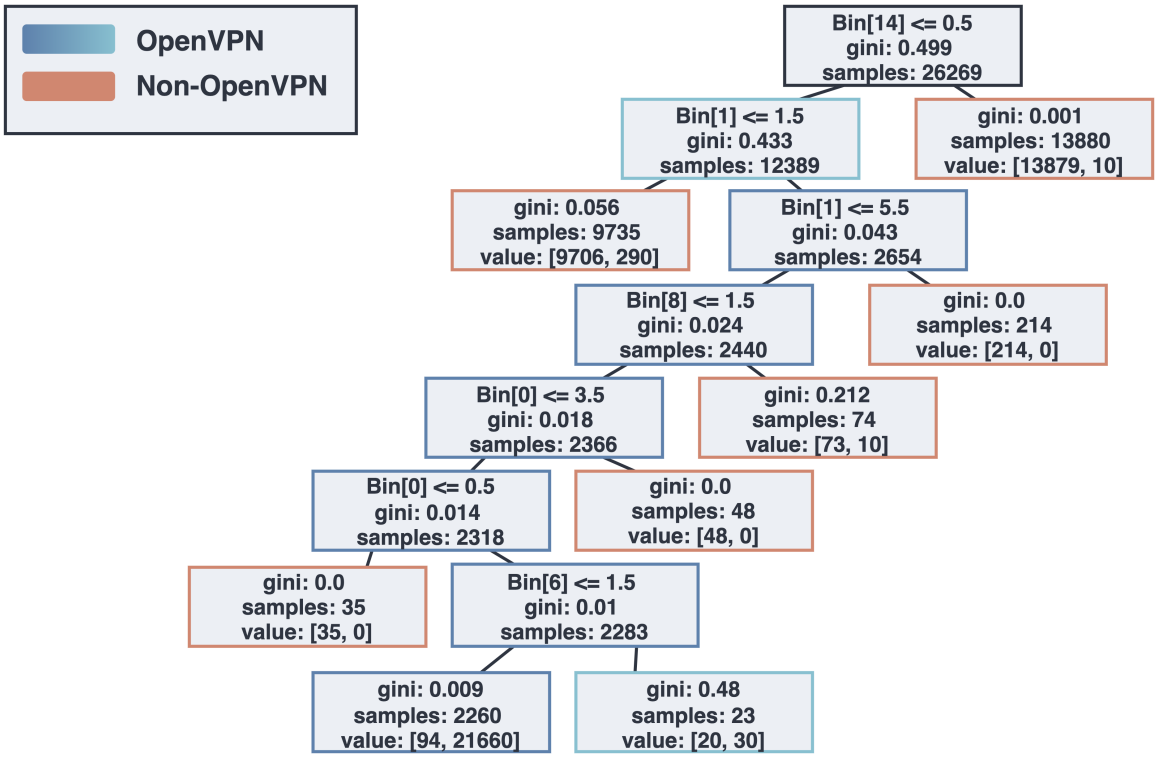


Figure 2.7: **ACK fingerprint DT from the ISP and VPN datasets.**

2.7.1 ACK Fingerprint Thresholds

We quantify the exact ACK fingerprint based on the *ISP* and *VPN Dataset*. We only include flows with at least 150 data packets (15 bins), which leaves us with 24,069 ISP flows and 2,200 VPN flows. A classification decision tree is constructed based on the two labeled sets with weights applied to account for the imbalanced data size. Figure 2.7 shows the constructed tree (depth and leaf limited, a complete graph can be found in Appendix Figure A.2). The ACK fingerprint is a sequence of thresholds based on the derived decision tree, as shown in Table 2.2. (*Bini* refers to the number of ACK-size packets for i^{th} Bin.)

2.7.2 Choice of Observation Window N

Previous works attempt to identify VPN traffic only *after* the flow terminates, making use of aggregated statistics such as connection duration [119, 89, 125]. However, detecting disallowed traffic only after the flow is finished may be of limited interest to a real-world

censor [39]. We therefore have two objectives for our *Filter*: to reduce probing targets by being as selective as possible, *and* to detect OpenVPN as soon as possible within a flow.

Inspired by [39, 198], we consider the windowing strategy of limiting the inspection to only the first N data packets of a flow. We tested N from [10, 20, 30, ... 200] on the *ISP* and *VPN Dataset*. As shown in Figure 2.8 (a), the number of ISP flows that are flagged by the *Filter* declines from over 62,000 to 322 as we increase the observation window. However, we note that a window size of 100 packets has already achieved a precision within 2% of the best performing (200 packets).

Detection Speed and Potential Impact on Blocking A smaller window size can sever a connection at an earlier stage, thereby reducing transfer of data to a censored endpoint, while a more conservative windowing strategy excels at accuracy. In our deployment, we use 100 packets as the window size to balance detection speed and accuracy. To put this choice into perspective, we note that the Great Firewall of China (GFW) was previously observed to send confirmation probes to suspected endpoints in 15-minute intervals, and it has only recently moved to near real-time operation [65]. Recent work on how it detects Shadowsocks shows the median delay between the beginning of a connection and probing is about a minute, with probes being replayed for up to 47 times for confirmation [30]. In comparison, our deployment with a window size of 100 packets gives a median time of 7.9 seconds for the filter to flag an OpenVPN connection. We believe even with this delay, our system is still useful for censors who are interested in blocking OpenVPN connections. In addition, we note that a motivated adversary can further optimize this delay and speed the detection by tuning window size and probing rate, but with some potential loss of accuracy.

2.7.3 Effects of Packet Loss

We investigate the effects of packet loss on the performance of the *Filter*. An adversary analyzing traffic on a busy network needs to keep pace with the packet arrival rate, or

otherwise packet drops will start to occur due to a CPU bottleneck. For the opcode and ACK fingerprinting, we need to inspect the raw contents of each reassembled packet until the observation window is reached, for all flows. This means all traffic must be passed on to Zeek's scripting layer, which may lead to a CPU bottleneck. In addition, the Network Interface Card (NIC) may also become an upstream bottleneck and lead to end-to-end packet loss. We therefore explore what to expect from the *Filter* when packet loss is inevitable.

We configure Zeek to ignore events signaling new packets with different probabilities in order to simulate random packet loss. We test loss rates from 1% up to 80%. The experiment is repeated three times, and the average result is reported in Figure 2.8 (b). We find that packet loss starts to have noticeable effects on the *Filter*'s outputs once the loss rate surpasses 10%. Notably, when packet loss starts to affect the detection accuracy, both opcode and ACK fingerprint vulnerabilities always produce *underblocking* instead of *overblocking*, which is favoured by real-world censors [182]. Still, in order to minimize the effects of packet loss, we always configure the Monitoring Station to sample flows with a given rate (adjusted with CPU resources and traffic volume) for our online evaluation.

2.7.4 Server Churn for Asynchronous Probing

After the *Filter* generates a list of probing targets, the *Prober* can either send probes synchronously as soon as a target is emitted, or asynchronously, waiting for a pre-configured interval before sending probes to targets in batches. Sending probes synchronously has the advantage of obtaining the most accurate results before the server IP is churned. However, this requires the probing system to be online the whole time. In contrast, sending probes in batches is more efficient and easier to manage, but the server IP may be churned if the interval between the filtering phase and the probing phase is excessively long. We explore a probing frequency that achieves efficiency and accounts for possible server IP churn. To do this, we monitor the 180,858 known OpenVPN servers from the *Censys Set* described in

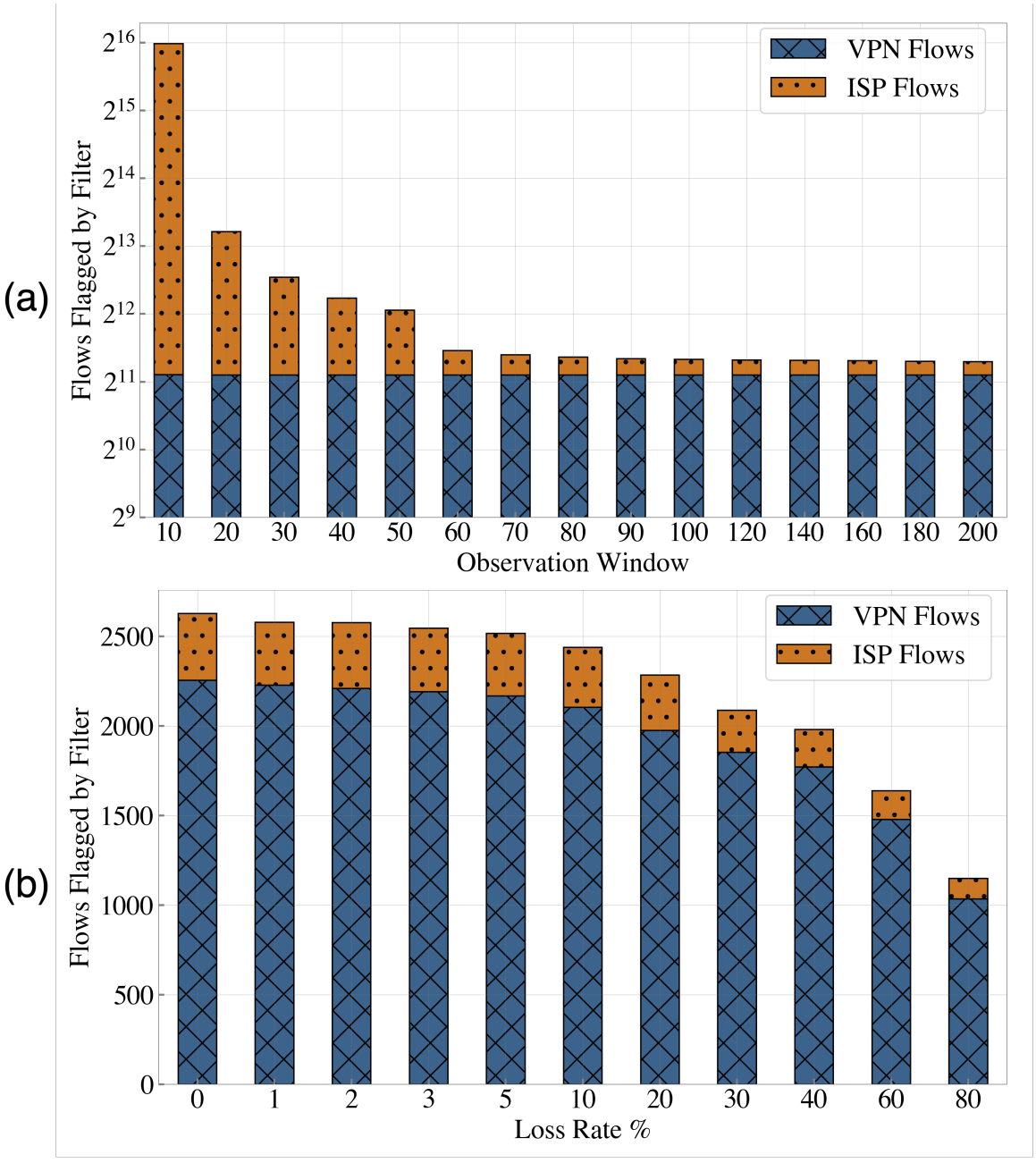


Figure 2.8: **Effects of (a) observation window and (b) average packet loss rate on *Filter*'s performance**

2.6.3. Starting from August 2nd, 18:00 EDT, we probe the servers every 3 hours for a week and record their responses.

As shown in Figure 2.9, even after a week, only 2.39% of OpenVPN servers either are not in the listening state or have been replaced by a different service. This suggests that the majority of OpenVPN servers are not churned frequently. In our online evaluation, we choose to probe targets in batches on a daily basis to balance between efficiency and potential IP churn. Based on the result of this test, approximately 0.9% of servers may be churned within 24 hours.

2.7.5 Probe UDP and Obfuscated OpenVPN Servers

The active probing scheme in the previous section primarily targets vanilla OpenVPN TCP servers, as it exploits the header length field that is unique to TCP mode that requires packetization. In addition, it works effectively against XOR-obfuscated servers because the length field is prefixed *after* the XOR encryption is applied to an OpenVPN packet. This construction allows us to probe XOR-obfuscated servers in the same way as if they had no obfuscation at all.

For UDP or other obfuscated servers, our probes are no longer effective because the length field is either not present (UDP) or encrypted (tunnel-based obfuscation). However, a critical observation is that most commercial VPN providers usually offer vanilla TCP servers along with UDP and/or obfuscated variants. This is expected as commercial VPN providers attempt to optimize their VPN's performance as well as reliability, since tunnel-based obfuscation adds overhead and UDP traffic may encounter more problems than TCP in a firewalled network. Furthermore, the vanilla TCP service is often co-located with the UDP or obfuscated OpenVPN services, presumably due to lower hosting and maintenance cost. They could be on the same host by listening on different ports, or they could be located in adjacent IPs in the same VPN provider subnet. In other words, probing adjacent netblocks of a suspected UDP or obfuscated endpoint may reveal nearby vanilla TCP servers, whose

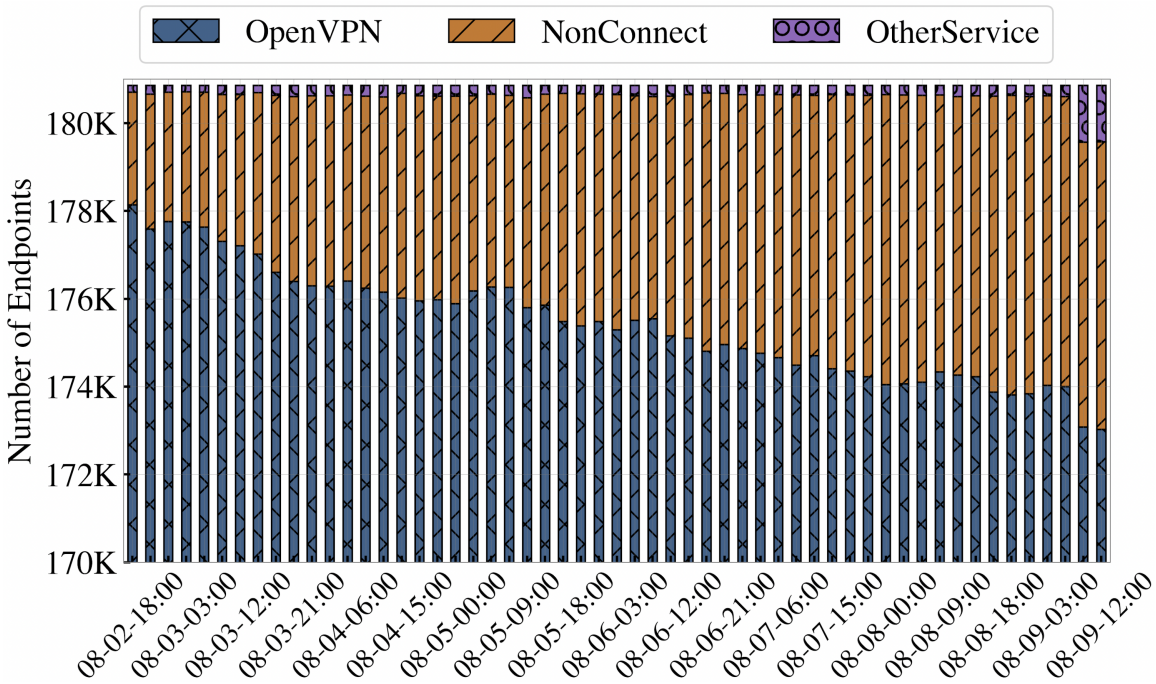


Figure 2.9: **OpenVPN server churns over time.**

existence corroborates the *Filter* results. For our *Prober* deployment on two dedicated measurement machines, we limit our probing to the /29 subnet the target IP belongs to over all TCP ports. This specific subnet size is chosen primarily due to probing resources limitation, and a more well-resourced adversary may expand the probing to larger subnets. With only two measurement machines, the parallelized /29 *Prober* is able to probe targets generated by a *Filter* monitoring a 5 Gbps network interface.

2.8 Real-world Deployment Setup

We set out to explore if an ISP or censor can fingerprint OpenVPN connections at scale, without significant collateral damage. Adopting the viewpoint of an adversarial ISP, we deploy our framework inside *Merit*, as shown in Figure 2.2. Our evaluation is two-fold: we generate *control* vanilla and obfuscated flows with commercial VPN providers and attempt to identify them as a network intermediary; we also process other traffic passing through our Monitoring Station in order to estimate the false positive rate of our framework.

We set up our framework on a 16-core server (Monitoring Station) inside *Merit* with two mirroring interfaces that have an aggregated 20 Gbps bandwidth. Due to the large traffic volume, we optimize our deployment with PF_RING [148] in order to improve the packet processing speed. We employ PF_RING in zero-copy mode and spread the traffic load across a Zeek cluster of 15 workers. Nonetheless, due to limited CPU resources, we only sample 12.5% of all TCP and UDP flows arriving at the network interfaces in order to minimize the effect of packet loss. The sampling is based on IP pairs so that all bi-directional traffic of a flow will be selected/dropped together. With these settings, we are able to operate with an end-to-end packet loss rate under 3%. Even though we process only a fraction of all traffic, our *Filter* still handles over 15 Terabytes of traffic from over 2 billion flows on an average day on a single server. In addition, processing all traffic without sampling is feasible through parallelism or using faster CPUs.

Next, we set up *Probers* on two dedicated measurement machines, each provisioned with 10 IPv4 and 1 IPv6 addresses. By the end of each day during the evaluation, the *Probers* fetch filtering logs from the Monitoring Station. For each target, we run a Masscan [121] to the /29 subnet the IP belongs to over all TCP ports (1-65535). We follow up each discovered open port by running our probing scheme, and endpoints confirmed through probing are recorded for manual analysis.

To select VPN services for evaluation, we first generate a list of "top" VPN services ranked by popularity. We combine 80 providers, most of which are paid premium VPN services, from top VPN recommendation sites based on previous work [156], listed in Appendix Table A.1. Next, we visit the websites of these VPN providers searching for "Obfuscation", "Stealth", or "Camouflage Mode" *etc*, and include providers that offer at least one obfuscated VPN configuration. In total, we find 24 providers offering obfuscated services. We test all obfuscation configurations if more than one is offered as well as vanilla OpenVPN for each provider. If TCP and UDP modes are both available, we test them separately. In total, we have 81 configurations, 41 of which are obfuscated ones.

Control Flows	Overall Recall	3141/4120 (76.24%)
	Filter Recall	3635/4020 (90.42%)
	Prober Recall	3186/3635 (87.65%)
	Vanilla Recall	1718/2000 (85.90%)
	Obfuscated Recall	1468/2020 (72.67%)
All Flows	Flow Count	23183039736
	Bytes Processed	124.67 Terabyte
	Flows $\geq$ Observation Window	10070994
	Filter Outputs	75850
	Probing Outputs	3638
	Confirmed OpenVPN Flows	3245
	Remaining Unclassified	393 (0.0039%)%

Table 2.3: **High-level evaluation statistics on *Merit*.**

We configure the Client Station inside *Merit* to act as a VPN client. Both upstream and downstream traffic of the Client Station go through the router that mirrors traffic to the Monitoring Station. In addition, we exclude this server from our random sampling so that *all* traffic to/from this server will be analyzed. On the client, we run an automated script to generate *control* traffic for our evaluation. For each iteration, we start the VPN client application and connect to the "default / recommended" server using Pywinauto [152]. After a random wait of 20 to 180 seconds, we confirm that the VPN tunnel is active and generate random browsing traffic with Selenium [167] by sending requests to a random website from the Alexa top 500. Finally, we disconnect from the VPN server and wait for 180 seconds before proceeding to the next iteration. For each VPN configuration, we repeat the process 50 times and collect packet captures for reference.

2.9 Evaluation & Findings

We started the evaluation on August 13, 2021, and kept the monitor running for a week until August 20. Table 2.3 contains high-level statistics of the evaluation. A more detailed result that breaks down statistics by each *control* VPN configuration can be found in Appendix Table A.2.

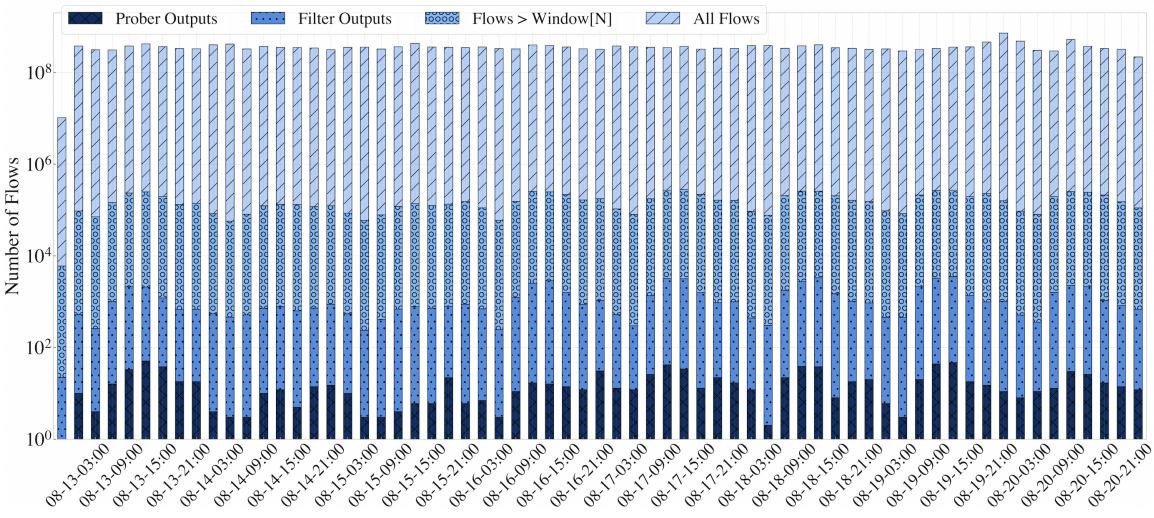


Figure 2.10: ***Merit* evaluation results over days, excluding Control VPN flows.**

2.9.1 Results for *control* VPN flows

Overall, we are able to identify 1,718 out of 2,000 vanilla flows, corresponding to 39 out of 40 unique configurations. This suggests the majority of OpenVPN traffic and servers are vulnerable to passive filtering and active probing, respectively. The few exceptions correspond to VPN providers that only offer UDP-based services or hide their servers behind IDS [57], which thwarts our probing attempts. Surprisingly, we also identify over two-thirds of all obfuscated flows, corresponding to 34 out of 41 obfuscated configurations. This result is mostly due to obfuscated services using OpenVPN as their backbone protocol and insufficient obfuscation failing to mask OpenVPN's fingerprints. Alarmingly, out of the "top 10" VPN providers ranked by *top10vpn.com* [178], eight provide obfuscation services of some sort, suggesting that being undetectable is within the providers' threat model for their clients. Yet, *all* of them are flagged as suspect flows due to either insufficient encryption (Opcode) or insufficient obfuscation over packet length (ACK). Considering that these obfuscated VPN services usually claim to be "undetectable" or claim that the obfuscation "keeps you out of trouble" [45, 180], this result is alarming as users who use these services may have a false sense of privacy and "unobservability".

4 out of the "top 5" VPN providers use XOR-based obfuscation, which is easily

fingerprintable.. We find that among the "top 5" VPN providers [178], four offer obfuscated services, all of which nonetheless are flagged as OpenVPN flows by our *Filter* over 90% of the time. A closer look at the raw packet capture suggests that *all* of them employ obfuscations that are almost identical to the unofficial XOR patch, thereby making them vulnerable to fingerprinting. Alarmingly, the XOR-based obfuscation—despite being rejected by OpenVPN developers [183]—appears to be a major obfuscation strategy adopted by the majority of VPN providers we test, who often praise the patch for its simplicity and low overhead. Although the patch can bypass some of the most basic filters adopted by existing open-source DPI tools, we have demonstrated that even a slightly more sophisticated filter will be able to reliably and accurately detect them.

Wrapping OpenVPN inside encrypted tunnels is a popular obfuscation strategy, yet some flows are still recognizable due to a lack of random padding.. Another popular class of obfuscation strategies is tunnel-based, which wraps OpenVPN traffic inside an encrypted tunnel to frustrate any analysis over packet payloads. Examples include Stunnel, SSH tunnel, Shadowsocks, obfs{2/3/4}, and V2Ray(VMess). Overall, we find 20 obfuscated configurations deployed by 14 VPN providers that are tunnel-based. However, most of these tunnels do not add random padding to the payload being tunneled, with the only exceptions being obfs4 and VMess which can draw packet sizes from certain distributions. Among the 20 tunnel-based obfuscated services, only three of them deploy obfs4 and only one deploys VMess, leaving the remaining 16 vulnerable to ACK fingerprinting. We note that this does not mean these tunneling tools do not work, but rather that protection against traffic analysis is not among the design goals. For example, the threat model of obfs3, which is deployed by Perfect Privacy VPN, states that the obfuscator "does not try to protect against non-content protocol fingerprints, like the packet size or timing" [139]. Yet, we have demonstrated that for applications with distinct signature over packet length, such as OpenVPN, even the simplest threshold-based detection can identify them with reasonable accuracy.

UDP and obfuscated servers often share infrastructure with vanilla TCP servers,

leaving them "guilty by association".. We discover that the majority of UDP and obfuscated OpenVPN services are co-located with vanilla TCP servers. For example, TorGuard hosts vanilla and stunnel-obfuscated OpenVPN instances on the same host but different ports, whereas Perfect Privacy hosts them in neighboring IPs (*.*.193.26 for vanilla, *.*.193.27 for Stunnel, *.*.193.28 for SSH, and *.*.193.29 for obfs3). We find that for 34 out of 41 obfuscated services, at least one vanilla OpenVPN TCP server can be found within the server's /29 subnet. Similarly, we were able to actively probe 18 out of 20 UDP configurations due to their co-location with TCP servers. In addition, we also find five providers sharing infrastructures used by their obfuscated services. For example, one IP (23.95.*.*) hosts Cryptostorm's SSH-obfuscated service as well as SecureVPN's vanilla servers. This result is only a lower bound as we did not connect to every single server available from each provider. Obfuscated services using shared infrastructures may be easier for adversaries to identify and block.

On the positive side, some deployed services successfully evade our detection. Some providers deploy randomizers such as obfs4, v2ray, or proprietary protocols with random padding, which stopped us at the filtering stage (e.g., Tunnelbear). In addition, some providers deploy their obfuscated servers behind a firewall or IDS, which would respond with SYN-ACKs to every arriving SYN packet on almost all TCP ports [103, 80]. Since we limit probe targets to 2,000 open ports per IP for practical considerations, this leads to false negatives when none of the probes hit an OpenVPN-listening port. Moreover, some VPN providers do not host vanilla OpenVPN TCP servers at all, such as VyprVPN, which currently only supports UDP as transport. For these providers, even though our *Filter* flags their flows as suspected OpenVPN, we were not able to confirm with subsequent probing. Finally, some providers host UDP or obfuscated servers outside the /29 probing range of the vanilla TCP ones, and we miss them due to probing resource constraints.

2.9.2 Results for all flows

Figure 2.10 shows an hour-level breakdown of the evaluation statistics, excluding *control* flows. Overall, both the *Filter* and *Prober* are able to reduce the number of suspected flows by several orders of magnitude, which when combined flagged 3,638 flows as OpenVPN connections. We manually analyze these flows to confirm our detection results.

Among the 3,638 flows, the destination servers for 469 of them respond to our *Base Probe #1* with an explicit server reset, indicating the presence of a legitimate OpenVPN server not configured with HMAC protection. For the remaining 3,169 flows, we first noticed that 2,580 of them are between a single IP pair. Based on our log, the client initiates a connection every 4 minutes to the server on port 1194 (assigned to OpenVPN). Reverse DNS lookup associates the client IP with the "lib-locker" subdomain under a private university in the US. Furthermore, the server runs a TLS service listening on port 443, which sends a certificate belonging to a smart locker company with subject and issuer CN as "vpn._-COMPANY_.com". Based on these evidence, we believe the captured flows correspond to the secure communications between a deployed smart locker and the infrastructure that controls it. This also suggests that the fingerprintability of OpenVPN may not only be a problem concerning censorship circumvention, but it may also be used for reconnaissance to identify and target IoT devices that communicate to their servers over an OpenVPN channel. Finally, we attempt to further characterize the remaining 589 flows based on circumstantial evidence about the destination endpoint.

Co-location with TLS In practice, TLS is the most common application we have seen that is co-located with an OpenVPN instance. For each of the remaining flows, we probe its destination endpoint with a TLS Client Hello and analyze the certificate and web page returned. Endpoints of 40 flows return certificates whose subject or issuer CN suggest VPN activity, such as `*.vpn.ipvanish.com`, `*.vpn.wlvpn.com`, `*.virtualshield.org`, and `OpenVPN Web CA`. In addition, 16 endpoints serve OpenVPN web interfaces over TLS.

WHOIS, DNS PTR, ISP Name We look up the WHOIS and DNS PTR records of the destination endpoints. 11 server IPs of 41 flows contain WHOIS records that can be linked back to a VPN provider, such as `protonvpn-*`, `PRIVADO-*`, and `secureconnectivity-*`. In addition, 2 servers have DNS PTR records as `*.strong.blackoakcomputers.com` and `fosvpncluster.fos.*.com`.

IP Context Service Several online platforms claim to offer VPN IP database or IP context services. We found 124 flows that can be linked to a commercial VPN server IP by the lookup service hosted on `spur.us`. However, these services do not disclose their specific methodology and their accuracy has not been systematically evaluated.

Our 7-day evaluation flagged 3,638 flows that are identified as "OpenVPN" from over 10 million flows that exceed our observation window. Among these, we are able find evidence that supports our detection result for 3,245 flows. The majority of the remaining 393 flows have server IPs belonging to cloud hosting services, and we are not able to further classify them. Conservatively, we can upper bound the false positive rate to 0.0039%, which is three orders of magnitude lower than previous ML-based approaches (1.4%-5.5%) [37, 125, 84]

2.10 Discussion and Mitigations

ISPs and government censors are motivated to detect OpenVPN flows in order to enforce traffic policies and information controls. We demonstrate that tracking and blocking the use of OpenVPN, even with most deployed obfuscation methods, is practical at scale and with minimal collateral damage. We note that many VPN providers' claims that their obfuscated services are unobservable appear to be misleading and potentially dangerous, especially to users from countries where personal VPN usage is illegal. In light of our findings, users should *not* expect complete unobservability even when connected to "obfuscated" OpenVPN-based services.

Putting the human danger aside, the ease of fingerprinting makes OpenVPN more sus-

ceptible to throttling or blocking from ISPs and governments. Previous research suggests that some censors already use two-stage pipelines, which are highly similar to our deployment, to detect other protocols such as Tor or Shadowsocks [209, 30]. These adversaries can quickly adapt such infrastructure to detect OpenVPN traffic by simply adding protocol-specific fingerprints and probes. Furthermore, while we focus on OpenVPN due to its overwhelming popularity among commercial VPNs, it is possible to extend our two-stage framework to other VPN protocols (e.g.,, WireGuard and StrongSwan) by analyzing their communication patterns and server behaviors. Governments can also quickly adopt these fingerprints to track and block VPN usage during sensitive times, like political upheavals, when VPN connections are most vital to the free flow of information.

Short-Term Mitigation There are several defensive strategies to achieve near-term protection from the fingerprinting attack we describe. First, VPN providers offering both vanilla and obfuscated OpenVPN services should avoid co-locating them. Ideally, obfuscation servers should be well separated from OpenVPN instances in the network address space and operate as "bridge servers" that forward client traffic to VPN servers elsewhere. For example, Mullvad VPN offers a Shadowsocks-based obfuscator service as a dedicated bridge, separating the VPN servers from the obfuscation [130].

Second, VPN providers should switch from static to random padding for their obfuscated services. As we have shown, for protocols with a stable and distinctive handshake phase, even the most basic threshold-based detector is able to fingerprint them by packet sizes. Ideally, the obfuscation layer should be able to send zero-length packets (packets whose payloads are all padding) to break the one-to-one correspondence between the unobfuscated and obfuscated packet streams [212]. Yet it is worth noting that previous work has shown that even fully randomized obfuscators (e.g.,, obfs4) can themselves be vulnerable to entropy-based fingerprinting attacks [198].

Third, we suggest that the OpenVPN developers follow recommendations from previ-

ous work with regard to how servers respond to failed handshake attempts. Servers closing failed connections immediately or in a predictable manner has enabled active probing attacks against a variety of other protocols [30, 186, 80]. In response, these protocols have implemented either unlimited timeouts (reading from the buffer indefinitely) or diversified close behaviors (in which each server instance closes failed connections in a different manner).

Long-Term Defenses In the long term, we fear that the cat-and-mouse game between censors and circumvention tools, such as the Great Firewall and Tor, will occur in the VPN ecosystem as well, and developers and providers will have to adapt their obfuscation strategies to the evolving adversaries. We urge commercial VPN providers to adopt more standardized obfuscation solutions, such as Pluggable Transports [150], and to be more transparent about the techniques used by their obfuscated services. This transparency will help foster development of stronger obfuscation methods and encourage developers to design better techniques to overcome the progress of information control technologies. Additional future work is needed to characterize the performance costs of different approaches to VPN obfuscation and to help users with varying threat models make appropriate trade-offs between performance and resilient unobservability.

2.11 Conclusion of Chapter II

We have demonstrated that OpenVPN, even with widely applied obfuscation techniques, can be reliably detected and blocked at-scale by network-based adversaries. Inspired by previous real-world censorship events, we designed a two-phase system that performs passive filtering followed by active probing to fingerprint OpenVPN flows. We evaluated the practicality of our approach in partnership with a mid-size ISP, and we were able to identify the majority of vanilla and obfuscated OpenVPN flows with only negligible false positives, which supports that the techniques we describe would be practical even for adversaries averse to collateral damage. Users worldwide rely on VPNs to protect their security and

privacy and to escape Internet censorship, yet the ease of fingerprinting OpenVPN traffic and the commodification of DPI technologies bring monitoring and blocking of popular VPN services within reach for almost any network operator. We propose several short-term mitigations that can help defend against these threats, but in the long term, we urge VPN providers to adopt more resilient and better standardized obfuscation approaches.

CHAPTER III

Protocol-Agnostic Traffic Fingerprinting Attacks[1]

Research over the past decade has extensively examined various fingerprinting attacks against individual proxy protocols and their respective countermeasures, including the results demonstrated in the previous chapter (Chapter II). In this chapter, however, I demonstrate the feasibility of a protocol-agnostic approach to proxy detection, enabled by the shared characteristic of nested protocol stacks inherent to all forms of proxying and tunneling activities. I showcase the practicality of such an approach by identifying one specific fingerprint—encapsulated TLS handshakes—that results from nested protocol stacks, and building similarity-based classifiers to isolate this unique fingerprint within encrypted traffic streams.

Assuming the role of a censor, I build a detection framework and deploy it within a mid-size ISP serving upwards of one million users. The evaluation demonstrates that the traffic of obfuscated proxies, even with random padding and multiple layers of encapsulations, can be reliably detected with minimal collateral damage by fingerprinting encapsulated TLS handshakes. While stream multiplexing shows promise as a viable countermeasure, the results from this chapter further caution that existing obfuscations based on multiplexing

[1]This chapter is based on: Diwen Xue, Michalis Kallitsis, Amir Houmansadr, Roya Ensafi. 2024. **Fingerprinting Obfuscated Proxy Traffic with Encapsulated TLS Handshakes** In USENIX Security Symposium 2024 (USENIX'24) [223] This work was made possible by the National Science Foundation under Grant Numbers CNS-2237552, CNS-2141512, the Defense Advanced Research Projects Agency (DARPA) under Agreement HR00112190127, and the Young Faculty Award program of the Defense Advanced Research Projects Agency (DARPA) under Grant DARPA-RA-21-03-09-YFA9-FP-003.

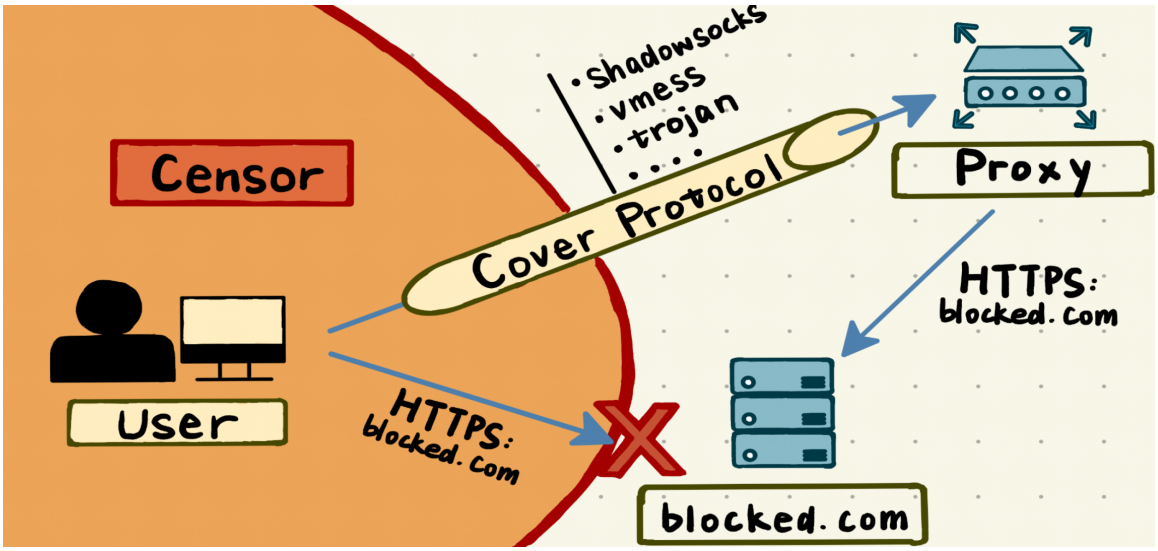


Figure 3.1: **Channel-based Circumvention.** Users' traffic is encapsulated within a cover protocol in order to pass through censors without being directly exposed. Figure inspired by [74].◇

and random padding *alone* are inherently limited, due to their inability to reduce the size of traffic bursts or the number of round trips within a connection. Proxy developers should be aware of these limitations, anticipate the potential exploitation of encapsulated TLS handshakes by the censors, and equip their tools with proactive countermeasures.

3.1 Introduction

The recent decades have witnessed a global escalation in censorship and surveillance by various nation-state actors [164]. Among the most notorious examples is China, where the Great Firewall (GFW) has been blocking foreign websites, filtering search results, and interfering with private communications [27, 95, 155, 14]. In Iran, the government blocks social media and throttles targeted protocols during periods of political unrest [34, 35, 12]. In Russia, "Sovereign Internet" prevents access to news and media during the Ukraine war, demonstrating how easily censors can create information bubbles and isolate specific regions from the broader Internet [225, 157].

