CenKL: Information-Theoretic Visibility into Censorship Circumvention Traffic Patterns

Aaron Ortwein* Armin Huremagic*† Shabbir Hussain* Holly Bergen‡
Joshua J. Daymude‡ Diwen Xue§ Jedidiah R. Crandall‡ Roya Ensafi*
*University of Michigan †Johannes Kepler University Linz
‡Arizona State University §The Chinese University of Hong Kong

***Abstract*—Censorship circumvention has long focused on making the *contents* of the TLS handshake, such as the SNI and cipher suites, appear ordinary. But recently, censors have developed powerful attacks that exploit patterns in the *shape* of circumvention traffic. In response, developers have designed diverse obfuscation techniques to disguise patterns in packet sizes, directions, and timings. However, the effectiveness of these defenses has been unclear.**

In this paper, we present *CenKL*, a system that evaluates how distinguishable obfuscated proxy traffic is from real-world TLS traffic using an approximation of KL divergence. Moreover, *CenKL* localizes the packets contributing most to distinguishability, giving developers diagnostic information to improve their tools *before* censors can block them.

We apply *CenKL* to nine popular obfuscated protocols with tens of millions of users, showing that existing obfuscation techniques offer limited protection against traffic analysis. Worse yet, these protocols are distinguishable as early as the first post-handshake packet. Finally, as a proof of concept, we use *CenKL* to reshape the traffic of two of these protocols to better blend into real-world TLS, reducing detection effectiveness while increasing collateral damage across several classifiers.

1. Introduction

Censors have traditionally targeted circumvention traffic by fingerprinting individual circumvention tools. In recent years, however, censors have developed protocol-agnostic attacks that exploit patterns intrinsic to proxy traffic and therefore enable blocking many protocols at once. In November 2021, China's Great Firewall (GFW) began blocking fully encrypted protocols such as Shadowsocks, VMess, and obfs4 based on the entropy of the first data-carrying packet of a connection [111]. China's GFW and Russia's TSPU blocked many popular proxy protocols in October 2022 and May 2024, respectively, with the mechanism widely speculated to involve inspecting the first few packets of a connection for indications of TLS handshakes encapsulated within an outer tunnel [34], [110]. These events challenged the long-held belief within the anti-censorship community that each new circumvention tool would require separate analysis to block, such that maintaining a diverse ecosystem of these tools could overwhelm censors.

The ability to block obfuscated proxy protocols at scale sparked widespread alarm within the community [34], [110], with some lamenting that "everything is over" [76] for censorship circumvention and that "there is no tool that can resist detection" [77]. This pessimism has been further exacerbated by research showing that proxy traffic exhibits fingerprintable structure across *multiple* dimensions, including packet sizes and timings [112], [113]. In response, circumvention tool developers have attempted to disguise the use of proxies through techniques such as padding packet sizes to obscure payload lengths (*e.g.,* XTLS-Vision [115]), delaying packet transmissions to obscure patterns in their inter-arrival times (*e.g.,* obfs4 [114]), and multiplexing to interleave traffic patterns from multiple application-layer streams within a single tunnel connection (*e.g.,* NaïveProxy [20]).

Unfortunately, the effectiveness of these approaches remains unclear, *i.e.,* how successful are these solutions in practice? Once a new solution is proposed, how can we determine whether it will resist detection? More broadly, how can we proactively evaluate circumvention tools, breaking out of the reactive development cycle in which defenses are implemented only after blocking occurs? Addressing these questions requires meeting three core needs: (1) knowledge of what real-world traffic looks like, as echoed by klzgrad in Net4People BBS [47]; (2) a framework for evaluating how distinguishable circumvention tools are from real-world traffic; and (3) methods for reducing the distinguishability of circumvention tool traffic from real-world traffic, as echoed by wkrp in Net4People BBS [108].

Currently, over 90% of all web traffic is TLS [62], making it the ideal reference for how obfuscated proxy traffic should look. However, as the anti-censorship community has learned the hard way, simply tunneling circumvention traffic through TLS is insufficient. Tor has repeatedly been blocked based on handshake fingerprints, including its distinctive Diffie–Hellman parameters and cipher suites [2], [23], [75]. These events motivated the development of uTLS [30], a library that leverages real-world user traffic to mimic the handshakes of popular TLS implementations and has since been adopted by widely used circumvention tools such as meek [85], Snowflake [86], Lantern [11], Psiphon [69], and Xray [67]. While uTLS has mitigated handshake finger-

printing, protocol-agnostic attacks targeting the shape (*e.g.,* the distribution of timing, direction, and size) of encrypted application data remain possible. Although perfect mimicry is unattainable [42], increasing similarity to real-world TLS traffic makes it more difficult for censors to accurately detect and block circumvention tools. To characterize and mimic the shape of real-world TLS, we first need to model its observable features.

In this work, we present *CenKL*, a system for evaluating how distinguishable obfuscated proxy protocol traffic is from real-world TLS traffic. *CenKL* operates in three core phases. First, it collects TLS traffic from a regional ISP and constructs a model of packet timings, directions, and sizes. *CenKL* addresses the statistical challenges of high-dimensional data by applying modeling assumptions that balance tractability and accuracy given the collected data. Second, it collects obfuscated proxy traffic in a controlled testbed designed to generate traffic at scale while still preserving key characteristics of real-world traffic and network conditions—including visits to popular websites, a mix of TLS versions, and network latency—and constructs a corresponding model. Finally, it compares these models using an approximation of Kullback-Leibler (KL) divergence that we call TLS Divergence, which provides a classifier-agnostic measure of distinguishability. TLS Divergence is interpretable in that it localizes the packets contributing to distinguishability, providing circumvention tool developers with the diagnostic information necessary to improve their tools before they can be blocked by censors.