In reaction to such escalating censorship measures, users have resorted to proxying and tunneling tools. As shown in Figure 3.1, users in censored regions encapsulate their

application traffic within a *cover* (i.e., proxy) protocol and transmit it to a proxy server outside the censor's jurisdiction, which then forwards the traffic to its destination. The cover protocol encrypts its payload, so that application traffic containing censored keywords does not trigger blocking. This strategy, referred to as "channel-based circumvention" by previous work [182], necessitates obfuscation over the cover protocol – otherwise, the channel itself could be fingerprinted and blocked. Over the past decade, the circumvention community has developed numerous obfuscation approaches, either making the channels look like protocols that are allowed [64, 127, 47, 4], or disguising the channels to ensure they do not resemble protocols that are prohibited [19, 26, 140].

On the other hand, advancements like carrier-grade deep packet inspection (DPI) have enabled censors to evolve from basic IP or keyword filtering to more sophisticated modes of detection. This has led to an ongoing arms race between circumvention proxies, who implement obfuscation mechanisms to avoid detection, and censors, who aim to see through such obfuscation: censors use active probing to identify proxy servers, driving proxies to implement probe-resistance mechanisms [66, 210, 80]. Censors block circumvention proxies by their unique TLS ciphersuites [2, 210, 78], in response to which proxies change their ciphersuites to mimic mainstream browsers [81]. Censors block fully encrypted proxies based on high entropy [216]. Proxy developers then modify the byte patterns to show less entropy [88, 11]. Such back-and-forth interactions represent the current state of the circumvention arms race, with censors exploiting the design and implementation flaws of individual cover protocols, and developers patching the flaws and experimenting with new protocols. A common assumption shared in the community is that each new cover protocol requires its own unique set of features and separate analysis by the censor to block [88].

What we demonstrate in this chapter, however, is the feasibility of a *protocol-agnostic* approach to proxy detection. We observe that, despite the varying designs and implementations of cover protocols, the fundamental concept underpinning all forms of proxying and

tunneling is that of nested protocol stacks, where one protocol stack is encapsulated within the payload of another. For example, users' web browsing traffic (HTTPS) can be encapsulated within another TLS or application-layer protocol, which serves both as a cover and transport for the former, in order to pass through censors without being directly exposed. The near-universal presence of nested protocol stacks in proxied connections, contrasted with the rarity of finding such behavior in direct client-server communications, establishes them as a fingerprinting vulnerability that is shared across various proxy protocols and is orthogonal to existing attacks and countermeasures.

In this chapter, we focus on one specific fingerprint of this kind, termed as *encapsulated TLS handshakes*, and the practicality of exploiting it in detecting obfuscated proxy traffic. Encapsulated TLS handshakes refer to TLS handshakes that take place within an encrypted or obfuscated cover protocol. As shown in Figure 3.2 (red), these handshakes are generated by user applications (e.g.,, browsers), and the proxy protocol then encrypts and transports them to the proxy server, which in turn forwards them to the final destination (e.g.,, web servers). This differentiates them from standard TLS that has plaintext structures and is visible to network intermediaries. We explain how the presence of encapsulated TLS handshakes indicates nested or redundant protocol stacking, which in turn signals that the connection carries proxied traffic. While encapsulated TLS handshakes cannot be easily identified by protocol parsers, we show that the packets exchanged during an (encapsulated) TLS handshake exhibit distinct patterns in their size, timing, and direction. These patterns remain visible even after encryption, enabling a censor to reliably identify encapsulated handshakes without breaking the encryption of the cover protocol.

We evaluate the practicality of fingerprinting obfuscated proxy traffic with encapsulated TLS handshakes, from the perspective of a censor or an adversarial ISP. We aim to answer: *How practical is this fingerprinting attack? Can censors deploy it at scale without significant collateral damage?* Addressing these questions requires not only identifying a vulnerability but also demonstrating feasible exploits, all while considering the real-world operating

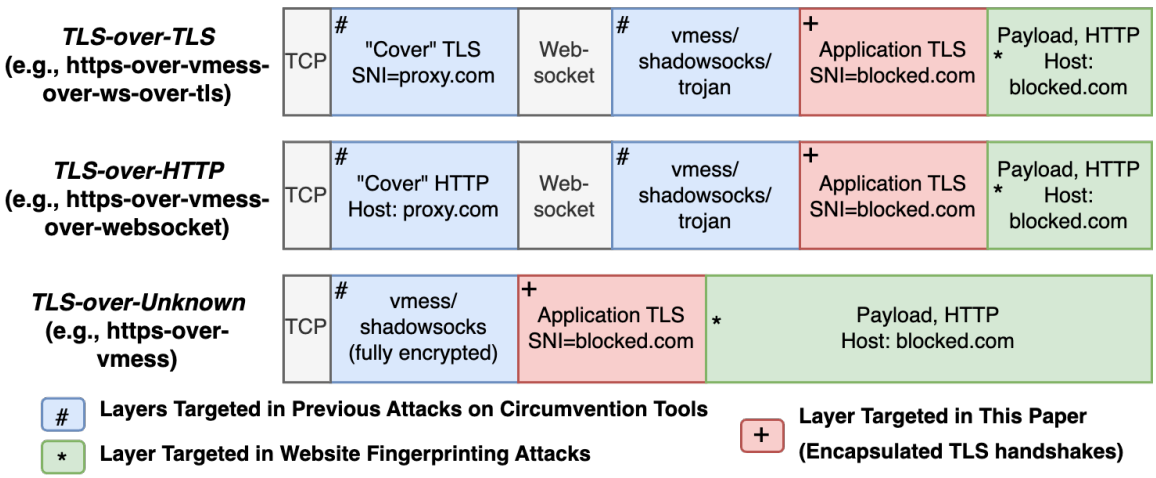


Figure 3.2: **Comparison with previous attacks on circumvention tools.** Prior work targeted cover or proxy protocol's failure to achieve indistinguishability, while this paper exploits fingerprints of the encapsulated layers within, which are agnostic to proxy protocols.◇

constraints of the censors, such as their sensitivity to false positives. To this end, we develop a detection framework using similarity-based classifiers, following a *conservative* censor capability model informed by empirical studies on real-world censorship events. Next, in collaboration with *Merit*, we deploy our framework within the ISP's network, adopting the position of a censor to evaluate the potential impact and collateral damage if the fingerprinting attack were to be widely deployed.

We test 23 obfuscated proxy configurations, including major circumvention protocols used by millions around the world like *shadowsocks*, *vmess*, *trojan*, and *vless*, and protocols proposed in previous research like *httpt* [19, 26, 24, 83]. We find that *all* tested obfuscated proxy protocols, in their standard configurations, are vulnerable to fingerprinting based on encapsulated TLS handshakes, with true positive rates (TPRs) all exceeding 70%. Such TPRs carry significant implications (§ 3.3), especially as a single detection can result in a censor blocking all subsequent connections to the corresponding proxy server. Even advanced configurations that encapsulate proxy traffic within additional cover layers, such as *shadowsocks* within websocket within TLS, provide limited defense and only marginally lower the detection rates. Contrary to common belief, we find that random padding slightly increases the fingerprinting difficulty, but not to a degree sufficient to prevent the exploit.

Even for more aggressive padding schemes like *XTLS-vision* and *obfs4*, we demonstrate that censors could easily adapt their statistical models and rely only on packet order and direction, rather than size, for detection.

During a 30-day evaluation, our deployment within *Merit* processed over 110 million flows, maintaining an upper-bound false positive rate as low as 0.0544%. Such precision positions our fingerprinting attack on par with approaches previously deployed by real censors [216]. Notably, our technique remains fully passive, agnostic to unpadded obfuscated protocols, and can be adapted to identify padded variants.

Obfuscated proxies are essential tools for facilitating unrestricted access to information and resources on the Internet. However, our findings suggest that traffic of obfuscated proxies, even with padding and layers of encapsulation, can be effectively detected by fingerprinting encapsulated TLS handshakes. Our approach shifts the focus from the cover layer to the encapsulated layers within, introducing a new threat dimension that is orthogonal and potentially complementary to existing attacks against circumvention tools. While we show that stream multiplexing could be a viable mitigation, we caution that obfuscations relying solely on padding and multiplexing are inherently limited. Proxy developers should be aware of the limitations of current obfuscation strategies, anticipate the potential exploitation of encapsulated TLS handshakes by censors, and equip their tools with proactive countermeasures.

3.2 Background & Related Work

3.2.1 Internet Censorship and Circumvention

News, anecdotes, and measurement studies collectively suggest that governments have been increasingly censoring the Internet on a global scale [164, 154, 137]. These studies document how governments practice censorship and, in particular, censors' technical capabilities, ranging from blanket IP-based blocking [146, 23, 67], content-based blocking [155,

147, 95, 49], to blocking of protocols and circumvention tools [225, 210, 66, 31, 216]. In addition to exposing otherwise covert practices of Internet censorship, these measurement studies contribute to the understanding of real-world censors and assist in approximating a realistic threat model [182]. Developers and researchers rely on this knowledge when building circumvention tools and evaluating the practicality of potential attacks against them.

Users from regions with restrictive censorship policies have been seeking ways to circumvent the censorship using various approaches, which can be roughly classified into strategy-based and channel-based circumvention. Strategy-based circumvention [202, 43, 53, 116] involves the client or server implementing custom packet manipulation rules, such as dropping or injecting specially crafted packets, in order to confuse middleboxes into not recognizing censorship triggers. On the other hand, channel-based circumvention [182] involves users establishing *channels* to *forwarders* located outside the censored region, which then route the users' traffic to its final destination. Obfuscated proxies fall into the latter category.

Channel-based circumvention necessitates obfuscation mechanisms. Most obfuscation mechanisms either make channels look like protocols that are allowed (i.e., mimicry), or disguise channels so they don't look like protocols that are forbidden (e.g.,, randomization). Mimicry-based obfuscation simulates traffic characteristics of popular, permitted protocols (e.g.,, HTTP, Skype [64, 127, 208]). Some even take the mimicry approach to the extreme by tunneling circumvention traffic through actual, legitimate network services like cloud storage, email, VoIP, and DNS [47, 4, 233, 99], resulting in more plausible protocol fingerprints [97]. On the other hand, randomization-based obfuscation approaches aim to eliminate all fingerprints by encrypting traffic into bits indistinguishable from random. Examples of these include *shadowsocks*, *vmess*, and *obfs4* [19, 26, 140], which defeat protocol parsers with the absence of a fixed protocol structure.

3.2.2 Attacks on Obfuscated Circumvention Tools

Previous research has documented several active and passive attacks by real-world censors against obfuscated circumvention tools. Active attacks involve censors sending specially-crafted probes to suspected endpoints and analyzing their responses. The GFW was able to identify Tor, VPN, and shadowsocks servers by sending connection requests of a targeted protocol and monitoring if the responses conform to the expected protocol behaviors [66, 31, 210]. In response, "probe-resistant" proxies were developed, which remain silent when probed by an unauthenticated client [80]. However, recent work shows that it is still possible to identify a server even without explicit server responses, due to application-specific features at the network level, such as how and when a server closes a connection [227, 196, 21].

Passive attacks have been used to detect both mimicry and randomization-based obfuscations. Houmansadr et al. argue that obfuscation by imitation is "fundamentally flawed" as seamlessly simulating all implementation-specific behaviors is too difficult and suggest tunneling as an alternative [97]. However, even when circumvention traffic is tunneled, detection remains possible through fingerprinting implementation discrepancies of the cover protocol. Past examples include the blocking of Tor due to the unique fingerprints in its custom TLS implementation, such as uncommon ciphersuites [210, 81, 22]. Snowflake [20], a Tor obfuscation based on WebRTC, was similarly fingerprinted by its unusual DTLS behaviors [50, 78]. Non-content features can also be exploited, such as excessive TCP ACK packets [197] or abnormal packet size distribution [38, 87]. While randomization-based approaches eliminate many of these potential fingerprints, looking like random could itself become a feature. Wang et al. demonstrate that packet length and entropy could identify fully encrypted proxies like obfs4 [197], and more recently, the GFW was found to block fully encrypted traffic using entropy as one of the features [216]. In response, developers have implemented several countermeasures, such as allowing users more fine-grained control over the fingerprints and entropy of the cover protocol [81, 88, 11].

3.2.2.1 Contextualizing Our Fingerprinting Approach

The fingerprinting attack discussed in this paper is orthogonal to all aforementioned attacks that target circumvention protocols. Both active and passive attacks found in previous work exploit implementation flaws in the cover/proxy protocols (depicted in blue in Figure 3.2). In contrast, this work focuses on the fingerprints generated by encapsulated application traffic (user-generated traffic inside the tunnel, e.g.,, from browsers), illustrated in red in Figure 3.2. This means that 1) our attack can potentially complement existing active or passive fingerprinting strategies targeting the cover protocols to further improve accuracy; and 2) existing countermeasures such as *utls* [81] are not effective against our approach, as they act on the cover protocol rather than the encapsulated layers within. A particular point of emphasis is to distinguish our approach from "TLS fingerprinting" as characterized in previous work [81, 28, 101]. These previous efforts aim to identify TLS implementations by looking at fields within ClientHellos of the cover/surface layer (e.g.,, distinguishing between TLS flows generated by Tor browsers versus those by Chrome). In contrast, our work looks for patterns that indicate the presence of TLS handshakes that are encapsulated within encrypted/obfuscated protocols.

The works most closely related to ours are those that infer the underlying protocol and protocol semantics from encrypted streams [33, 214, 89, 149], using features that remain visible after encryption, such as packet size, timing, and direction.

3.3 Threat Model

We outline a realistic threat model for the fingerprinting attack evaluated in this paper. Although real-world censorship practices are often covert, we approximate a *conservative* censor capability model informed by previous work on the arms race between censors and circumvention tools [182, 97, 66, 31, 216]. We assume an *on-path* censor performing passive traffic fingerprinting attacks on all connections passing through its network. The censor can-

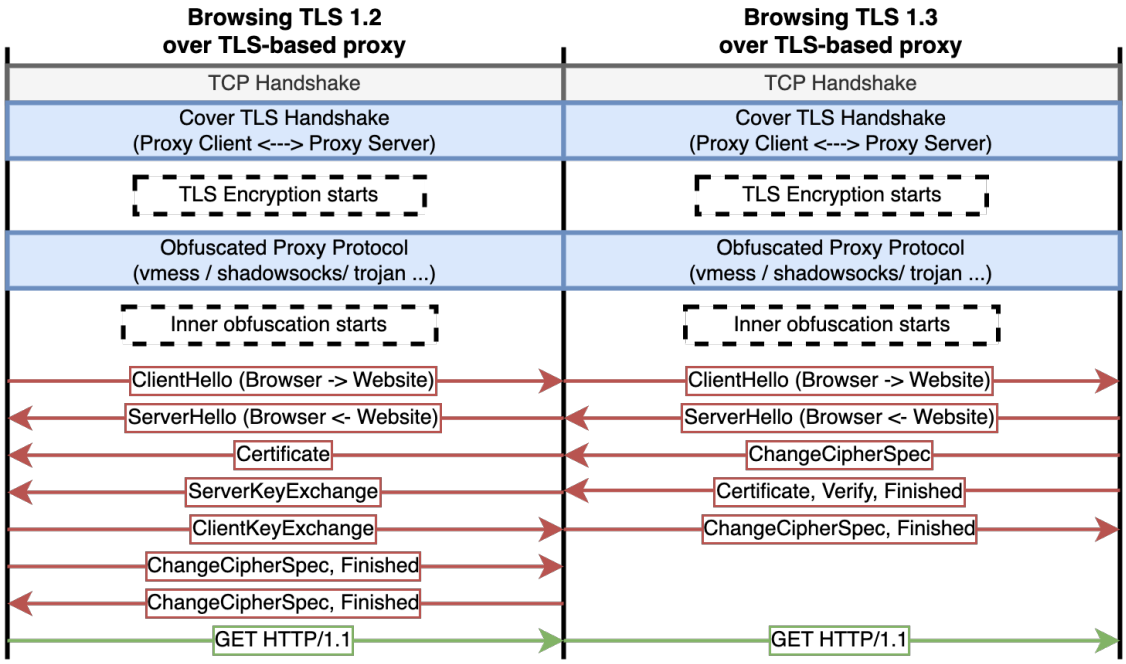


Figure 3.3: **TLS handshakes inside TLS-based proxy.** Messages transmitted during the handshake stage are well-specified and exhibit characteristics in their size, timing, and direction.◇

not inject, drop, or modify any passing packets, nor actively probe suspected servers. While the censor is stateful, it is bounded by memory and computational resources. This means the censor can maintain only a limited number of states per connection within a limited observation window. The censor is familiar with the targeted obfuscated proxy protocols and can adapt its detection methods as needed. However, it cannot break the encryption or obfuscation of the targeted protocol if standard cryptographic primitives are used. We note that these assumptions align with a *weak* adversary as defined in previous work [97].

The censor's reactions to false positives and false negatives are asymmetric. The censor is highly sensitive to collateral damage – the economic costs associated with falsely blocking legitimate traffic during censorship efforts. The censor's reluctance to cause collateral damage is essentially what enables circumvention to be possible [74]. Furthermore, due to the low base rate of circumvention traffic, even a seemingly low false positive rate (1%) can still be economically impractical [227, 197]. On the other hand, a moderate false negative rate can be considered acceptable. This is because circumvention tools users are likely to

use them on a daily basis, generating hundreds of flows for each browsing session, and the censor only needs to detect a single flow to block the proxy server.

3.4 Encapsulated TLS Handshakes

We begin by providing the intuition of how we fingerprint obfuscated proxy traffic. Fundamental to all proxying and tunneling activities is the concept of nested protocol stacks, where one protocol stack is encapsulated within the payload of another, in order to pass through network intermediaries without being directly exposed. This observation forms the basis of the fingerprinting attack we evaluate in this paper and renders the approach agnostic to the choice of proxy protocols. Nested protocol stacks are rare in regular, direct client-server connections. In proxied connections, however, a cover protocol serves as a transport between the client and proxy, while an encapsulated application protocol facilitates end-to-end communication between the client and the final destination server. Nested protocol stacks can result in violations of the layering of the OSI model, such as when the same protocol is encapsulated within itself, as in the case of a TLS-based proxy encapsulating HTTPS traffic (i.e., *TLS-over-TLS*).

The presence of nested protocol stacks is indicative of proxying. Identifying this behavior, however, can be challenging, as the cover protocol encrypts or obfuscates its payloads. Previous work has approached this problem by modeling non-proxied connections and detecting anomalies by comparing their behaviors to the model [63]. Nevertheless, precisely characterizing "legitimate" traffic can be challenging due to the diversity of traffic and protocols on the Internet. For example, while HTTPS constitutes a significant fraction of TLS traffic, TLS payloads deviating from the expected HTTP behaviors do not always imply a second protocol stack, since there are other protocols that use TLS for encryption.

Instead, our approach focuses on fingerprinting a unique behavior that results from nested protocol stacks, termed *Encapsulated TLS Handshakes*. Encapsulated TLS handshakes refer to the TLS handshakes of user traffic transmitted within an encrypted or obfuscated cover

protocol (including an outer layer of TLS). This contrasts with standard TLS with plaintext headers and structures that are identifiable by protocol parsers, as shown in Figure 3.2. The rationale for using encapsulated TLS handshakes as indicators of proxying is intuitive: since TLS is designed to provide end-to-end security over an unsecured network, an encapsulated TLS session within an already-encrypted cover channel suggests redundant protocol stacking and implies that the outer encryption is not carried through to the final destination but is instead terminated earlier, e.g.,, at the forwarding proxy server.

Several aspects of encapsulated TLS handshakes make them an ideal fingerprint for proxy traffic. **Distinct:** Similar to standard TLS handshakes, encapsulated TLS handshakes are also well-defined, and the packets transmitted during the handshake stage exhibit distinct size, timing, and directional characteristics, as shown in Figure 3.3. For example, most *ClientHello* messages typically range from 200 to 550 bytes depending on the TLS version, whereas *ServerHello* messages with certificates can span several thousand bytes and exhibit a higher degree of variation across flows. Moreover, as each handshake message logically depends on the preceding message transmitted, an on-path adversary can exploit the order in which packets carrying these messages are observed on the wire. These characteristics remain visible even after encryption and thus allow us to identify encapsulated TLS handshakes without breaking the cover protocol. **Reliable:** TLS is ubiquitous on the Internet, providing a foundation for secure communication across a variety of applications. As such, encapsulated TLS handshakes serve as a reliable fingerprint, as it would be impractical for proxy users to avoid generating such fingerprints by not using TLS. **Precise:** As described earlier, encapsulated TLS handshakes are indicative of proxied connections. In § 3.7, we provide empirical evidence to demonstrate that this behavior is fairly unique to the use of proxies. Consequently, using encapsulated TLS handshakes as a fingerprint results in minimal collateral damage for censors.

if *is_obfuscated_or_encryptedflow* :

extract: *size, timing, direction* from *flow[payload]*

if *matches_TLS_handshakesize, timing, direction* :

log_as_proxy_trafficflow

The pseudo-method above conceptualizes how censors could exploit encapsulated TLS handshakes for proxy detection. In the following sections, we demonstrate the practicality of this approach by building a detection framework using similarity-based classifiers and evaluating it in a real network.

3.5 Ethical Considerations

Research on censorship and circumvention is inherently sensitive, especially when it involves raw network traffic containing real users' data. For this, we sought approval from our university's IRB for our research plan. Although the IRB determined that the work is "Not Regulated", we take extensive procedural and technical measures to minimize potential risks.

A primary ethical concern relates to the handling of raw network traffic. We access live ISP traffic via our *Monitor* within *Merit* to train classifiers and evaluate the proposed fingerprinting attack. Our *Monitor* deployment is overseen by *Merit*, who has extensive collaboration experience and well-established ethics and privacy guidelines for such projects. The *Monitor* receives only a mirrored copy of traffic without handling the actual routing, ensuring that *Merit*'s service remains unaffected. The raw traffic is processed by a Zeek [231] cluster that carries out protocol parsing and feature extraction (§ 3.6.2). Instead of recording raw traffic, we log only extracted features, consisting of sequences of packet sizes and timing, without including any packet payloads. These logs are stored and processed on a *Merit*-maintained server, accessible only to select members of the team on a least-privilege

basis. We emphasize that *at no point during the project's duration are packet payloads recorded to disk or inspected by humans*.

As with all research on attack methodologies, there is a risk that the fingerprinting attack evaluated in this paper could be deployed by real-world adversaries. We are currently disclosing relevant information to the developers of the proxy protocols examined in our study. It is worth noting, however, that there have already been speculations in the community regarding the potential exploitation of TLS handshakes by GFW in its recent attempts to block circumvention tools [41, 13]. As these obfuscated proxies are used by millions of people to circumvent censorship, we believe it is crucial to bring these vulnerabilities to light, especially considering the realistic threat they pose, as evidenced in this paper. Our work intends to provide users with accurate information about the proxy tools they depend on and encourage proxy developers to implement principled countermeasures.

3.6 Similarity-Based Classification

Our objective is to distinguish traffic generated by obfuscated proxies from non-proxied user traffic. At the same time, it's equally important to identify the specific features that contribute to detection, as this knowledge will inform future developments in obfuscated proxy protocols. For example, although deep learning–based classifiers have been used in previous research to evaluate the distinguishability of proxy/VPN traffic [119, 84, 201, 3, 25], the limited explainability of deep learning models makes them less suitable for investigating specific fingerprinting vulnerabilities, such as the encapsulated TLS handshakes in our study.

We approach our problem as a binary classification problem – determining whether (obfuscated) network traffic flows contain TLS handshake sequences. We explored several options for classification, such as clustering or supervised learning techniques. We found that classifiers constructed with similarity-based metrics achieve a precision level (in terms of false positive rates) that we consider practical for a censor to deploy, while being computationally efficient. In similarity-based classification, similarity scores are derived

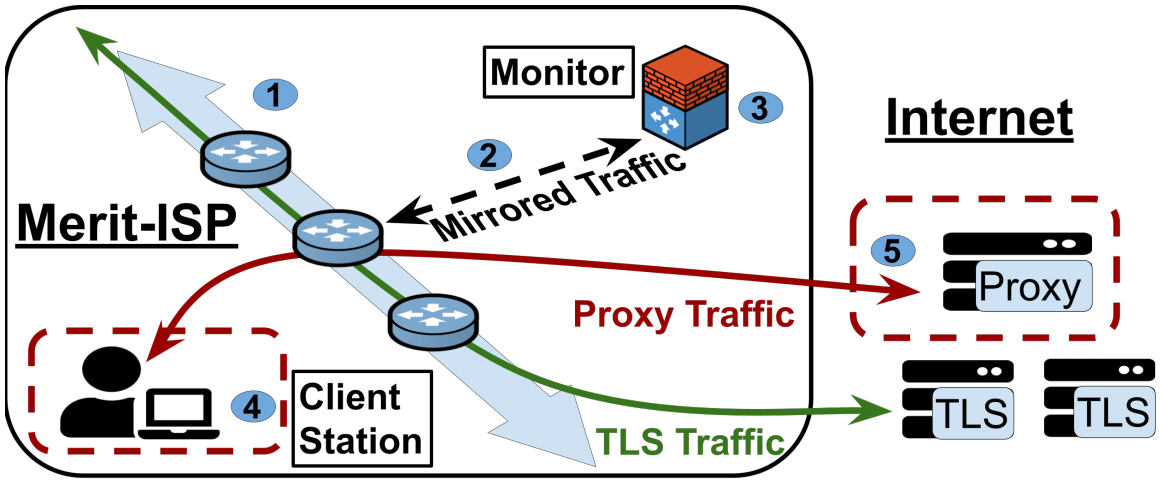


Figure 3.4: ***Merit* Setup** for collecting plain TLS/non-TLS traffic (1–3:§ 3.6.1) as well as obfuscated proxy flows (4–5:§ 3.7.1.2).◇

for each class by applying selected distance metrics to the test sample and training datasets. Next, a class label is estimated for the test sample based on its similarity scores to each class [51].

In this section, we collect a large training dataset comprising TLS/non-TLS flow records by applying a protocol parser to traffic passing through a major *Merit* point-of-presence. Next, we build two classifiers using the training datasets and task them with identifying TLS handshakes *using only features visible post-encryption*, i.e., packet size, direction, and inter-arrival time. We evaluate our classifiers first on *plain* TLS/non-TLS traffic for both TLS version 1.2 and 1.3. In the following section, we adapt our classifiers to detect *encapsulated* TLS handshakes from obfuscated proxy traffic.

3.6.1 Data Collection

To compile a training dataset with TLS and non-TLS traffic, we set up our monitoring station within *Merit*, as shown in Figure 3.4, which oversees 50 Gbps of ingress and egress traffic mirrored from a backbone router. First, a user inside *Merit* initiates a TLS connection to an external TLS server (1). As the traffic passes through the monitored router, a copy of the traffic is sent to our *Monitor* (2). The *Monitor* operates a cluster of 23 Zeek [231] instances running our custom script (3). Due to the large traffic volume, we optimize our

TLS flows	All flows	26,500,694
	TLS 1.2	10,851,340 (40.95%)
	TLS 1.3	15,649,354 (59.05%)
Non-TLS flows	All flows	7,020,287
	Unknown Protocols	479,982 (6.84%)

Table 3.1: **TLS/non-TLS dataset for training and testing classifiers.**

setup with PF_Ring to enhance packet processing speed [148]. We apply the following filtering rules when creating datasets for training and testing: For both TLS and non-TLS flows, we filter only TCP and require the observation of both a SYN packet and a SYN-ACK packet in the reverse direction. This ensures symmetric routing and that we see the beginning of the connection, where protocol handshakes are expected to occur. We record the first W_o data-carrying TCP packets transmitted in either direction for analysis. We discard short-lived flows with fewer packets than the observation window W_o. For TLS flows, we differentiate between version 1.2 and 1.3 based on the *Supported Version* extension within *ServerHello* records [17]. We filter out flows with an empty *Server Name Indication* field and flows where no TLS records with *Application Data* type are observed (e.g.,, failed handshake).

Despite the optimizations implemented to improve packet processing speed, handling traffic at 50Gbps on a single server still results in non-negligible packet loss due to CPU bottlenecks. Therefore, we sample only 1/8 of all TCP flows arriving at the *Monitor*. This sampling is based on IP pairs. A detailed breakdown of the collected datasets is shown in Table 3.1.

3.6.2 Feature Extraction

Each flow is represented as a sequence of integers, where the absolute value of each integer corresponds to the size of the TCP payload, and the sign indicates the packet's direction. We use SEQ/ACK analysis to establish a total ordering in which the packets should be processed. An example of such a sequence is $517, -1400, -1400, 80$. Additionally, we record the inter-arrival times between packets.

We extract n-grams from these sequences of packet sizes. n-grams have been used in previous work to analyze encrypted DNS and VoIP traffic [215, 117, 171]. In our case, the intuition is that, compared to 1-grams, n-gram representations can better differentiate between bulk transfer and interactive communications, such as protocol handshakes. Moreover, for protocols with well-defined handshake processes like TLS, we anticipate that n-grams can capture patterns in size tuples during state transitions. For example, the first 3-gram extracted from the previous packet sequence is $517, -1400, -1400$, which often signals an outgoing *ClientHello* followed by a segmented *ServerHello*. Such patterns would be obscured in a 1-gram representation. We experimented with 2-gram, 3-gram, and 4-gram, and found that 3-gram provides the best balance between performance and memory requirements.

We augment the n-gram representation with "burst" features, which consider packet directions and inter-arrival times. Bursts refer to sequences of consecutive packets traveling in the same direction, often occurring when TCP splits large application layer *write()* into smaller pieces for transport. Burst sequences are more robust to minor changes at the packet level, as they consider only the aggregated traffic volume and the direction in which it travels. Specifically, given an original TCP flow record and the corresponding inter-arrival times, we first estimate the connection's RTT as the time elapsed between the observation of the SYN packet and the arrival of the first ACK packet sent from the client at the *Monitor*. To construct burst sequences, we aggregate the sizes of consecutive packets that 1) travel in the same direction and 2) have an inter-arrival time less than three times the estimated RTT.

3.6.3 Classifiers

To make a classification, we deploy two tests – a Chi-squared (χ^2) test over the 3-gram representation, and a Mahalanobis distance metric over the burst sequence. We classify a flow as TLS only when both tests are satisfied. The intuition behind this approach is that the

Algorithm 2 Chi-squared Test over 3-grams.
T_0, T_1:training sets; G:set of all 3-grams; δ:decision threshold
s:test sample; M:size mapping; f:dimensionality parameter

Input: $T_0, T_1, G, \delta, s, M, f$
Training:
$T_c = Mx$ for x in $T_c, c \in \{0,1\}$
$Prc, g = \frac{1}{t \in T_c N_t} t \in T_c N_t \times Prt, g, c \in \{0,1\}, g \in G$
$Distincg = _{c \in \{0,1\}} \frac{1}{|T_c|} t \in T_c Prt, g - Prc, g^2$ /
$\frac{1}{c \in \{0,1\} |T_c|} c \in \{0,1\} t \in T_c Prt, g - Pr\bar{c}, g^2 \ \varepsilon, g \in G$
$F = SortedG, key = Distincg, reverse = 1, g \in G0 : f$
Testing:
$s = Ms$
$Ds, c = _{g \in F} \frac{1}{Prc, g} Prs, g - Prc, g^2, c \in \{0,1\}$
Return $Ds, 1Ds, 0 \ \varepsilon \geq \delta$

former test captures local orders of packet sequences and state-transition patterns, whereas the latter offers a more aggregated view of a flow's dynamics.

3.6.3.1 Chi-squared Test

Chi-squared test is a statistical method for determining whether the distributions of categorical variables significantly differ from one another, by comparing the observed frequencies with the expectation for each category. Specifically, the expected number of occurrences of each 3-gram can be found given training sets T_0 and T_1 for non-TLS and TLS, respectively, as well as a set of all 3-grams G. Next, given a test sample s, Chi-squared distances for each class are derived, and we classify the sample by comparing the ratio of the two distances with a decision threshold δ.

However, using raw packet sizes to generate all possible 3-grams leads to an extremely high-dimensional feature space. For a standard MTU of 1500 bytes, the number of unique 3-gram features would be 3000^3. We implement two measures to reduce dimensionality. First, raw packet sizes are discretized into groups $\in L$ before generating 3-grams. We experimented with different grouping strategies (e.g.,, equal-width, equal-frequency) and found that the strongest results are obtained when packets carrying the same semantic mean-

ings (i.e., TLS Record Type) are grouped together. Using Zeek [231], we label handshake packets from the training set according to their TLS handshake types. Next, given group size $|L|$, we derive a custom mapping that attempts to map sizes associated with the same TLS handshake type into the same group. The mapping function preserves directionality; that is, same packet types in different directions are grouped together, distinguished by their respective polarities. For our analysis, we set $|L| = 4$ to differentiate *ClientHello*, *ServerHello*, *ChangeCipherSpec*, which exhibit the most distinct size patterns, from other packets. We then search for a mapping that separates each of the three handshake types while minimizing the packets deviating from their respective groups. We use the mapping $M = L_1 : 1-160, L_2 : 161-600, L_3 : 601-1210, L_4 : 1211$. For example, 99.26% of *ClientHellos* in our training dataset are mapped into L_2. This step reduces the dimensionality to $4 * 2^3$. Furthermore, inspired by [215], we only use 3-grams that provide the best distinguishability by selecting those that exhibit low variance among samples from the same class but high variance among samples from different classes.

The overall training and testing process is outlined in Algorithm 2, where N_t and Prt, g correspond to the number of all 3-grams and the probability of the specific 3-gram g in flow t, respectively. The variables f and δ control the number of 3-gram features considered in the model and the decision threshold. $\bar{c}$ denotes the opposite class of c. As an example, Table 3.2 lists the top five 3-gram features ranked by the distinguishability they provide. As expected, we observe that 3-grams most likely representing state-transition patterns are among the top features that offer the best distinguishability.

3.6.3.2 Mahalanobis Distance

The Mahalanobis distance provides a method to measure the multivariate distance between a point and a distribution, incorporating both the variances of each variable and the inter-variable correlations. In our use case, we apply the Mahalanobis distance to the burst representation of a given flow, where each burst is considered as a separate dimension. Our

TLS Labels	Distinc.
(*ClientHello*, *ServerHello*, *ClientKeyExchange*)	7.226
(*ServerHello*, *ServerHello (cont.)*, *ServerHello (cont.)*)	5.886
(*ServerHello*, *ClientKeyEchange*, *ServerChangeCipherSpec*)	2.879
(*ServerHello*, *ServerHello (cont.)*, *ServerKeyExchange*)	2.780
(*ClientHello*, *ServerHello*, *ServerHello (cont.)*)	2.416

Table 3.2: **Top Tri-grams with TLS labels.**$\diamond$

rationale for this approach stems from the observation that each phase of the TLS handshake typically forms its own burst, as successive stages transmit packets in alternating directions. Additionally, while individual packets within different TLS handshakes can exhibit wider variability, the aggregated dynamics of a flow – characterized by traffic volume and direction – are likely more consistent across samples. In contrast to distance metrics like the Euclidean distance, the Mahalanobis distance effectively normalizes for the variance within each dimension by considering the covariance matrix. In other words, it takes into account the relative importance of changes in each dimension based on their expected variability. For example, the Mahalanobis distance computation will inherently recognize and accommodate the fact that the sizes of *ServerHello* bursts might vary more across samples than the sizes of *ClientHello* bursts.

Algorithm 3 outlines the process of measuring the Mahalanobis distance, given a TLS training set T_1, a test sample s, a burst window W_b, and a detection threshold γ. We set W_b to be $2 \times RT$ 1, where RT is the minimum number of round trips required for a *full* TLS handshake. Therefore, for TLS 1.2 and 1.3, W_b is set to 5 and 3, respectively.

3.6.4 Evaluation on plain TLS/non-TLS traffic

We train and test our classifiers using the TLS/non-TLS dataset, which is fully labeled with protocol parsers as described in § 3.6.1. We partition the TLS class flows based on their respective TLS versions and train a separate classifier for each version. The negative class consists of non-TLS flows, shared between both classifiers. We set the observation window as $W_o = 25$, a length that we found to be sufficient for over 99.95% of TLS handshakes in our

dataset. Additionally, we set the number of 3-gram features under consideration as $f = 100$. Upon completing the initial round of training, we compute the Chi-squared distance and Mahalanobis distance for each sample within the training set. Then, we remove outliers exhibiting abnormally large distances from the models and re-derive the models with the remaining samples.

We evaluate the classifiers under varying Chi-squared and Mahalanobis distance thresholds. Figure B.1 shows the performance of both classifiers using the parameter sets that yield the highest TPRs under specific FPR constraints. As expected, the TLS 1.2 classifier outperforms the 1.3 classifier across a range of threshold configurations, especially in terms of minimizing the false positive rate (FPR). This can be intuitively explained by the fact that a full handshake in TLS 1.2 requires one additional round trip compared to version 1.3, which manages to exchange all necessary information in the initial round trip, resulting in shorter handshake sequences. As our Mahalanobis classifier uses a moving window to identify potential handshake sequences, this means that in TLS 1.3, fewer elements need to match consecutively in the stream, increasing the likelihood of coincidental matches. We emphasize that the low FPR of classifying TLS 1.2 traffic makes it a more attractive target for censors, who tend to prioritize minimizing collateral damage. Regarding the source of false negatives, while we couldn't manually inspect raw traffic due to ethical concerns (§ 3.5), based on packet labels and sizes, we conjecture that irregular handshakes resulting from optimizations such as TLS Session Resumption and False Start [15, 16] may contribute to the majority of false negative instances.

3.7 Detecting Obfuscated Proxy Traffic

In this section, we evaluate the practicality of exploiting encapsulated TLS handshakes to detect traffic belonging to obfuscated proxies. In line with previous studies on censorship and fingerprintability [227], our goal is to conduct our evaluation in a realistic setting while incorporating perspectives on how censors operate in practice. To this end, we adopt

Algorithm 3 Mahalanobis Distance over Bursts
T_1:TLS training set; s:test sample; γ: detection threshold
$W_b = 5$ for TLS 1.2 and $W_b = 3$ for TLS 1.3

Input: T_1, s, W_b, γ
Training:
  $T_1 = x0 : W_b$ for x in T_1
  $M = GetMeanOfBurst T_1$
  $C = GetCovarianceMatrix T_1$
Testing:
  $Dis \leftarrow \{\}$
  for i in $0...|s| - W_b$ **do**
    Dis.add$\sqrt{si : i\ W_b - M^T C^{-1} si : i\ W_b - M}$
  end for
  Return $minDis \leq \gamma$

the viewpoint of a potential censor by deploying our detection framework within *Merit*. As illustrated in Figure 3.4, our framework processes two types of traffic: obfuscated proxy traffic generated from a control machine (4) to a proxy server (5), with our *Monitor* positioned on-path, as well as legitimate, passing network traffic produced by *real* users served by *Merit*. Throughout this section, we aim to address three research questions: *(1) How effective is the proposed fingerprinting attack? (2) Can current countermeasures, such as random padding and stream multiplexing, effectively defend against the attack? (3) How practical is it for censors to deploy at scale without causing significant collateral damage?*

3.7.1 Experiment Setup

3.7.1.1 Selection of Proxy Configurations

We compile a list of obfuscated proxy configurations based on previous research [182, 31, 66, 216], online censorship discussion forums [9, 1], and our experience with circumvention tools used in regions with strict information control. We do not include configurations that lack obfuscation, such as plain HTTP/SOCKS proxy or VPN protocols, as they are trivially detectable. We also exclude "stealth VPNs" as marketed by commercial VPN providers, since previous work has shown that most of them do not deploy standard obfus-

cation mechanisms and can be easily probed and detected [227]. The proxy configurations we examined are shown in Table 3.3.

Basic Configurations. include raw protocols such as *vmess*, *Shadowsocks*, *Trojan* (*trojan-go*), and *vless* [26, 19, 24]. These protocols are specifically designed for censorship circumvention, and as such, obfuscation is among their top priorities. *Shadowsocks* and *vmess* are random-looking proxies that use encryption to make their traffic indistinguishable from random bytes. *Trojan* and *vless* are TLS-based protocols, encapsulating traffic inside TLS tunnels to avoid detection.

Advanced Configurations. build upon raw protocols, adding additional "cover" layers to encapsulate proxy traffic inside popular, legitimate protocols. Examples include vmess-over-tls, vmess-over-websocket, shadowsocks-over-websocket-over-tls, *etc*. This category also includes dedicated pluggable transports like Cloak and shadowTLS [5, 7], which improve the censorship-resistance of *Shadowsocks*, and httpt and gost [6, 83], which are probe-resistant proxy systems that hide proxy traffic inside TLS tunnels.

MUX Configurations. combine multiple application streams within a single TCP connection. Without multiplexing, each TCP stream from local applications arriving at the proxy client would result in a new outgoing TCP connection to the proxy server. With multiplexing enabled, a small number of long-lived connections are maintained between the proxy client and server, with all application streams being multiplexed inside these connections. Although multiplexing is often discussed as a performance optimization, previous work has found that multiplexing can mitigate the effect of traffic analysis attacks [129, 106]. Table B.1 in Appendix lists the implementations examined and their multiplexing support.

Random Padding Configurations. involve appending dummy data to payloads in order to obscure patterns in packet sizes. For obfuscated proxies, padding is implemented as part of the traffic obfuscation mechanism. The method for choosing a padding length varies across protocols: some select a padding length uniformly at random from a range

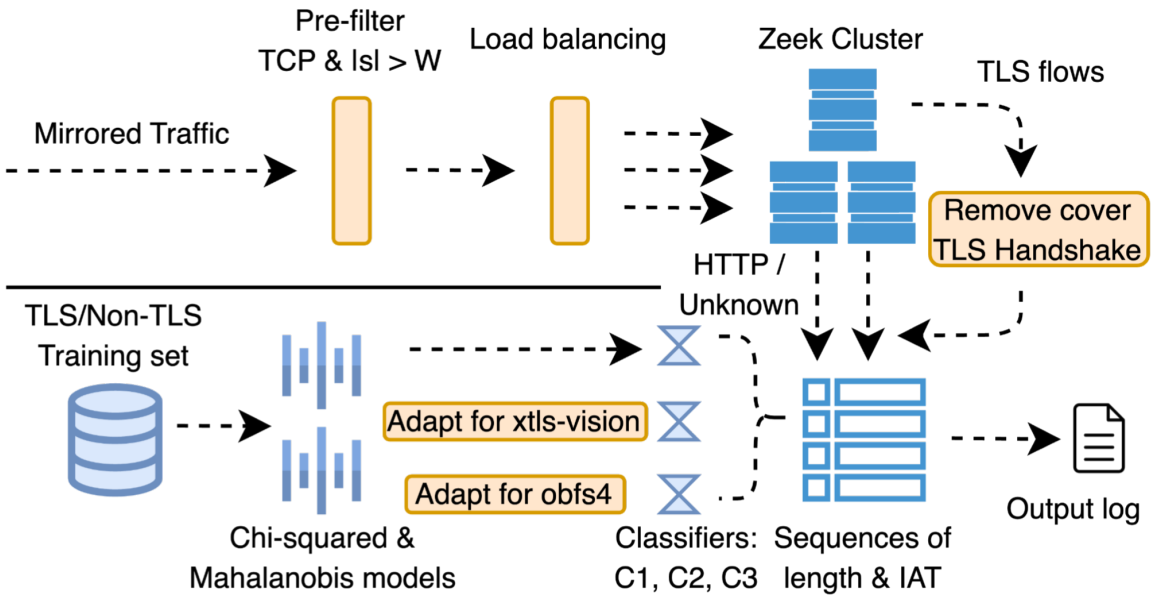


Figure 3.5: **Detection Framework in *Merit*.** For TLS flows, we remove the handshake of the cover TLS layer before feature extraction in order to only identify handshakes that are encapsulated.◇

and append to each packet [26, 8], while other proxies pad all packets to a target range [219] or according to a specific distribution [140].

3.7.1.2 Proxy Traffic Generation

To generate and test on obfuscated proxy traffic, we use the *Client Station* inside *Merit* to connect to a server running proxy protocols. As shown in Figure 3.4, all traffic to/from the *Client Station* passes through a *Merit* backbone router, which is configured to send mirrored traffic to the *Monitor*.

On the *Client Station*, we run an automated script to generate proxy traffic. For each obfuscated proxy configuration, we connect to the proxy server and use Selenium [18] Firefox to visit the top 1K domains as ranked by Cloudflare [10] while collecting packet captures for reference. We note that the goal of browsing 1K domains is to generate a diverse set of TLS traces on proxy. However, the exploit is not dependent on web browsing – any proxied application using TLS will generate encapsulated TLS handshakes. We configure the Selenium browser to use 1.2 only when negotiating TLS versions. In § 3.7.4, we extend

the experiment to include TLS 1.3 and show that, at the moment, it is in censors' interest to focus on TLS 1.2, due to the higher precision that TLS 1.2 handshake fingerprints offer. For configurations with stream multiplexing, we restart the proxy client between consecutive domain visits. On average, this process generates over 15,000 flows exceeding the observation window $W_o = 25$ for non-multiplexed configurations. We note that some of these proxied flows might not be TLS and therefore do not contain an encapsulated TLS handshake, resulting in an underestimated True Positive Rate in our evaluation. However, as shown in § 3.7.4, we estimate that only 0.39% of flows are non-TLS connections when conducting experiments on Cloudflare's 1K list; thus, we ignore them when reporting TPRs.

3.7.2 Detection Framework

Our *Monitor* inside *Merit* receives up to 50 Gbps of mirrored traffic, which necessitates sampling due to CPU bottleneck in our single-server setup. We use a sampling rate of 1/8 to minimize the effect of packet loss, and we sample traffic based on 4-tuples so that both directions of a flow are either selected for analysis or dropped together. Traffic to/from the *Client Station* is not subject to sampling. We note that while nation-state censors could potentially scale up their deployment to avoid sampling, a recent work shows that GFW operates under an approximate sampling rate of 1/4, presumably to conserve resources and limit collateral damage [216].

As shown in Figure 3.5, once mirrored traffic arrives, it is first processed by a cluster of Zeek [231] instances running our custom script. We then use protocol parsers to extract the application-layer protocol for each flow. For flows that parsers cannot identify (e.g.,, fully encrypted protocols like *shadowsocks*), we label them as *Unknown*. Since all our proxy configurations have either *Unknown*, *TLS*, or *HTTP* as their (application-layer) cover protocols, we filter out flows with any other labels, which account for 0.89% of all flows exceeding the observation window W_o. [2] Next, we extract features including TCP payload

[2]Note that non-obfuscated protocols that use TLS for encryption, e.g.,, SMTP/SSL, POP3/SSL, IMAP/SSL, *etc*, are filtered out at this stage.

sizes and inter-arrival times (IAT) of the first W_o TCP data-carrying packets and pass them into our classifiers. Importantly, *for TLS flows, we remove packets forming the (cover) TLS handshakes prior to feature extraction* so that the classifiers disregard the "cover" TLS and instead only focus on identifying potential TLS handshakes that are encapsulated in the payload of the cover TLS (i.e., TLS-over-TLS).

We follow the process outlined in § 3.6 to derive the statistical models. Notably, the training datasets used here are the same as in § 3.6.1. This highlights an advantage of the proposed exploit over previous work: censors do not need to generate a synthetic training dataset of proxy traffic. Instead, they can model encapsulated TLS handshakes by observing real TLS traffic in their network. It's important to clarify that all evaluations in this section are conducted on live and proxy traffic, entirely separate from the training traces. To adapt the classifiers for detecting proxy traffic, we modify the mapping of packet sizes M to reflect the framing overheads introduced by the proxy protocols. Although this overhead varies between proxy protocols, they tend to be small in size (typically ranging between 20 to 60 bytes) due to performance considerations. While adjusting specific framing overheads for each proxy protocol might lead to marginally improved results, for the purpose of evaluation, we choose the median overhead size for all proxy protocols that have fixed framing overheads.

However, for protocols with complex random padding schemes, framing overheads vary across packets, making it challenging to separate packets with different semantic meanings based on sizes. For example, *XTLS-vision* [219] pads *all* packets at the beginning of a flow to a range of 900-1400 bytes, effectively nullifying individual packet sizes as a feature. Yet, packet directions can still serve as a feature, although less informative, because proxies we examine only pad payloads from upstream applications and don't send "dummy" packets when there is nothing to transmit. Thus, for *XTLS-vision* and *obfs4* [219, 140], we use a mapping that groups packets by their directions, regardless of their sizes ($|L| = 1$). Furthermore, we adapt the Mahalanobis model as these padding schemes lead to increased burst sizes and higher variances across flows, which needs to be accounted for when

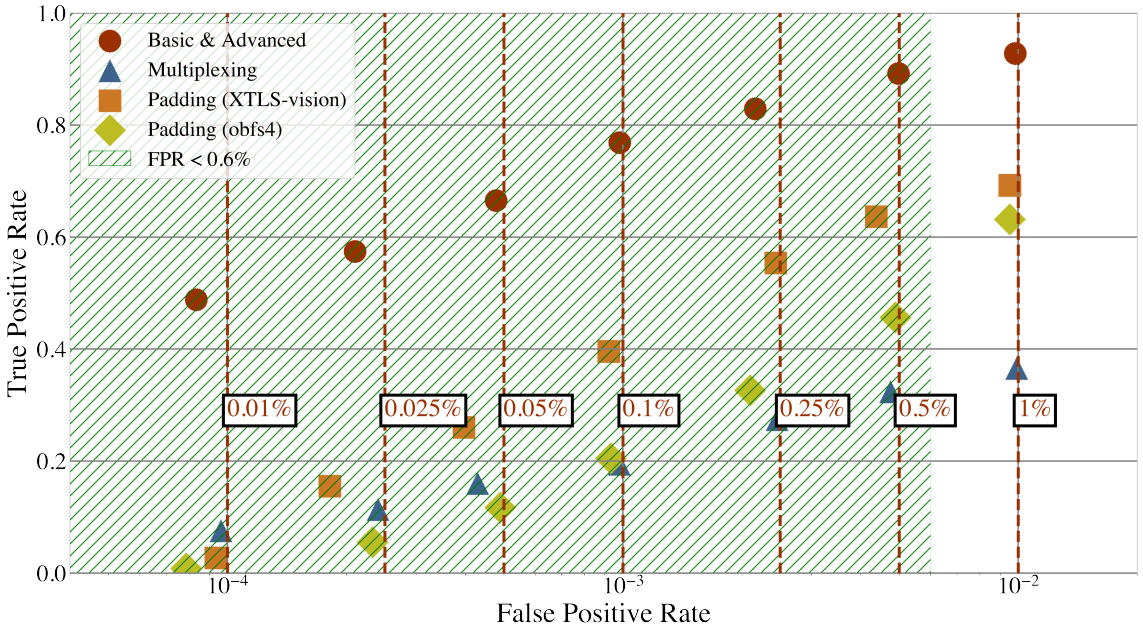


Figure 3.6: **Results by configurations.** Data points are obtained using the parameter set of Chi-squared threshold δ and Mahalanobis threshold γ that yield the highest TPRs under a specific FPR constraint. $\diamond$

computing distances. Specifically, we apply the padding procedures as implemented by *XTLS-vision* and *obfs4* to each flow in the training dataset, as detailed in Appendix B.1. Then, we derive the new means and covariance matrices based on the padded flow sequences.

3.7.3 Results and Findings

Beginning March 16, 2023, we deploy our detection framework on the *Monitor* for 30 days to evaluate it on both proxy and ISP traffic. Despite the sampling, the *Monitor* still processes over 36 terabytes of traffic daily, consisting of 34 billion packets from over 3.9 million flows exceeding the observation window $W_o = 25$. As *Merit* is located in a country with minimal censorship on network traffic, we expect that the vast majority of traffic seen on *Merit* does not contain traffic from *obfuscated* proxies designed for censorship circumvention. For this reason, we consider all ISP flows that our framework labels as proxy traffic to be false positives to derive a conservative upper-bound False Positive Rate (FPR).

Classifier	Group	Configuration	Application-layer Protocal	Random Padding	Multiplexing	TPR
C_1	Basic	vmess	Unknown	Yes	No	0.77135
C_1	Basic	shadowsocks	Unknown	No	No	0.85383
C_1	Basic	vless over tls	TLS	No	No	0.74830
C_1	Basic	trojan over tls	TLS	No	No	0.73705
C_1	Advanced	vmess over websocket	HTTP/websocket	Yes	No	0.78454
C_1	Advanced	vmess over tls	TLS	Yes	No	0.74463
C_1	Advanced	vmess over tls w/o padding	TLS	No	No	0.84071
C_1	Advanced	vmess over websocket over tls	TLS	Yes	No	0.68782
C_1	Advanced	vmess over websocket over tls w/o padding	TLS	No	No	0.85907
C_1	Advanced	shadowsocks over websocket	HTTP/websocket	No	No	0.83597
C_1	Advanced	shadowsocks over websocket over tls	TLS	No	No	0.69679
C_1	Advanced	shadowsocks over Cloak	TLS	No	No	0.78748
C_1	Advanced	shadowsocks over shadowTLS	TLS	No	No	0.82857
C_1	Advanced	vless over websocket over tls	TLS	No	No	0.70652
C_1	Advanced	httpt	TLS	No	No	0.87758
C_1	Advanced	gost	TLS	No	No	0.73543
C_1	MUX	vmess (concurrency=2)	Unknown	Yes	Yes	0.22520
C_1	MUX	vmess (concurrency=4)	Unknown	Yes	Yes	0.17635
C_1	MUX	vmess (concurrency=8)	Unknown	Yes	Yes	0.16754
C_1	MUX	vmess over tls (concurrency=8)	TLS	No	Yes	0.14841
C_1	MUX	vmess over websocket over tls (concurrency=8)	TLS	No	Yes	0.12534
C_1	MUX	trojan over tls (concurrency=8)	TLS	No	Yes	0.17943
C_1	MUX	shadowsocks	Unknown	No	Yes	0.18832
C_1	MUX	vmess over HTTP/2 over tls	TLS	No	Yes	0.36772
C_1	Padding	naiveproxy	TLS	Yes	Yes	0.32772
C_2	Padding	XTLS-vision	TLS	Yes	No	0.51281
C_3	Padding	SOCKS over obfs4	Unknown	Yes	No	0.43830

Proto.	Flow Count	Share	C_1 False Positive	C_2 False Positive	C_3 False Positive	Classifier	Overall FPR
TLS	105,542,111	0.8994	57,464 (0.0544%)	209,987 (0.1989%)	N/A	C_1	0.0544%
HTTP	10,021,983	0.0854	3,205 (0.0319%)	N/A	N/A	C_2	0.1989%
Unknown	731,446	0.0062	3,218 (0.4399%)	N/A	4,482 (0.6127%)	C_3	0.6127%

Table 3.3: **Evaluation Results.** Top: performance of the detection framework, broken down by proxy configurations. Bottom: False Positive Rates estimated on ISP traffic. We do not include *vmess* in the Random Padding group, as its padding scheme is limited (0-63 bytes).◇

We extract distances for each proxy and ISP flow, labeling a flow as "proxy" if both distances fell below specific thresholds. We identify the "optimal" thresholds and their corresponding True Positive Rates (TPRs), contingent upon the network operator's tolerance to false positives (how aggressive the censor is). Figure 3.6 presents the results, broken down by configuration groups, under varying FPR constraints. To provide context on the framework's practicality, we highlight the region where $FPR < 0.6\%$, aligning with the estimated FPR of the inferred algorithm used by the GFW to block fully encrypted proxies [216]. We highlight that our approach not only captures the majority of non-multiplexed proxy flows but also retains precision comparable to attacks that employ protocol-specific active probing [227], while being applicable to a broader range of circumvention proxies. Table 3.3 details evaluation results using one set of detection thresholds, grouped by configurations. Note that for C_3 (obfs4), we used a more permissive FPR constraint, as it applies only to traffic whose application-layer protocol is *Unknown*, a category that censors have historically approached more aggressively.

Major obfuscated proxies, in their standard configurations, are vulnerable to fingerprinting based on encapsulated TLS handshakes.. We are able to detect over 70% of flows generated by any of the four most widely used obfuscated proxy protocols, specifically *vmess*, *shadowsocks*, *vless*, and *trojan*. The implication of such detection rates is amplified when considering that a censor only needs to identify a single flow to block the corresponding proxy server. Although *vmess* features a built-in payload padding scheme, the padding size is drawn from a limited range of 0-63 bytes [26]. While such a scheme may be effective in obscuring values within request header fields, such as the destination URL, our findings has demonstrated that it falls short in obfuscating the size and directional traffic characteristics of (encapsulated) TLS handshakes from even simple statistical models.

Wrapping proxy traffic inside additional cover protocols does not provide an effective defense against traffic analysis.. Encapsulating proxy connections within another protocol is a popular strategy to provide extra security that comes with the cover protocol,

such as the probe-resistance of TLS. However, such connections can still be accurately distinguished from legitimate sessions running the cover protocols (HTTP, TLS), since TLS handshakes are not expected to occur *on top of* these cover protocols in their typical use cases. We note that our exploit does not assume implementation flaws and discrepancies in the cover protocols, which could become additional fingerprintable features orthogonal to our attack [81, 196, 80]. For example, Cloak [5] encapsulates proxy traffic in a cover TLS tunnel, which preserves the syntactic validity of TLS but not its semantics [97] (e.g.,, the absence of server certificate). A censor could complement the exploit of encapsulated TLS handshakes with discrepancies at the cover protocol layer to further improve detection accuracy.

Random padding is not the final word when it comes to obfuscating patterns within packet size sequences.. Implementing a random padding scheme is often the first step taken by developers of obfuscated proxies to hide patterns in packet sizes. Yet, we show that simple random padding schemes drawing padding sizes from limited ranges (e.g.,, *vmess*: [0, 63]), while increasing the challenge, doesn't render detection infeasible (TPR: $0.859 \rightarrow 0.687$ for vmess-ws-tls after padding). For more aggressive padding schemes e.g.,, *XTLS-vision* and *obfs4*, censors could adapt their statistical models by mirroring the padding schemes, training on padded versions of datasets, and leveraging packet directions over sizes for classification. Although doing so substantially increases FPR ($0.0544\% \rightarrow 0.6127\%$), empirical evidence suggests that a motivated censor may be able to tolerate this level of collateral damage during politically sensitive times [216].[3]

It's worth emphasizing that the need for dedicated classifiers for protocols like *XTLS-vision* and *obfs4* inherently demands more from censors in terms of resources, efforts, and potential costs from collateral damage. This additional complexity contrasts with configurations from the *Basic* and *Advanced* category, which can be fingerprinted in an effectively protocol-agnostic manner.

[3]The FPRs in Table 3.3 are broken down by application-layer protocols. That is, the FPR for C_3 (0.6127%) applies to *Unknown* flows only – FPR:0.0038% when all traffic is considered.

Multiplexing proves to be a more effective mechanism for masking the fingerprint of encapsulated TLS handshakes.. On a positive note, introducing connection multiplexing significantly reduces the detection rate, regardless of the underlying proxy protocol used. Even the most basic form of multiplexing – combining every two application streams in one proxy connection – lowers the TPR by over 70%. Intuitively, when a TLS handshake takes place concurrently with data transmission from another stream sharing the multiplexed connection, individual TLS handshake packets are likely to interleave with other data packets, thus disrupting the size and timing patterns typically expected from a non-multiplexed TLS handshake. We note that the actual impact of multiplexing may be more substantial than the differences in TPR suggest, as there will be significantly fewer proxy connections crossing the censor's firewall. One caveat to consider is that the effectiveness of multiplexing is limited when there is only one application stream. Without packets from co-existing streams to interleave, patterns in packet sizes remain exposed to fingerprinting in the same way as non-multiplexed flows.

False Positives:. It is possible that some of the flows labeled as obfuscated proxies are actually true positives. To investigate this possibility, we look for circumstantial evidence, independent of the encapsulated TLS fingerprint we exploit, to corroborate our detection results. Specifically, we extract 7,100 unique server endpoints from the 63,887 connections labeled by C_1 over the 30-day evaluation period. For each endpoint, we then examine the number of unique client IPs (and SNIs, if applicable). Intuitively, we expect endpoints running proxy protocols to have fewer unique client IPs and SNI values. As illustrated in Figure B.2, while labeled endpoints indeed connect fewer unique clients, they also tend to have more unique SNIs than average. A closer look at WHOIS and SNIs of the labeled endpoints reveals that streaming, gaming, and CDN services are among the top heavy hitters, suggesting that the majority of labeled endpoints are unlikely to be obfuscated proxies.

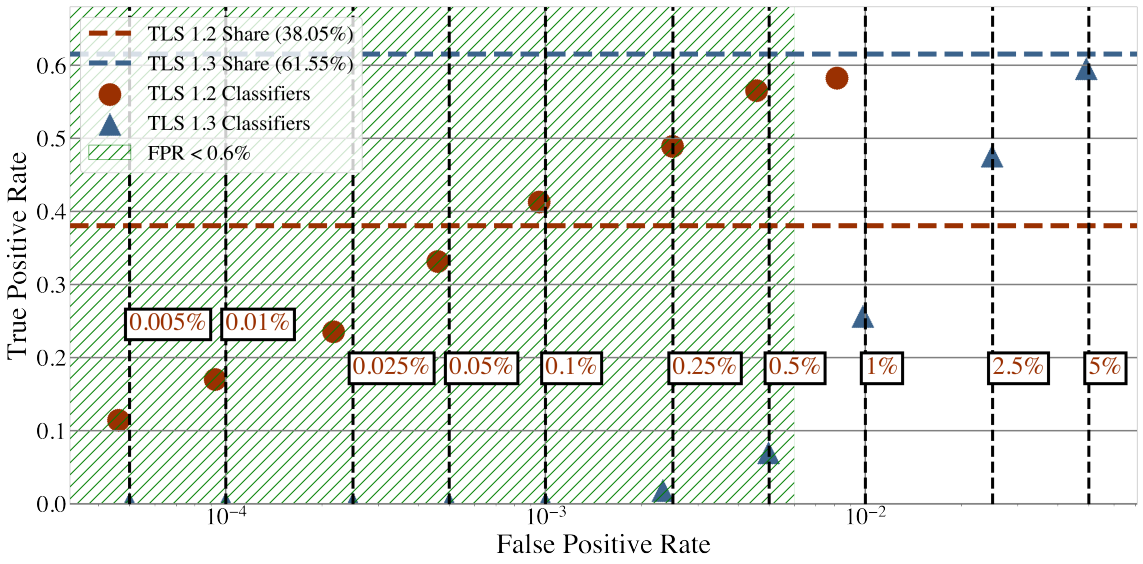


Figure 3.7: **Performance of classifiers on mixed TLS 1.2/1.3 traffic.** TPR is averaged from *vmess*, *shadowsocks*, *vless*, and *trojan*.◇

3.7.4 TLS 1.2 vs TLS 1.3

To investigate the effect of the increasing adoption of TLS 1.3 on the proposed fingerprinting attack, we repeat the experiment described in § 3.7.1.2 using Cloudflare's 1K domains [10]. This time, instead of forcing the browser to negotiate TLS 1.2, we allow the browser and server to decide on their preferred protocol version. We tested on two classifiers, each trained specifically for a particular version of TLS. For each version, we conducted the experiment five times, using the *Basic* configurations, and reported the average results in Figure 3.7.

We find that although most top domains themselves support TLS 1.3, resources loaded on the page (e.g.,, images, ad trackers) may lag in TLS 1.3 adoption. As a result, on average, when visiting the top 1K domains, TLS 1.2, 1.3, and HTTP represent 38.05%, 61.55%, and 0.39% of all TCP flows, respectively. These figures align with what we observed in *Merit* traffic (Table 3.1) and a recent TLS Telemetry Report [205]. Regarding classifier performance, we observe that although the TLS 1.3 classifier offers a relatively higher detection rate, it comes at the expense of a significantly elevated FPR, which can be attributed to its shorter (and thus less specific) handshakes. In contrast, the TLS 1.2 classifier effectively

detects the majority of proxied TLS 1.2 flows, despite being limited by the version share, while maintaining a lower FPR which makes it more practical for deployment. As discussed in § 3.3, a moderate TPR is unlikely to be a major concern for a censor, as a non-multiplexed proxy would generate hundreds of flows during a typical browsing session, and the censor only needs to detect one of them. Considering that TLS 1.2 still represents a significant share of Internet traffic, we believe that, at this moment, the optimal strategy for a perspective censor implementing the proposed exploit would be to exclusively target encapsulated handshakes of TLS 1.2.

3.8 Discussion and Mitigation

Obfuscated proxies play a crucial role in facilitating unrestricted access to information and resources on the Internet, particularly in regions where censorship is prevalent. However, we demonstrate that detecting obfuscated proxy traffic, even with random padding and layers of encapsulation added, is highly effective by exploiting encapsulated TLS handshakes. The ability of censors to identify proxy connections could pose significant risks to users, especially in countries where circumventing censorship is deemed illegal.

Would censors deploy the fingerprinting attack based on encapsulated TLS handshakes?. Our evaluation on *Merit* shows that, with conservative detection thresholds, the proposed fingerprinting attack yields an exceptionally low FPR, while only requiring passive monitoring and remaining largely agnostic to the choice of circumvention protocol. [4] Furthermore, this exploit targets layers encapsulated *within* the proxy protocol, thereby making it orthogonal and potentially complementary to attacks targeting the proxy protocols themselves [197, 87, 81] as well as host-based network analysis [195]. For this reason, we believe that proxies featuring fully encrypted cover layers, e.g.,, raw *shadowsocks* and *vmess*, are especially vulnerable: given that censors have already demonstrated a willingness to

[4] The approach to fingerprint proxy flows with encapsulated TLS handshakes is generally protocol-agnostic. However, classifiers may need adjustments for protocols using random padding (§ 3.7.2).

block them based on entropy [216], the additional use of encapsulated TLS handshakes as a complementary feature could further reduce the collateral damage.

However, we note that even with a false positive rate of 1 in 2000 (§ 3.7.3), due to the sheer volume of traffic passing through a national firewall and the low base rate of circumvention traffic in the wild, our fingerprinting attack would likely still label more legitimate connections as proxied than actual proxied connections [197]. Furthermore, a nation-state censor is not just a traffic classifier, and deploying a new censorship technique involves considerations beyond false positive rates, such as public sentiment and operational costs. As such, we caution against prematurely declaring any of the tested proxies as "broken" based on our findings alone. Rather, proxy developers should anticipate the potential exploitation of encapsulated TLS handshakes as a shared fingerprint of proxy traffic, and proactively equip their tools with mitigation mechanisms.

Mitigation. Encapsulated TLS handshakes cannot be easily eliminated from proxy traffic without compromising end-to-end security. Hence, the responsibility falls on proxies to implement effective obfuscation mechanisms to hide them from traffic analysis. Our evaluation suggests that multiplexing could be a more viable strategy compared to random padding alone. Intuitively, multiplexing interleaves packets from multiple streams, disrupting traffic patterns not only in packet size but also timing and direction. Different co-existing streams introduce additional unpredictability. However, the effectiveness of multiplexing as an obfuscation mechanism depends on the presence of co-existing flows. In situations where there is only a single flow, or where the co-existing flows are inactive, patterns in packet sequences will remain as exposed as they are with non-multiplexed proxies. One direction for future work could be exploring methods for generating synthetic co-existing flows when they are not naturally present.

However, obfuscation mechanisms based on padding and multiplexing alone are subject to fundamental limitations. Specifically, *padding can only increase the size of a packet or a burst, not decrease it. Stream multiplexing can only increase the number of round trips*

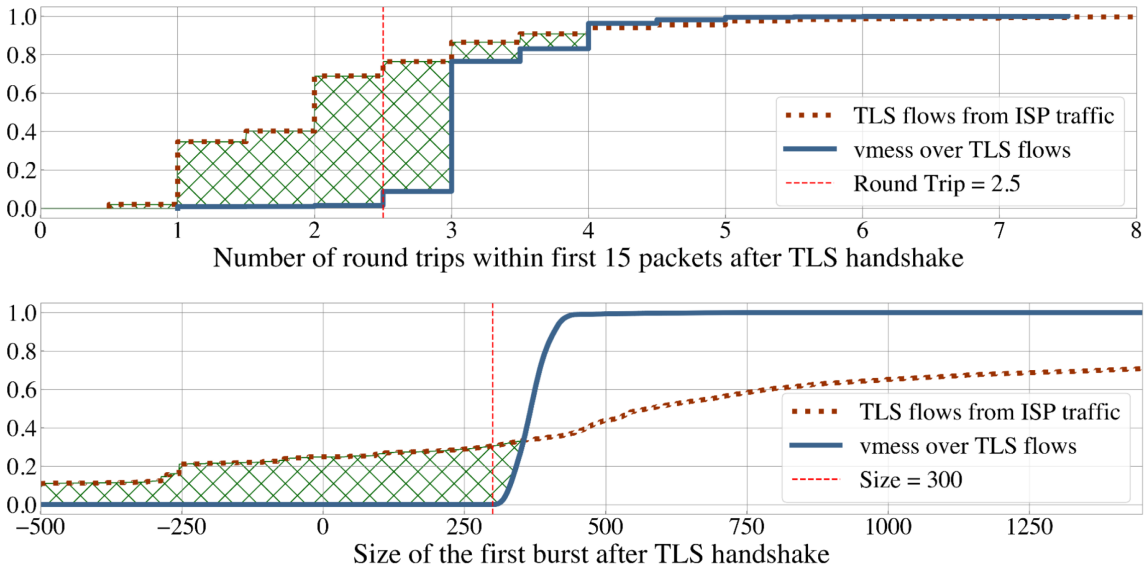


Figure 3.8: **Round trip count and size of first burst after TLS handshakes.** *TLS-over-TLS* requires more round trips and a larger initial burst due to a second (encapsulated) handshake. Shaded areas highlight dissimilarities that padding and multiplexing cannot obfuscate.

in a connection, not decrease them. These limitations become significant when the pattern requiring obfuscation is inherently characterized by larger sizes and more round trips than what might be considered "normal" within the context.

Consider normal HTTPS versus TLS-based proxies. In a typical HTTPS connection, the packets that immediately follow the TLS handshake usually consist of just one round trip: an outgoing GET and an incoming response segmented into multiple packets. In contrast, when accessing HTTPS over a TLS-based proxy (i.e., *TLS-over-TLS*), what immediately follows the initial TLS handshake is a second (encapsulated) TLS handshake, thereby requiring more round trips and resulting in larger packet sizes due to multiple framing overheads. Figure 3.8 shows the number of round trips and the size of the first burst following the initial handshake for normal TLS and *TLS-over-TLS*. Importantly, the shaded areas represent *dissimilarities that padding and multiplexing cannot obfuscate*. In theory, a censor could combine two simple filtering rules ($RT < 2.5, Size < 300$) to effectively filter out 82.5% of legitimate and 1.5% of proxied connections. More sophisticated analysis could then be applied to the remaining connections. This example illustrates that, if obfuscated only with

padding and multiplexing, traffic of TLS-based proxies can always be distinguished from the majority of genuine HTTPS traffic.

A dedicated obfuscation layer. An effective obfuscation mechanism should transform patterns in packet size, timing, and direction such that traffic observable on the wire does not reflect any dynamics of the traffic encapsulated within, thereby hiding fingerprintable features such as encapsulated TLS handshakes. Existing padding and multiplexing schemes, unfortunately, do not meet this criterion. We believe that to achieve this, proxies need to decouple obfuscation from encapsulated application streams, in order to allow maximum flexibility to simulate any given traffic shape [76, 71]. For example, one approach is to implement a traffic scheduler at the obfuscation layer that dictates when, how, and how much traffic should be sent at any given time. Such scheduler goes beyond existing padding schemes as it would require proxies to send dummy packets when there is no application data to send, or buffer application data when the scheduler demands quiet time. Abstracting an obfuscation layer would also allow proxies not only to multiplex but also de-multiplex application streams. For example, the obfuscation layer could use two separate network connections to transmit upstream and downstream application traffic. For each connection, half of the traffic is dummy data that could be arbitrarily interleaved with the uni-directional application streams.

A dedicated obfuscation layer would ensure a more principled and sustainable defense against traffic analysis. However, defining a "legitimate" traffic shape for proxies' obfuscation to simulate, and balancing obfuscation and performance, remain open questions for future research.

Generalizing to UDP-based Proxies and VPNs. While our work primarily focuses on TCP-based proxies, the principles behind our approach are transferable to UDP-based proxies such as MASQUE or vmess/shadowsocks over QUIC. This is because the patterns for fingerprinting are generated by encapsulated layers inside a proxy's payload, rather than the

characteristics of the proxy layer itself. Adapting the classifiers to these contexts would need to consider factors such as the unreliability of UDP and the potential frame padding of QUIC.

Our fingerprinting attack is predicated on the presence of nested protocol stacks – the rarity of finding a TLS layer inside another TLS or application layer in direct connections makes this behavior an ideal fingerprint for proxy flows. While this paper explores one specific attack, it points to broader risks across all proxying and tunneling tools that rely on nested protocol stacks. For example, a layer-3 analogy would be detecting obfuscated VPN flows by considering encapsulated *TCP* handshakes (and the rarity of finding TCP handshakes inside another transport layer protocol).

3.9 Conclusion of Chapter III

We explore the practicality of exploiting encapsulated TLS handshakes as a fingerprint to detect obfuscated proxy traffic. Our evaluation within an ISP demonstrates that, even with random padding and layers of encapsulation, traffic of obfuscated proxies can still be reliably detected with minimal collateral damage. Considering that useras in censored regions depend on obfuscated proxies for access to information, it is critical for proxy developers to anticipate this emerging threat and to design and deploy proactive countermeasures.

CHAPTER IV

Timing-Based Fingerprinting against Traffic Obfuscation[1]

In this chapter, I bring out the presence of an inherent fingerprint for detecting obfuscated proxy traffic. The fingerprint is created by the misalignment of transport- and application-layer sessions in proxy routing, which is reflected in the discrepancy in Round Trip Times (RTTs) across network layers. Importantly, being protocol-agnostic, the fingerprint enables an adversary to effectively target multiple proxy protocols simultaneously. I conduct an extensive evaluation using both controlled testbeds and real-world traffic, collected from a partner ISP, to assess the fingerprint's potential for exploitation by censors. In addition to being of interest on its own, the timing-based fingerprinting vulnerability surfaced by this chapter highlights the deficiencies in existing obfuscation approaches. I hope this chapter brings the attention of the circumvention community to packet timing as an area of concern and leads to the development of more sustainable countermeasures.

4.1 Introduction

On May 30, 2024, Myanmar imposed a blocking on user traffic to VPN and proxy servers, affecting citizens who had relied on these services to circumvent censorship put

[1]This chapter is based on: Diwen Xue, Michalis Kallitsis, Amir Houmansadr, Roya Ensafi. 2025. **The Discriminative Power of Cross-layer RTTs in Fingerprinting Proxy Traffic** In Network and Distributed System Security Symposium 2025 (NDSS'25) [229] This work was made possible by the National Science Foundation under Grant Numbers CNS-2237552 and CNS-2141512.

in place after the military coup [90]. On April 25, 2024, Russia launched a new method to fingerprint and block proxy-enabled circumvention traffic, reinforcing the government's hold on information flow and further isolating its citizens from the global Internet [135]. In November 2021, China deployed its latest fingerprinting attack against traffic of fully encrypted proxies, showcasing both its advancement in censorship capabilities and its persistent commitment to suppress circumvention efforts [216].

These are not isolated events. They are part of a global trend where escalating Internet censorship motivates a growing reliance on circumvention tools, leading to an ongoing arms race [182]: on one side, developers and users of circumvention tools aim to bypass restrictions on information; on the other, censors deploy increasingly sophisticated fingerprinting attacks to detect and block circumvention traffic. Censors used deep packet inspection (DPI) to detect circumvention traffic, driving developers to obfuscate it to resemble popular protocols like TLS [134, 227]. As censors blocked circumvention traffic based on their distinctive ciphersuites, developers adopted ciphersuites from mainstream browsers [2, 81]. When censors deployed active probing to identify circumvention servers, developers in response implemented probe-resistant protections [66, 80]. The incidents in Myanmar, Russia, and China mark the latest developments in this circumvention arms race.

Obfuscated proxies remain one of the most popular methods for circumventing censorship, providing flexible, general-purpose access to information with good usability for average users. Similar to standard proxies, an obfuscated proxy setup involves a user in a censored region connecting to a proxy outside that jurisdiction in order to access an application server that is otherwise blocked. The proxy, upon receiving payloads from the user, establishes a new connection with the application server and forwards the traffic as received. Obfuscated proxies differ from standard proxies by including an extra obfuscation layer to avoid being detected by censors. Over the past decade, the circumvention community has developed various obfuscation schemes, such as by mimicking allowed

protocols [64, 197, 127] or by randomizing proxy's payloads so they cannot be statically identified [19, 26, 140].

The main focus of this chapter is to bring out the presence of a fingerprint for detecting traffic from obfuscated proxies and to empirically evaluate its potential for practical fingerprinting attacks by nation-state censors. At a high level, the fingerprint is created by the inherent misalignment of *sessions* across OSI layers that results from the use of a proxy. Transport-layer sessions break into two parts at the proxy – client to proxy, and proxy to server – while the application-layer maintains an end-to-end session that extends directly from client to server. This architecture enables the proxy to relay communication between client and server while also preserving end-to-end features of higher layers like TLS. However, because transport- and application-layer sessions terminate at different endpoints, their round-trip times (RTTs) may show significant discrepancies, as the paths they traverse span different distances. Throughout this chapter, we focus on this discrepancy in cross-layer RTTs, RTT_{diff}, and demonstrate the distinguishing power it holds in fingerprinting proxy traffic.