We applied *CenKL* to nine TLS-based proxy protocols with tens of millions of users and found that *all* of them are distinguishable from real-world TLS. To better understand *why*, we investigated the largest contributions to TLS Divergence and *found several recurring causes:* (1) Obfuscated protocols inherit artifacts from their underlying TLS stacks. Notably, protocols implemented in Go often begin with a 1208-byte packet when tunneling post-quantum TLS, a result of the `ClientHello` exceeding the packet size limit of Go's dynamic record sizing optimization. As post-quantum cryptography gains widespread adoption, this pattern will become more prevalent. (2) Nested protocol handshakes (*e.g.,* VLESS-over-WebSocket) often exhibit near-deterministic patterns in packet sizes and directions. By layering additional protocols within the tunnel, protocol nesting reduces the effective cover population from all TLS traffic to only TLS traffic nested in the same way. (3) Request-response exchanges of nested handshakes can introduce inter-packet delays that are consistently larger than those in normal TLS traffic.

Finally, we develop *ProxyShaper*, a proof-of-concept traffic shaper that uses *CenKL*'s TLS model to shape the sizes and directions, but not the timing, of the first 10 packets of VMess [92] and VLESS [68] connections. We evaluate *ProxyShaper*'s effect on several potentially deployable statistical and ML classifiers and find that it reduces precision and recall while increasing the false positive rate—corresponding to reduced effectiveness in detecting circumvention traffic and increased collateral damage.

By providing circumvention tool developers with an interpretable and classifier-agnostic framework for assessing how closely their traffic resembles real-world TLS, *CenKL* offers a path toward more proactive circumvention tool development. We have open sourced both *CenKL* and *ProxyShaper*. We hope *CenKL* can serve not just as a static benchmark, but as an active instrument for guiding the iterative development of obfuscation protocols.

2. Background and Related Work

2.1. History of Obfuscation

Internet censorship is fundamentally possible because disallowed flows leak information enabling censors to identify and block them without incurring prohibitive costs, including the processing and memory overhead of real-time detection and collateral damage (the economic or social consequences of overblocking). Traditionally, censorship has relied on deep packet inspection to identify forbidden traffic by destination, first by IP address, and later by domains exposed in plaintext application-layer fields (*e.g.,* DNS query name, HTTP `Host` header, or TLS Server Name Indication) as CDNs started hosting many domains on a single IP address [93]. With the widespread adoption of TLS, the Server Name Indication (SNI) became a key target for censors.

Hiding Destination Identity. Several techniques have been proposed to hide destination identity (Figure 1). TLS extensions such as Encrypted SNI (eSNI) and its successor Encrypted Client Hello (ECH) do so by encrypting the SNI and `ClientHello`, respectively. Both have seen limited server-side support, and some countries have simply blocked all connections using them [9], [41], [60], [109]. Domain fronting [27] similarly conceals the destination by using a front domain in the TLS SNI and the true domain in the HTTP request, relying on the server—which must host both domains—to route the request internally. However, this technique has largely been decommissioned by CDNs [19].

The most widely used approach is proxy-based circumvention, which conceals destination identity at the network and application layers by encrypting traffic and relaying it through a proxy server. Early versions of popular proxies such as VPNs and Tor were not designed to resist censorship [6], [23] and exhibited distinctive protocol fingerprints: VPN protocols such as OpenVPN, WireGuard, and IPsec are identifiable with standard protocol parsers, and vanilla Tor transmits traffic in fixed-size cells. Obfuscation has since sought to make proxy protocols less detectable through one of three approaches: *mimicry*, *polymorphism*, or *tunneling*.

Mimicry. Mimicry makes proxy traffic look like something a censor allows by imitating (or "parroting") the features of a benign *cover protocol* such as HTTP [24], [53], [101], VoIP [100], or Skype [58]. Conventional wisdom held that a censor must develop a separate classifier for each cover protocol, giving rise to a "let a thousand flowers bloom" approach to obfuscation that sought to overwhelm censors by diversifying protocols faster than they could

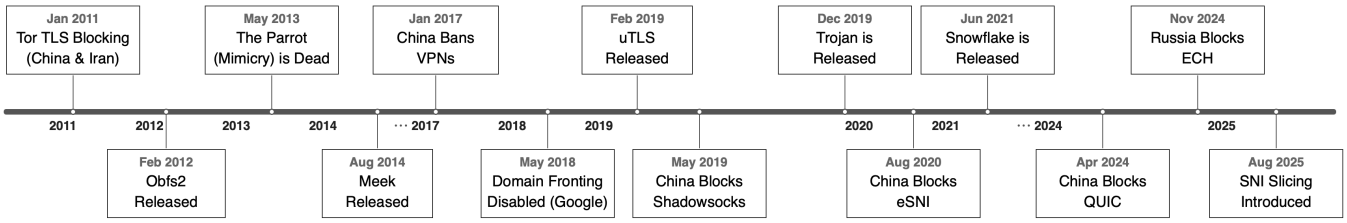


Figure 1: **History of Obfuscation.** We highlight major developments in censor capabilities and obfuscation techniques.

be fingerprinted [5]. However, prior work has shown that mimicry is often trivially detectable [42], [99]. Houmansadr et al. further argued that mimicry is unlikely to ever be viable, reasoning that parrot protocols must faithfully mimic not only the cover protocol's specification but also the idiosyncrasies of concrete implementations [42]. Protocol mimicry has ultimately seen limited real-world deployment and has largely been abandoned in practice.

Polymorphism. Polymorphism, in contrast, transforms proxy traffic such that it looks like nothing a censor blocks, exploiting the censor's unwillingness to implement strict allowlisting that would block the long tail of legitimate but less common protocols. Many circumvention tools including Shadowsocks [21], VMess [92], obfs4 [114], Outline [61], ScrambleSuit [107], and Dust [104] achieve polymorphism through content *randomization*, which aims to eliminate all static plaintext fingerprints by fully encrypting traffic from the first byte of the connection, making it indistinguishable from random bits. While seemingly more robust than mimicry because there are no regular expressions for DPI systems to match against, randomness is itself a fingerprint uncharacteristic of most Internet traffic. Even encrypted protocols such as TLS and SSH begin with a plaintext handshake to negotiate encryption parameters. The GFW has therefore used the entropy of the first data-carrying packet to identify potential Shadowsocks traffic [4] and subsequently to block all fully encrypted protocols [111].