We emphasize that this fingerprint represents more than just another potential way to detect proxy traffic. Up until now, the vast majority of fingerprinting attacks against obfuscated proxies, whether deployed by censors or envisioned by researchers, focused on *specific* protocols – censors exploit the obfuscation flaws of individual protocols, while developers patch these issues and introduce new protocols in an attempt to outpace a resource-limited censor. Both sides find themselves locked in this ongoing arms race, where each side must constantly balance between efficacy and expenditure [182]. The fingerprinting vulnerability discussed in this chapter, however, belongs to a class of fingerprints that are inherent to *all* proxy protocols. Not only does this protocol-agnostic approach complement existing protocol-specific attacks; it also enables censors to efficiently target multiple protocols at once with little manual effort. Such a development could shift the balance in the ongoing circumvention arms race.

Assuming the role of a censor, we empirically evaluate the practicality of exploiting the RTT_{diff} fingerprint with only passive monitoring capabilities. We detail our construction of the exploit: first, from features visible to an on-path censor, we estimate the RTTs at the transport and application layers through cross-correlation and derive their discrepancy. Next, we frame a sequential hypothesis testing (SHT) problem to determine whether the observed discrepancy in delays suggest a direct or proxied connection. Using eight geographic locations, we conducted extensive evaluation of the exploit using a mixture of controlled proxy traffic and real user traffic collected from an Internet provider, to assess the sensitivity and specificity of RTT_{diff} as a standalone fingerprint.

We found that the efficacy of the exploit is largely independent of the client's location or the censor's position relative to the client; nor is it affected by the specific proxy protocol tested, whether obfuscated or otherwise. For a client visiting the top 5K domains via an obfuscated proxy, we found that about 80% of the domains visited would generate at least one traffic flow characterized by the RTT discrepancy detectable by our fingerprint, with half of these detections made within the first 60 packets of a flow, all while maintaining a False Positive Rate (FPR) on par with those of deployed attacks. While we demonstrate the general feasibility of this fingerprint, we also discuss practical factors – such as the handling of DNS resolution, the presence of CDNs, and the potential for categorical collateral damage – that would affect the fingerprint's efficacy and compel a potential adversary to weigh the costs and implications of deployment.

Our evaluation on this timing-based fingerprint highlights the deficiencies in existing obfuscation schemes. Over the years, researchers have long recognized that distributions of packet sizes hold significant classification power for detecting circumvention traffic [197, 111], and censors have indeed exploited packet sizes in practice [135, 216]. As such, many obfuscated proxy protocols have incorporated (random) padding schemes directly into their designs [26, 219]. While padding provides a straightforward defense to obfuscate patterns over packet sizes, it leave packet timings unprotected, a source of information often

overlooked by designers of obfuscation schemes. What this chapter showcases, however, is one example of what a censor can accomplish in fingerprinting proxy traffic using *only* features derived from packet timings.

While we demonstrate several potential short-term mitigations, such as configuring TCP delayed ACK on the proxy server or enabling connection multiplexing, we caution that these strategies result in atypical behaviors that might themselves be fingerprintable. In particular, we showcase how existing delay-based timing obfuscation schemes, such as obfs4/scramblesuit, are fundamentally limited in their ability to mitigate the proposed fingerprint and, paradoxically, may even worsen the issue. Additionally, while our focus is on transport-layer obfuscated proxies, we also explore how the underlying principles of our timing-based fingerprint extend to network-layer VPNs. We hope this study highlights packet timing as an area that warrants further attention from the circumvention community and leads to the development of more principled, sustainable countermeasures.

4.2 Background

4.2.1 Internet Censorship

News reports, personal anecdotes, and measurement studies collectively suggest an increasing trend in nation-state Internet censorship on a global scale, as governments exert control over the flow of information deemed undesirable within and across their borders [164, 122, 108, 158]. For over two decades, researchers and Internet freedom activists have studied government censorship policies and their enforcement through technical means. Building upon seminal works from the early 2000s [55, 235, 145], censorship research has advanced our understanding of how nation-state firewalls interfere with users' traffic [155, 95, 94], their technical capabilities [203, 120, 44], and their architecture and geographical distribution [225, 69, 222]. This body of research not only exposes the otherwise covert practice of censorship but also assists in developing a realistic threat model. In general,

censorship involves a two-step process: *Detect* and *Disrupt*. The first phase classifies traffic as either allowed or prohibited using a range of techniques from simple keyword matching to more sophisticated traffic analysis. Traffic considered undesirable is then actively interfered with by dropping packets or RST-ing connections [182, 195]. Note that while *Disrupt* often requires active engagement, *Detect* can operate passively, remaining completely invisible to the communicating parties. Our investigation into proxy fingerprinting falls under the detection step.

4.2.2 Circumvention Arms Race

As censorship measures intensify, users in affected regions increasingly seek methods to circumvent these restrictions. The most common and flexible circumvention mechanisms follow a channel-based approach—users establish *channels* to *forwarders* located outside the censor's jurisdiction, which in turn forward their traffic to its final destinations [182]. This setup commonly involves proxying protocols (both network-layer and transport-layer). However, standard proxying protocols such as OpenVPN and SOCKS feature plaintext headers that can be easily detected and blocked, which necessitates the use of obfuscation techniques, or more specifically, the indistinguishability provided by such obfuscation, across all channel-based systems.

This has led to an ongoing arms race between developers of obfuscated proxies, who obfuscate the traffic of their protocols to evade detection, and censors, who aim to see through such obfuscation. The dynamics of this conflict are best illustrated by the interactions between the Great Firewall of China (GFW) and various circumvention tools: The GFW was able to detect and block Tor traffic using manually-crafted signatures that exploited the unique ciphersuites of Tor's custom TLS implementation [2]. In response, developers introduced dedicated obfuscators mimicking TLS implementations from mainstream browsers [81, 160]. The GFW blocked servers hosting Tor, VPN, and other circumvention services by sending specially crafted probes to suspected endpoints and checking if the

responses matched the expected protocol behaviors [66, 31, 210]. Circumvention developers countered by deploying "probe-resistant" proxies that remain silent until the client authenticates itself [80]. The GFW fingerprinted traffic from fully encrypted proxies based on their atypically high entropy [216], and developers in response changed the byte patterns of their traffic to show lower entropy [11, 88]. This back-and-forth characterizes the current state of the circumvention arms race, where each side continually adjusts their approaches in reaction to the advancements in fingerprinting and obfuscation.

The specific fingerprint evaluated in this paper represents a potential furthering of this arms-race: unlike aforementioned attacks that targeted specific circumvention protocols by exploiting vulnerabilities in their obfuscation designs, this timing-based fingerprint stems from an inherent characteristic shared by all proxying and tunneling protocols – the misalignment between layers. This protocol-agnostic approach could overcome the trade-off between efficacy and cost in the circumvention arms-race by enabling censors to target multiple protocols at once.

4.2.3 Timing-based Fingerprinting Attacks

In previous traffic fingerprinting attacks, particularly those on website fingerprinting, the focus has traditionally been on features derived from packet sizes rather than those based on packet timings. Several early work discussed that timing features, specifically packet inter-arrival times, are unreliable due to their dependency on network conditions like jitters [110, 230, 40]. Jaber et al., however, demonstrated that the distribution of packet timings holds discriminative power for classifying network traffic [104]. Feghhi et al. proposed a website fingerprinting attack that leverages only timing information, bypassing defenses based on packet padding [70]. More recently, Rahman et al. studied the utility of packet timing and found that timing indeed provides meaningful classification power [153].

There is limited prior work on detecting proxy traffic with timing features. Webb et al. proposed an approach for detecting proxy usage at the server side by measuring the RTT for

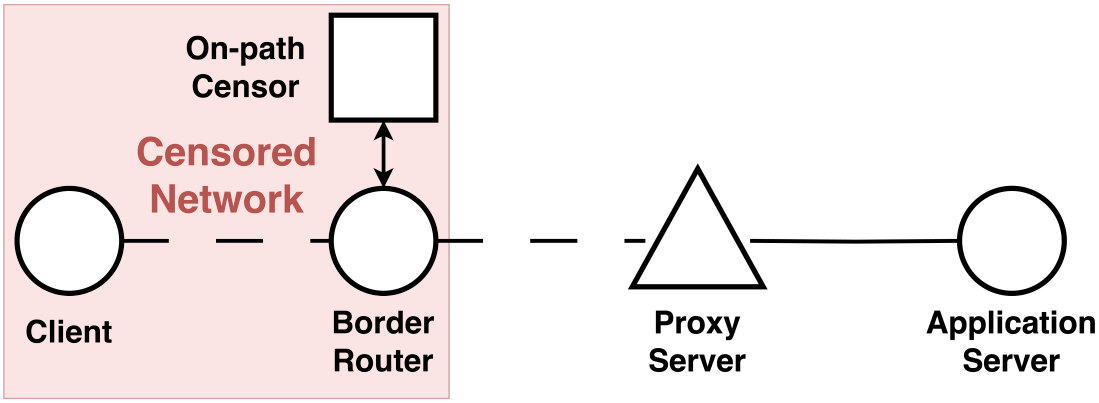


Figure 4.1: **Threat model:** A censor positioned *on-path* relative to the client and proxy, receiving a copy of traffic. Figure inspired by [70].

each known IP and identifying anomalies in their distributions, requiring separate training for each IP address [207]. Ramesh et al. developed a proxy detection method using RTTs, but their approach requires access to an endpoint of a connection (e.g.,, at the server) and active measurements [159]. In contrast, the fingerprinting vulnerability evaluated in this paper requires only passive monitoring capability with no server-side deployment and, in principle, can be exploited at any network location between the client and proxy server.

4.3 Threat Model and Scope

Previous measurement studies have shown that nation-state censors possess the capability for both passive monitoring and active interference [66, 182], with some demonstrating even more intrusive behaviors such as injecting malicious traffic or intercepting end-to-end encryption [120, 172]. For the purpose of this study, we adopt a conservatively *weak* but practical adversary model as defined in previous work [182]. We assume that the censor is limited to passive monitoring capabilities, examining but not injecting, removing, or otherwise altering the traffic. The censor is positioned *on-path* relative to the client and the proxy (e.g.,, deployed at chokepoints or border ASes), as shown in Figure 4.1. We also assume the censor to be stateful, allowing it to accumulate information about each connection it observes. However, computational constraints limit the censor's real-time, per-flow analysis

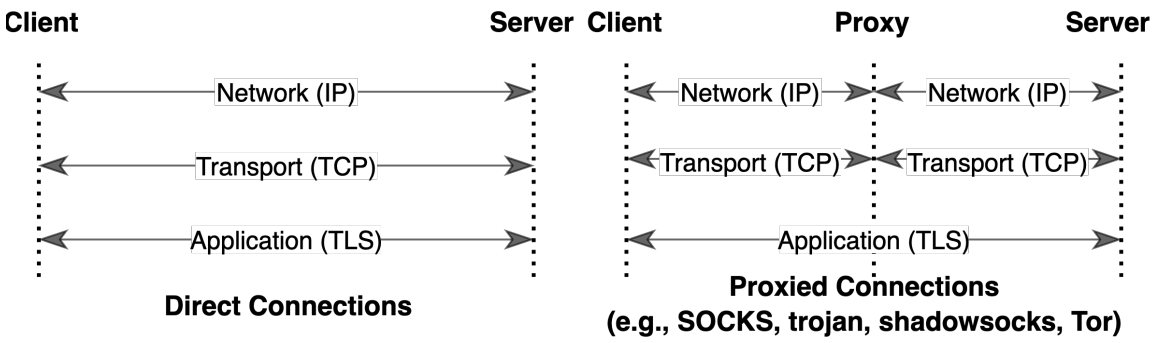


Figure 4.2: **Protocol layerings in Direct vs. Proxied Connections**. In proxied connections, transport sessions terminate at the proxy, whereas the application layer connection remains end-to-end. ◇

to basic statistical methods; more complex methods like deep learning have not yet been observed in real-world deployments. Finally, for our discussion, the censor makes use only of features derived from packet timestamps and sizes, ignoring other potentially identifiable information such as IP addresses, ports, or payloads.

While powerful, real-world censors operate within certain constraints, chief among them being the *cost* – whether as economic losses or civil discontent – caused by accidentally blocking traffic that should have been allowed (i.e., false positives). The cost of false positives, known as "*collateral damage*", is the only reason preventing censors from simply shutting down the entire Internet and trivially blocking all circumvention traffic with perfect recall [74]. Evaluating the practicality of a potential fingerprint, therefore, involves assessing both the reliability of the fingerprint as an indicator of circumvention traffic, as well as the rarity of finding the same fingerprint in legitimate traffic.

4.3.1 Scope

The focus of this work is transport-layer obfuscated proxies, such as shadowsocks, Trojan, VMess, *etc* These proxy protocols function similarly to traditional SOCKS proxies by forwarding traffic between clients and servers without modification (preserving application-layer end-to-end features), but incorporate additional encryption and obfuscation to evade detection.

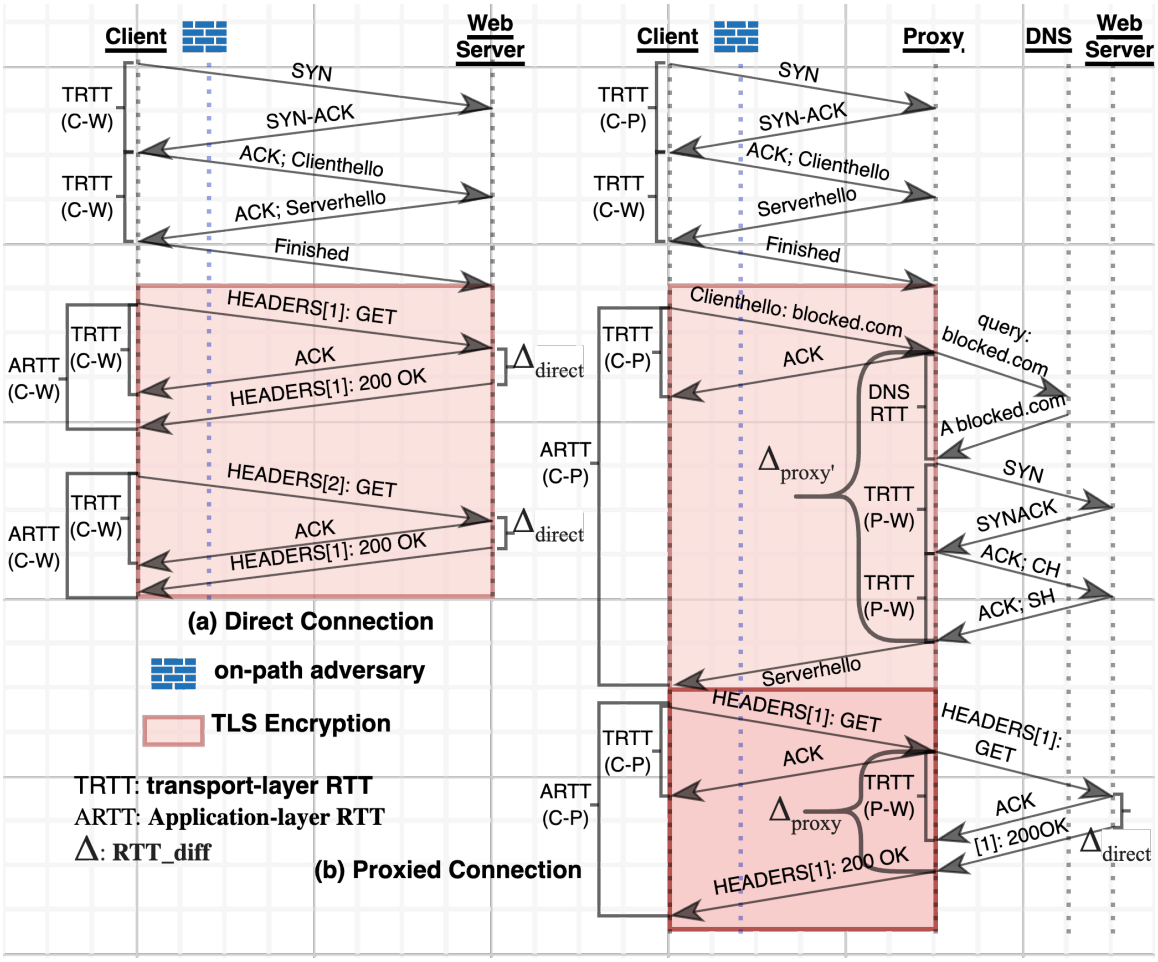


Figure 4.3: **Sequence timing diagram for (a) direct HTTPS session vs. (b) TLS-based proxy (vless-over-tls) session.** For the proxied connection, we observe a much higher discrepancy in transport vs. application-layer delay as the sessions at two layers terminate at different endpoints (proxy and the web server, respectively). ◇

We do not consider TLS-terminating proxies in this work (i.e., MITM proxy). Although there has been some discussion about their use for censorship circumvention, they have not been adopted by mainstream circumvention tools due to the security risks they pose by breaking the PKI model [136, 48, 73]. Additionally, while our focus is on transport-layer proxies, we recognize that the underlying principles of our timing-based exploit may have implications for network-layer VPNs as well, and we explore this potential extension in § 4.7.2.

4.4 Cross-layer RTT Difference

We begin by describing the different connection segments and their corresponding RTTs for a proxy connection and the intuition of how these differences can be exploited as a fingerprint. At a high level, the idea is that using a proxy inevitably affects the layering of the OSI model, leading to a situation where the sessions at the transport layer and the application layer terminate at different endpoints. Especially when the application layer session extends significantly beyond the transport layer – i.e., when there is a substantial geographical distance – we demonstrate that the difference in RTT becomes noticeable and can then be exploited by an on-path adversary to infer the presence of the proxy. Table C.1 in Appendix compiles a list of all symbols we use throughout this paper.

4.4.1 Proxy Architecture and Round-Trip-Time

A typical proxied connection involves three parties: a client who wants to access a service that is blocked (due to censorship, geoblocking, *etc*); an application server offering the desired service; and a proxy that relays (and optionally obfuscates/encrypts) the client's traffic to the application server.

Figure 4.2 describes the layering of protocol stacks for direct and proxied connections. In direct connections, sessions at the network (e.g.,, IP), transport (e.g.,, TCP), and application (e.g.,, HTTPS) layers often have a one-to-one correspondence: a higher-layer session is fully encapsulated within the one below, and their notions of endpoints (client and server) are aligned across layers, spanning the same distance. Proxied connections, however, introduce segmentation at one or more layers of the stack. For single-hop proxies, sessions at the transport layers could be split into two segments: client-to-proxy and proxy-to-server, while the application layer remains end-to-end (client-to-server). Such architecture enables the proxy to forward communication between the client and server, while preserving end-to-end features of higher layers, like TLS.

The way that proxies change the layering of protocol stacks also affects the Round-

Trip-Times (RTTs) observed across different layers. In direct connections, the RTTs at the application and transport layers are aligned, since they share the same endpoints and path distances. For proxied connections, however, sessions at lower layers terminate at the proxy. An observer located between the client and the proxy would perceive shorter transport-layer RTTs that only account for the distance to the proxy. *The proxy's acknowledgment of data receipt appears to complete the round trip from the observer's standpoint.* In contrast, the end-to-end application-layer RTT additionally encompasses the time the proxy takes to forward the request/response to/from the server. This additional segment results in noticeable differences in cross-layer RTTs, which can be consistently observed when there is sufficient distance between the proxy and the application-layer server. This discrepancy forms the basis of the timing-based fingerprint that we discuss in this paper.

4.4.2 Example: HTTPS vs. VLESS-TLS

Figure 4.3 compares the difference in cross-layer RTTs between a direct HTTPS session and a proxied HTTPS session using a transport-layer TLS-based proxy, e.g.,, vless, trojan [24, 26], *etc* In both scenarios, a client attempts to visit the same web server while an on-path adversary monitors traffic close to the client. Note that after the initial handshake (with either the web server or proxy), the adversary only sees TCP control packets (e.g.,, ACKs) and the encrypted streams.

Following the initial handshake, in the direct scenario, the client sends a request to the server. The server acknowledges the reception of the request, which returns to the client exactly one transport-layer RTT after the request was sent. Depending on the type of request and load, the server might take some time to process the request and prepare a response, resulting in a typically small difference (Δ_{direct}) between the transport-layer RTT (TRTT) and application-layer RTT (ARTT). On the other hand, in the proxied case, the first packet after the proxy handshake (in this case, TLS) typically contains both the SOCKS address (indicating where to forward the traffic, e.g.,, blocked.com) and the first

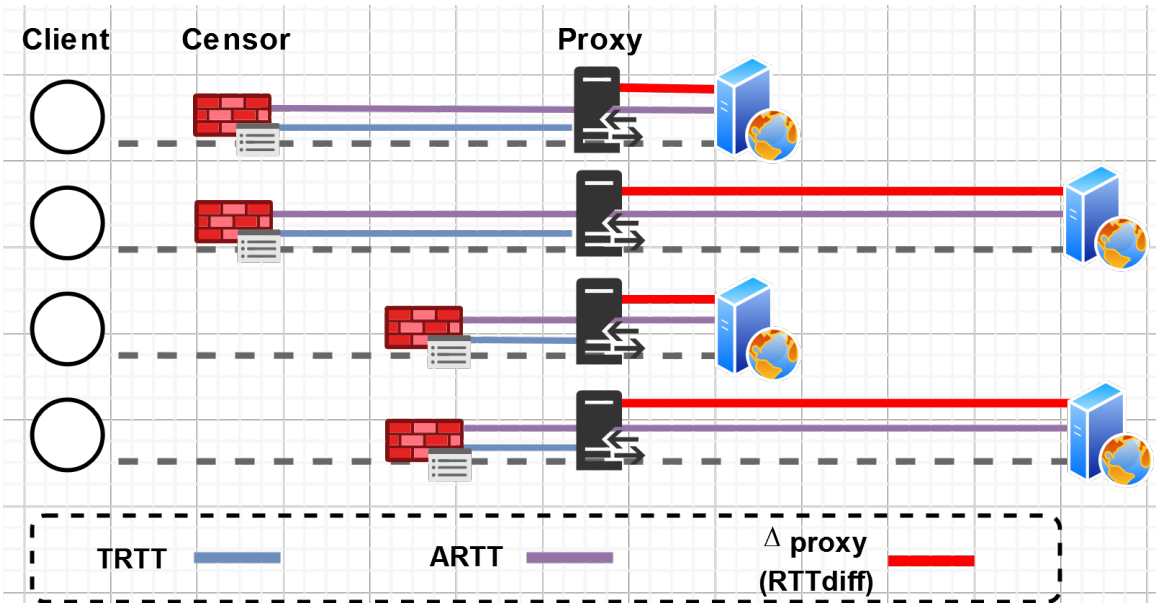


Figure 4.4: RTT_{diff} **for varying server locations and varying censor position between the client and proxy.** RTT_{diff} (the red line) depends only on the relative distance between proxy and application server and is independent of the client's or censor's location. ◇

data packet (e.g.,, a clienthello for blocked.com) [2]. Upon receiving this packet, the proxy initiates a 3-way TCP handshake with the application server, taking one TRTT between the proxy and the application server ($TRTT_{PW}$). The proxy then forwards the clienthello, receives the serverhello, and returns it to the client after another $TRTT_{PW}$. Subsequent requests/responses follow the same pattern. If the client opts to resolve hostnames via the proxy, an extra initial round trip to the proxy's DNS server is required.

Comparing the cross-layer RTTs in the two scenarios reveals that for direct connections, the difference (*ARTT-TRTT*) is just the server processing delay Δ_{direct}. However, for proxied connections, the first cross-layer difference $\Delta_{proxy'}$ includes a *DNSRTT* and two $TRTT_{PW}$ (additional path delay), while subsequent Δ_{proxy} contains one $TRTT_{PW}$ and additionally the server processing delay Δ_{direct} on top of that. The task of fingerprinting proxy traffic is thus reduced to determining whether an observed cross-layer RTT difference *ARTT-TRTT* is an instance of Δ_{direct} or Δ_{proxy}.

[2] Unlike standard SOCKS protocol that sends address and the first payload packet separately, most transport-layer circumvention tools (Outline, shadowsocks, v2ray, *etc*), merge them for performance and obfuscation purposes [105].

4.4.3 Efficacy and Assumptions

We note that this timing-based fingerprint does *not* depend on the client's location or the distance between the client and the proxy. As long as the adversary is positioned between the client and the proxy, RTT measurements can simply be redefined from client-server to adversary-server. Thus, the client's relative location to the adversary or its distance to the proxy becomes irrelevant, as illustrated in Figure 4.4.

What matters, however, is the proxy's position relative to the application server: the proxy must not be geographically co-located or near the server to ensure that the transmission delay introduced by the distance between them is not negligible, an assumption that may not always hold due to the prevalence of CDNs. Moreover, exploiting the fingerprint relies on the adversary's ability to distinguish Δ_{proxy} from Δ_{direct} by consistently observing the effect of $TRTT_{PW}$ on the cross-layer RTT difference. In other words, *the delay introduced by transmission ($TRTT_{PW}$) must be more pronounced than the variance introduced by the processing time at the server* (Δ_{direct}). We also assume that the proxy does not intentionally delay the transmission of TCP acknowledgments. Finally, and perhaps most importantly, the exploit depends on the adversary's ability to observe the cross-layer RTT differences. While transport-layer RTT is straightforward to measure, determining application-layer RTT is more challenging due to encryption.

In the following sections, we will examine each of these assumptions to evaluate the fingerprint's potential for exploitation and assess the susceptibility of proxy traffic to the exploit.

4.5 Constructing a Cross-Layer RTT Exploit

This section outlines a practical instantiation of a possible fingerprinting attack based on the previously laid conceptual groundwork. At its core, the exploit leverages RTT discrepancies to identify mismatches in session alignments across network layers, which

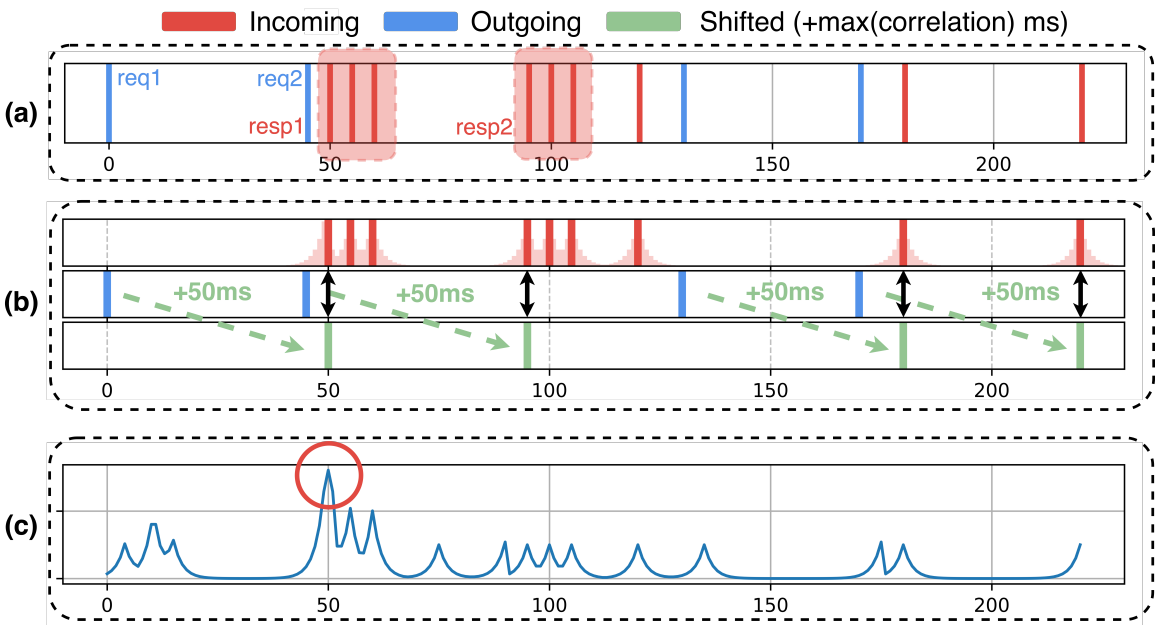


Figure 4.5: **Cross-correlation for application-layer RTT Estimation.** (a) Bi-directional packet flow, packets ordered by SEQ/ACK; (b) Cross-correlating outgoing against a weighted distribution centered around incoming packets; (c) Correlation scores at different shifts. Shift corresponding the highest correlation is the estimated $ARTT$.

are indicative of proxy use. However, adversaries aiming to apply the cross-layer RTT fingerprint in the real world need to consider factors such as the limited visibility into encrypted streams and the variances introduced by network jitter or server load. Our approach to implementing this exploit involves two primary phases: 1) From the traffic visible to an on-path adversary, estimating the RTTs at both the transport and application layers and calculating the cross-layer differences; 2) Determining whether the observed RTT differences align more closely with those expected from a direct connection or a proxy setup.

4.5.1 Estimating Cross-layer RTT Differences

The initial step of the exploit involves measuring the cross-layer RTT differences RTT_{diff} in a potential proxy connection. This measurement is achieved by subtracting transport-layer RTT from application-layer RTT. To obtain the former, we use SEQ/ACK analysis to identify each ACK-eliciting packet sent by the client and its corresponding ACK. If

multiple such packet pairs exist within an observation window, we use the median as the transport-layer RTT for that interval.

However, estimating application-layer RTTs poses significant challenges, especially for our threat model where an on-path network adversary performs passive monitoring and does not have visibility into the encrypted streams. The problem lies in the fact that the logical sequence of operations at the application layer does not directly map to the observable sequence of packet exchanges at the transport layer.

Consider the example of a client interacting with a web service through an encrypted proxy. HTTP pipelining allows multiple transactions to reuse the same connection, and the client may send several requests back-to-back without waiting for each response. Figure 4.5 (a) shows a TCP flow of incoming and outgoing data-carrying packets observed by an on-path censor located between the client and the proxy. While the transport-layer sequence and acknowledgement might suggest an ordering of $req1, req2, resp1, resp2$, the actual application-layer logical dependencies are $req1, resp1$, $req2, resp2$. Without visibility into the encrypted streams, the adversary cannot easily determine the true application-layer dependencies. Attempting to estimate $ARTT$ as $Tresp1 - Treq2$ would be incorrect, as $resp1$ and $req2$ are not a pair at the application layer.

Previous work on traffic analysis based on RTTs often relies on having access to one end of the application-layer connection. For example, Ramesh et al. estimates the application-layer RTT by actively sending WebSocket PINGs from the server side [159]. Such approaches benefit from directly observing and interacting with application-layer transactions. However, such techniques are not applicable in our threat model, where we assume the adversary is on-path, not on either endpoint, and limited to passive monitoring only.

Approach: Cross-correlation for RTT Estimation. We propose a cross-correlation-based approach to estimate the application-layer RTT in the presence of encryption. This approach is based on the assumption that a temporal relationship exists between the request and response traffic patterns. In typical client-server communication, a client sends a

request to the server, and the client receives a corresponding server response, with the delay between these actions reflecting the RTT at the application layer. Cross-correlation is a statistical technique that measures the similarity between two time series as a function of relative displacement or delay. By *measuring the similarity between sequences of incoming and outgoing packets as a function of delays*, cross-correlation could identify the delay interval at which the similarity is maximized, indicating the most probable RTT.

Specifically, to apply cross-correlation for RTT estimation, we first (1) extract the timing information of TCP data-carrying packets in both directions. Next, (2) we initiate a moving window of size W, where W represents the number of potential request-response pairs. A potential request-response pair is defined as one or more consecutive outgoing packets followed by one or more consecutive incoming packet. For example, in a packet sequence of out, in, in, out, there is only one potential request-response pair, while in a sequence of out, in, out, in, there are two such pairs. This window slides through the packet sequence, ensuring it always contains W pairs, although the total number of packets within each window can vary. This step is needed as cross-correlation requires multiple instances to accurately determine the most likely delay. Next, (3) we categorize packets by direction, creating separate time series for outgoing and incoming packets, as illustrated in Figure 4.5 (b). We then apply a shift/delay of S milliseconds (where $S \geq 0$) to the outgoing series and compute the cross-correlation function between the shifted outgoing series and the incoming series. We continue this for each incremental delay (+1ms) up to a maximum (e.g.,, 1000 ms). Finally, (4) we identify the delay at which the correlation is highest, as shown in Figure 4.5 (c). This peak correlation represents the most common delay between requests and responses within the current window, which is our estimate for application-layer RTT.

Impact of Correlation Window: The choice of window size W in our approach affects the accuracy and reliability of the cross-correlation results. Setting W to 1, the method analyzes just one outgoing-incoming pair at a time, essentially reducing cross-correlation to a simple analysis similar to transport-layer SEQ/ACK. With a larger window size that spans

much of or the entirety of a persistent connection, cross-correlation will attempt to find a common delay across all request-response pairs within that window. However, the presence of transient network issues or congestion fluctuations can lead to a broad delay distribution within a large window, in which case our approach may not provide an ARTT estimate that is representative across the span of the entire window. In practice, the optimal window size should be adapted and tuned based on the observed network conditions. It should be sufficiently large to cover multiple request-response pairs, allowing cross-correlation to estimate RTT from common delays. Yet, it should not be so large that it covers significant variations in network conditions within the same window. See § 4.6.1 for evaluation of correlation accuracy across different W values.

Impact of Network Jitter: We note that in the real-world, propagation delay can fluctuate even within a short observation window. To address this, instead of relying on exact matches between outgoing and incoming timestamps, our cross-correlation incorporates a weighted distribution centered around each incoming packet's timestamp (as shown in Figure 4.5 (b)), assigning higher weights to packet matches closer in time while allowing for small deviations in delays. Consistent with previous work [131, 132, 98], we model network delays as $i.i.d.$ exponential, which implies that network jitter, defined as the difference of two delays, follows a zero-mean Laplace distribution denoted by $Lap0, b_\delta$. The variance of this distribution should be adapted based on the observed network conditions and expected variability in RTTs. With this, the cross-correlation approach becomes more resilient to slight variations in RTT across a given window.

Constraints: We impose constraints when cross-correlating packet series, ensuring that outgoing packets are only matched with incoming packets that arrive after them to accurately reflect RTTs. Additionally, we restrict peak selection from cross-correlation results to the range $TRTT, \text{max_shift}$ as the application-layer RTT should be at least as large as the transport-layer RTT. Our method assumes a RESTful request-response pattern,

which applies to many protocols like HTTP(S) and FTP but may not hold for all proxy traffic, and relies on consistent RTTs across request-response pairs within a time window.

4.5.2 Distinguishing Δ_{direct} vs. Δ_{proxy}

From the previous discussions, we explained that a key characteristic of proxy traffic is that it typically shows greater discrepancies between RTTs at different layers compared to direct traffic. Having measured transport-layer RTTs and estimated application-layer RTTs, the next step is to determine whether the observed RTT differences RTT_{diff} are more likely to be caused solely by the inherent server processing delays ($\sim \Delta_{direct}$), or are dictated instead by the additional path delays $TRTT_{PW}$ introduced by the distance between the proxy and the application server ($\sim \Delta_{proxy}$). In this section, we formulate a detection problem that provides the basis for a practical algorithm to detect proxy traffic on-the-fly, while also striving for high precision and coverage.

Framework: Sequential Hypothesis Testing. Ultimately, we aim to determine whether the observed RTT_{diff} is primarily caused by the additional transport delays incurred when a proxy forwards data to/from the application server. This scenario can be represented as a statistical detection problem, where the goal is to identify the presence or absence of a specific prior condition (i.e., the spatial separation between the transport and the application endpoints), based on the separation of the distributions under different priors. We adopt the Sequential Hypothesis Testing (SHT) framework, which leverages repeated trials/observations and known outcome probabilities to distinguish between multiple hypotheses. Previous work has used SHT for intrusion detection and censorship detection [146, 107]. SHT aligns well with our problem, where RTT measurements can be streamed continuously into one decision making process.

We begin our construction of SHT by defining an observation to be made when the monitor has observed enough individual packets to form W request-response pairs, where W is the size of the correlation window. We classify the $RTT_{diff} = ARTT - TRTT$ of an

observation as either "inflated" or "matched" by comparing the RTT_{diff} to a predefined threshold T. For each flow (as defined by its four-tuple), let Y_i be a random variable for the ith observation of the flow, such that:

$$Y_i = \begin{cases} 0 & \text{if } RTT_{diff} < T \text{ (matched)} \\ 1 & \text{if } RTT_{diff} \geq T \text{ (inflated)} \end{cases} \tag{4.1}$$

Threshold T is selected as a lower bound of the expected delay introduced by the proxy's additional network path relative to the application server ($TRTT_{PW}$). For example, if the proxy server is located in New York and the application server in Detroit, the physical distance between these two locations would introduce a minimum RTT of roughly 15 ms. In this case, RTT_{diff} should consistently exceed 15 ms, as the transport delay inherent in proxy routing (i.e., the extra distance the data must travel) cannot be eliminated, and additional delays due to application-layer processing might be added on top of the baseline transport delay.

We consider two hypotheses for our problem: the Null Hypothesis (H_0), which suggests that the connection is direct, i.e., $RTT_{diff} \sim \Delta_{direct}$, and the Alternative Hypothesis (H_1), which suggests that the apparent transport-layer server is actually a proxy, i.e., $RTT_{diff} \sim \Delta_{proxy}$. For now, let's assume that RTT_{diff} observations are independent and identically distributed (i.i.d). Then, to implement SHT, we need to identify the prior conditional probabilities for each hypothesis:

$$\begin{aligned} &Pr Y_i = 0 | H_0 = \theta_0 && Pr Y_i = 1 | H_0 = 1 - \theta_0 \\ &Pr Y_i = 0 | H_1 = \theta_1 && Pr Y_i = 1 | H_1 = 1 - \theta_1 \end{aligned} \tag{4.2}$$

These probabilities reflect the initial beliefs about the likelihood of each scenario before observing any RTT_{diff} measurements. From § 4.4, with a reasonably chosen threshold T, we expect that an inflated RTT_{diff} is more likely to occur in a proxied connection than a

direct one: $1 - \theta_1 > 1 - \theta_0$. In an ideal scenario, with perfect network conditions and no processing delays, we can expect θ_0 to approach 1, as the RTTs are expected to be closely matched across layers. On the other hand, with a perfect choice of threshold T, θ_1 would be close to 0, as the additional path delay ($TRTT_{PW}$) should consistently result in larger RTT differences. In practice, however, variability in network condition and server load, as well as the uncertainty around the location of the application server (especially those served by CDNs), make it challenging to maintain a static threshold T that is consistently reliable.

From the construction, we define a likelihood ratio test (ΛY) as each RTT_{diff} observation is made, such that:

$$\Lambda Y = \frac{\Pr Y_1, \ldots, Y_N \mid H_1}{\Pr Y_1, \ldots, Y_N \mid H_0} = \prod_{n=1}^{N} \frac{\Pr Y_n | H_1}{\Pr Y_n | H_0} \geq \eta \tag{4.3}$$

where Y represents the set of available RTT_{diff} observations at any point. Upon receiving each new observation, the algorithm updates the cumulative metric ΛY based on the probability priors and compares the current probability ratio with an upper threshold η, where η is a scalar value that quantifies how much more likely the observations are under H_1 compared to H_0. The product form of (3) is derived from our earlier assumption that the RTT_{diff} observations are independent (they are, in fact, not independent, and we need to account for their dependence, which we expand in Appendix C.2). If $\Lambda Y \geq \eta$, we accept H_1, i.e., presence of a proxy. An optional lower bound could be implemented to conclusively accept H_0. But here we follow a conservative approach where we do not explicitly separate H_0 and $Undetermined$.

Estimating Priors: A critical step of our SHT construction is to empirically measure the probability priors for the two hypotheses regarding direct and proxy connections. However, directly measuring the priors, particularly for the proxy class H_1, is exceedingly challenging. The timing of proxy connections can vary significantly based on factors such as proxy

configuration and the relative distance between the proxy and the web server, making it infeasible to directly collect measurements representative of every possible proxy scenario.

Instead, we adopt an indirect approach to approximate the priors for the proxy class using measurements from non-proxy connections. Specifically, from three geographically distributed vantage points – Atlanta, Singapore, and Amsterdam – we visit the top 5K websites as ranked by Chrome CrUX [91, 166] while capturing traffic traces and TLS keys. Next, from the decrypted PCAP traces, we identify pairs of application-layer request-response interactions, such as Clienthello $\leftrightarrow$ Serverhello or HTTP GET and the corresponding 200 OK responses. For each pair, we record three timestamps: T_0 as the timestamp of the outgoing request; T_1 as the timestamp of the first packet with an ACK number greater than the sequence number of the request; and T_2 as the timestamp of the first packet carrying the corresponding response.

$T_1 - T_0$ is the transport delay and $T_2 - T_0$ is the application delay. To estimate the priors for the direct class (H_0), we analyze the distribution of $T_2 - T_0 - T_1 - T_0$, which is the interval between when the server acknowledges the request and when it responds. This directly measures the typical processing delays inherent in direct connections (i.e., RTT_{diff} for H_0). For the proxy scenario (H_1), we instead consider the distribution of the complete application-layer round-trip time $T_2 - T_0$. The rationale is that *the apparent processing delay at the proxy would resemble the full path latency between the proxy and the application server, plus the processing delay at the server*. In other words, $T_2 - T_0$ measured at the proxy server would be equal to $T_2 - T_1$ measured at the proxy client. This approach essentially simulates the distribution of RTT_{diff} for connections routed through proxies located in these three locations.

Figure 4.6 presents the CDFs of the measured prior probabilities, comparing $T_2 - T_1$ (for H_0) against $T_2 - T_0$ (for H_1). These distributions match our initial assumptions: the propagation delay introduced by additional transmission paths in proxy scenarios are markedly more pronounced than the variations in processing delays, and an inflated RTT_{diff} is more likely to result from a proxy connection than a direct connection. For example, at

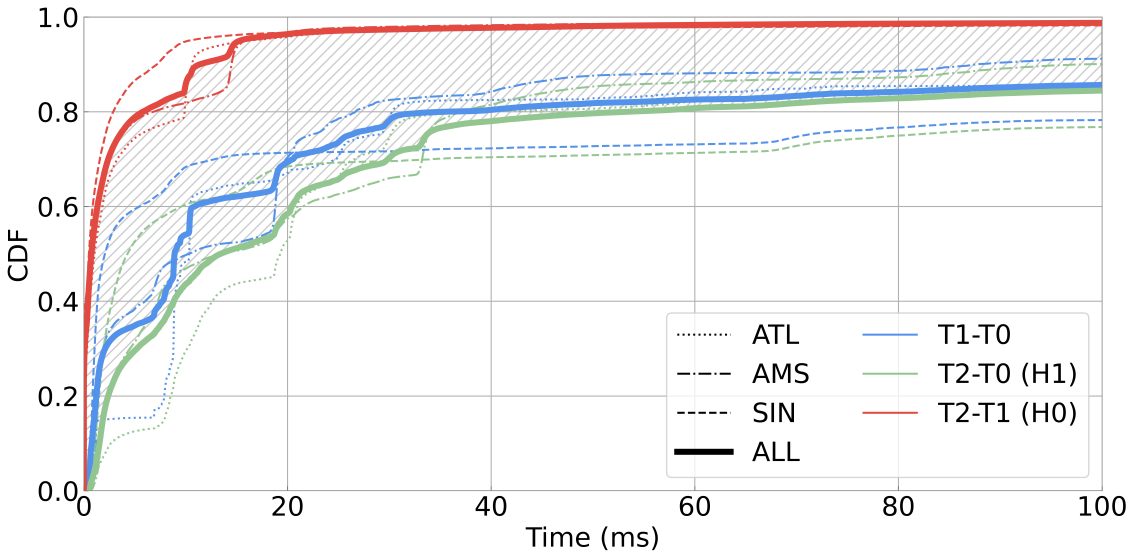


Figure 4.6: **Distributions of RTT differences for prior estimation.** T1-T0 is TRTT, T2-T0 is ARTT and T2-T1 is RTTdiff. Shaded area represents the separation of the distributions under different priors. ◇

$T = 15$, we observe $\theta_0 \approx 0.95$ and $\theta_1 \approx 0.50$. Furthermore, Figure 4.6 also highlights the distributions of RTT differences across the three vantage points, located in three continents. While there are some variability among their distributions, the overall separation between classes is generally consistent. This consistency suggests that 1) the observation – inflated RTT_{diff} are more commonly associated with proxy connections – holds true across different geographical regions; and that 2) the same estimated priors are likely to be applicable regardless of the specific location of the proxy server.

4.5.3 Implementation

The RTT_{diff} exploit begins when the traffic monitor identifies a new connection flow. The monitor accumulates packets, recording relevant fields such as timestamps, SEQ/ACK, *etc*, until a sufficient number of packets have been collected to form a correlation window. Every time a correlation window is complete, the monitor performs cross-correlation, as detailed in § 4.5.1, to estimate the RTT_{diff} for that window.

If the flow's cumulative probability ratio (ΛY) has been previously initialized, the

monitor updates the ratio following the process described in § 4.5.2. If ΛY has not been initialized and the current RTT_{diff} exceeds the threshold T, the monitor initializes ΛY for the flow. Otherwise, the monitor discards the current window and awaits the next set of observations. This step is designed to accommodate proxy protocols with varying handshake lengths, during which the packet exchange is strictly between the client and the proxy, resulting in RTT_{diff} similar to those of a direct connection. The monitor repeats this step until ΛY is either initialized or at least three request-response pairs have been observed [3]. The monitor compares ΛY with η after each update to the cumulative ratio, and logs the flow as proxy if the threshold is exceeded. Otherwise, the monitoring continues until the connection terminates (refer Figure C.1 in Appendix for the workflow).

We have implemented a PoC exploit using Zeek, an open-source network monitoring and analysis framework [231]. This implementation takes the form of a plugin that accepts a sequence of packet metadata (timestamps, size, SEQ/ACK) and estimates a sequence of RTT_{diff}.

We emphasize that our instantiation of the fingerprinting concept is not meant to be the most optimized or effective form of exploitation possible. Other approaches might exploit the same fingerprint more efficiently. Our intention is to demonstrate the fingerprint's potential for exploitation and to highlight the necessity for proactive countermeasures.

4.6 Evaluation

We evaluate the practicality of exploiting RTT discrepancy across layers to detect (obfuscated) proxy traffic. Our goal is to conduct evaluation under realistic conditions, mirroring the operational constraints typically faced by censors.

Before evaluating RTT_{diff} as a standalone fingerprint, we start by focusing on individual components of our exploit. In § 4.6.1 and § 4.6.2, we evaluate the effectiveness of cross-

[3] Some proxy protocols like Shadowsocks support 0-RTT mode, combining the proxy's handshake packets with the first application payload. For others, the handshake procedure typically completes with less than three round trips.

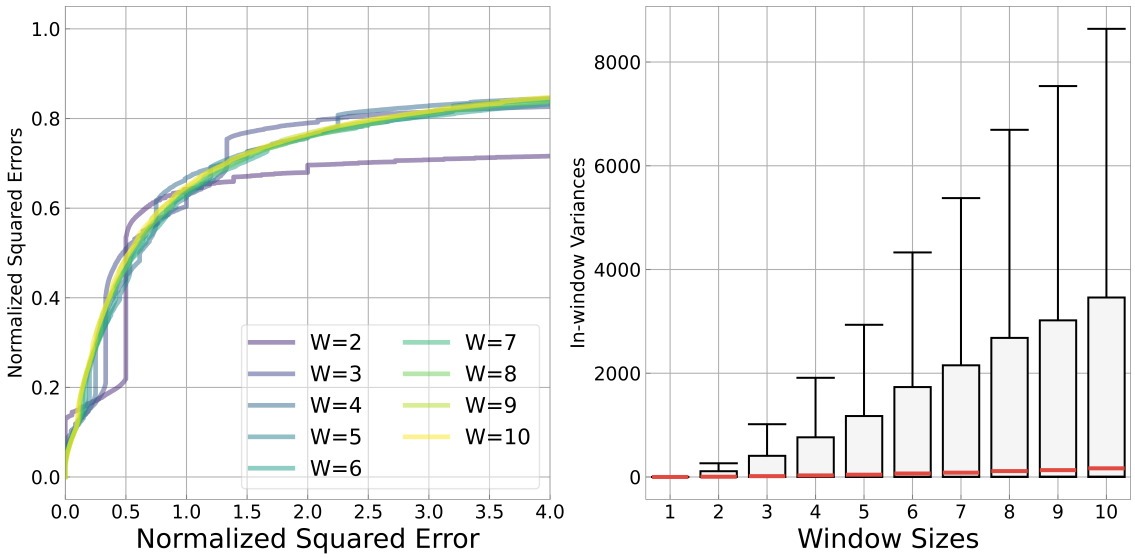


Figure 4.7: **Left:** Distributions of normalized correlation errors. **Right:** ARTT variance with different window sizes (median in red).

correlation in estimating the application-layer RTT, as well as exploring how the size of the correlation window W and the threshold T affect the efficacy of the exploit. Next, in § 4.6.3, we thoroughly evaluate the RTT_{diff} using a combination of controlled proxy traffic and real user traffic collected from an Internet provider. Throughout this evaluation, we aim to understand both the *sensitivity* of the fingerprint – the extent to which proxy traffic is vulnerable and factors that can impact this vulnerability – and its *specificity* – the uniqueness of the fingerprint to reliably distinguish proxy from regular traffic.

4.6.1 Evaluating Cross-correlation Method for ARTT Estimation

We evaluate the reliability of our cross-correlation-based method for estimating application-layer RTTs (ARTTs), as the success of subsequent exploit phases depends on these estimations being accurate. As mentioned before, for $W > 1$, there's a possibility that request-response may exhibit a broad range of delays even within the same window. For this, we first analyze the inherent variability of ARTTs within the same correlation windows. Using the dataset collected in § 4.5.2, we quantify the ARTT in-window variances across different window sizes W. As shown in Figure 4.7, the variance of ARTTs within each window tends

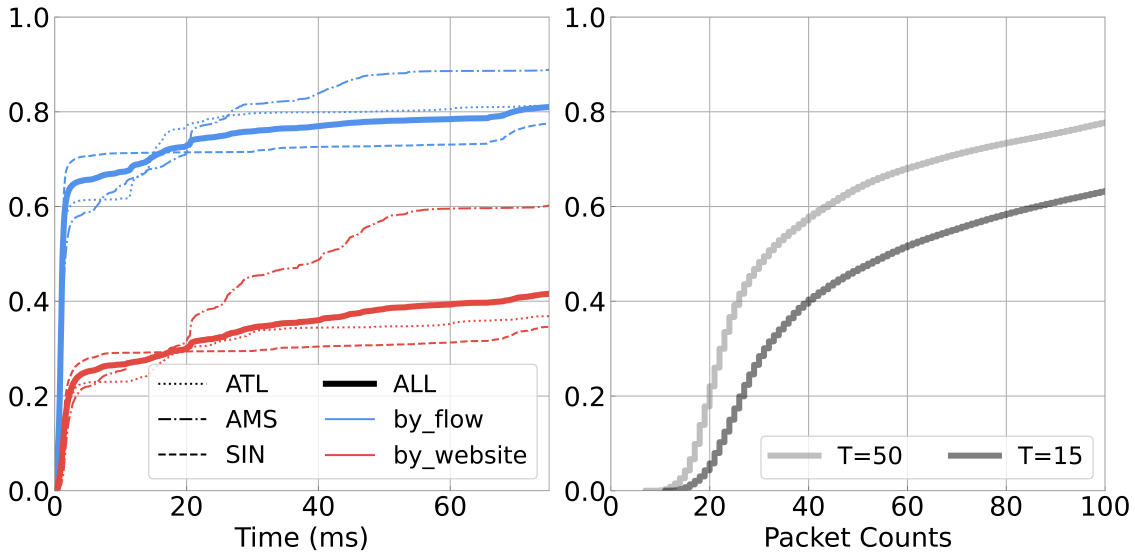


Figure 4.8: **Left:** Distribution of transport-layer delay from proxy locations. Blue lines represent the delay of all the flows generated and the red lines represent the delay of selected flows that had the maximum delay for each website visit. **Right:** Number of packets observed before detection for $T = 15ms$ and $T = 50ms$, fixing FPR=.01.

to increase with larger window sizes, which is expected as more request-response pairs can introduce more variability due to differing network conditions and server response times. For example, the median variance increases from 4.5 for $W = 2$ to 168 for $W = 10$.

Accounting for this inherent variance, we use normalized squared errors as a measure to evaluate the accuracy of our ARTT estimation: $\frac{estimated_ARTT - median_ARTT^2}{Variance}$. This measure compares the difference between the estimated and the median ARTT of a window to the baseline variance of the window. As shown in Figure 4.7, the distribution of the normalized errors suggests that the majority of our estimations yield relatively low errors. Balancing between need for multiple request-response pairs for effective correlation against the increased variance with larger window sizes, we use $W = 3$ for subsequent evaluations. With $W = 3$, over 60% of our estimations have a normalized error less than 1, implying that our estimates typically fall within the distribution of actual ARTTs. Additionally, about 80% of estimations have errors less than 2, in which case estimates are within 1.4 standard deviations of actual median ARTT.

4.6.2 Evaluating the Susceptibility of Proxy Traffic and the selection of threshold T

We evaluate the extent to which proxy traffic is susceptible to the exploit and how threshold T affects this susceptibility. While the concept of cross-layer RTT differences is inherent to all proxy activity, an adversary's ability to reliably detect such differences depends significantly on the proximity of the proxy to the application server. As such, the selection of T is crucial as it determines what constitutes a "significant" RTT discrepancy indicative of proxy use. As mentioned before, the threshold T should exceed the transport delay between the proxy and the server, which is an unavoidable baseline delay that is reflected in the cross-layer RTT discrepancy.

We measure the distribution of transport-layer delays from three proxy locations while visiting the top 5K domains to assess this susceptibility and to inform the selection of T. The results are shown in Figure 4.8. The blue curves represent the distribution of transport delays for individual flows between the proxy and the website server (i.e., *per-flow exposure*). Surprisingly, more than half of these flows exhibit minimal transport delays, which means that the difference in RTT caused by proxy routing would also be minimal. Apart from the fact that our proxies are located in data centers with potentially better connectivity than average hosts, we note that most of these low-delay flows are to CDNs like Cloudflare, Google, and Fastly (refer to Table C.2 in Appendix). The prevalence of CDNs, to some extent, lessens the visibility of additional delays caused by proxy routing. Despite this, the proposed exploit still poses a realistic threat: because a single website visit typically generates dozens of individual flows, a proxy server could still be exposed even if only *some* of these flows exhibit significant delays. To understand the possibility of observing inflated RTT_{diff} during a user session, we extracted out the flows with maximum transport delays for each website visit. The red curves in the same figure illustrate this distribution, indicating that nearly every visit likely includes at least one flow with a significant RTT_{diff} indicative of proxying (i.e., *per-website exposure*).

The choice of T also affects the speed of detection. A lower T leads to a smaller ratio

Protocol	Type	Changes Timing
Plain SOCKS	Plaintext	No
VMess	Obfuscated (fully encrypted)	No
Shadowsocks	Obfuscated (fully encrypted)	No
VLESS over TLS	Obfuscated (TLS-based)	No
Trojan	Obfuscated (TLS-based)	No
VMess over Websocket	Obfuscated (HTTP)	No
Shadowsocks over WS/TLS	Obfuscated (TLS-based)	No
VLESS over WS/TLS	Obfuscated (TLS-based)	No
XTLS-Vision	Obfuscated (TLS-based)	No
SOCKS over obfs4	Obfuscated (fully encrypted)	Yes

Table 4.1: List of proxy protocols tested in evaluation. ◇

of $PrT|H_1$ over $PrT|H_0$, thereby requiring more observations to maintain a similar level of specificity (False Positive Rate) as a higher T. Figure 4.8 shows the number of packets required to achieve the same level of detection confidence (FPR=.01) for $T = 15$ and $T = 50$. We use $T = 15$ for the following evaluation. Using the conversion factor of $\frac{4}{9}c$ from prior work [109], a 15ms network delay roughly translates to a physical distance of 600 miles, approximately the distance between New York and Detroit, or San Francisco and San Diego. In other words, as long as the proxy and the application server are separated by at least this distance, a threshold of $T = 15$ should consistently observe inflated RTT_{diff} caused by proxy routing.

4.6.3 Evaluating RTT_{diff} as a Standalone Fingerprint

We perform an in-depth evaluation of the RTT_{diff} fingerprint using both controlled proxy traffic from our testbed and real user network traffic sourced from an Internet provider.

4.6.3.1 Evaluation Setup

Since subtle differences in traffic timing create the fingerprint, our evaluation aims for diversity among proxy clients, servers, and application servers to cover a wide range of proxy scenarios. We set up five geographically distributed hosts to act as proxy clients. These include locations in Hong Kong, China (HKG) and Chelyabinsk, Russia (CEK),

reflecting the typical user base of censorship circumvention tools. Notably, Hong Kong, as a special administrative region, is not subject to the same GFW restrictions as mainland China [155]. Similarly, our Russian site is located in a data center, which is exempted from the typical residential censorship restrictions [225]. We confirmed the absence of filtering behaviors commonly associated with GFW and Russia's TSPU [225, 44, 155, 95], and as such our proxy flows are not to be affected by any local censorship measures. Nonetheless, we deployed additional clients in nearby Tokyo (TYO) and Stockholm (ARN). We set up the fifth client in Detroit, USA.

The proxy servers routing this traffic were placed in Singapore (SIN), Amsterdam (AMS), and Atlanta (ATL). Like the clients, these servers are single-tenant bare metal machines that are well-provisioned with processing and network resources. This minimizes any impact on packet timings from processing or bandwidth constraints, as each proxy client/server handles only one website visit at a time. Using a script with Selenium, we had each client use each of the proxy servers to visit the top 5K domains as ranked by the CrUX report [91] (either ranked globally or region-specific).

We selected and tested a range of popular proxy protocols in our evaluation. Our selection is informed by discussions in censorship forums [9, 1], targeted attacks by nation-state censors [31, 216, 135, 79], and protocols selected for evaluation by prior work [223, 59, 77]. These include fully encrypted protocols like shadowsocks and VMess, and TLS-based obfuscated proxies like VLESS and Trojan. A complete list of the proxy protocols tested is provided in Table 4.1.

To evaluate the specificity of the RTT_{diff} fingerprint and estimate potential false positives should the fingerprint be deployed, we partnered with Merit Network. Real-user traffic was mirrored from a major Point-of-Presence of the ISP to our monitoring server, with traffic volumes reaching up to 50Gbps. Such volume necessitates sampling, as we perform all feature extraction and analysis directly on Merit's server due to ethics and privacy concerns. We applied a conservative sampling rate of 1/8, based on connection 4-tuple, to minimize

the effect of packet loss and accommodate traffic spikes. On this server, we deployed a Zeek cluster loaded with our plugin. Several sanity checks were performed on each new flow, including verifying a complete TCP handshake and the presence of a PSH flag. The number of flows that passed these checks was used as the baseline (denominator) for calculating the False Positive Rate (FPR). This approach is consistent with prior study [216], such that FPR metrics are directly comparable. Our evaluation on ISP traffic extended over a 10-day period, during which we analyzed over 102 million TCP flows after sanity checks. We applied our exploit to both the controlled proxy traffic and the ISP traffic using the exact same configuration and parameters.

4.6.3.2 Sensitivity

We first note that, aside from obfs4, *the results across all tested proxy protocols were highly identical*, including most widely-used obfuscated proxies such as shadowsocks or VLESS. The fingerprinting vulnerability is largely protocol-agnostic, as it does not target specific obfuscations but rather exploits an *inherent* characteristic of proxying. Most proxy protocols, including those with sophisticated obfuscations, do not by design alter packet timings. Even in protocols that incorporate packet-level padding, the padding is appended only to application-layer payloads, without sending dummy packets when there's no data to send. Obfs4, which was designed to obfuscate packet inter-arrival times, is an exception and is further discussed in § 4.7.1.

The decision threshold in hypothesis testing, η, balances sensitivity against specificity. Raising η allows us to lower false positives to an arbitrary level, albeit at the cost of missing detections for many proxy flows that do not reach the decision threshold before they terminate. Selecting a threshold comes down to maximizing the detection of proxy traffic while maintaining a tolerable level of blocking legitimate traffic—a censor's operational constraint. As part of our evaluation, we measured the fingerprint's sensitivity on controlled

Proxy/Client	DTW	HKG	TYO	CEK	ARN
Remote DNS Resolution, CrUX Global 5K					
ATL	.207 / .819	.233 / .828	.219 / .784	.201 / .802	.215 / .791
SIN	.177 / .738	.172 / .727	.180 / .743	.199 / .732	.201 / .738
AMS	.201 / .775	.181 / .747	.201 / .759	.197 / .711	.172 / .766
Local DNS Resolution, CrUX Global 5K					
ATL	.372 / .905	.340 / .876	.377 / .880	.443 / .927	.448 / .907
SIN	.455 / .898	.313 / .842	.315 / .851	.438 / .892	.424 / .880
AMS	.435 / .905	.337 / .839	.328 / .856	.293 / .854	.339 / .877
Remote DNS Resolution, CrUX Regional 5K					
ATL	- / -	.186 / .712	.193 / .765	.410 / .879	.364 / .851
SIN	- / -	.147 / .719	.133 / .748	.330 / .851	.313 / .842
AMS	- / -	.176 / .658	.190 / .722	.352 / .827	.221 / .808

Table 4.2: Evaluation results on proxy traffic. Results are shown for both per-flow detection rate / per-website detection rate. Threshold η selected for FPR = .01

proxy traffic while fixing an upper bound FPR at .01 evaluated on ISP traffic. We further discuss false positives in the next section.

Table 4.2 shows the results of this case study. Looking at the results for each proxy location in the top section of the table, we note that the detection rates across all client locations are largely consistent. This matches our expectation, as the RTT_{diff} fingerprint is generated by the routing between the proxy and application server, rendering the client's proximity to the proxy irrelevant. The detection rates at the per-flow level are only moderately effective, with approximately 20% of flows correctly flagged. This can be largely attributed to the connectivity provided by CDNs, which to some extent offsets the additional delays caused by proxy routing. Most of the proxy flows that we fail to detect are requests to static content cached at CDNs or advertising and tracking, typically occurring within 5ms of our proxy locations, as measured in Figure 4.8. We highlight that our evaluation uses traffic to top domains instead of a random sample of websites to simulate traffic that users are most likely to route through proxies. But since top domains are commonly served by CDNs, our

findings likely represent a lower bound of the fingerprint's sensitivity. Still, even for top domains, the implication from such per-flow exposure becomes even more alarming when aggregated by website visits: at the per-website level, detection rates exceed 70% across all client and proxy locations. This means that the adversary can almost guarantee that the client and server IPs of proxy flows will be flagged after just a few website visits.

Our results also highlight the increased susceptibility of proxy traffic to the fingerprinting exploit under certain configurations. Specifically, most obfuscated proxy tools expose a SOCKS interface for local applications, allowing the user to supply either a destination IP (DNS resolved locally) or a domain name (resolving DNS at the proxy). Users who resolve DNS locally are considerably more susceptible to the exploit, with per-flow rates approximately doubling across all client/proxy locations. When DNS is resolved at the proxy, it uses nearby DNS servers to retrieve IPs that are optimally reachable from the proxy. Local DNS resolution, however, may direct the proxy to farther endpoints, particularly back towards the client's region, thus increasing the distance and consequently the discrepancy in RTTs.

Lastly, we substituted the global site list with region-specific lists (CrUX-China for clients in Hong Kong and Tokyo, and CrUX-Russia for Russian and European clients). For the Russian list, this has shown to significantly increase the sensitivity of the fingerprint. Regional lists, more tailored to local browsing behaviors, often include sites less likely to be hosted on global networks (e.g.,, local education, government, finance) [165] and thus may exhibit larger RTT discrepancies. In the following mitigation section, we will discuss implications of our results on proxy configurations, such as using routing rules to ensure local traffic remains local.

4.6.3.3 Specificity

Analysis on collateral damage is crucial in both the design of circumvention protocols and in envisioning potential fingerprinting attacks against them. Given that *Merit Network*

Category	Identifier	Percentage of All Positives (%)
Rmt Port	443	57.88
	993	33.29
	80	4.47
	5222	0.43
	9001	0.30
SNI	apple.imap.mail.*.com	14.89
	imap.*.com	5.32
	android.imap.mail.*.com	2.91
	imap.mail.*.com	2.90
	*.*health.com	2.13
	(empty) / Not applicable	17.47

Table 4.3: Distribution of Detection Positives by Remote Port and SNI. Note: '*' in SNIs masks service names, not wildcards.

operates in a region with minimal network censorship, we *conservatively* consider all detections from the ISP traffic as false positives, which we upper-bound at .01 for our evaluation.

We note the false positives are not evenly distributed. Table 4.3 groups the top heavy hitters either by remote port or by SNI. Notably, traffic to port 993, commonly used for IMAP/SSL and comprising less than 1% of the total traffic, accounts for nearly a third of all potential false positives. Similarly, results aggregated by SNIs also suggest that IMAP traffic frequently results in what we term *categorical false positives*, on which we provide further analysis in Appendix C.5. We note that the assumption underlying our correlation-based ARTT estimation is a RESTful request/response communication pattern, which might not perform well on traffic that deviates from this pattern. Implementing a pre-filter to whitelist IMAP traffic further reduces the FPR by one third.

This level of per-flow FPR (0.6%-0.7%) is lower than that reported in fingerprinting attacks discussed in prior work [197], and is on par with the estimated FPR of attacks *deployed* by GFW against fully encrypted proxies [216]. Nonetheless, this FPR *still implies a non-negligible amount of collateral damage* – especially considering the generally low base-rate of circumvention traffic. Yet, the standalone fingerprint RTT_{diff} can be complemented with

protocol-specific fingerprinting attacks, and can additionally be applied on a per-host basis by accumulating per-flow RTT_{diff} fingerprints over time for each destination host. Prior work has shown that, even with looser classifiers, host-based analysis can significantly reduce false positives – even to zero – by aggregating multiple flow-level observations [195]. As such, we believe that the evaluated vulnerability presents a realistic threat that necessitates vigilance and proactive countermeasures from the circumvention community.

4.7 Discussion

Previous sections demonstrate the practicality of exploiting cross-layer RTT discrepancies to fingerprint proxy traffic. This poses a significant threat to the availability of proxies, particularly obfuscated ones that are essential for facilitating information access across the world's most restricted networks.

Exploiting the RTT_{diff} fingerprint requires only passive monitoring and limited stateful connection tracking, simplifying its deployment and making it within the reach of virtually any network operator. Unlike previous attacks targeting specific proxy or obfuscation protocols [197, 87, 81, 80], this timing-based fingerprint is an inherent characteristic of proxy routing. The protocol-agnostic aspect of the fingerprint broadens its applicability, which in turn heightens the severity of its implications. More, the fact that the exploit makes on-the-fly detections provides censors with more options for intervention. In cases where proxies are hosted on shared IPs (e.g.,, CDN), detecting proxy mid-flow allows flow-level actions such as resetting or throttling the violating flow, when traditional blanket IP blacklisting could incur a higher cost for censors.

Despite the demonstrated practicality, **we advise caution in interpreting our results**: our real-world evaluation suggests a low, but non-trivial rate of false positives, including categorical false positives, which would compel any censor to carefully weigh the deployment costs. We posit that a potential adversary is more likely to use the RTT_{diff} fingerprint in combination with existing protocol-specific attacks, which are orthogonal and complemen-

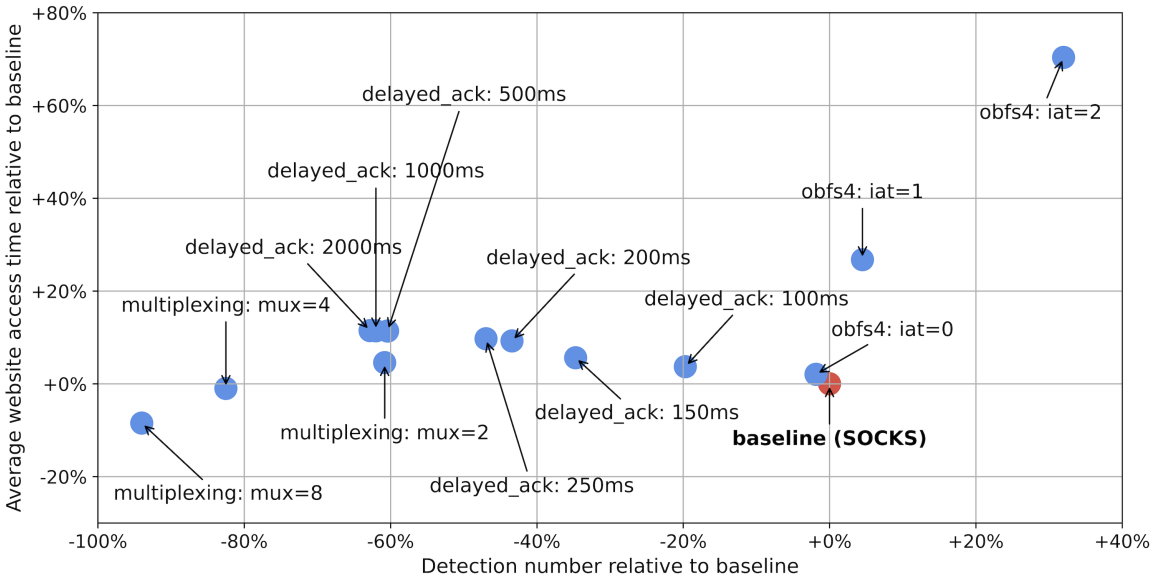


Figure 4.9: **Per-flow detection and performance of potential mitigations.** *mux=N* combines every N logical streams into one flow.◇

tary to our approach, rather than as a standalone method. Also, the nature of the fingerprint – relying on subtle patterns in packet timings – means that its efficacy could be affected by unpredictable network changes along the path. Yet, it would not be a sound strategy to depend solely on the unreliability of the network as a defense against such timing-based fingerprints. The responsibility therefore falls on proxies to implement proactive countermeasures. Below, we discuss a few potential mitigation approaches and, for some, provide a preliminary evaluation on their efficacy.

4.7.1 Potential Mitigation Approaches

Proxy Configuration. Our results suggest that the RTT_{diff} fingerprint is independent of the client's location. Still, certain configurations on the client side can reduce the susceptibility to some extent. For example, domain name resolution should be handled at the proxy to prefer IPs that are optimally reachable from the proxy, reducing the additional delay introduced by proxy routing. Additionally, specific routing rules should be implemented on the client to ensure that local traffic – traffic to services located on the same side of the firewall as the client – remains local and is not routed through the proxy. Instances of

categorical false positives are particularly relevant not only to adversaries implementing the attack but also to proxy users who might leverage these false positives to their advantage. For example, proxy users configuring obfuscated proxies with a decoy domain [7, 220] should consider selecting a domain that naturally exhibits inflated or unpredictable RTT_{diff}, such as IMAP services, to increase the uncertainty of the fingerprinting attack.

Obfs4. The evaluated fingerprint is based on timing patterns introduced by proxy routing. Obfs4/scramblesuit [140, 211] is a well-known protocol specifically designed to obfuscate packet lengths and inter-arrival times through pseudo-random padding and delays. Figure 4.9 illustrates the effect of incorporating obfs4 into a proxy setup compared to the baseline. We find incorporating obfs4 degrades the performance *but also increases the traffic's susceptibility to the fingerprint*. This somewhat counter-intuitive outcome can be attributed to obfs4's method of randomizing inter-arrival timings across packets: it injects delays following the arrival of application-layer data at the sending buffer. As such, application-layer transactions always appear to have a larger RTT after obfs4's obfuscation, widening the discrepancies in cross-layer RTTs.

Delayed ACK. The RTT_{diff} fingerprint can be viewed as the result of a proxy server *prematurely* acknowledging data packets that are, in fact, still being transmitted to the application server, i.e., ACKs are being sent "too early". One way to reduce the RTT discrepancy is through the TCP Delayed Acknowledgment mechanism [52], which provides a native way to delay the transmission of ACK packets. This feature is typically enabled by default on Linux systems with a delay timer of 40ms, so that ACKs can ideally be sent alongside data-carrying packets to minimize overheads. By adjusting the delay timer to 500ms (the maximum allowed by RFC [46]), we find a significant reduction in the number of detections by over half compared to the baseline, as shown in Figure 4.9. This method, however, does result in performance trade-offs. Moreover, it is unclear whether having an atypical ACK delay timeout might itself become an exploitable fingerprint.

Multiplexing. Traffic analysis attacks will always have a range of uncertainty leading to

false positives, and the goal of obfuscation is to further increase this uncertainty [211]. One way to introduce additional entropy into packet timings is through connection multiplexing, which interleaves packets from multiple coexisting logical streams into a single network flow, obfuscating patterns in packet size, timing, and direction. The variability in the number and type of the logical streams (i.e., interactive or bulk transfer) introduces additional unpredictability. Prior work has shown that connection multiplexing could be an effective defense against fingerprinting over packet sizes [223]. Similarly, we also find that even a basic form of multiplexing – combining every two logical streams into one network flow – lowers detections both by numbers and rates, as shown in Figure 4.9.

However, multiplexing might also introduce new types of fingerprints. For one, multiplexed flows tend to live longer and carry more packets compared to non-multiplexed flows. For example, the median number of request-response pairs in multiplexed flows is higher than over 97% of all flows observed from the ISP, which already makes them outliers and more conspicuous. But even in comparisons with this narrow 3%, multiplexed proxy flows could still be differentiated: since multiplexing interleaves packets from different streams, our estimation of application-layer RTT using correlation may not converge, adding a layer of variability which, paradoxically, might itself be fingerprintable. As shown in Figure C.2 in Appendix, the sequence of estimated RTT_{diff} from multiplexed proxy flows exhibit wider confidence intervals compared to that of ISP traffic. We therefore advise caution in adopting multiplexing as the only mitigation strategy without further research.

Traffic Splitting. Similar to connection multiplexing, traffic splitting provides an alternative or complementary approach to obfuscation. Traffic splitting decouples the upstream and downstream of a traffic flow, reducing the information exposed to an adversary with limited network perspective [199, 58]. Specifically, in the case of the RTT_{diff} fingerprint, such decoupling distorts the timing pattern our fingerprinting attack relies on, effectively nullifying cross-correlation as a method for estimating application-layer RTTs. An example implementation of this strategy is the *SplitHTTP* transport developed by XTLS [126].

While not a proxy protocol itself, *SplitHTTP* can be integrated with existing protocols like shadowsocks in a proxy setup. When splitting upstream and downstream into separate network flows, the RTT_{diff} exploit was unable to make any detections, as no single flow contains paired request&response that can be used for cross-correlating RTTs. Further research is needed to understand whether a sophisticated adversary could perform cross-flow correlation based on host IP addresses [199].

Application-independent Traffic Scheduler. It might seem surprising that obfs4, a protocol designed to obscure packet timing, only seems to exacerbate the issue rather than mitigate it. In essence, this is because obfs4 implements timing obfuscation by randomly *delaying* the transmission of application-layer data already within the sending buffer: when the application has data to send, obfs4 sends the data possibly with some delay; when the application is quiet – or, as in our case, if data is still in transit – obfs4 remains quiet as well. This observation is analogous to previous research noting the inability of padding-based size obfuscations in mitigating detection based on packet sizes that are conspicuously large [223].

The current generation of obfuscation schemes operates under an "application-dependent" model: they pad or delay data as it moves from the application layer to the obfuscation layer. Their capability to obfuscate is limited to one direction only – larger size, more delay [75, 72]. While acceptable for certain scenarios, this limitation becomes critical when the traffic to be obfuscated inherently involves longer RTTs or larger sizes than what is considered "normal", as in our case. A more principled obfuscation framework must operate independently of application data, centered around a traffic scheduler that dictates the timing and volume of traffic according to a predefined schedule, regardless of the underlying application-layer traffic pattern. Such a framework would require an overhaul of existing obfuscation implementations. The focus of the obfuscation layer should shift from responding to incoming application data, to adhering to the demands of the traffic schedule, i.e., from being triggered by input to aligning with the required output. For example,

suppose the traffic scheduler demands a packet to be sent at the moment, but no application data is ready. In that case, the obfuscation layer should still send a padding-only dummy packet, thereby capable of simulating both larger and smaller RTTs [75, 72].

VLESS is in the process of revamping their obfuscation implementation to provide the flexibility necessary to support an application-independent traffic scheduler [221]. A traffic schedule that demands at least one `send()` operation every transport-layer RTT seconds would fundamentally eliminate the cross-layer RTT discrepancy pattern exploited in this work. Such an application-independent scheduler, however, introduces new questions – defining what constitutes a "legitimate" shape for proxies to simulate, evaluating the performance implication of such scheduling, and assessing whether a widely adopted schedule might itself converge into a pattern presenting new fingerprinting vulnerability – all remain open questions for future research.

4.7.2 Extending to Network-layer VPNs

Our work focuses on transport-layer obfuscated proxies used for censorship circumvention. However, the underlying principles of our approach – sessions spanning different distances and reflecting in cross-layer RTT discrepancies – are applicable to network-layer VPNs as well when used without traffic multiplexing. Figure 4.10 illustrates the sequence timing diagram for a TCP-based VPN connection, showing how RTT_{diff} is defined in this scenario and how the same exploit can be directly applied (for UDP-based VPNs, an additional step is required to estimate the transport RTT of the tunnel).