Tunneling. Tunneling, often described as the logical extreme of mimicry, does not merely *emulate* a cover protocol, but instead *uses* it to transport circumvention traffic. Prior work has explored tunneling over many protocols and applications, including DNS [26], VoIP [43], video conferencing [7], email [116], cloud storage services [12], video games [39], [96], WebRTC [8], [10], [28], [91], [95], and existing application traffic [71]. Today, however, most widely deployed circumvention tools tunnel over TLS. Although tunneling is often assumed to be unblockable without significantly disrupting legitimate use of the cover protocol, this assumption has broken down in two key ways. First, circumvention tools have relied on niche implementations of cover protocols whose fingerprints are effectively unique to the tool [2], [73], [74], [75], [106]. Addressing these fingerprints has led to a renewed reliance on limited forms of mimicry, most notably through uTLS [30], which enables tunnels to mimic TLS `ClientHello` messages commonly observed in real user traffic. Second, tunnels are often detectable because they fail to match the *shape* of their cover protocol in terms of packet timings, directions, and sizes [33], [40], [49], [72], [112], [113].

2.2. Related Work on Traffic Analysis

Proactive Evaluation of Defenses. Evaluating defenses against specific classifiers has led to an arms race in multiple subfields of encrypted traffic analysis, in which defenses can only react to advances in attacks. Prior work has sought to break this reactive development cycle in the context of web application side-channel attacks and website fingerprinting by theoretically quantifying distinguishability independent of any particular classifier (or equivalently, under the optimal classifier) [13], [16], [18], [50], [54]. However, the definitions of distinguishability used in these works do not directly transfer to censorship. The ideal these definitions formalize is for traffic from different application activities or websites to be mutually indistinguishable, without necessarily resembling ordinary traffic. For censorship circumvention, however, the ideal is for proxy traffic to be indistinguishable from ordinary cover-protocol traffic.

Only recently have approaches emerged that empower censorship circumvention tool developers to identify and address distinguishability *before* their tools are blocked. Sheffey et al. built CensorLab, a framework for testing circumvention tools against emulations of past, present, and anticipated (*e.g.,* machine-learning) censors [78]. More closely related to our work, Pereira et al. [65] and Khoury et al. [46] argue that indistinguishability claims should be formalized and justified theoretically rather than evaluated solely through ad hoc empirical testing. In practice, however, their evaluations quantify distinguishability through the true and false positive rates of concrete classifiers. All three approaches have thus far considered only the classifiers that researchers can conceive, providing limited assurance against future detection strategies deployed by censors. By contrast, *CenKL* quantifies differences between network traffic distributions independently of any particular classifier.

Adapting Proxy Traffic. Several approaches have sought to transform or exert fine-grained control over the appearance of proxy traffic. Programmable protocol systems such as Marionette [25], Proteus [97], WATER [17], and UPGen [98] enable circumvention tool developers to rapidly specify and deploy new protocols in response to evolving censorship techniques. Traffic shapers [44], [64],

[105] instead shape packet sizes and timings according to a given traffic schedule. While these systems provide mechanisms for transforming or shaping traffic, precisely *what* the resulting traffic should look like has remained an open question. *CenKL* addresses this gap by modeling the distribution of real-world TLS traffic, providing a concrete, empirically grounded target for these systems.

Adversarial machine learning provides an alternative approach to obfuscation, exploiting the practical limitations of machine learning classifiers by inducing misclassification through small perturbations to packet features [37], [51], [52], [94]. Although adversarial examples generated by these perturbations often transfer across classifiers—since different models tend to learn similar decision boundaries [89]—they do not necessarily make traffic more closely resemble the cover protocol, unlike *ProxyShaper*, potentially limiting their long-term efficacy against adaptive censors.

3. *CenKL* System Overview

In this section, we detail *CenKL*, our system for evaluating how well circumvention protocols are obfuscated. We define obfuscation as the extent to which a protocol's traffic is indistinguishable from some notion of baseline traffic. TLS serves as a natural baseline due to its ubiquity and the diversity of application traffic it carries, making it costly for censors to block indiscriminately. We present an overview of *CenKL* in Figure 2. At a high level, *CenKL* collects both TLS and obfuscated protocol traffic, constructs statistical models of each, and compares them to quantify how closely the obfuscated protocol matches the characteristics of TLS.

The first step is constructing a model of TLS traffic. The efficacy of any metric for quantifying distinguishability depends on the fidelity of this model. We therefore partner with a regional ISP to build our model from real-world TLS traffic. However, modeling high-dimensional and large-scale network traffic is statistically and computationally challenging. *CenKL* balances modeling tractability and fidelity by applying dimensionality-reducing assumptions as expressive as the available data permits.

Next, *CenKL* constructs models of obfuscated protocol traffic. In this work, we consider nine popular protocols that tunnel circumvention traffic through TLS connections to a proxy server. We focus on TLS-based protocols because non-TLS traffic is trivially distinguishable from TLS using features unrelated to traffic shape. To collect traffic for these models, we build a controlled testbed that runs obfuscated proxies and routes automated web requests through them.

Finally, we require a method for comparing these models. A common approach for demonstrating the indistinguishability of circumvention protocols is to train a machine learning classifier to distinguish obfuscated protocol traffic from benign cover traffic and show that it performs no better than random guessing [8], [71]. However, this approach is fundamentally limited. First, it ties indistinguishability to a specific classifier, whereas an *ideal* obfuscation protocol must defend against *all possible* classifiers. Second, while machine learning classifiers are highly effective at determining *whether* a protocol is distinguishable, they offer limited insight into *why*. *CenKL* instead uses an approximation of KL divergence, which quantifies differences between the models independently of any classifier and localizes the packets responsible for distinguishability.