To demonstrate the potential applicability to network-layer VPNs, we conduct a supplementary experiment with the three popular VPN protocols: OpenVPN (TCP), WireGuard, and OpenConnect, which are generally used across VPN products, either commercial or otherwise. We target the index pages of the top 1K CrUX domains (without redirects), applying the RTT_{diff} exploit using the same parameters from § 4.6. (For WireGuard, which operates over UDP, the transport-layer RTTs are estimated from the initial handshake.) Fig-

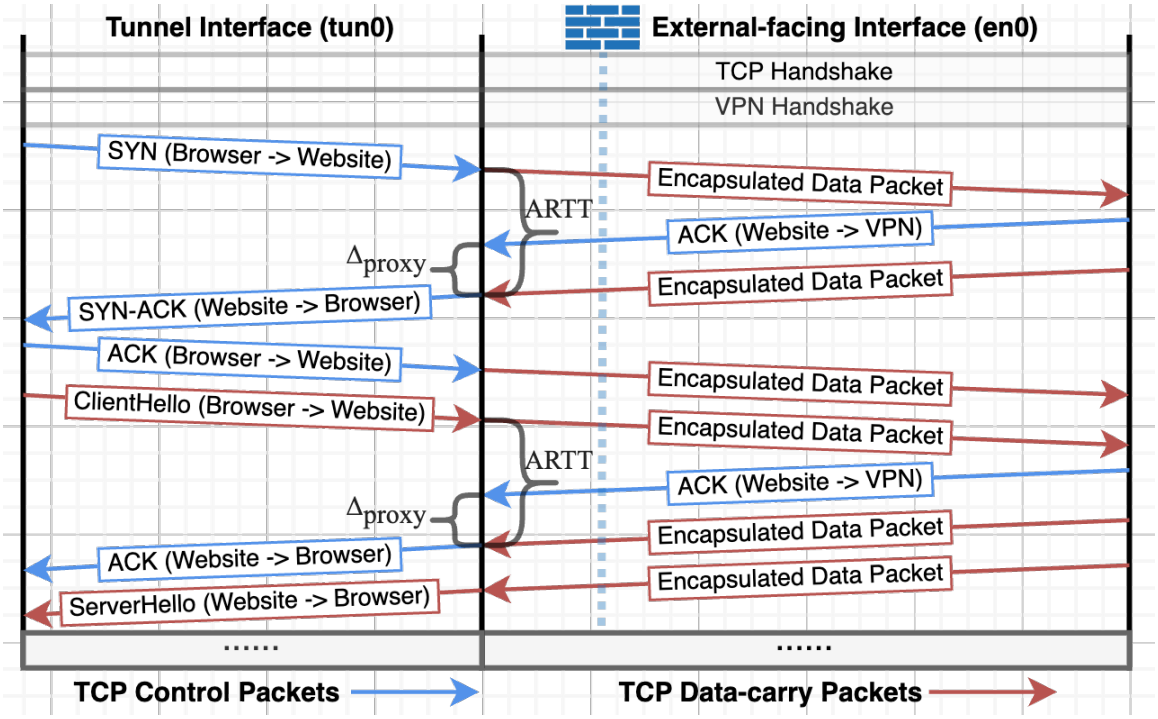


Figure 4.10: **Sequence timing diagram illustrating cross-layer RTT discrepancies in network-layer VPN connections.** Both data-carrying and TCP control packets of the encapsulated flow exhibit cross-layer RTT discrepancies. ◇

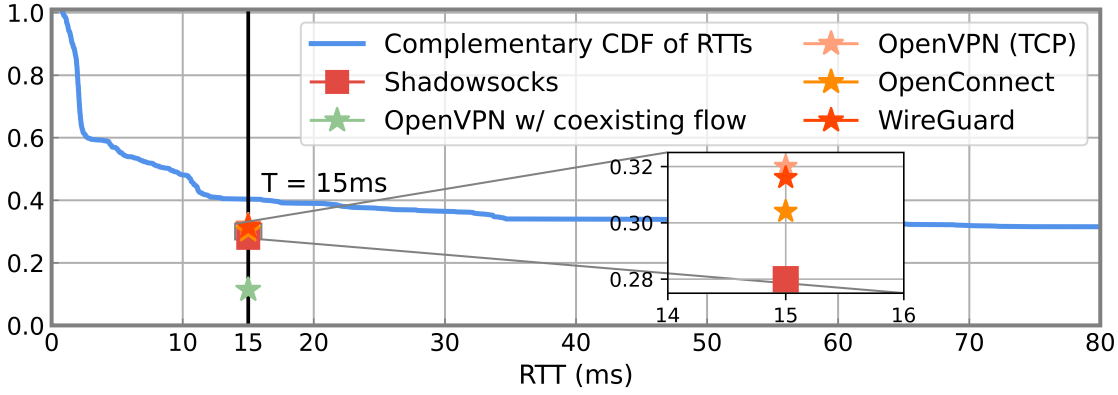


Figure 4.11: **Detection rates for network-layer VPNs**, with the Transport RTT Distributions for CrUX Top 1K domains. A request to *google.com* is used as the coexisting flow. ◇

ure 4.11 shows the detection rates across the VPN protocols, with shadowsocks included for comparison. The exploit was effective across all protocols, with detection rates approaching theoretical expectations (the C-CDF of RTT), as these protocols do not alter traffic timing. Notably, VPNs showed slightly higher detection rates than shadowsocks, which can be attributed to the nature of network-layer tunneling tools where not only data-carrying (e.g.,, clienthello, serverhello) but also TCP control packets (e.g.,, SYN, ACK) of the encapsulated end-to-end flow can be used to identify cross-layer RTT discrepancies, as illustrated in Figure 4.10.

However, extending the exploit to network-layer VPNs *in practice* might not be feasible due to most VPNs configuring themselves as the "default-gateway" on a user's device, effectively multiplexing multiple logical streams into one network flow. As previously discussed, this complicates the cross-correlation of application-layer RTTs and, therefore, reduces detection rates (as highlighted in Figure 4.11 for OpenVPN with two coexisting flows). Yet, it's important to note that most VPN protocols lack any form of obfuscation to begin with and can be trivially detected by protocol parsers. Even VPNs that claim to be "obfuscated" have been shown to be vulnerable to more straightforward fingerprinting attacks that are orthogonal to packet timing, as demonstrated by prior work [227, 92].

4.8 Conclusion of Chapter IV

In this chapter, we examined the potential for exploiting a timing-based fingerprint to detect traffic from obfuscated proxies. This fingerprint is created by the inherent misalignment between transport- and application-layer sessions in proxy routing, which is reflected in the discrepancies between transport and application RTTs. We detailed a possible exploit by an on-path censor using this fingerprint to detect proxy traffic and demonstrated its general feasibility. We hope our study draws the attention of the circumvention community to packet timing as an area of concern and leads to the development of more sustainable countermeasures.

CHAPTER V

The Discriminative Power of Congestion Control Behaviors[1]

In this chapter, I investigate the discriminative power of congestion control behaviors and how they can be exploited to fingerprint obfuscated traffic. Obfuscated proxies often have to operate under adverse network conditions, especially over cross-border links with high packet loss. These scenarios motivated the development of proxies that implement custom congestion control algorithms (CCAs) that aggressively sustain high sending rates by disregarding standard congestion responses. In this chapte, however, I argue that such custom CCAs are fundamentally at odds with the core principles of censorship circumvention. Using Hysteria and TCP-Brutal as case studies, I demonstrate how these custom CCAs produce traffic patterns that significantly deviate from standard TCP and QUIC behaviors, and further develop simple, threshold-based classifiers to show how an adversary can distinguish such proxy traffic by its lack of response to congestion signals. The results from this chapter emphasize that any performance optimizations for vulnerable traffic (i.e., traffic being targeted for blocking) must be grounded in standard protocol behaviors to maintain the indistinguishability required for effective traffic obfuscation.

[1]This chapter is based on: Wayne Wang, Diwen Xue, Piyush Kumar, Ayush Mishra, Roya Ensafi. 2025. **Is Custom Congestion Control a Bad Idea for Circumvention Tools?** In Free and Open Communications on the Internet 2025 (FOCI'25) [200] This work was made possible by the National Science Foundation under Grant Numbers CNS-2237552 and CNS-2141512.

5.1 Introduction

The global escalation in censorship efforts has heightened the demand for circumvention tools. In a typical scenario, a user in a censored region connects to a proxy server located outside the jurisdiction to access application servers that are otherwise blocked. These proxy connections face several unique challenges inherent to the adversarial nature of censorship circumvention, with one of the most extensively studied being the need for proxies to obfuscate their traffic to avoid detection and blocking [182]. Over the past decades, proxy developers have continuously adopted various obfuscation strategies, whereas censors deployed increasingly sophisticated fingerprinting techniques to detect and disrupt proxy activities [82, 32, 217].

A less examined yet critical issue affecting the usability of proxy tools is their performance under unreliable and lossy network conditions. Proxy connections often need to traverse geographically distant and topologically disparate networks, in particular cross-border or intercontinental links that are characterized by frequent congestion and high packet loss. For example, prior studies have measured that inbound traffic crossing the Chinese border can experience packet loss rates ranging from 10% to 50% [234]. To optimize performance under such network conditions, some proxy tools have adopted custom, non-standard Congestion Control Algorithms (CCAs) designed to aggressively transmit packets in spite of packet loss. Notable examples include Hysteria [112] and TCP-Brutal [114], which maintain higher sending rates compared to conventional CCAs, allowing them to outcompete co-existing flows and secure an (unfairly) disproportionate share of throughput on congested links.

We argue in this chapter that such custom CCA implementations raise concerns for deviating from the design principles of censorship circumvention tools. The essence of effective censorship circumvention lies in obfuscating proxy traffic such that it is indistinguishable from the broader category of traffic valued by censors, thereby leveraging "collateral damage" to achieve some degree of unblockability. Over the years, practitioners in the field

have recognized that to do this well, circumvention tools must adopt standard, ubiquitous protocols like TLS and strive to align their behaviors with those expected from mainstream implementations (e.g.,, browsers). Custom congestion control that heavily diverges from standard behavior, on the other hand, is a clear deviation from this principle: by reacting to packet loss in a manner inconsistent with standard TCP/QUIC behaviors, these implementations not only contribute to network congestion but also generate traffic patterns that diverge from the norm—departing the goal of blending in.

We characterize the custom CCAs implemented by Hysteria and TCP-Brutal by quantifying their aggressive sending behaviors. Using a controlled testbed, we emulate various network conditions and demonstrate that these tools could be reliably differentiated from standard TCP and QUIC implementations across multiple mainstream CCAs, including TCP Cubic, YeAH, Vegas, HTCP and BBR. Furthermore, these distinctions generalize across varying round-trip-times (RTTs) and packet loss rates. To illustrate the feasibility of exploiting these behaviors for proxy detection, we develop a proof-of-concept two-stage classifier and evaluate it on a dataset of synthetic flows generated for a thousand different network conditions. Even with a simple threshold-based approach, the classifier is able to differentiate the custom CCA from standard TCP/QUIC flows with a rare few false positives and negatives.

By analyzing Hysteria and TCP-Brutal as case studies, we showcase examples of proxy tools' fingerprintability stemming from the non-standard behaviors of the custom CCAs that they employ. However, we do not discount the importance of performance for circumvention tools; rather, we argue that such solutions must be sought within the framework of standard protocols and behaviors so that performance optimization does not come at the expense of fingerprintability. In our experiments, we found that BBR performs reasonably well under most network conditions, coming close to the performance levels achieved by the custom CCA, particularly when packet loss remains below 20%. As such, we see no compelling justification for adopting custom and aggresive CCAs in circumvention tools under high

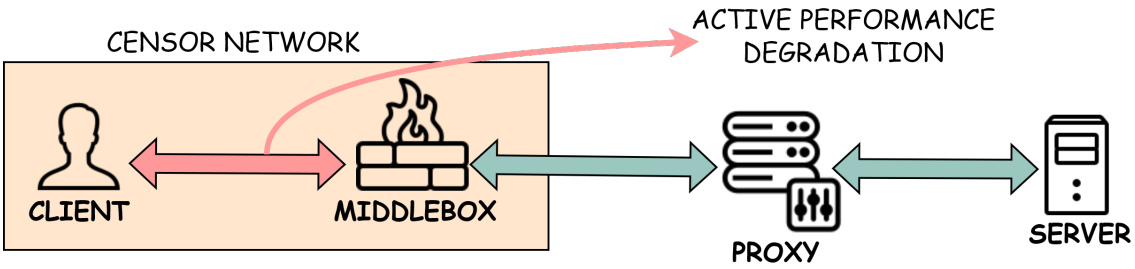


Figure 5.1: Testbed Setup. The middlebox can actively introduce loss or constrain bandwidth of traffic. It can also take advantage of the natural performance degradation on cross-border links. It can observe traffic before and after the middlebox device.

packet-loss conditions – after all, any marginal performance gains are futile if the tools get blocked from fingerprintable traffic patterns.

5.2 Threat Model

We envision an adversary aiming to detect proxy tools that aggressively try to obtain better performance in constrained network environments. The adversary capabilities are based on recent and past documented instances of adversaries intentionally restricting end-user performance [228, 161, 162]. Specifically, we assume an adversary along the network path, akin to an ISP, with the ability to impose restrictions on packet flows by either introducing artificial packet losses or constraining bandwidth. Further, the functioning of the proxy tools is assumed to be public knowledge, and the adversary can based on it to develop heuristics to detect their usage.

Testbed Setup We build our testbed following the threat model. We use four physical machines, all of which are configured in-lab to run Debian instances. We follow the topology as shown in Figure 5.1 and assume that a client machine residing inside a censored networks is connected to a proxy host, traversing through a hop where the censor deploys a middlebox. The client and middlebox are assumed to be within the censor's network boundary with other entities outside the censor's purview. We configure and run appropriate proxy tools under test (and popular CCAs) on the client and proxy machine. Network traffic

was generated through large file transfer from server to client, using a file sufficiently large to ensure sustained connection throughout the experiment duration. The link between the client and middlebox can be configured with arbitrary network conditions such as loss, RTT or bandwidth constraints using the `netem` and `tbf` modules in `tc` [100]. When evaluating standard CCAs in the TCP/QUIC stack, direct connections to the web server were established, bypassing the proxy server. The analysis for detecting the presence of studied proxy tools is conducted on the middlebox machine, which also captures raw PCAPs for reference.

5.3 Behavior Characterization

The focus of our investigation is a class of transports used by obfuscated proxy protocols that are designed to enhance the performance over unreliable and lossy networks. Notable examples of these protocols include Hysteria [112] (QUIC-based) and TCP-Brutal [114]. These tools are promoted for use in specific scenarios such as tunneling cross-border connections, where network congestion leads to packet loss rates typically in the range of 5% to 15%, with peak loss rates as high as 50% [234]. To optimize the proxy's performance under such situations, Hysteria and TCP-Brutal use custom, non-standard Congestion Control Algorithms (CCAs) that aggressively transmit packets at a higher rate and outcompete standard CCAs on congested links.

Before we proceed with quantifying these deviations, we first provide the high-level intuition that guided our study: traditional CCAs – whether loss-based, delay-based, or rate-based – implement a "back off" mechanism in response to congestion signals. This mechanism aims to avoid excessive bandwidth usage and to ensure fairness when network conditions change, such as increased RTTs or reduced bandwidth capacity. On the other hand, the custom CCAs used by Hysteria and TCP-Brutal are designed to sustain high performance even under congestion. Instead of cooperating with competing flows by adapting to congestion, these protocols disregard congestion signals and even attempt to

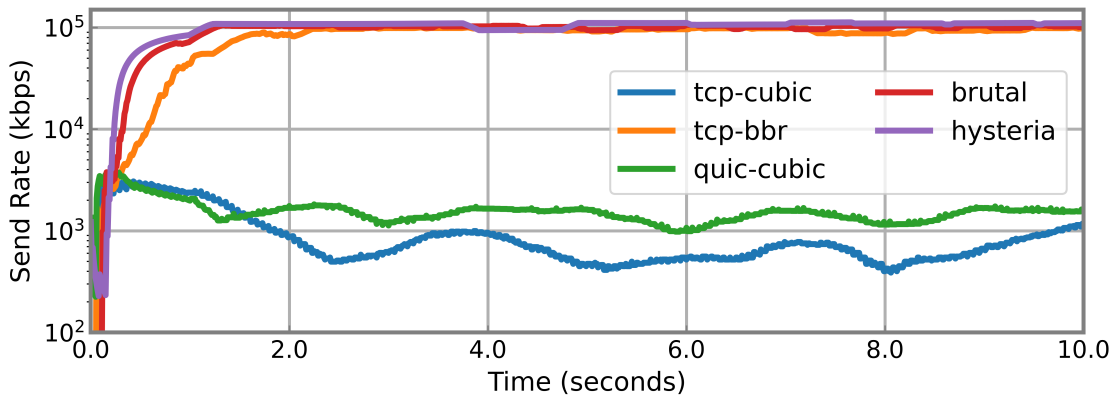


Figure 5.2: Comparing Aggressive Proxy Tools with Traditional CCAs. The link has a bandwidth of 100 Mbps, an RTT of 60ms, and an uniform packet loss rate of 5%.

compensate for packet loss by further *increasing* their sending rate, therefore maintaining the throughput configured by the user-defined "download bandwidth". Such behavior represents a fundamental violation of the congestion control principles defined in both TCP and QUIC [42, 102]. Apart from disrupting the intended fairness of network resource allocation and inducing potential congestion collapse, such behaviors also generate distinctive traffic patterns that fundamentally undermine the objectives of censorship circumvention.

In this section, we empirically demonstrate the non-compliant behaviors of Hysteria and Brutal through a series of experiments. In § 5.3.1, we illustrate how the behaviors of Hysteria and Brutal diverge from those of typical TCP Cubic or QUIC Cubic under a specific set of network conditions. § 5.3.2 extends this analysis by showing consistent deviations across various Hysteria and Brutal configurations, in comparison to several loss-based TCP CCAs, such as Cubic, YeAH, HTCP, and Vegas. In § 5.3.3, we generalize these observations to a broad range of network conditions with varying loss rates and RTTs. Finally, in § 5.3.4, we further distinguish Hysteria and Brutal against rate-based CCAs like BBR.

5.3.1 Aggressive Sending Rates

Figure 5.2 shows the sending rates of five connections: Hysteria, TCP Brutal, QUIC-Cubic, TCP-Cubic, and TCP-BBR, over a link with a bandwidth of 100 Mbps, an RTT of

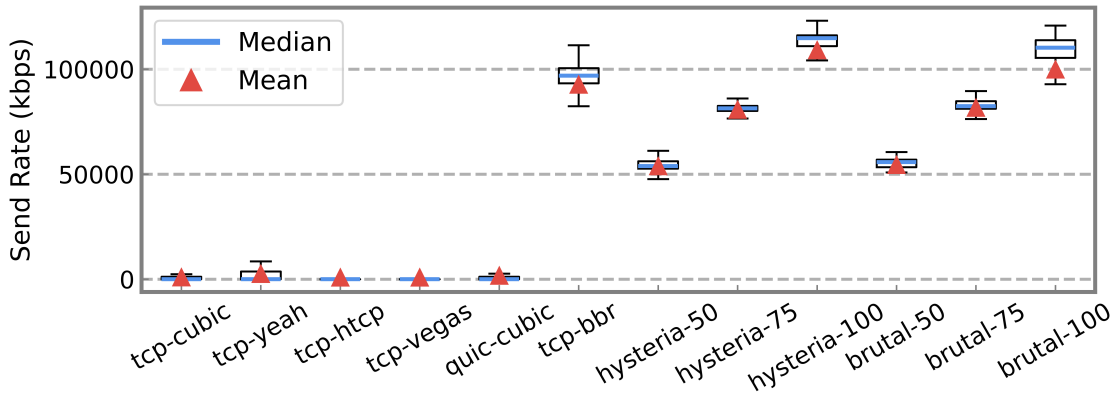


Figure 5.3: Differentiating Hysteria and Brutal from Loss-based CCAs based on Sending Rate. Sending rates over time is captured by box-plots. Hysteria and Brutal are configured at 50Mbps, 75Mbps, and 100Mbps on a link with 100Mbps bandwidth, 60ms RTT, and 5% uniform loss rate.

60 ms, and a random packet loss rate of 5% (uniform loss). For Hysteria and Brutal, the download bandwidth was configured to match the link capacity, as recommended by their protocol specifications [113].

As expected, Cubic, in both TCP and QUIC, being a loss-based CCA, suffers significant performance degradation at a 5% loss rate, leading it to transmit at rates several orders of magnitude lower than those achieved by Hysteria and Brutal. The custom CCA adopt a fixed-sending-rate model, effectively utilizing the available bandwidth regardless of packet loss. This difference lends us the intuition that in networks with non-negligible packet loss, such as cross-border or transcontinental links, Hysteria and Brutal can be easily distinguished from CCAs that interpret packet loss as a congestion signal and consequently back off from sending.

It is worth noting, however, that TCP-BBR [2], which adopts a model/rate-based CCA that does not directly respond to packet loss, exhibits sending behavior similar to that of Hysteria and Brutal, which necessitates further differentiation.

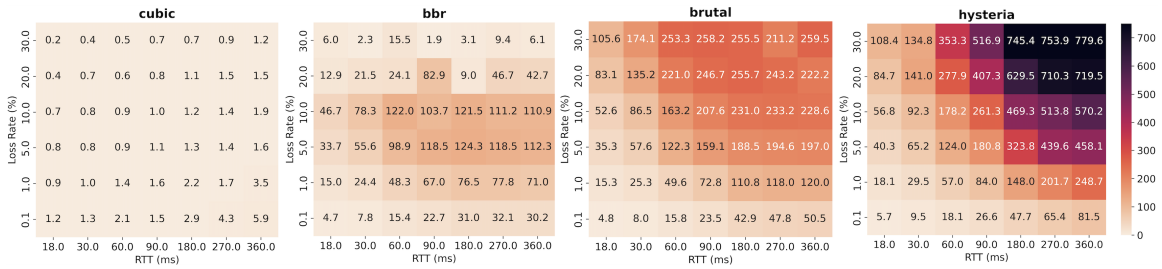


Figure 5.4: Differentiating Hysteria and Brutal across Varying Network Conditions. We quantify the average sending rate ratio of TCP-Cubic, TCP-BBR, Brutal, and Hysteria, respectively, with the estimated sending rate using the Mathis equation.

5.3.2 Differentiation from loss-reactive CCAs

Figure 5.3 compares the sending rates of Brutal and Hysteria with those of other standard CCAs, including TCP-YeAH, HTCP, and Vegas, in addition to Cubic and BBR. Additionally, Brutal and Hysteria are configured with different settings, including when the "Download Bandwidth" is set below the actual link capacity. In these configurations, they effectively acts as a self-imposed rate limiter. Despite this, both custom tools, which rely on a fixed, pre-configured sending rate, remain reliably distinguishable from loss-based CCAs.

BBR, however, is an exception among the standard CCAs. As a rate-based algorithm, BBR aims to maximize available network capacity rather than react to packet loss as a congestion signal. Surprisingly, BBR outperforms Hysteria and Brutal when these tools are configured with a lower-than-actual bandwidth. It's worth noting that Hysteria has built-in support for BBR as an alternative to its custom CCA, presumably due to BBR's ability in sustaining stable throughput even in moderately lossy environments.

5.3.3 Generalizing across Network Conditions

We demonstrate that the observed differentiating behaviors between the custom and standard CCAs generalize across varying network conditions – specifically network reliability (loss rate) and latency (RTT). We conducted experiments using our testbed to empirically measure the performance of different CCAs under varying loss-RTT combinations, using

[2]We use the BBR version included in Linux, which is version 1.

the Mathis Equation [123]:

$$Throughput = \frac{MSS}{RTT\sqrt{P_{loss}}} \tag{5.1}$$

as a shared baseline for comparison [3]. Figure 5.4 shows the results as a heatmap, with the value in each entry corresponding to the ratio of the specific CCA's performance relative to Mathis estimation.

We found that Mathis Equation provides an accurate estimation for Cubic under most of the tested network conditions, whereas Brutal and Hysteria consistently send at multiples of the estimated rates, especially under high loss-rate and/or high RTT scenarios. This is consistent with our understanding that the absence of slowing down under packet loss clearly distinguishes the two custom tools from CCAs that react to packet loss. However, BBR also shows significantly higher sending rates than those estimated when packet loss and RTT increase. As shown in Figure D.1 in the Appendix, under most network conditions, Brutal and Hysteria have a sending rate similar to that of BBR (except when the loss rate reaches above 20%). Thus, a different approach is needed to differentiate CCAs like BBR that do not directly react to packet loss.

5.3.4 Active Throttling

To differentiate Brutal & Hysteria from TCP BBR, a potential censor could actively impose throttling mid-connection to induce BBR's response to changes in bandwidth and contrast it with the behavior of the custom tools, which lack a similar adaptive mechanism. Specifically, in our experiments, we reduced the bandwidth by half (e.g.,, 200 Mbps $\rightarrow$ 100 Mbps, 50 Mbps $\rightarrow$ 25 Mbps) in the middle of each connection using a `token bucket filter` to simulate throttling. As illustrated in Figure 5.5, BBR is designed to respond to changing network conditions by adjusting its sending rate. Upon detecting reduced bandwidth, BBR begins probing for available capacity, eventually converging to the lowered

[3] We use a standard $MSS = 1460 bytes$

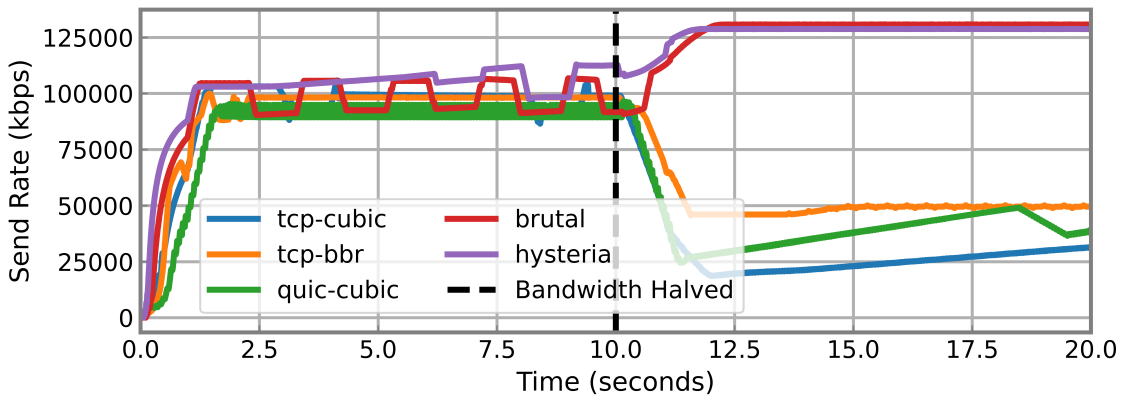


Figure 5.5: Differentiating Hysteria and Brutal with Active Throttling. The link has a bandwidth of 100Mbps, which is reduced to 50Mbps mid-connection, an RTT of 60ms, and no base loss rate.

rate after a few probing cycles. In contrast, Brutal and Hysteria interpret the reduced bandwidth as an increase in packet loss and thus attempt to compensate by increasing their sending rate even further.

5.4 Evaluation

We present a simple, threshold-based classifier to demonstrate the feasibility of differentiating the custom CCA among standard TCP and QUIC flows. The proof-of-concept classifier is then evaluated using a dataset of 10,080 traffic flows generated in controlled lab environments across varied network conditions.

5.4.1 Threshold-based Classifiers

The classification consists of two stages that are designed to distinguish the custom CCA from loss-based and rate-based CCAs respectively, as shown in Fig.5.6. In the first stage, we apply a simple threshold to determine if a connection exhibits a back-off response when encountering packet loss. Specifically, a connection is labeled as non-loss-based if its observed average sending rate surpasses the estimated performance by a scaling factor of $s = 5$. The expected performance is determined as $minbtl_bw, s * mathis_eqn$, where btl_bw is the

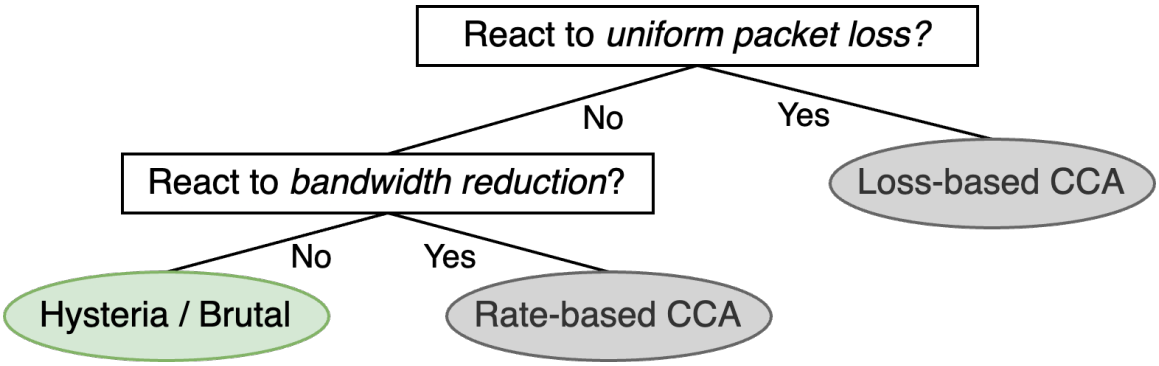


Figure 5.6: Classification Decision Tree. The two-stage classifier distinguishes Brutal from loss-based and rate-based CCAs based on reaction to packet loss and bandwidth reduction mid-connection.

bottleneck bandwidth and *mathis_eqn* is derived from (5.1). Since the Mathis equation considers packet loss rate as the sole limiting factor, when a bottleneck bandwidth exists, loss-based CCAs are expected to operate below the minimum of these two constraints. By identifying flows that exceed this threshold, we isolate non-loss-based CCAs (including Hysteria and TCP-Brutal) from those CCAs that respond to packet loss using only passive measurement data.

The second stage further examines flows classified as non-loss-based to distinguish Brutal from rate-based CCAs like BBR that don't react to packet loss directly through back-off but rather conform to the new bandwidth in case of a bandwidth reduction. Here, we calculate the sending rate in 10ms windows to capture all peak transmissions and determine whether a flow continues to probe above the original bandwidth after the 7s mark, 2 seconds after a 50% bandwidth reduction is introduced to allow BBR to adjust its sending rate within a few bandwidth probing cycles, as described in § 5.3.4. Flows that persist in over-sending are flagged, while those conforming to the new reduced bandwidth are identified as rate-based CCAs.

5.4.2 Results

The classifier was evaluated using traffic flows generated with our controlled testbed involving eight different CCA implementations. We configured various network conditions

Stage	Label/CCA	Brutal	Hysteria	BBR	CUBIC	YeAH	HTCP	Vegas	QUIC
Stage 1: All flows	Loss-based	3	3	97	1260	1260	1260	1260	1260
	Non-loss	1257	1257	1163	0	0	0	0	0
Stage 2: Non-loss flows	Rate-based	0	0	1147	0	0	0	0	0
	Other(custom)	1257	1257	16	0	0	0	0	0

Table 5.1: Evaluation Results. Each tool was measured under 1260 different network condition combinations. The first stage labels flows as either loss-based or non-loss-based. The second stage then classifies non-loss-based flows further into rate-based or custom (e.g.,, Hysteria and TCP Brutal).

20 different persistent RTTs, distributed evenly from 15ms to 300ms, and 21 different packet loss rates, ranging from 0.1% to 20%, under 3 different bandwidth capacity (200 Mbps, 100 Mbps, and 50 Mbps), totaling 10080 experiments. Each experiment lasts 10s, with a 50% reduction of available bandwidth at 5s mark. For Hysteria and Brutal, we configure them according to the protocol specifications to use the link bandwidth limit as the download bandwidth configuration [113].

Table 5.1 presents the classification results. The first stage achieved 100% accuracy in differentiating loss-based CCA flows from non-loss-based ones. Although 97 BBR flows were initially categorized as loss-based due to performance degradation under high packet loss – similar to loss-based CCAs – these did not contribute to false positives. In the second stage, the classifier correctly identified all traffic flows exhibiting custom congestion behaviors while producing only 16 false positives from BBR flows, mostly clustered in high-RTT, low-bandwidth scenarios.

5.4.3 Limitations

This work explores two custom implementations of CCAs – Hysteria and TCP-Brutal. However, there exists other non-standard CCAs that have been or could easily be integrated with circumvention tools [185, 204], each potentially deviating from standard congestion control to varying degrees. Additionally, our analysis is limited to circumvention tools built on QUIC and TCP, both of which offer diverse yet standardized congestion control algorithms. We do not examine pure UDP-based circumvention tools that employ custom

CCA implementations, such as mKCP [185], where the absence of standard UDP congestion control complicates comparisons. Future research should explore the fingerprintability of such alternative CCAs.

This study examines potential fingerprintability of Hysteria and TCP-Brutal under in-lab, simulated network conditions. While we were able to explore a broad range of bandwidth and latency parameters, our analysis does not fully capture the complexities of real-world networks. We encourage future research to investigate the fingerprinting of custom CCAs in real-world traffics.

5.5 Conclusion of Chapter V

The results from this chapter emphasize that the custom congestion control implemented in circumvention proxies, whose behavior diverges heavily from standard ones found in TCP/QUIC stack, fundamentally conflicts with need for blending into mainstream traffic patterns to remain undetected. Through lab-based case studies on Hysteria and TCP-Brutal, we demonstrated how their aggressive sending behaviors can render such proxies identifiable and thus vulnerable to fingerprinting and blocking by censors. We argue that any performance optimizations for circumvention tools must align with standard protocol behaviors to maintain the efficacy of obfuscation efforts.

CHAPTER VI

Conclusion and Future Vision

In this dissertation, I have demonstrated that widely adopted traffic obfuscation approaches remain highly vulnerable to practical and readily deployable fingerprinting attacks. These attacks exploit non-content traffic characteristics, including packet sizes, timing patterns, and distinctive flow- and server-level behaviors. My findings highlight the gaps in current obfuscation methods as well as the urgency of developing more systematic, principled, and robust obfuscation approaches.

Across the four chapters, I proposed several short-term mitigation strategies tailored to the specific fingerprinting attacks detailed in each chapter. These mitigations, such as connection multiplexing and strategic padding, provide immediate, targeted relief and can be readily deployed. However, the long-term efficacy of these defenses remain unclear. Mitigations that harden obfuscation against one attack may inadvertently introduce new, distinct fingerprints. For example, while connection multiplexing effectively disrupts timing-based analysis, our preliminary analysis from Chapter IV suggests it might also produce characteristic traffic patterns distinguishable from standard, non-multiplexed traffic. As such, there is a risk that the fragility of these short-term, targeted mitigations might perpetuate an arms race between traffic obfuscation and fingerprinting: as soon as a promising defense emerges, adversaries adapt their fingerprinting heuristics, in turn pressuring obfuscation developers to develop new responses.

A mid- to long-term priority for the obfuscation research community is to move beyond this arms-race dynamic. Current state-of-the-art research in this field, including the four chapters from this dissertation, typically follow a reactive cycle—defining specific attacks, revealing vulnerabilities in current obfuscation methods, and then proposing countermeasures. This process is effective in surfacing vulnerabilities but tends to evaluate obfuscations only by "whether they fail under a known attack". Even if we deploy the recommended hardening steps from this dissertation, immunity against the fingerprinting attacks described here does not guarantee broad indistinguishability from other yet-undiscovered fingerprints and the evolving capabilities of adversaries.

Current methodology for evaluating obfuscation traps research efforts in a continuous arms race of adaptation and counter-adaptation, limiting progress towards genuinely robust obfuscation. In my view, breaking out from this arms race requires **new evaluation methodologies that do not assume specific fingerprinting attacks or what adversaries are looking for**. In other words, we should be able to decide "this padding/delay strategy provides a high degree of obfuscation" independently of how adversaries build their fingerprinting heuristics. One promising direction involves leveraging principles from information theory, particularly entropy, which quantifies uncertainty about a given entity or process [170, 54, 29]. By applying an information-theoretic lens, we can measure an adversary's uncertainty regarding whether the observed traffic originates from objectionable or benign processes. A higher degree of uncertainty (i.e., entropy) implies better obfuscation since the adversary risks inadvertently affecting legitimate traffic if it attempts to perform targeted interference. This approach allows us to predict an obfuscation protocol's ability to evade fingerprinting based solely on the uncertainty it induces, without assumptions about the specific features or strategies the adversary uses for fingerprinting.

The overarching vision of this research field is to address foundational questions about what it truly means to be indistinguishable from general Internet traffic—the collective stream of flows traversing network intermediaries. An obfuscation protocol that randomly

spreads out all of its observable traffic characteristics, such as packet sizes and timings, may evade targeted fingerprinting but could still appear anomalous compared to typical traffic behavior. Understanding and defining "normal" Internet traffic at scale is therefore essential. My ongoing research explores mathematical representations that capture the stochastic nature of traffic flows, empirically learn these a representation for "normal" from massive volumes of network traffic at global Internet gateways, and finally construct specialized network stacks that emulate such "normal" shape as obfuscation.