3.1. *CenKL* TLS Model

CenKL begins by constructing a statistical model of the shape of baseline TLS traffic from empirical observations, implicitly assuming that traffic shape is the primary feature distinguishing obfuscated traffic from TLS. In other words, the destination IP has not already been identified as a proxy and fingerprints beyond traffic shape (*e.g.,* TLS handshake characteristics) have already been mitigated.

Traffic shape can be characterized at multiple granularities, including per-packet, per-burst, and per-flow [103]. *CenKL* opts for per-packet features because censors aim to interfere with disallowed connections as early as possible, precluding the use of aggregate features that require observing a significant portion of or the entire flow. *CenKL* therefore models a flow as a sequence of packets $\langle x_1, \ldots, x_\ell \rangle$ and a packet at index i as the triple $x_i = (t_i, d_i, s_i)$, where $t_i \in [0, \infty)$ denotes timing, $d_i \in \{-1, +1\}$ denotes direction, and $s_i \in \{1, \ldots, 1460\}$ denotes payload size in bytes of a data-carrying TCP packet belonging to a TLS connection. While directions and sizes are largely determined by the application, timings are heavily influenced by network conditions. To improve model generalizability across diverse networks, we normalize timing by first calculating the delay since the previous packet sent by the same endpoint, then dividing by the round-trip time (RTT) estimated from the TCP three-way handshake as $t_{\mathrm{ACK}} - t_{\mathrm{SYN}}$. Taking inter-packet delays from the same sender removes one-way network latency to the observation point, but the sending rate still reflects path length and congestion; dividing by the RTT mitigates the remaining network-dependent variance.

Traffic Collection. We collected TLS traffic from a network tap within a regional ISP that observes traffic volumes of up to 100 Gbps. Packets were captured and analyzed using PF_RING [22] and a Zeek [63] cluster. We assume that virtually all TLS flows identified by Zeek's SSL/TLS analyzer correspond to regular TLS connections rather than TLS tunnels. Prior work applying obfuscated proxy traffic detection to real-world ISP traffic found that only 0.0544%–0.6% of flows were classified as tunnels [111], [112]. While these flows were conservatively treated as false positives, even if they were truly tunnels, they would not substantively affect *CenKL*'s TLS model.

We applied strict filters to the collected traffic to model a worst-case censor with ideal observation conditions. In particular, we assume a censor with full visibility of the connection in both directions and that the traffic observed by the censor does not exhibit any packet loss, reordering, or retransmissions. If a protocol is indistinguishable from TLS under these idealized conditions, it will also be indistinguishable under the more limited observation conditions

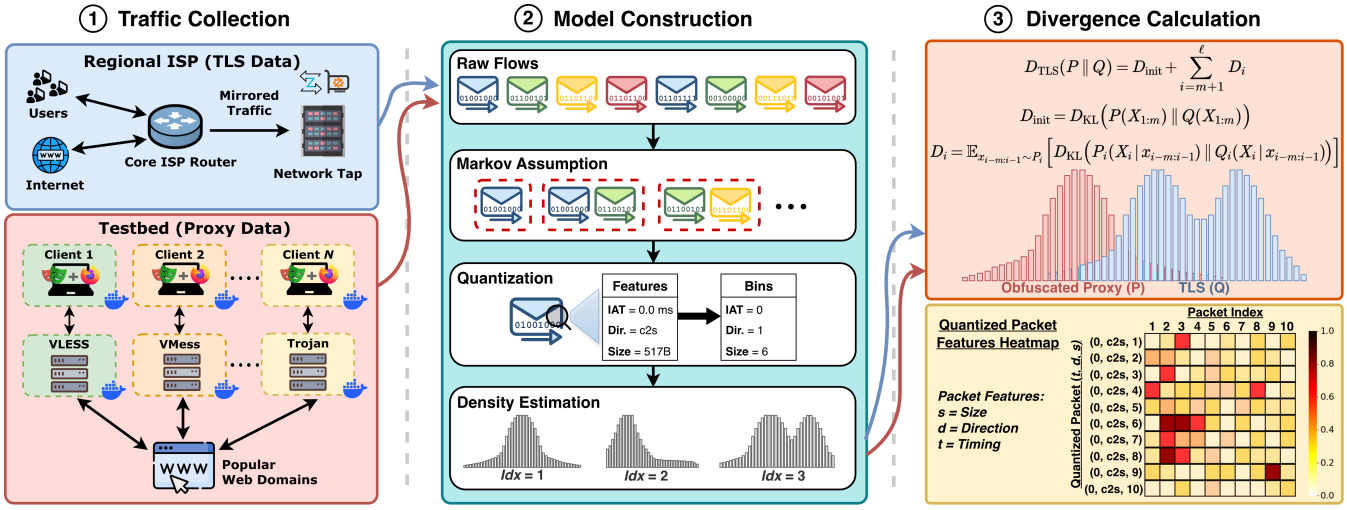


Figure 2: ***CenKL* System Overview.** *CenKL* evaluates the distinguishability of obfuscated proxy traffic from ordinary TLS traffic. To do so, it first collects real-world TLS traffic from a regional ISP and obfuscated traffic from a controlled proxy testbed. Next, it constructs a statistical model of each, applying a Markov assumption and quantization of packet features to reduce model complexity. Finally, it computes an approximation of KL divergence, TLS Divergence, between the proxy and TLS models. The largest terms identify the packet features contributing most to distinguishability.

faced by real censors, which may see only one direction of traffic, miss packets, or monitor only the first few packets of the connection. We therefore required observation of the entire TCP and TLS handshakes and no TCP sequence number gaps. We also required observing at least one Application Data record after the TLS connection was established according to Zeek's `ssl_established` event.

Flows were logged when Zeek stopped tracking the connection (*i.e.,* the flow terminated or was idle for five minutes) or upon reaching a 50-packet threshold, whichever occurred first. For each packet, we recorded only its (t_i, d_i, s_i) triple. For each flow, we recorded the timestamps of the TCP three-way handshake, the TLS version, and the index of the last TLS handshake packet. Traffic was collected over three weeks from November 2025 to December 2025, resulting in a dataset of 326,531,763 TLS flows.

Observing real users' network traffic raises important privacy considerations. We therefore sought approval from our IRB for our data collection plan. Although our work was determined to be "Not Regulated," we nevertheless took several steps to preserve user privacy. First, we restricted inspection to TLS traffic by considering only traffic on TCP port 443 and limited logging to the minimum information required by *CenKL*. We ignored all non-TLS traffic and recorded neither connection endpoint information nor packet payloads. Second, all traffic data was processed directly on the network tap—accessible only to select authors on a least-privilege basis—and was not removed from the ISP network except in aggregated form as part of our TLS model.

Modeling Assumptions. The reliability of *CenKL*'s evaluation of obfuscated protocols depends on the fidelity of its TLS traffic model. Unfortunately, accurately estimating the full joint distribution $Q(X_1, \ldots, X_\ell)$ of TLS traffic flows is intractable. The domain of timing alone is continuous and therefore infinite, and even after discretization, the state space grows exponentially with flow length ℓ as $k = (b_t \cdot b_d \cdot b_s)^\ell$, where b_t, b_d, and b_s denote the domain sizes of the discretized timing, direction, and size features, respectively. To determine the largest state space our data can reliably support, we turn to *sample complexity*, which gives a sufficient number of samples n to approximate a discrete distribution over k outcomes with error tolerance ε (in terms of total variation distance from the true distribution) and confidence $1 - \delta$, for $\varepsilon, \delta \in (0, 1]$ [14]:

$$n \geq \max\left(\frac{k}{\varepsilon^2}, \frac{2}{\varepsilon^2}\ln\left(\frac{2}{\delta}\right)\right)$$

For fixed $n \geq \frac{2}{\varepsilon^2}\ln(\frac{2}{\delta})$, inverting this expression yields $k \leq \lfloor n\varepsilon^2 \rfloor$. To satisfy this bound on k, *CenKL* adopts two dimensionality-reducing modeling assumptions commonly used in encrypted traffic analysis: (1) a finite-order *Markov assumption* [48], [81], under which each packet depends only on a small number of preceding packets, and (2) *quantization* [51], where feature values are grouped into bins to limit the size of the state space. The Markov assumption reduces the exponent ℓ while quantization reduces the base $b_t \cdot b_d \cdot b_s$. These assumptions create a trade-off: for fixed k, increasing modeling complexity in one dimension requires reducing it in the other.

Markov Assumption. Instead of estimating the full joint distribution $Q(X_1, \ldots, X_\ell)$, we estimate ℓ joint distributions $\langle Q_1, \ldots, Q_\ell \rangle$, where, for Markov order m, Q_i is defined over $X_{i-m:i} = (X_{i-m}, \ldots, X_i)$, omitting indices less than 1. Since only small values of m are practical—prior work considers only $m \leq 2$—we leave the choice of m to the user.

Quantization. We also quantize the domains of packet features for each packet position i independently by grouping the observed values of X_i into bins. Selecting an appropriate binning strategy is critical for preserving the distinctive structure of each feature's distribution. The simplest approach, equal-width binning, assumes feature values are uniformly distributed—an assumption violated by both timing and size, albeit in different ways, necessitating feature-dependent quantization.

Packet sizes are multimodal, typically concentrated either at the maximum segment size (MSS) or at small values (*e.g.,* due to TCP segmentation). We therefore bin sizes using *Maxdiff* [66], a variable-width binning strategy that, for b bins, places boundaries at the $b-1$ largest differences in frequency between adjacent values, thereby reducing within-bin variation and, for sufficiently large b, often isolating the most frequently occurring sizes into their own bins.

Packet timings, on the other hand, are right-skewed and therefore require logarithmic bins to provide finer resolution at smaller values and coarser resolution at larger values. However, packet timings introduce several additional challenges for quantization. First, timings are unbounded in principle, so binning them without fixing an upper bound still results in an infinite domain. Second, the upper bound must be selected carefully: binning between 0 and an arbitrarily large value (*e.g.,* the largest number representable by a 64-bit float) would leave many bins empty. Third, the range of observed timings varies significantly across packet indices, so a single fixed range would also allocate bins inefficiently. To address these challenges, we bin timings between lower and upper bounds that vary with the packet index. Rather than using the minimum and maximum observed values at each packet index, we use the 1st and 99th percentiles to avoid allocating bins to extreme outliers; values below the 1st percentile or above the 99th percentile are assigned to the first and last bins, respectively.

CenKL then automatically selects the number of bins $b_{t,i}$, $b_{d,i}$, and $b_{s,i}$ at each packet index i such that—for a given Markov order m, number of observations n_i, and error tolerance ε—the state space size $(b_{t,i} \cdot b_{d,i} \cdot b_{s,i})^{m+1}$ is at most $k \leq \lfloor n_i \varepsilon^2 \rfloor$. To analyze subsets of features, a feature f can be *masked* by setting $b_{f,i} = 1$ for all i. Assuming all features are unmasked, there are always $b_{d,i} = 2$ packet directions. The remaining budget $B = \frac{\sqrt[m+1]{k}}{2}$ is then split between timing and size. A straightforward approach would allocate $b_{t,i} = b_{s,i} = \sqrt{B}$, but timing stands to benefit more from additional resolution because it is continuous. We therefore bias the allocation toward timing by allowing $b_{s,i}$ to be less than $\sqrt{B}$ when the size distribution can be adequately represented with fewer bins. We first apply the Freedman–Diaconis rule [29], which proposes a uniform bin width of $w_i = 2 \cdot \mathrm{IQR} \cdot n_i^{-1/3}$, where IQR is the interquartile range of the size distribution. We then set $b_{s,i} = \min(\lceil \frac{1460}{w_i} \rceil, \sqrt{B})$ and allocate the remaining budget to timing as $b_{t,i} = B/b_{s,i}$. Most Internet flows are short-lived, low-volume mice flows [3], [57]; thus, later packet indices have fewer observations n_i, limiting the model to smaller state spaces under the sample complexity bound. As a result, *CenKL* uses finer-grained bins for earlier packets and coarser-grained bins for later ones.