APPENDICES

APPENDIX A

Appendix for Fingerprinting Obfuscated Tunneling Traffic at Scale

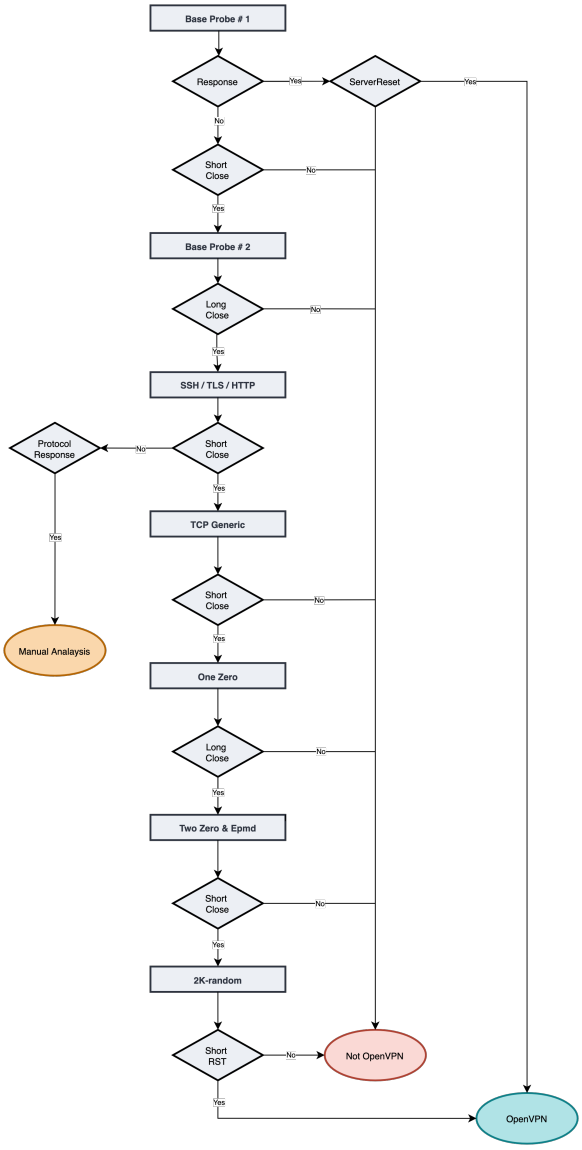


Figure A.1: **Evaluation Process for Active Server Fingerprinting.**

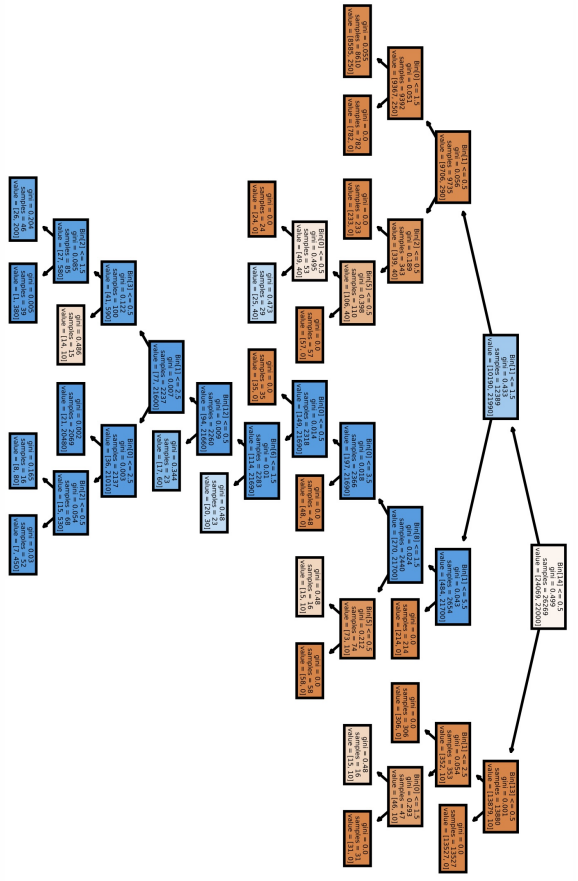


Figure A.2: **Decision tree derived from ISP and VPN datasets.**

VPN Recommendation Sites Used
https://www.security.org/vpn/best/
https://www.techradar.com/vpn/best-vpn
https://www.cnet.com/news/best-vpn/
https://www.tomsguide.com/best-picks/best-vpn
https://www.pcmag.com/picks/the-best-vpn-services
https://thebestvpn.com/
https://www.wired.co.uk/article/best-vpn
https://www.zdnet.com/article/best-vpn/
https://www.cloudwards.net/best-vpn/
https://www.internetsecurity.org/compare/usa
https://www.top10vpn.com/best-vpn-for-usa/v/d/?bsid=c33se1kw011
https://bestvaluevpn.com/usd/best-vpn/?utm_campaign=ggls-en-usa-gen
https://www.nytimes.com/wirecutter/reviews/best-vpn-service/
https://cybernews.com/best-vpn/
https://vpnoverview.com/best-vpn/top-5-best-vpn/
https://www.guru99.com/best-vpn-usa.html
https://www.crazyegg.com/blog/best-vpn-services/
https://www.forbes.com/advisor/business/software/best-vpn/
https://blog.flashrouters.com/vpn/
https://vpnpro.com/best-vpn-services/
https://bestvpn.org/best-vpns-for-the-usa/
https://www.safetydetectives.com/best-vpns (formerly thatoneprivacyguy)
https://www.tomsguide.com/best-picks/best-free-vpn
https://www.top50vpn.com/best-vpn
https://www.top10vpn.com/best-vpn/

Table A.1: **Recommendation Websites Used**

VPN Provider Name	Transport	Variant	Collection Size	Filter	Filter Rate	Opcode/ACK	Prober	Overall Rate
AirVPN	TCP	SSL Tunnel	50	49	0.98	0/49	49	0.98
AirVPN	TCP	SSH Tunnel	50	16	0.32	0/16	16	0.32
AirVPN	TCP	Vanilla	50	48	0.96	48/48	37	0.74
AirVPN	UDP	Vanilla	50	49	0.98	49/49	49	0.98
Astrill	TCP	Proprietary	50	47	0.94	0/47	9	0.18
Astrill	UDP	Proprietory/XOR*	50	50	1	50/6	0	0
Astrill	UDP	Vanilla	50	49	0.98	49/7	4	0.08
BolehVPN	TCP	XOR*	50	50	1	50/50	50	1
BolehVPN	UDP	XOR*	50	49	0.98	49/46	49	0.98
BolehVPN	TCP	Vanilla	50	50	1	50/50	50	1
BolehVPN	UDP	Vanilla	50	50	1	50/49	50	1
CactusVPN	TCP	Obfsproxy	50	0	0	0/0	0	0
CactusVPN	TCP	Vanilla	50	50	1	48/50	50	1
CactusVPN	UDP	Vanilla	50	50	1	50/50	50	1
Cryptostorm	TCP	HTTPS Tunnel	50	48	0.96	0/48	19	0.38
Cryptostorm	TCP	SSH Tunnel	50	27	0.54	0/27	7	0.14
Cryptostorm	TCP	Vanilla	50	48	0.96	48/48	13	0.26
Cryptostorm	UDP	Vanilla	50	49	0.98	49/49	16	0.32
ExpressVPN	TCP/UDP	XOR*	20	20	1	20/4	16	0.8
ExpressVPN	TCP	Vanilla	50	49	0.98	49/49	29	0.58
ExpressVPN	UDP	Vanilla	50	50	1	50/50	32	0.64
Hide.me	TCP	Vanilla	50	49	0.98	49/49	49	0.98
Hide.me	UDP	Vanilla	50	45	0.9	45/45	41	0.82
IPVanish	TCP	XOR*	50	49	0.98	49/49	49	0.98
IPVanish	UDP	XOR*	50	50	1	50/50	50	1
IPVanish	TCP	Vanilla	50	50	1	50/50	50	1
IPVanish	UDP	Vanilla	50	47	0.94	47/47	47	0.94
IVPN	TCP	Obfsproxy	50	0	0	0/0	0	0
IVPN	TCP	Vanilla	50	50	1	50/50	50	1
IVPN	UDP	Vanilla	50	50	1	50/50	50	1
KeepSolid/Unlimited	TCP	Proprietory/XOR*	50	50	1	50/50	50	1
KeepSolid/Unlimited	UDP	Proprietory/XOR*	50	50	1	50/50	50	1
KeepSolid/Unlimited	TCP	Vanilla	50	50	1	50/50	50	1
Mullvad	TCP	Shadowsocks	50	39	0.78	0/39	0	0
Mullvad	TCP	Vanilla	50	47	0.94	47/47	47	0.94
Mullvad	UDP	Vanilla	50	49	0.98	49/49	49	0.98
NordVPN	TCP/UDP	XOR*	50	50	1	50/50	50	1
NordVPN	TCP	Vanilla	50	50	1	48/50	50	1
NordVPN	UDP	Vanilla	50	50	1	50/50	50	1
PerfectPrivacy	TCP	Stunnel	50	47	0.94	0/47	47	0.94
PerfectPrivacy	TCP	SSH	50	39	0.78	0/39	39	0.78
PerfectPrivacy	TCP	Obfsproxy3	50	42	0.84	0/42	42	0.84
PerfectPrivacy	TCP	Vanilla	50	50	1	50/50	50	1
PerfectPrivacy	UDP	Vanilla	50	49	0.98	49/49	49	0.98
PrivateInternetAccess	TCP	Shadowsocks	50	50	1	0/50	50	1
PrivateInternetAccess	TCP	Vanilla	50	50	1	50/50	50	1
PrivateInternetAccess	UDP	Vanilla	50	50	1	50/50	50	1
PrivateVPN	TCP	Shadowsocks	50	50	1	0/50	50	1
PrivateVPN	TCP	Vanilla	50	50	1	49/50	50	1
PrivateVPN	UDP	Vanilla	50	50	1	50/49	50	1
SecureVPN	TCP	SSH Tunnel	50	50	1	0/50	50	1
SecureVPN	TCP	SSL Tunnel	50	49	0.98	0/49	49	0.98
SecureVPN	TCP	Vanilla	50	50	1	50/50	50	1
SecureVPN	UDP	Vanilla	50	49	0.98	48/49	49	0.98
StrongVPN	TCP	XOR	50	50	1	50/50	50	1
StrongVPN	UDP	XOR	50	50	1	50/50	50	1
StrongVPN	TCP	Vanilla	50	49	0.98	49/49	49	0.98
StrongVPN	UDP	Vanilla	50	49	0.98	49/49	49	0.98
SurfShark	TCP	Proprietary/XOR*	50	50	1	50/50	50	1
SurfShark	UDP	Proprietary/XOR*	50	50	1	50/50	45	0.9
TorGuard	TCP	XOR*	50	50	1	49/50	50	1
TorGuard	UDP	XOR*	50	50	1	50/50	50	1
TorGuard	TCP	SSL Tunnel	50	44	0.88	0/44	44	0.88
TorGuard	TCP	Vanilla	50	50	1	50/50	50	1
TorGuard	UDP	Vanilla	50	49	0.98	49/48	49	0.98
TunnelBear	TCP	Obfsproxy	50	0	0	0/0	0	0
TunnelBear	TCP	Vanilla	50	50	1	50/50	50	1
VPN.AC	TCP	XOR	50	50	1	50/50	50	1
VPN.AC	UDP	XOR	50	50	1	50/50	50	1
VPN.AC	TCP	V2Ray	50	0	0	0/0	0	0
VPN.AC	TCP	Vanilla	50	50	1	50/50	50	1
VPN.AC	UDP	Vanilla	50	50	1	50/50	50	1
VPNArea	TCP	Stunnel	50	49	0.98	0/49	49	0.98
VPNArea	TCP	Vanilla	50	50	1	50/50	50	1
VPNArea	UDP	Vanilla	50	50	1	50/49	50	1
VyprVPN	UDP	Proprietory	50	0	0	0/0	0	0
VyprVPN	UDP	Vanilla	50	50	1	50/50	0	0
Windscribe	TCP	TLS Tunnel	50	49	0.98	0/49	25	0.5
Windscribe	TCP	WebSocket Tunnel	50	48	0.96	0/48	24	0.48
Windscribe	TCP	Vanilla	50	50	1	50/50	50	1
Windscribe	UDP	Vanilla	50	50	1	50/50	50	1

Table A.2: **Evaluation results on *Merit*, breakdown by configuration.** Highlighted rows are "obfuscated" configurations. Variants marked with stars mean that the VPN provider does not disclose which obfuscation technique is used and we can only infer the variant type based on packet captures. Note Hide.me claims the *tls-crypt* option alone is enough to "obfuscate entire traffic" [93]. However, this option only encrypts control channel payloads but not the OpenVPN packer headers.

APPENDIX B

Appendix for Protocol-Agnostic Traffic Fingerprinting Attacks

B.1 *XTLS-vision* and *obfs4* Padding Schemes

Algorithm 4 and 5 outlines the padding schemes as implemented by *XTLS-vision* and *obfs4* (iat-mode=0/1). For *XTLS-vision*, TLS handshake packets are padded to a range of 900-1400 bytes, and the first 7 packets transmitted in either direction are padded with a random size of 0-255 bytes. For *obfs4*, both the client and server maintain a padding distribution. For each burst in a flow sequence, a padding size is determined by sampling from the sender's distribution.

Implementation	Protocol	Multiplexing
v2ray/v2fly	{vmess, vless, trojan} over {raw, TLS, websocket}	Recommended
outlineVPN	shadowsocks	Not Supported
shadowTLS	shadowsocks over shadowTLS	Default off
gost	gost	Default off
HTTPT	HTTPT	Not Supported
Cloak	shadowsocks over Cloak	Default on
XTLS-vision	XTLS-vision	Default off
naiveproxy	naiveproxy	Default on

Table B.1: **Multiplexing support for implementations examined in the paper.**◇

Algorithm 4 *XTLS-vision* Padding Scheme

Input: S

$outgoing \leftarrow 0$, $incoming \leftarrow 0$
for i in $0...|S|$ **do**
  $P \leftarrow 0$
  if $i < TLSEstablished$ **then**
    if $|Si| < 900$ **then**
      $P = randint0, 500\ 900 - |Si|$
    else
      $P = randint0, 255$
    end if
  else if $Si > 0$ & $outgoing < 7$ || $Si < 0$ & $incoming < 7$ **then**
    $P = randint0, 255$
  end if
  $UpdatePacketCountoutgoing, incoming$
  if $P > 0$ **then**
    $Si = signSi|Si|\ P$
  end if
end for

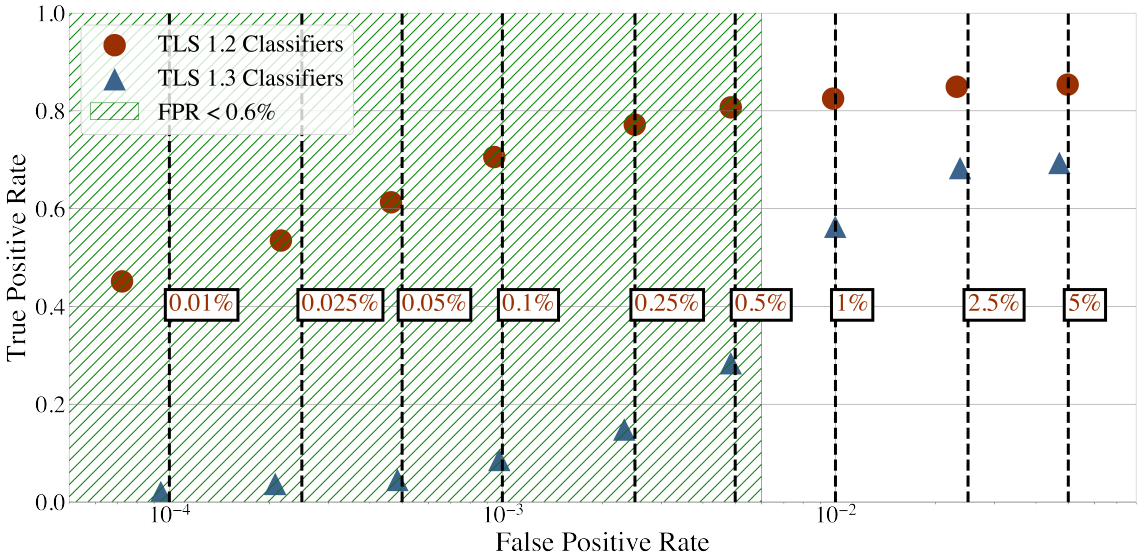


Figure B.1: **Performance of classifiers on plain TLS traffic** TPR and FPR for classifiers with varying detection thresholds $\delta \in 0.1, 0.15, ...1.0, \gamma \in 1, 1.25, ...5$. For each given FPR constraint, we select the parameter set that yield the highest TPR.◇

Algorithm 5 *obfs4* Padding Scheme

Input: S

```
procedure padPacket(size, toPadTo)
    tail ← size%1448
    P ← 0
    if toPadTo ≥ tail then
        P = toPadTo − tail
    else
        P = 1448 − tail toPadTo
    end if
    if P > 21 then
        return size P − 21
    else if P > 0 then
        return size P 1427
    end if
    return size
end procedure
cliDist ← NewDistnewSeed, cliDist ∈ 0, 1448
serverDist ← NewDistnewSeed, serverDist ∈ 0, 1448
S ← extractBurstS
for i in 0...|S| do
    if Si > 0 then
        Si = padPacketSi, cliDist.sample
    else
        Si = −padPacket|Si|, serverDist.sample
    end if
end for
```

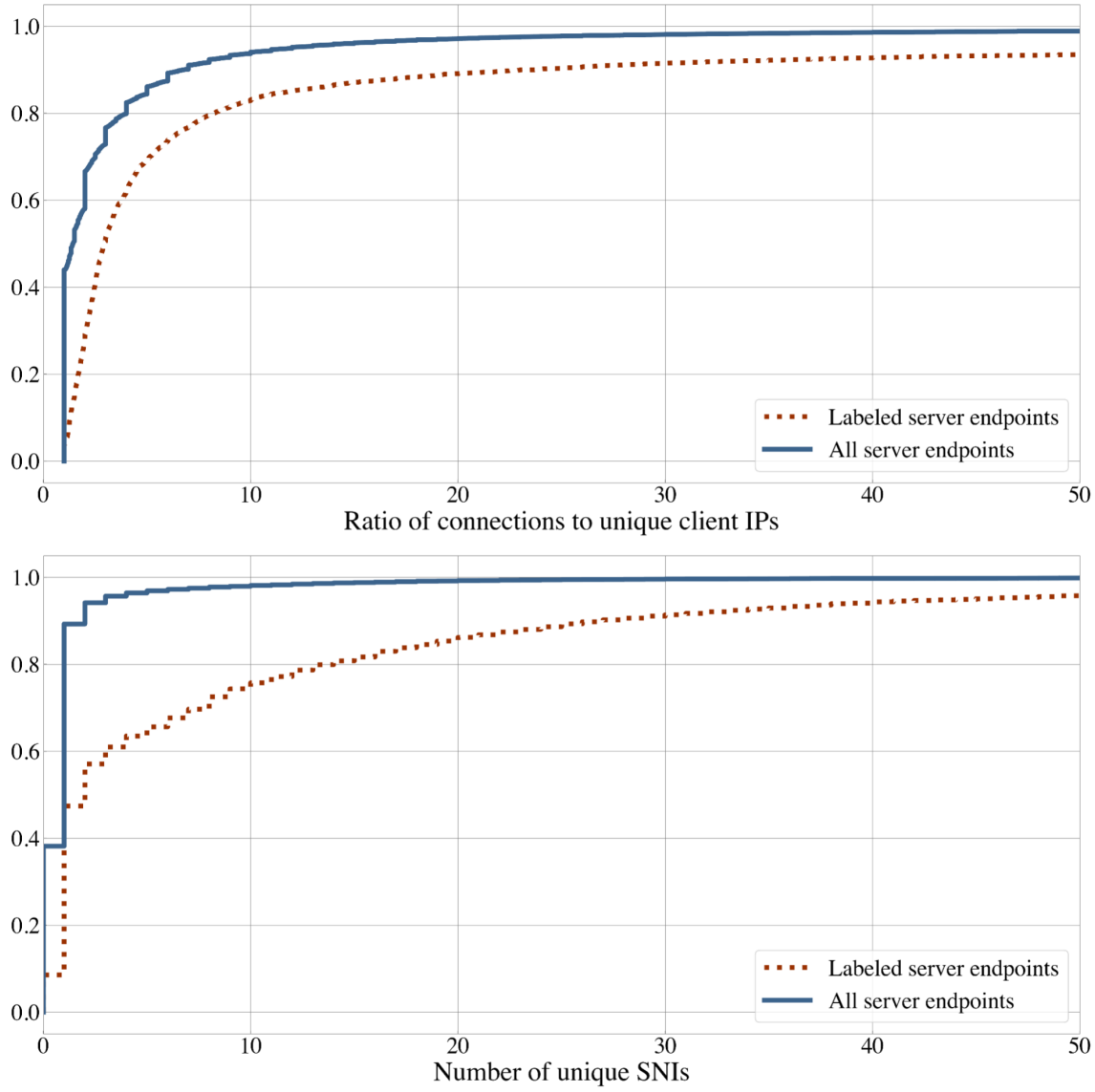


Figure B.2: **Analysis of server endpoints labeled as proxies** Top: CDF for the ratio of number of connections to the number of unique client IPs. Bottom: CDF for the number of unique SNIs.◇

APPENDIX C

Appendix for Timing-Based Fingerprinting against Traffic Obfuscation

C.1 Notation & Symbols

Table C.1 defines the notation and symbols used in the paper.

C.2 Dependence between RTT_{diff} observations

Previously, we assumed that sequential RTT_{diff} observations, conditional on whether a connection is direct or proxied, would independently exhibit similar probabilities of being either inflated or matched. This assumption of independence, however, often does not align with real-world conditions. Factors like transient network conditions, server loads, or congestion can affect consecutive measurements under both hypotheses. We note that in our approach, each correlation window is determined by the number of request-response pairs, rather than a fixed number of packets. This introduces variability in both the timing and the number of packets contained in each observation, potentially reducing temporal correlations (compared to fixed sampling intervals).

Symbol	Definition
Δ_{direct}	difference between client transport- and application-layer RTT in direct connection
Δ_{proxy}	difference between client transport- and application-layer RTT in proxy connection
$TRTT_{AB}$	transport layer round trip time between A and B
$ARTT_{AB}$	application layer round trip time between A and B
RTT_{diff}	for a specific observation in a flow, defined as $ARRT - TRTT$
W	number of request response pairs included in correlation
S	shift/delay in milliseconds applied to outgoing series in cross-correlation
T	threshold over which we consider an RTT_{diff} inflated
Y_i	random variable for the ith observation of a flow, defined as in equation (1)
H_0	Null Hypothesis: the connection is direct
H_1	Alternative Hypothesis: the apparent transport layer is a proxy
θ_0	prior conditional probability for the Null Hypothesis, as defined in equation (2)
θ_1	prior conditional probability for the Alternative Hypothesis, as defined in equation (2)

Table C.1: List of Symbols with Definitions

However, another form of dependence arises specifically under the proxy scenario: if the additional path latency $TRTT_{PW}$ consistently exceeds the threshold T, then an inflated RTT_{diff} is likely not a one-off occurrence but a persistent feature throughout the session. In this case, dependence results to positive correlation among consecutive observations, which invalidates the last equality in (3). The computational complexity and data requirements to reliably estimate the full chain joint probability, however, are not feasible. For this, we define the dependency structure as a direct dependence on the immediately preceding observation (i.e., a first-order Markov chain). Then, the probability priors to be estimated are:

$$\begin{aligned} &PrY_i = 0|Y_{i-1} = 0, H_0 && PrY_i = 1|Y_{i-1} = 0, H_0 \\ &PrY_i = 0|Y_{i-1} = 1, H_0 && PrY_i = 1|Y_{i-1} = 1, H_0 \end{aligned} \tag{C.1}$$

and similarly for H_1. We estimate the transition probabilities in a similar way using empirical data, but instead of independently analyzing each application-layer request-response pairs, we aggregate them by connections they belong to and learn the transition probabilities from consecutive pairs within the same connection. Then, the calculation of the cumulative

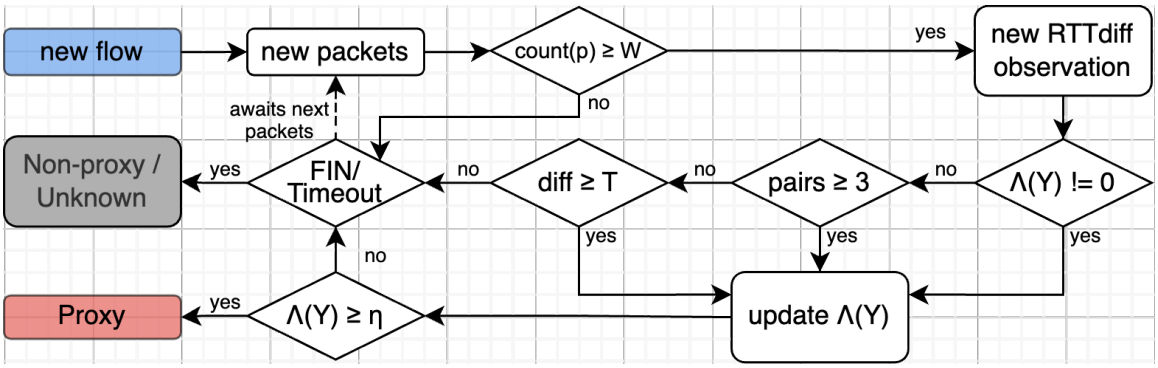


Figure C.1: **Workflow.** The monitoring continues until either $\Lambda Y \geq \eta$ or the connection terminates. ◇

ASN	Percentage	Organization
13335	5.94%	Cloudflare
15169	4.24%	Google
16509	3.20%	Amazon
54113	2.04%	Fastly
16625	1.03%	Akamai
60068	0.97%	cdn77
20940	0.90%	Akamai
32934	0.83%	Facebook
396982	0.64%	Google
62713	0.54%	PubMatic

Table C.2: Top server ASNs for flows exhibiting low transport-delay ($< 5ms$) from proxy locations.◇

likelihood ratio is updated as follows:

$$\Lambda Y = \frac{\Pr Y \mid H_1}{\Pr Y \mid H_0} = \frac{\Pr Y_1 \mid H_1}{\Pr Y_1 \mid H_0} \prod_{n=2}^{N} \frac{\Pr Y_n \mid Y_{n-1}, H_1}{\Pr Y_n \mid Y_{n-1}, H_0} \tag{C.2}$$

C.3 Distribution of Transport Delay from Proxy Locations

Table C.2 shows the top server ASNs for flows exhibiting low ($< 5ms$) transport delay. We note that the prevalence of CDNs mitigates the attack to some extent, as the connectivity they provide reduces the additional delay caused by proxy routing.

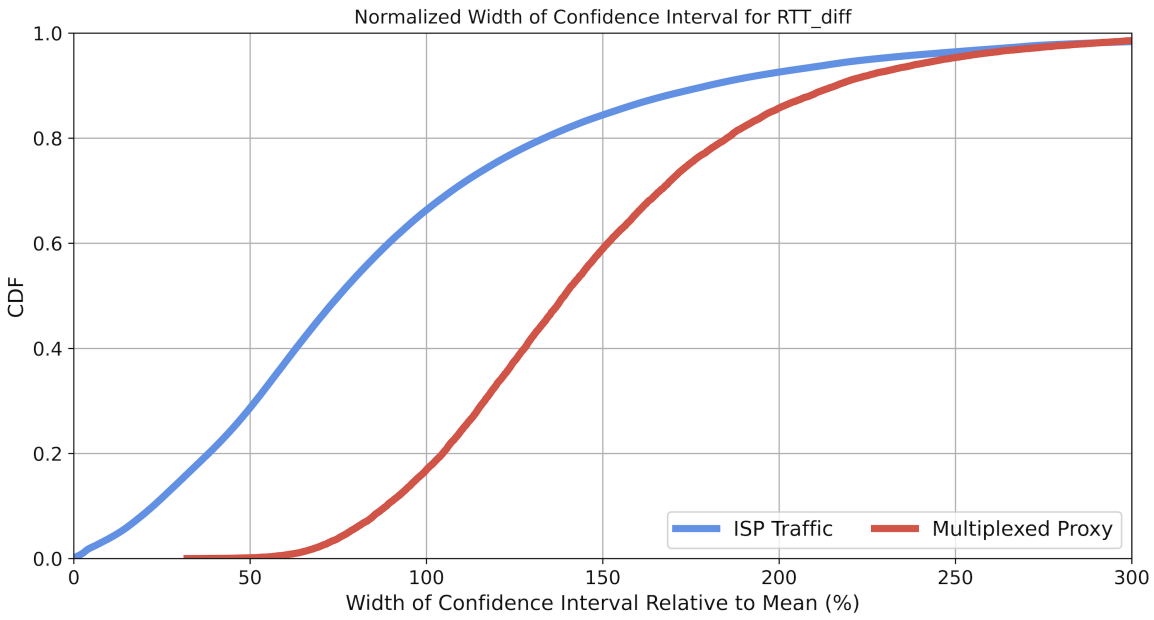


Figure C.2: **Comparing variability in RTT_{diff} for ISP and multiplexed proxy flows.** We use the ratio of the width of the confidence interval (confidence level = .95) to the mean as a normalized measure for variability.◇

C.4 The Effect of Multiplexing on Estimating RTT_{diff}

While connection multiplexing has shown to be able to lower the detection rates, it might also introduce new types of fingerprints. Figure C.2 shows the normalized width of confidence intervals of RTT_{diff} for both the ISP traffic and the multiplexed proxy traffic. We note that the variability introduced by connection multiplexing could itself hold distinguishing power.

C.5 Categorical False Positives with IMAP Traffic

In our evaluation, we found that IMAP traffic could potentially become categorical false positives if the RTT_{diff} fingerprint were to be deployed. To understand why IMAP traffic disproportionately matches with the fingerprint, although privacy constraints prevent further inspection into false positive ISP flows, we were able to generate similar traffic from our *own* devices with the same heavy-hitter SNIs to three IMAP email service providers. We

found that for all three services, actions such as fetching mailbox updates consistently result in delays that substantially exceed those at the transport layer, which explains why IMAP traffic emerged as false positives in our evaluation. See Table C.3 for IMAP examples.

Time (s)	tRTT (ms)	aRTT (ms)	Decrypted & Decoded Contents	Comments
			Provider 1 IMAP	
0.00	–	–	TCP SYN	–
0.03	–	–	TCP SYN/ACK	–
0.04	27	30	Client Hello (SNI=apple.imap.mail.*.com)	–
0.07	–	–	Server Hello	–
0.11	–	–	Response: * OK Welcome!	–
0.14	51	51	Request: 1 CAPABILITY	IMAP Client Capability Request
0.20	–	–	Response: * CAPABILITY	IMAP Server Capability Response
0.43	–	–	Response: 1 OK CAPABILITY completed	Completed Capability Request
0.43	28	378	Request: 2 AUTHENTICATE XOAUTH2	Client Authentication Request
0.81	–	–	Response: 2 OK [COMPRESS=DEFLATE]	Authentication Response
0.91	28	48	Request: 3 ID (”name” ”iPhone Mail” ”version” ...)	Client ID Request
0.96	–	–	Response: * ID (”version” ”1.1.22274” ”host” ...)	Server ID Response
1.20	–	–	Response: 3 OK ID completed	Completed ID Request
1.32	67	297	Request: 4 COMPRESS DEFLATE	Deflate Compression Request
1.61	–	–	Response: 4 OK COMPRESS DEFLATE active	Deflate Compression Response
1.72	28	501	Request: 5 LIST ”” ”*” RETURN (SPECIAL-USE STATUS (UNSEEN))	Mailbox Request
2.22	–	–	Response: * LIST (Archive HasNoChildren) ”/” ”Archive”	Mailbox Response
			.....	.....
			Provider 2 IMAP	
0.00	–	–	TCP SYN	–
0.03	–	–	TCP SYN/ACK	–
0.08	32	32	Client Hello (SNI=imap.*.com)	–
0.11	–	–	Server Hello	–
0.15	–	–	Response: * OK Welcome!	–
0.26	71	71	Request: 1 CAPABILITY	IMAP Client Capability Request
0.33	–	–	Response: * CAPABILITY	IMAP Server Capability Response
0.60	–	–	Response: 1 OK CAPABILITY completed	Completed Capability Request
0.72	73	431	Request: 2 AUTHENTICATE XOAUTH2	Client Authentication Request
1.15	–	–	Response: 2 OK [COMPRESS=DEFLATE]	Authentication Response
1.16	33	67	Request: 3 ID (”name” ”iPhone Mail” ”version” ...)	Client ID Request
1.23	–	–	Response: * ID (”version” ”1.1.22274” ”host” ...)	Server ID Response
1.48	–	–	Response: 3 OK ID completed	Completed ID Request
1.50	71	308	Request: 4 COMPRESS DEFLATE	Deflate Compression Request
1.80	–	–	Response: 4 OK COMPRESS DEFLATE active	Deflate Compression Response
1.82	32	626	Request: 5 LIST ”” ”*” RETURN (SPECIAL-USE STATUS (UNSEEN))	Mailbox Request
2.45	–	–	Response: * LIST (Archive HasNoChildren) ”/” ”Archive”	Mailbox Response
			.....	.....
			Provider 3 IMAP	
0.00	–	–	TCP SYN	–
0.03	–	–	TCP SYN/ACK	–
0.04	20	20	Client Hello (SNI=imap.*.com)	–
0.06	–	–	Server Hello	–
0.11	–	–	Response: * OK * ready for requests from	–
0.15	23	46	Request: 1 CAPABILITY	IMAP Client Capability Request
0.20	–	–	Response: * CAPABILITY	IMAP Server Capability Response
0.21	24	480	Request: 2 AUTHENTICATE XOAUTH2	Client Authentication Request
0.69	–	–	Response: * OK authenticated (Success)	Authentication Response
0.74	19	180	Request: 3 CAPABILITY	IMAP Client Capability Request
0.92	–	–	Response: * CAPABILITY 3 OK Success	IMAP Server Capability Response
0.95	20	182	Request: 4 ID (”name” ”iPhone Mail” ”version” ...)	Client ID Request
1.13	–	–	Response: * ID (”name” ”*” ”vendor” ”*” ...)	Server ID Response
1.15	21	181	Request: 5 COMPRESS DEFLATE	Deflate Compression Request
1.33	–	–	Response: 5 OK Success	Deflate Compression Response
1.35	19	441	Request: 6 LIST ”” ”*” RETURN (SPECIAL-USE STATUS (UNSEEN))	Mailbox Request
1.79	–	–	Response: * LIST (HasNoChildren) ”/” ”Messages”	Mailbox Response
			.....	.....

Figure C.3: Interaction with IMAP servers for providers 1, 2, and 3. For all three services, actions such as fetching mailbox updates consistently result in delays that substantially exceed those at the transport layer. Note that it is hard to know precisely why IMAP leads to such delays without visibility into the functioning of the back-end servers. However, this behavior seems to be associated with only IMAP (see Table 4.3) as no other protocol flows within the ISP traffic qualified to consistently show a significant delay difference.

APPENDIX D

The Discriminative Power of Congestion Control Behaviors

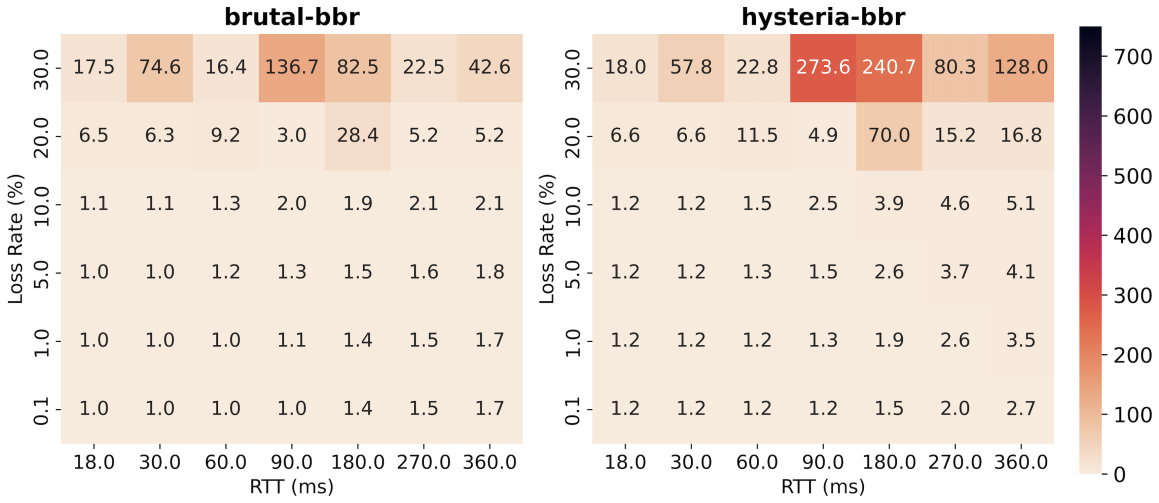


Figure D.1: **Average sending ratio comparison of Brutal and Hysteria with BBR.** Both Hysteria and Brutal are similar to BBR for lower loss rates, whereas they significantly differ at higher loss rates ($> 20\%$).

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Censorship circumvention methods & software — ntc.party. https://ntc.party/c/censorship-circumvention-software/. [Accessed 24-Apr-2023].

[2] Cyberoam firewall blocks meek by TLS signature — groups.google.com. https://groups.google.com/g/traffic-obf/c/BpFSCVgi5rs. [Accessed 05-May-2023].

[3] Deep packet inspection to classify V2Ray traffic · Issue v2ray/discussion — github.com. https://github.com/v2ray/discussion/issues/569. [Accessed 19-Apr-2023].

[4] dnstt; DoH- and DoT-capable DNS tunnel — bamsoftware.com. https://www.bamsoftware.com/software/dnstt/. [Accessed 05-May-2023].

[5] GitHub - cbeuw/Cloak: A censorship circumvention tool to evade detection by authoritarian state adversaries — github.com. https://github.com/cbeuw/Cloak. [Accessed 24-Apr-2023].

[6] GitHub - ginuerzh/gost: GO Simple Tunnel - a simple tunnel written in golang — github.com. https://github.com/ginuerzh/gost. [Accessed 12-May-2023].

[7] GitHub - ihciah/shadow-tls: A proxy to expose real tls handshake to the firewall — github.com. https://github.com/ihciah/shadow-tls. [Accessed 25-May-2023].

[8] GitHub - klzgrad/naiveproxy: Make a fortune quietly — github.com. https://github.com/klzgrad/naiveproxy/. [Accessed 24-Apr-2023].

[9] GitHub - net4people/bbs: Forum for discussing Internet censorship circumvention — github.com. https://github.com/net4people/bbs. [Accessed 24-Apr-2023].

[10] Goodbye, Alexa. Hello, Cloudflare Radar Domain Rankings — blog.cloudflare.com. https://blog.cloudflare.com/radar-domain-rankings/. [Accessed 24-Apr-2023].

[11] index/prefixing - outlinevpn — reddit.com. https://www.reddit.com/r/outlinevpn/wiki/index/prefixing/. [Accessed 05-May-2023].

[12] Iran blocks social media, app stores and encrypted DNS amid Mahsa Amini protests — ooni.org. https://ooni.org/post/2022-iran-blocks-social-media-mahsa-amini-protests/. [Accessed 09-May-2023].

[13] Large scale blocking of TLS-based censorship circumvention tools in China · Issue-129 · net4people/bbs — github.com. https://github.com/net4people/bbs/issues/129. [Accessed 08-May-2023].

[14] Missing Links: A comparison of search censorship in China - The Citizen Lab — citizenlab.ca. https://citizenlab.ca/2023/04/a-comparison-of-search-censorship-in-china/. [Accessed 09-May-2023].

[15] RFC 5077: Transport Layer Security (TLS) Session Resumption without Server-Side State — rfc-editor.org. https://www.rfc-editor.org/rfc/rfc5077. [Accessed 23-Apr-2023].