Distribution Estimation. Finally, for each packet index i, we estimate $Q_i(X_{i-m}, \ldots, X_i)$. We first quantize each subsequence $X_{i-m:i}$ using the bins for index i, then count occurrences of the resulting sequences q. Because our dataset is finite, it is unclear whether unobserved sequences are strictly impossible or merely low-probability. Assigning zero probability to such a sequence q has the undesirable side effect that any model of obfuscated traffic capable of producing q even once would be considered absolutely distinguishable from Q, even if it is otherwise quite similar to normal TLS traffic. Instead of using empirical probabilities directly, we apply Laplace smoothing by adding a *pseudocount* of one to all possible packet sequences, treating each possible sequence as having been observed at least once. The smoothed probability of a sequence q is then:

$$\hat{p}_{\text{smoothed}}(q) = \frac{\text{count}(q) + 1}{\sum_{q' \in Q_i} \text{count}(q') + (b_{t,i} \cdot b_{d,i} \cdot b_{s,i})^{m+1}}$$

3.2. *CenKL* Obfuscated Protocol Testbed

Selected Protocols. We select a set of widely used obfuscation protocols and configurations that tunnel traffic over TLS, spanning multiple design strategies: several Xray-based configurations (VMess [92], VLESS [68], Trojan [90], VLESS-over-WebSocket, VLESS-over-gRPC, and XTLS-Vision [115]), NaïveProxy [20], GOST [35], and WebTunnel [80]. We provide an overview of these protocols in Table 1 and describe each below.

VMess is a fully encrypted protocol originally developed for the V2Ray proxy platform. Because fully encrypted protocols can be detected using the entropy of the first packet [111], we tunnel VMess over TLS. ***VLESS*** is more lightweight than VMess and lacks built-in encryption, instead relying on TLS for security. VLESS is frequently paired with *transports*—which define how data is encapsulated and delivered—such as TCP (*VLESS*), WebSocket (***VLESS-over-WebSocket***), or gRPC (***VLESS-over-gRPC***). ***XTLS-Vision*** is a VLESS-specific mode designed to defend against TLS-in-TLS detection by padding the packets of the inner TLS handshake.

Trojan was designed under the assumption that TLS alone—due to its ubiquity—is sufficient for circumvention, without additional obfuscation. After the TLS handshake, Trojan authenticates clients using the hash of a pre-shared password sent in the client request [90]. ***NaïveProxy*** reuses Chromium's network stack to tunnel proxy traffic through HTTP/2 (or HTTP/3) `CONNECT` tunnels [20]. ***GOST*** (or GO Simple Tunnel) is a versatile tunnel that allows for stacking a proxy layer (*e.g.,* SOCKS, HTTP, Shadowsocks) over a range of transports (*e.g.,* TLS, WebSocket, HTTP/2). We layer SOCKS over TLS for our experiments. ***WebTunnel*** is a Tor pluggable transport based on HTTPT [31], a probe-resistant proxy designed to hide behind a web server that reverse-proxies traffic to a Tor bridge [80].

Version	Protocol	GitHub Stars	Encapsulation (Inner > Outer)
Xray-core v25.10.15	VMess	★39.5K	Inner TLS > VMess Encrypted Data Chunks > VMess AEAD Auth Header > Outer TLS > TCP
	VLESS	★39.5K	Inner TLS > VLESS Header > Outer TLS > TCP
	VLESS over gRPC	★39.5K	Inner TLS > VLESS Header > gRPC Message > HTTP/2 DATA Frame > Outer TLS > TCP
	VLESS over WebSocket	★39.5K	Inner TLS > VLESS Header > WebSocket Frame > Outer TLS > TCP
	VLESS with XTLS	★39.5K	Inner TLS > VLESS Header + Vision Padding > Outer TLS > TCP (Phase 1) Inner TLS > direct splice (zero framing) > Outer TLS > TCP (Phase 2)
	Trojan	★39.5K	Inner TLS > Trojan Header > Outer TLS > TCP
GOST v3.2.6	GOST	★18K	Inner TLS > GOST Relay Header > Outer TLS > TCP
NaïveProxy v2.10.2	NaïveProxy	★9.1K	Inner TLS > Padding Frame > HTTP/2 CONNECT Tunnel > Outer TLS > TCP
WebTunnel v0.0.3	WebTunnel	★*	Inner TLS > Tor OR Protocol > Raw Tunnel > Outer TLS > TCP

TABLE 1: **Overview of Testbed Proxy Protocols.** Note: WebTunnel is hosted on Tor's GitLab; GitHub stars do not apply.

Traffic Collection. To collect obfuscated protocol traffic, we built a controlled testbed that deploys the proxy client and server for each protocol and automates web browsing traffic through them. The testbed is designed to collect traffic at scale while preserving realistic traffic and network conditions, ensuring that observed differences from TLS primarily reflect the behavior of the obfuscated protocols rather than artifacts of the testbed setup.

Each proxy client and server runs in a separate Docker container, enabling scaling both within and across machines: the testbed can be readily replicated across hosts, while container isolation simplifies collecting traffic from multiple protocols on a single machine. Traffic collection for a single protocol can be further scaled by increasing the number of clients connecting to the proxy server. Because the testbed colocates the client and proxy server, client-to-proxy round-trip times would be near zero, inflating the normalized timing values t_i. We therefore use `netem` to emulate client-to-proxy round-trip times matching the distribution observed in our ISP dataset.

The testbed generates traffic by using Playwright [56] to drive Firefox browser traffic through each proxy. Specifically, we repeatedly cycle through the top 15,000 domains from the Alexa Top 1M [59]. For protocols that support stream multiplexing, we restarted the browser process between each domain visit to maximize the number of flows collected. This allows multiplexing of concurrent streams used to fetch resources during a single page load while preventing traffic from multiple domain visits from being carried over the same long-lived proxy connection. We disable QUIC so that all TLS traffic is carried over TCP, ensuring compatibility with *CenKL*'s TLS model. To make the generated traffic more representative of real-world TLS traffic, we use browser flags to enforce a TLS version distribution roughly matching our ISP dataset, with 34% of connections using TLS 1.2 and 66% using TLS 1.3. We similarly use browser flags to ensure that roughly two-thirds of TLS 1.3 connections use post-quantum handshakes, based on statistics reported by Cloudflare [102].

Client-to-proxy traffic is recorded by running `tcpdump` inside the proxy server container. To ensure that the packetization in the capture matches what is transmitted on the wire, we disable TCP segmentation offload in all containers. Packet timing, direction, and size features are then extracted using the same Zeek script used for ISP traffic collection.

Model Construction. Model construction for obfuscated protocols largely follows the same procedure as for TLS, although with two key differences. First, the Markov order and feature bin sizes are fixed to those chosen for the TLS model rather than being selected independently, ensuring that the two models are directly comparable. Second, while *CenKL* applies Laplace smoothing to the TLS model, obfuscated protocol distributions are not smoothed. This asymmetry reflects the one-way nature of the comparison: it is more consequential for obfuscated traffic to exhibit packet sequences absent from TLS than vice versa.

3.3. *CenKL* TLS Divergence

The models of TLS and obfuscated protocol traffic constructed in Sections 3.1 and 3.2 define probability distributions. A common approach for comparing two probability distributions P and Q is to use information-theoretic distance or divergence measures such as *total variation distance* or *Kullback–Leibler (KL) divergence*. These measures capture the inherent distinguishability between P and Q, independent of any particular classifier.

CenKL approximates KL divergence, which, informally, measures the "surprise" resulting from observing samples drawn from P when they are believed to be drawn from Q. By treating obfuscated traffic as P and baseline TLS as Q, KL divergence quantifies the degree to which obfuscated traffic deviates from the expected shape of TLS. Formally, for P and Q defined over a random variable X, the KL divergence $D_{\mathrm{KL}}(P \parallel Q)$ is defined as:

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in X} P(x) \ln \frac{P(x)}{Q(x)}$$

KL divergence is measured in units of *nats* and ranges from 0 to ∞, where 0 indicates that P and Q are identical, and ∞ indicates that there exists an outcome x such that $P(x) > 0$ and $Q(x) = 0$. However, ***CenKL*** never computes a KL divergence of ∞ because Laplace smoothing ensures $Q(x) > 0$, *i.e.*, that all sequences have non-zero probability in the TLS model.

A key property of KL divergence that makes it an ideal basis for comparing ***CenKL***'s TLS and obfuscated traffic models is its *chain rule*, which expresses the KL divergence between joint distributions $P(X, Y)$ and $Q(X, Y)$ as:

$$D_{\mathrm{KL}}(P(X,Y) \parallel Q(X,Y)) \\ = D_{\mathrm{KL}}(P(X) \parallel Q(X)) + D_{\mathrm{KL}}(P(Y \mid X) \parallel Q(Y \mid X))$$

Repeatedly applying the chain rule decomposes the KL divergence between the full joint distributions $P(X_1, \ldots, X_\ell)$ and $Q(X_1, \ldots, X_\ell)$ into a sum of KL divergences between lower-dimensional conditional distributions $P(X_i \mid h_i)$ and $Q(X_i \mid h_i)$, each averaged over histories $h_i = (x_1, \ldots, x_{i-1})$ under P. This decomposition aligns naturally with ***CenKL***'s model of traffic as a sequence of packet-indexed distributions:

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{i=1}^{\ell} \mathbb{E}_{h_i \sim P}\big[D_{\mathrm{KL}}\big(P(X_i \mid h_i) \parallel Q(X_i \mid h_i)\big)\big] \\ = \sum_{i=1}^{\ell} \sum_{h_i, x_i} P(h_i, x_i) \ln \frac{P(x_i \mid h_i)}{Q(x_i \mid h_i)}$$

However, because ***CenKL*** applies a Markov assumption and quantization instead of modeling the full joint distributions exactly, we compute the following approximation:

$$D_{\mathrm{KL}}(P \parallel Q) \approx \sum_{i=1}^{\ell} \sum_{x_{i-m:i}} P_i(x_{i-m:i}) \ln \frac{P_i(x_i \mid x_{i-m:i-1})}{Q_i(x_i \mid x_{i-m:i-1})},$$

where the Markov assumption replaces the full history h_i with the fixed-length history $x_{i-m:i-1}$, and quantization reduces the number of possible values of $x_{i-m:i}$.

We refer to this approximation of KL divergence as the *TLS Divergence*. A higher TLS Divergence indicates greater distinguishability of the obfuscated traffic distribution from the TLS distribution. However, a TLS Divergence of 0 does not imply that the two traffic distributions are identical—only that they are indistinguishable under ***CenKL***'s modeling assumptions. A censor using a more expressive model (*e.g.,* a larger Markov order or finer-grained bins) may therefore still distinguish them.

Interpreting TLS Divergence. ***CenKL*** represents the nested sum in TLS Divergence as a matrix of its constituent terms, enabling inspection of contributions to divergence across packets and packet indices. Taking the maximum (or top k) contribution across packet indices identifies how deep into a flow the strongest distinguishing features occur. Distinguishing features that emerge later in a flow require censors to inspect more packets and maintain per-flow state for longer, increasing the memory needed to monitor the same traffic volume. Meeting these computational requirements can become prohibitively expensive at scale: censors have resorted to monitoring a subset of connections [111] or faced capacity limitations [32] even when blocking based only on the first data-carrying packet payload or `ClientHello`. We can further localize divergence within a packet index by taking the maximum term of the inner sum, providing actionable insight to fix the source of divergence.

4. *CenKL* Findings

In this section, we use ***CenKL*** to evaluate the distinguishability of nine obfuscated protocols. Because the TLS `ClientHello` is transmitted in plaintext, connections can be blocked based on its contents without shape-based analysis; we therefore stripped the TLS handshake from all flows before building models and focused only on patterns in TLS data packets. To isolate the contributions of packet features, at each Markov order $m \in \{0, 1\}$, we constructed separate models for sizes and directions (timing masked) and for timing (size and direction masked). All TLS models used $\varepsilon = \delta = 0.05$, such that the total variation distance between each estimated Q_i and its discretized true distribution is at most 0.05 with probability at least 95%.