[16] RFC 7918: Transport Layer Security (TLS) False Start — rfc-editor.org. https://www.rfc-editor.org/rfc/rfc7918. [Accessed 23-Apr-2023].

[17] RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3 — rfc-editor.org. https://www.rfc-editor.org/rfc/rfc8446#section-4.2.1. [Accessed 20-Apr-2023].

[18] Selenium — selenium.dev. https://www.selenium.dev/. [Accessed 24-Apr-2023].

[19] Shadowsocks | A fast tunnel proxy that helps you bypass firewalls. — shadowsocks.org. https://shadowsocks.org/. [Accessed 24-Apr-2023].

[20] Snowflake — snowflake.torproject.org. https://snowflake.torproject.org/. [Accessed 06-May-2023].

[21] Summary on recently discovered v2ray weaknesses. https://gfw.report/blog/v2ray$_{w}$eaknesses/en/. [Accessed 04-May-2023].

[22] TLSHistory · Wiki · The Tor Project / Organization · GitLab — gitlab.torproject.org. https://gitlab.torproject.org/tpo/team/-/wikis/projects/Tor/TLSHistory. [Accessed 05-Jun-2023].

[23] Tor partially blocked in China | Tor Project — blog.torproject.org. https://blog.torproject.org/tor-partially-blocked-china/. [Accessed 04-May-2023].

[24] Trojan Documentation — trojan-gfw.github.io. https://trojan-gfw.github.io/trojan/. [Accessed 24-Apr-2023].

[25] V2ray traffic identification method based on long short-term memory (lstm) networks · Issue-1898 · v2ray/v2ray-core — github.com. https://github.com/v2ray/v2ray-core/issues/1898. [Accessed 20-Apr-2023].

[26] VMess protocol | V2Fly.org — v2fly.org. https://www.v2fly.org/en$_{U}$S/developer/protocols/vmess.html. [Accessed 24-Apr-2023].

[27] We Chat, They Watch: How International Users Unwittingly Build up WeChat's Chinese Censorship Apparatus - The Citizen Lab — citizenlab.ca. https://citizenlab.ca/2020/05/we-chat-they-watch/. [Accessed 04-May-2023].

[28] Accurate TLS Fingerprinting using Destination Context and Knowledge Bases, author=Blake Anderson and David A. McGrew. *ArXiv*, abs/2009.01939, 2020.

[29] Christoph Adami. What is information? *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2063):1–14, 2016.

[30] Alice, Bob, Carol, Jan Beznazwy, and Amir Houmansadr. How China Detects and Blocks Shadowsocks. In *ACM Internet Measurement Conference (IMC)*, 2020.

[31] Alice, Bob, Carol, Jan Beznazwy, and Amir Houmansadr. How China detects and blocks Shadowsocks. In *Internet Measurement Conference*. ACM, 2020.

[32] Alice, Bob, Carol, Jan Beznazwy, and Amir Houmansadr. How China detects and blocks Shadowsocks. In *Internet Measurement Conference*. ACM, 2020.

[33] Blake Anderson, Andrew Chi, Scott Dunlop, and David McGrew. Limitless HTTP in an HTTPS World: Inferring the Semantics of the HTTPS Protocol without Decryption, 2018.

[34] Collin Anderson. Dimming the Internet: Detecting throttling as a mechanism of censorship in Iran. Technical report, University of Pennsylvania, 2013.

[35] Simurgh Aryan, Homa Aryan, and J. Alex Halderman. Internet censorship in Iran: A first look. In *Free and Open Communications on the Internet*. USENIX, 2013.

[36] Stealth VPN - Astrill VPN. https://www.astrill.com/features/vpn-protocols/stealth-vpn.

[37] Sikha Bagui, Xingang Fang, Ezhil Kalaimannan, Subhash C. Bagui, and Joseph Sheehan. Comparison of machine-learning algorithms for classification of vpn network traffic flow using time-related features. In *Journal of Cyber Security Technology*, 2017.

[38] Diogo Barradas, Nuno Santos, and Luís Rodrigues. Effective detection of multimedia protocol tunneling using machine learning. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 169–185, Baltimore, MD, August 2018. USENIX Association.

[39] Laurent Bernaille, Renata Teixeira, Ismael Akodjenou, Augustin Soule, and Kavé Salamatian. Traffic classification on the fly. In *Computer Communication Review, Association for Computing Machinery*, 2006.

[40] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early application identification. In *Proceedings of the 2006 ACM CoNEXT conference*, pages 1–12, 2006.

[41] Marc Bevand. My Experience With the Great Firewall of China — blog.zorinaq.com. https://blog.zorinaq.com/my-experience-with-the-great-firewall-of-china/, 2016. [Accessed 08-May-2023].

[42] Ethan Blanton, Dr. Vern Paxson, and Mark Allman. TCP Congestion Control. RFC 5681, September 2009.

[43] Kevin Bock, George Hughey, Xiao Qiang, and Dave Levin. Geneva: Evolving censorship evasion strategies. In *Computer and Communications Security*. ACM, 2019.

[44] Kevin Bock, Gabriel Naval, Kyle Reese, and Dave Levin. Even censors have a backup: Examining China's double HTTPS censorship middleboxes. In *Free and Open Communications on the Internet*. ACM, 2021.

[45] BolehVPN Traffic Obfuscation Keeps You out of Trouble. https://www.vpnmentor.com/blog/bolehvpn-traffic-obfuscation-keeps-you-out-of-trouble/.

[46] Robert T. Braden. RFC 1122: Requirements for Internet Hosts - Communication Layers — datatracker.ietf.org. https://datatracker.ietf.org/doc/html/rfc1122, 1989.

[47] Chad Brubaker, Amir Houmansadr, and Vitaly Shmatikov. CloudTransport: Using cloud storage for censorship-resistant networking. In *Privacy Enhancing Technologies Symposium*. Springer, 2014.

[48] Internet Storm Center. Explicit Trusted Proxy in HTTP/2.0 or...not so much - SANS Internet Storm Center — isc.sans.edu. https://isc.sans.edu/diary/ExplicitTrustedProxyinHTTP2.0or...notsomuch/17708, 2014.

[49] Zimo Chai, Amirhossein Ghafari, and Amir Houmansadr. On the importance of encrypted-SNI (ESNI) to censorship circumvention. In *Free and Open Communications on the Internet*. USENIX, 2019.

[50] Junqiang Chen, Guang Cheng, and Hantao Mei. F-accumul: A protocol fingerprint and accumulative payload length sample-based tor-snowflake traffic-identifying framework. *Applied Sciences*, 13(1):622, 2023.

[51] Yihua Chen, Eric Garcia, Maya Gupta, Ali Rahimi, and Luca Cazzanti. Similarity-based classification: Concepts and algorithms. *The Journal of Machine Learning Research*, 10:747–776, 03 2009.

[52] David D. Clark. RFC 813: Window and Acknowledgement Strategy in TCP — datatracker.ietf.org. https://datatracker.ietf.org/doc/html/rfc813, 1982.

[53] Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. Ignoring the Great Firewall of China. In *Privacy Enhancing Technologies*, pages 20–35. Springer, 2006.

[54] Thomas M. Cover and Joy A. Thomas. Entropy, Relative Entropy, and Mutual Information. In *Elements of Information Theory*, pages 13–55. John Wiley & Sons, Ltd., 2nd edition, 2006.

[55] Jedidiah R. Crandall, Daniel Zinn, Michael Byrd, Earl Barr, and Rich East. ConceptDoppler: A weather tracker for Internet censorship. In *Computer and Communications Security*, pages 352–365. ACM, 2007.

[56] E.F. Crist and J.J. Keijser. *Mastering OpenVPN*. Packt Publishing, 2015.

[57] Cryptostorm - Port Stripping v2. https://cryptostorm.is/blog/port-striping-v2.

[58] Wladimir De la Cadena, Asya Mitseva, Jens Hiller, Jan Pennekamp, Sebastian Reuter, Julian Filter, Thomas Engel, Klaus Wehrle, and Andriy Panchenko. TrafficSliver: Fighting Website Fingerprinting Attacks with Traffic Splitting. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, New York, NY, USA, 2020. Association for Computing Machinery.

[59] Ziye Deng, Zihan Liu, Zhouguo Chen, and Yubin Guo. The random forest based detection of shadowsock's traffic. In *Intelligent Human-Machine Systems and Cybernetics*. IEEE, 2017.

[60] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *13th USENIX Security Symposium (USENIX Security 04)*.

[61] Arun Dunna, Ciarán O'Brien, and Phillipa Gill. Analyzing china's blocking of unpublished tor bridges. In *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)*.

[62] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A search engine backed by Internet-wide scanning. In *22nd ACM Conference on Computer and Communications Security*, 2015.

[63] Maurizio Dusi, Manuel Crotti, and L. Salgarelli. Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting. *Computer Networks*, pages 81–97, 01 2009.

[64] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Protocol misidentification made easy with Format-Transforming Encryption. In *Computer and Communications Security*. ACM, 2013.

[65] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *Proceedings of the 2015 Internet Measurement Conference*.

[66] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. Examining how the Great Firewall discovers hidden circumvention servers. In *Internet Measurement Conference*. ACM, 2015.

[67] Roya Ensafi, Jeffrey Knockel, Geoffrey Alexander, and Jedidiah R. Crandall. Detecting intentional packet drops on the Internet via TCP/IP side channels. In *Passive and Active Measurement Conference*. Springer, 2014.

[68] Roya Ensafi, Philipp Winter, Abdullah Mueen, and Jedidiah Crandall. Analyzing the great firewall of china over space and time. *Proceedings on Privacy Enhancing Technologies*, 2015.

[69] Roya Ensafi, Philipp Winter, Abdullah Mueen, and Jedidiah R. Crandall. Analyzing the Great Firewall of China over space and time. *Privacy Enhancing Technologies*, 2015(1), 2015.

[70] Saman Feghhi and Douglas J Leith. A web traffic analysis attack using only timing information. *IEEE Transactions on Information Forensics and Security*, 11(8):1747–1759, 2016.

[71] Ellis Fenske and Aaron Johnson. Security notions for fully encrypted protocols. In *Free and Open Communications on the Internet*, 2023.

[72] Ellis Fenske and Aaron Johnson. Security notions for fully encrypted protocols. In *Free and Open Communications on the Internet*, 2023.

[73] David Fifield. GoAgent Security Advisory — bamsoftware.com. https://www.bamsoftware.com/sec/goagent-advisory.html, 2014.

[74] David Fifield. *Threat modeling and circumvention of Internet censorship*. University of California, Berkeley, 2017.

[75] David Fifield. [tor-dev] The limits of timing obfuscation in obfs4 — lists.torproject.org. https://lists.torproject.org/pipermail/tor-dev/2017-June/012310.html, 2017.

[76] David Fifield. Turbo Tunnel, a good way to design censorship circumvention protocols. In *Free and Open Communications on the Internet*. USENIX, 2020.

[77] David Fifield. Comments on certain past cryptographic flaws affecting fully encrypted censorship circumvention protocols. Cryptology ePrint Archive, Paper 2023/1362, 2023.

[78] David Fifield and Mia Gil Epner. Fingerprintability of webrtc, 2016.

[79] David Fifield and Lynn Tsai. Censors' delay in blocking circumvention proxies. In *Free and Open Communications on the Internet*. USENIX, 2016.

[80] Sergey Frolov, Jack Wampler, and Eric Wustrow. Detecting Probe-resistant Proxies. In *Network and Distributed System Security*, 2020.

[81] Sergey Frolov and Eric Wustrow. The Use of TLS in Censorship Circumvention. In *Network and Distributed System Security*. The Internet Society, 2019.

[82] Sergey Frolov and Eric Wustrow. The use of TLS in censorship circumvention. In *Network and Distributed System Security*. The Internet Society, 2019.

[83] Sergey Frolov and Eric Wustrow. HTTPT: A Probe-Resistant proxy. In *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*. USENIX Association, August 2020.

[84] Peipei Fu, Chang Liu, Qingya Yang, Zhenzhen Li, Gaopeng Gou, Gang Xiong, and Zhen Li. A NetFlow Sequence Attention Network for Virtual Private Network Traffic Detection. In *International Conference on Web Information Systems Engineering*.

[85] Ping Gao, Guangsong Li, Yanan Shi, and Yang Wang. VPN Traffic Classification Based on Payload Length Sequence. In *2020 International Conference on Networking and Network Applications (NaNA)*.

[86] Thiago Garrett, Ligia E Setenareski, Leticia M Peres, Luis CE Bona, and Elias P Duarte. Monitoring network neutrality: A survey on traffic differentiation detection. *IEEE Communications Surveys & Tutorials*, 2018.

[87] John Geddes, Max Schuchard, and Nicholas Hopper. Cover your ACKs: Pitfalls of covert channel censorship circumvention. In *Computer and Communications Security*. ACM, 2013.

[88] gfw report. Sharing a modified Shadowsocks as well as our thoughts on the cat-and-mouse game · Issue-136 · net4people/bbs — github.com. https://github.com/net4people/bbs/issues/136, 2022.

[89] Gerard Drapper Gil, Arash Habibi Lashkari, Mohammad Mamun, and Ali A. Ghorbani. Characterization of Encrypted and VPN Traffic Using Time-Related Features. In *the 2nd International Conference on Information Systems Security and Privacy(ICISSP)*, 2016.

[90] Wai Phyo Myint Golda Benjamin. Access Now — accessnow.org: Myanmar's dangerous vpn ban. https://www.accessnow.org/myanmar-vpn-ban/, 2024.

[91] Google. Overview of CrUX: Chrome UX Report. Chrome for Developers — developer.chrome.com. https://developer.chrome.com/docs/crux, 2021.

[92] Michelina Hanlon, Gerry Wan, Anna Ascheman, and Zakir Durumeric. Detecting VPN traffic through encapsulated TCP behavior. In *Free and Open Communications on the Internet*, 2024.

[93] Hide.me: Security Hardened OpenVPN Config with Traffic Obfuscation. https://hide.me/en/blog/security-hardened-openvpn-config-with-traffic-obfuscation/.

[94] Nguyen Phong Hoang, Jakub Dalek, Masashi Crete-Nishihata, Nicolas Christin, Vinod Yegneswaran, Michalis Polychronakis, and Nick Feamster. GFWeb: Measuring the Great Firewall's Web censorship at scale. In *USENIX Security Symposium*. USENIX, 2024.

[95] Nguyen Phong Hoang, Arian Akhavan Niaki, Jakub Dalek, Jeffrey Knockel, Pellaeon Lin, Bill Marczak, Masashi Crete-Nishihata, Phillipa Gill, and Michalis Polychronakis. How great is the Great Firewall? Measuring China's DNS censorship. In *USENIX Security Symposium*. USENIX, 2021.

[96] Hans Hoogstraaten. Evaluating server-side internet proxy detection methods (MSc thesis). 2018.

[97] A. Houmansadr, C. Brubaker, and V. Shmatikov. The Parrot Is Dead: Observing Unobservable Network Communications. In *2013 IEEE S&P*.

[98] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. Non-blind watermarking of network flows. *IEEE/ACM Transactions on Networking*, 22(4):1232–1244, 2013.

[99] Amir Houmansadr, Thomas Riedl, Nikita Borisov, and Andrew Singer. I want my voice to be heard: IP over voice-over-IP for unobservable censorship circumvention. In *Network and Distributed System Security*. The Internet Society, 2013.

[100] Bert Hubert. tc - show / manipulate traffic control settings, 2001. Accessed: 2024-11-04.

[101] Martin Husák, Milan Čermák, Tomáš Jirsík, and Pavel Čeleda. HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting. *EURASIP Journal on Information Security*, 2016:1–14, 2016.

[102] Jana Iyengar and Ian Swett. QUIC Loss Detection and Congestion Control. RFC 9002, May 2021.

[103] Liz Izhikevich, Renata Teixeira, and Zakir Durumeric. LZR: Identifying unexpected internet services. In *30th USENIX Security Symposium (USENIX Security 21)*.

[104] Mohamad Jaber, Roberto G Cascella, and Chadi Barakat. Can we trust the inter-packet time for traffic classification? In *2011 IEEE international conference on communications (ICC)*, pages 1–5. IEEE, 2011.

[105] Jigsaw. Shadowsocks resistance against detection and blocking. https://github.com/Jigsaw-Code/outline-server/blob/master/docs/shadowsocks.md, 2023.

[106] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[107] Jaeyeon Jung, Vern Paxson, Arthur W Berger, and Hari Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pages 211–225. IEEE, 2004.

[108] Divyank Katira, Gurshabad Grover, Kushagra Singh, and Varun Bansal. CensorWatch: On the implementation of online censorship in India. In *Free and Open Communications on the Internet*, 2023.

[109] Ethan Katz-Bassett, John P John, Arvind Krishnamurthy, David Wetherall, Thomas Anderson, and Yatin Chawathe. Towards ip geolocation using delay and topology measurements. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 71–84, 2006.

[110] Hyunchul Kim, Kimberly C Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT conference*, pages 1–12, 2008.

[111] Carmen Kwan, Paul Janiszewski, Shela Qiu, Cathy Wang, and Cecylia Bocovich. Exploring simple detection techniques for DNS-over-HTTPS tunnels. In *Free and Open Communications on the Internet*. ACM, 2021.

[112] Aperture Internet Laboratory. Hysteria: A high-performance proxy and relay tool, 2024. Accessed: 2024-11-04.

[113] Aperture Internet Laboratory. Hysteria: A high-performance proxy and relay tool, 2024. Accessed: 2024-11-04.

[114] Aperture Internet Laboratory. Tcp brutal: A high-performance congestion control algorithm, 2024. Accessed: 2024-11-04.

[115] Fangfan Li, Arian Akhavan Niaki, David Choffnes, Phillipa Gill, and Alan Mislove. A large-scale analysis of deployed traffic differentiation practices. In *Proceedings of the ACM Special Interest Group on Data Communication*. SIGCOMM, 2019.

[116] Fangfan Li, Abbas Razaghpanah, Arash Molavi Kakhki, Arian Akhavan Niaki, David Choffnes, Phillipa Gill, and Alan Mislove. lib·erate, (n): A library for exposing (traffic-classification) rules and avoiding them efficiently. In *Internet Measurement Conference*. ACM, 2017.

[117] Shuai Li, Mike Schliep, and Nick Hopper. Facet: Streaming over videoconferencing for censorship circumvention. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 163–172, 11 2014.

[118] libprotoident: Library for application protocol identification. https://github.com/wanduow/libprotoident.

[119] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: a novel approach for encrypted traffic classification using deep learning. In *Soft Comput 24*, 2019.

[120] Bill Marczak, Nicholas Weaver, Jakub Dalek, Roya Ensafi, David Fifield, Sarah McKune, Arn Rey, John Scott-Railton, Ron Deibert, and Vern Paxson. An analysis of China's "Great Cannon". In *Free and Open Communications on the Internet*. USENIX, 2015.

[121] MASSCAN: Mass IP port scanner. https://github.com/robertdavidgraham/masscan.

[122] Alexander Master and Christina Garman. A Worldwide View of Nation-state Internet Censorship. In *Free and Open Communications on the Internet*, 2023.

[123] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev.*, 27(3):67–82, July 1997.

[124] Merit network. https://www.merit.edu/.

[125] Shane Miller, Kevin Curran, and Tom Lunney. Multilayer Perceptron Neural Network for Detection of Encrypted VPN Network Traffic. In *2018 International Conference On Cyber Situational Awareness, Data Analytics And Assessment (Cyber SA)*.

[126] mmmray. SplitHTTP, Project X — xtls.github.io. https://xtls.github.io/en/config/transports/splithttp.html, year = 2024.

[127] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. SkypeMorph: Protocol obfuscation for Tor bridges. In *Computer and Communications Security*. ACM, 2012.

[128] Arash Molavi Kakhki, Abbas Razaghpanah, Anke Li, Hyungjoon Koo, Rajesh Golani, David Choffnes, Phillipa Gill, and Alan Mislove. Identifying traffic differentiation in mobile networks. In *IMC'15*.

[129] Ricardo Morla. Effect of pipelining and multiplexing in estimating http/2.0 web object sizes, 2017.

[130] MullvadVPN: Intro to Shadowsocks. https://mullvad.net/en/help/intro-shadowsocks/.

[131] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1962–1976, 2018.

[132] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2053–2069, 2017.

[133] nDPI: Open Deep Packet Inspection Library. https://www.ntop.org/products/deep-packet-inspection/ndpi/.

[134] Net4People. OpenVPN and WireGuard blocking reported in multiple ISPs in Russia, 2023-08-07. https://github.com/net4people/bbs/issues/274, 2023.

[135] Net4People. Blocking of fully encrypted protocols (Shadowsocks, VMess) in Russia, targeting HTTPS traffic fingerprints. https://github.com/net4people/bbs/issues/363, 2024.

[136] net4people. Is it possible to implement a man-in-the-middle (MITM) tool to bypass censorship? - Issue-373 - net4people/bbs — github.com. https://github.com/net4people/bbs/issues/373, 2024.

[137] Arian Akhavan Niaki, Shinyoung Cho, Zachary Weinberg, Nguyen Phong Hoang, Abbas Razaghpanah, Nicolas Christin, and Phillipa Gill. ICLab: A global, longitudinal internet censorship measurement platform. In *Symposium on Security & Privacy*. IEEE, 2020.

[138] Nim Programming Language. https://nim-lang.org/.

[139] Obfs3 threat model. https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs3.

[140] Learning more about the GFW's active probing system. https://blog.torproject.org/learning-more-about-gfws-active-probing-system.

[141] The History of OpenVPN. https://openvpn.net/blog/the-history-of-openvpn/.

[142] Question about tls-crypt and port 443 firewall ducking. https://sourceforge.net/p/openvpn/mailman/message/35560747/.

[143] OpenVPN_XORPatch. https://github.com/clayface/openvpn$_x$orpatch.

[144] Yi Pang, Shuyuan Jin, Shicong Li, Jilei Li, and Hao Ren. OpenVPN Traffic Identification Using Traffic Fingerprints and Statistical Characteristics. In *Internation Conference on Trustworthy Computing and Services*, 2012.

[145] Jong Chun Park and Jedidiah R Crandall. Empirical study of a national-scale distributed intrusion detection system: Backbone-level filtering of html responses in china. In *2010 IEEE 30th International Conference on Distributed Computing Systems*, pages 315–326. IEEE, 2010.

[146] Paul Pearce, Roya Ensafi, Frank Li, Nick Feamster, and Vern Paxson. Augur: Internet-wide detection of connectivity disruptions. In *Symposium on Security & Privacy*. IEEE, 2017.

[147] Paul Pearce, Ben Jones, Frank Li, Roya Ensafi, Nick Feamster, Nick Weaver, and Vern Paxson. Global measurement of DNS manipulation. In *USENIX Security Symposium*. USENIX, 2017.

[148] PF_RING ZC (Zero Copy). https://www.ntop.org/products/packet-capture/pf$_r$ing/pf$_r$ing-zc-zero-copy/.

[149] Julien Piet, Dubem Nwoji, and Vern Paxson. GGFAST: Automating Generation of Flexible Network Traffic Classifiers. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 850–866, New York, NY, USA, 2023. Association for Computing Machinery.

[150] Pluggable transports. https://pluggabletransports.info/.

[151] Port shadows via network alchemy. https://breakpointingbad.com/2021/09/08/Port-Shadows-via-Network-Alchemy.html.

[152] Pywinauto. https://github.com/pywinauto/pywinauto.

[153] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. Tik-tok: The utility of packet timing in website fingerprinting attacks. *arXiv preprint arXiv:1902.06421*, 2019.

[154] Ram Sundara Raman, Prerana Shenoy, Katharina Kohls, and Roya Ensafi. Censored Planet: An Internet-wide, longitudinal censorship observatory. In *Computer and Communications Security*. ACM, 2020.

[155] Raymond Rambert, Zachary Weinberg, Diogo Barradas, and Nicolas Christin. Chinese wall or Swiss cheese? keyword filtering in the Great Firewall of China. In *WWW*. ACM, 2021.

[156] Reethika Ramesh, Leonid Evdokimov, Diwen Xue, and Roya Ensafi. VPNalyzer: Systematic Investigation of the VPN Ecosystem. In *Network and Distributed System Security*, 2022.

[157] Reethika Ramesh, Ram Sundara Raman, Matthew Bernhard, Victor Ongkowijaya, Leonid Evdokimov, Anne Edmundson, Steven Sprecher, Muhammad Ikram, and Roya Ensafi. Decentralized Control: A Case Study of Russia. In *Network and Distributed System Security*, 2020.

[158] Reethika Ramesh, Ram Sundara Raman, Apurva Virkud, Alexandra Dirksen, Armin Huremagic, David Fifield, Dirk Rodenburg, Rod Hynes, Doug Madory, and Roya Ensafi. Network Responses to Russia's Invasion of Ukraine in 2022: A Cautionary Tale for Internet Freedom. In *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA, 2023. USENIX Association.

[159] Reethika Ramesh, Phillip Winter, S. Korman, and Roya Ensafi. CalcuLatency: Leveraging Cross-Layer Network Latency Measurements to Detect Proxy-Enabled Abuse. In *33nd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[160] refraction networking. GitHub - refraction-networking/utls: Fork of the Go standard TLS library, providing low-level access to the ClientHello for mimicry purposes. — github.com. https://github.com/refraction-networking/utls, 2021.

[161] OONI Reports. Throttling of news media amid kazakhstan's 2022 presidential election, 2023. Accessed: 2024-11-04.

[162] OONI Reports. Turkey: Throttling and dns blocking of twitter following deadly earthquake, 2024. Accessed: 2024-11-04.

[163] Attention to companies using vpn services in operation. https://rkn.gov.ru/news/rsoc/news73628.htm.

[164] Zach Rosson, Felicia Anthonio, Sage Cheng, Carolyn Tackett, and Alexia Skok. Internet shutdowns in 2022: the KeepItOn Report — accessnow.org. https://www.accessnow.org/internet-shutdowns-2022/, 2022. [Accessed 04-May-2023].

[165] Kimberly Ruth, Aurore Fass, Jonathan Azose, Mark Pearson, Emma Thomas, Caitlin Sadowski, and Zakir Durumeric. A world wide view of browsing the world wide web. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 317–336, 2022.

[166] Kimberly Ruth, Deepak Kumar, Brandon Wang, Luke Valenta, and Zakir Durumeric. Toppling top lists: Evaluating the accuracy of popular website lists. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 374–387, 2022.

[167] Selenium. https://www.selenium.dev/.

[168] Wendy Seltzer. Infrastructures of censorship and lessons from copyright resistance. In *Workshop on Free and Open Communications on the Internet (FOCI)*, 2011.

[169] Shadowsocks. Shadowsocks: A fast tunnel proxy that helps you bypass firewalls. https://shadowsocks.org/, 2023. Accessed: 2023-04-24.

[170] Claude E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[171] Sandra Siby, Marc Juarez, Claudia Diaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. Encrypted dns –> privacy? a traffic analysis perspective, 2019.

[172] Ram Sundara Raman, Leonid Evdokimov, Eric Wurstrow, J Alex Halderman, and Roya Ensafi. Investigating Large Scale HTTPS Interception in Kazakhstan. In *Proceedings of the 2020 ACM Internet Measurement Conference*.

[173] Ram Sundara Raman, Leonid Evdokimov, Eric Wustrow, Alex Halderman, and Roya Ensafi. Investigating Large Scale HTTPS Interception in Kazakhstan. 2020.

[174] Ram Sundara Raman, Adrian Stoll, Jakub Dalek, Reethika Ramesh, Will Scott, and Roya Ensafi. Measuring the Deployment of Network Censorship Filters at Global Scale. In *Network and Distributed System Security*, 2020.

[175] Surfshark camouflage. https://surfshark.com/features.

[176] Hardening OpenVPN Security. https://openvpn.net/community-resources/hardening-openvpn-security/.

[177] William J. Tolley, Beau Kujath, Mohammad Taha Khan, Narseo Vallina-Rodriguez, and Jedidiah R. Crandall. Blind In/On-Path Attacks and Applications to VPNs. In *30th USENIX Security Symposium (USENIX Security 21)*.

[178] Top10VPN: VPN Reviews. https://www.top10vpn.com/.

[179] Tor In China – The Onion Router. http://www.mediafactory.org.au/2015-media6-deepweb/2015/10/01/tor-in-china/.

[180] Stealth VPN Unblock Websites, Firewalls and VPN Blocks. https://torguard.net/stealth-vpn.php.

[181] GFW probes based on Tor's SSL cipher list. https://gitlab.torproject.org/legacy/trac/-/issues/4744.

[182] M. C. Tschantz, S. Afroz, Anonymous, and V. Paxson. SoK: Towards Grounding Censorship Circumvention in Empiricism. In *2016 IEEE Symposium on Security and Privacy (SP)*.

[183] Tunnelblick and openvpn_xorpatch. https://tunnelblick.net/cOpenvpn$_{x}$orpatch.html.

[184] V2Fly. Vmess protocol. https://www.v2fly.org/en$_{U}$S/developer/protocols/vmess.html, 2023. Accessed: 2023-04-24.

[185] V2Fly.org. Kcp(mkcp), 2022. Accessed: 2025-01-31.

[186] Summary on Recently Discovered V2Ray Weaknesses. https://gfw.report/blog/v2ray$_{w}$eaknesses/en/.

[187] Indian Parliamentary Committee Wants To Ban VPN Services In India. https://www.indiatimes.com/technology/news/vpn-ban-indian-govt-vpn-services-in-india-548493.html.

[188] Chinese government orders ISPs to block personal VPN use. https://privateinternetaccess.com/blog/great-firewall-china-chinese-government-orders-isps-block-personal-vpn-use-february-1st/.

[189] China's firewall technology upgrades virtual private network management and control tightening. https://www.rfa.org/mandarin/yataibaodao/cyl-12212012155229.html.

[190] VPN Downloads surge in response to Hong Kong Security Law. https://www.bloomberg.com/news/articles/2020-05-22/vpn-downloads-surge-in-response-to-hong-kong-security-law.

[191] PTA sets deadline for VPN users to register by June 30th. https://privateinternetaccess.com/blog/the-coming-pakistan-vpn-ban-pta-sets-deadline-for-vpn-users-to-register-by-june-30th/.

[192] Rain throttles Internet speeds for customers on VPNs. https://mybroadband.co.za/news/internet/384642-rain-throttles-internet-speeds-for-customers-on-vpns.html.

[193] Biggest VPN Trends for 2020: Possibilities and Dangers. https://openvpn.net/blog/biggest-vpn-trends-for-2020-possibilities-and-dangers/.

[194] How Chameleon Defeats VPN Blocking. https://www.vyprvpn.com/features/chameleon.

[195] Ryan Wails, George Arnold Sullivan, Micah Sherr, and Rob Jansen. On Precisely Detecting Censorship Circumvention in Real-World Networks. In *Network and Distributed System Security*, 2024.

[196] Gaukas Wang, Anonymous, Jackson Sippe, Hai Chi, and Eric Wustrow. Chasing shadows: A security analysis of the ShadowTLS proxy. In *Free and Open Communications on the Internet*, 2023.

[197] Liang Wang, Kevin Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Eric Shrimpton. Seeing through network-protocol obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[198] Liang Wang, Kevin Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Eric Shrimpton. Seeing through network-protocol obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[199] Mona Wang, Anunay Kulshrestha, Liang Wang, and Prateek Mittal. Leveraging strategic connection migration-powered traffic splitting for privacy. *arXiv preprint arXiv:2205.03326*, 2022.

[200] Wanye Wang, Diwen Xue, Piyush Kumar, Ayush Mishra, Anonymous, and Roya Ensafi. Is Custom Congestion Control a Bad Idea for Circumvention Tools? In *Free and Open Communications on the Internet*. Proceedings on Privacy Enhancing Technologies, 2025.

[201] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*.

[202] Zhongjie Wang, Yue Cao, Zhiyun Qian, Chengyu Song, and Srikanth V. Krishnamurthy. Your state is not mine: A closer look at evading stateful Internet censorship. In *Internet Measurement Conference*. ACM, 2017.

[203] Zhongjie Wang, Yue Cao, Zhiyun Qian, Chengyu Song, and Srikanth V. Krishnamurthy. Your state is not mine: A closer look at evading stateful Internet censorship. In *Internet Measurement Conference*. ACM, 2017.

[204] wangyu. Udpspeeder, 2023. Accessed: 2025-01-31.

[205] David Warburton. The 2021 TLS Telemetry Report | F5 Labs — f5.com. https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report. [Accessed 01-May-2023].

[206] Nicholas Weaver, Christian Kreibich, and Vern Paxson. Redirecting DNS for Ads and Profit. In *USENIX Workshop on Free and Open Communicationson the Internet*, 2011.

[207] Allen T Webb and AL Narasima Reddy. Finding proxy users at the service using anomaly detection. In *2016 IEEE Conference on Communications and Network Security (CNS)*. IEEE.

[208] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. StegoTorus: A camouflage proxy for the Tor anonymity system. In *Computer and Communications Security*. ACM, 2012.

[209] Philipp Winter and Stefan Lindskog. How the great firewall of china is blocking tor. In *2nd USENIX Workshop on Free and Open Communications on the Internet (FOCI 12)*, Bellevue, WA, August. USENIX Association.

[210] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is blocking Tor. In *Free and Open Communications on the Internet*. USENIX, 2012.

[211] Philipp Winter, Tobias Pulls, and Juergen Fuss. ScrambleSuit: A polymorphic network protocol to circumvent censorship. In *Workshop on Privacy in the Electronic Society*. ACM, 2013.

[212] WireGuard with obfuscation support. https://github.com/net4people/bbs/issues/88.

[213] WireGuard - Let's talk about obfuscation again. https://lists.zx2c4.com/pipermail/wireguard/2018-September/003289.html.

[214] Charles Wright, Fabian Monrose, and Gerald Masson. On inferring application protocol behaviors in encrypted network traffic. *Journal of Machine Learning Research*, 6:2745–2769, 12 2006.

[215] Charles V. Wright, Lucas Ballard, Fabian Monrose, and Gerald M. Masson. Language identification of encrypted VoIP traffic: Alejandra y roberto or alice and bob? In *16th USENIX Security Symposium (USENIX Security 07)*, Boston, MA, August 2007. USENIX Association.

[216] Mingshi Wu, Jackson Sippe, Danesh Sivakumar, Jack Burg, Peter Anderson, Xiaokang Wang, Kevin Bock, Amir Houmansadr, Dave Levin, and Eric Wustrow. How the great firewall of china detects and blocks fully encrypted traffic. In *32th USENIX Security Symposium (USENIX Security 23)*.

[217] Mingshi Wu, Jackson Sippe, Danesh Sivakumar, Jack Burg, Peter Anderson, Xiaokang Wang, Kevin Bock, Amir Houmansadr, Dave Levin, and Eric Wustrow. How the Great Firewall of China detects and blocks fully encrypted traffic. In *USENIX Security Symposium*. USENIX, 2023.

[218] Eric Wustrow, Colleen M. Swanson, and J. Alex Halderman. Tapdance: End-to-middle anticensorship without flow blocking. In *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association.

[219] XTLS. XTLS Vision, fixes TLS in TLS, to the star and beyond · XTLS/Xray-core. https://github.com/XTLS/Xray-core/discussions/1295, 2022.

[220] XTLS. GitHub - XTLS/REALITY: THE NEXT FUTURE — github.com. https://github.com/XTLS/REALITY, 2023.

[221] XTLS. Xray, Penetrates Everything. Also the best v2ray-core, with XTLS support. Fully compatible configuration. https://github.com/XTLS/Xray-core, 2024.

[222] Xueyang Xu, Z. Morley Mao, and J. Alex Halderman. Internet censorship in China: Where does the filtering occur? In *Passive and Active Measurement Conference*, pages 133–142. Springer, 2011.

[223] Diwen Xue, Michalis Kallitsis, Amir Houmansadr, and Roya Ensafi. Fingerprinting Obfuscated Proxy Traffic with Encapsulated TLS Handshakes. In *33st USENIX Security Symposium (USENIX Security 24)*, PHILADELPHIA, PA, 2024. USENIX Association.

[224] Diwen Xue, Benjamin Mixon-Baca, ValdikSS , Anna Ablove, Beau Kujath, Jedediah R. Crandall, and Roya Ensafi. TSPU: Russia's Decentralized Censorship System. 2022.

[225] Diwen Xue, Benjamin Mixon-Baca, ValdikSS, Anna Ablove, Beau Kujath, Jedidiah R. Crandall, and Roya Ensafi. TSPU: Russia's decentralized censorship system. In *Internet Measurement Conference*. ACM, 2022.

[226] Diwen Xue, Reethika Ramesh, Leonid Evdokimov, Andrey Viktorov, Arham Jain, Eric Wustrow, Simone Basso, and Roya Ensafi. Throttling Twitter: an emerging censorship technique in russia. 2021.

[227] Diwen Xue, Reethika Ramesh, Arham Jain, Michalis Kallitsis, J. Alex Halderman, Jedidiah R. Crandall, and Roya Ensafi. OpenVPN is open to VPN fingerprinting. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, 2022. USENIX Association.

[228] Diwen Xue, Reethika Ramesh, Valdik S S, Leonid Evdokimov, Andrey Viktorov, Arham Jain, Eric Wustrow, Simone Basso, and Roya Ensafi. Throttling twitter: an emerging censorship technique in russia. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC '21, page 435–443, New York, NY, USA, 2021. Association for Computing Machinery.

[229] Diwen Xue, Robert Stanley, Piyush Kumar, and Roya Ensafi. The Discriminative Power of Cross-layer RTTs in Fingerprinting Proxy Traffic. In *Network and Distributed System Security*. The Internet Society, 2025.

[230] Sebastian Zander, Thuy Nguyen, and Grenville Armitage. Self-learning ip traffic classification based on statistical flow characteristics. In *Passive and Active Network Measurement: 6th International Workshop, PAM 2005, Boston, MA, USA, March 31-April 1, 2005. Proceedings 6*, pages 325–328. Springer, 2005.

[231] The Zeek Network Security Monitor. https://zeek.org/.

[232] Qi Zhang, Juanru Li, Yuanyuan Zhang, Hui Wang, and Dawu Gu. Oh-Pwn-VPN! Security Analysis of OpenVPN-Based Android Apps. In *CANS*, 2017.

[233] Wenxuan Zhou, Amir Houmansadr, Matthew Caesar, and Nikita Borisov. SWEET: Serving the web by exploiting email tunnels. In *Hot Topics in Privacy Enhancing Technologies*. Springer, 2013.

[234] Pengxiong Zhu, Keyu Man, Zhongjie Wang, Zhiyun Qian, Roya Ensafi, J. Alex Halderman, and Haixin Duan. Characterizing transnational internet performance and the great bottleneck of china. In *Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '20, page 69–70, New York, NY, USA, 2020. Association for Computing Machinery.

[235] Jonathan Zittrain and Benjamin Edelman. Internet filtering in China. *IEEE Internet Computing*, 2003.