Guided by the per-packet-index contributions (in nats) to TLS Divergence shown in Figure 3, we further investigated the largest sources of distinguishability. To identify their causes, we examined the corresponding ***CenKL*** models, proxy source code and documentation, and—for proxies that support disabling the outer TLS tunnel or logging its encryption keys—packet captures of the inner traffic.

Go Dynamic Record Sizing. Figure 3 (a, b, c, d) shows that for most Xray-based protocols—including Trojan, VMess, VLESS, and XTLS-Vision—the largest contributions to TLS Divergence occur within the first three packets. These contributions correspond to unusually frequent 1208-byte packets sent client-to-server at $i = 1$ and server-to-client at $i = 2$ and $i = 3$. In ***CenKL***'s TLS model, only 0.38% of packets at $i = 1$ and 0.07% and 0.05% of packets at $i = 2$ and $i = 3$, respectively, match these patterns.

Reason: We found that this behavior is a consequence of Xray being implemented in Go. Go's `crypto/tls` package uses *dynamic record sizing*, a performance optimization that sizes the first TLS record in each direction to fit within a single TCP segment, allowing endpoints to decrypt the record and process application-layer data immediately rather than waiting for packet reassembly [82]. Go conservatively assumes a maximum segment size of 1208 bytes, derived from the minimum IPv6 MTU of 1280 bytes minus the 40-byte IPv6 header and 32-byte TCP header with timestamps [83]. In tunneled connections, `ClientHellos` carrying post-quantum key shares, as well as most `ServerHellos`, exceed this threshold, resulting in an initial 1208-byte packet. Although Go can be configured to disable dynamic record sizing [82], Xray does not do so.

Interestingly, XTLS-Vision produces 1208-byte client-to-server packets at $i = 1$ in 71.7% of connections, much more often than our testbed sends post-quantum

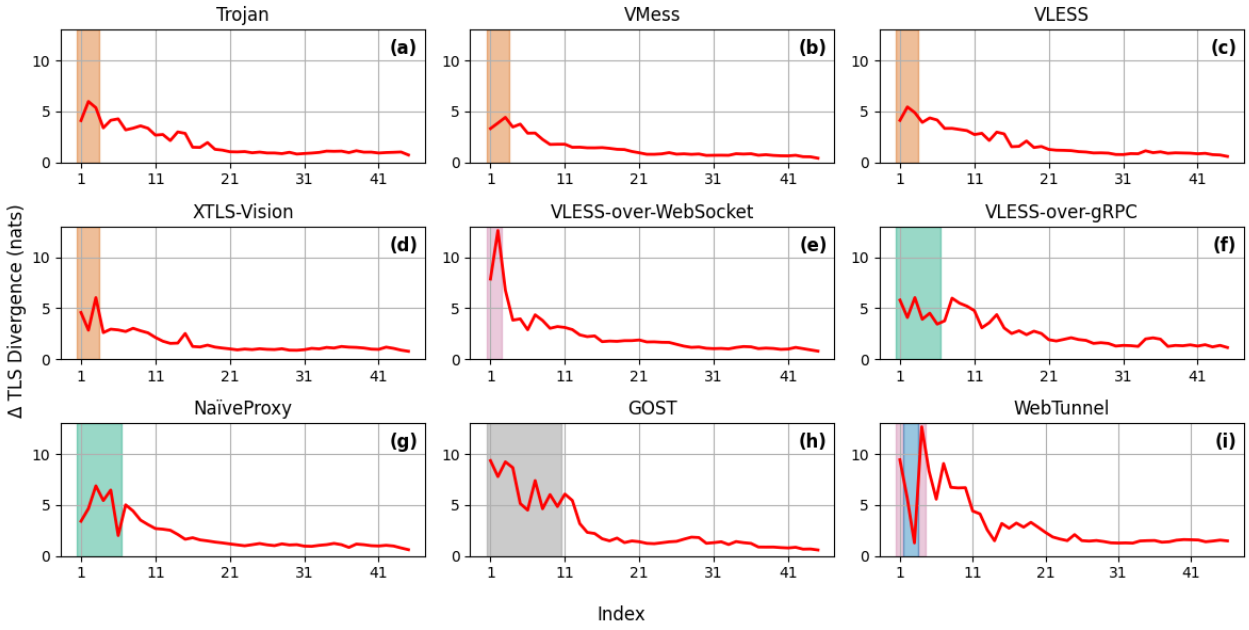


Figure 3: **Per-packet TLS Divergence for Obfuscated Proxies.** TLS Divergence for each evaluated obfuscated proxy, computed from post-TLS-handshake packets using packet sizes and directions with Markov order $m = 1$. Shaded regions indicate the packet indices examined in Section 4, where we observe high divergence due to implementation- and protocol-specific behavior of each proxy.

`ClientHellos`. This is caused by XTLS-Vision's inner TLS handshake padding scheme. Because 38.4% of the target packet size range of 900–1400 bytes lies above 1208 bytes, the padding substantially increases the fraction of proxy connections affected by dynamic record sizing.

OpenSSL New Session Tickets. Figure 3 (i) shows that WebTunnel's second and third packets contribute 7.01 nats to TLS Divergence, driven by the subsequence $\langle -271, -271 \rangle$. In *CenKL*'s TLS model, this subsequence occurs with probability 0.12% at $i = 2$ and $i = 3$.

Reason: We found that before completing the WebSocket handshake, the web server sends two TLS 1.3 `NewSessionTicket` messages. Our testbed uses nginx, which relies on OpenSSL for TLS. By default, OpenSSL sends two `NewSessionTicket` messages on new connections [15]. Although this pattern depends on the web server used, nginx is likely a common choice for WebTunnel deployments because Tor's WebTunnel documentation only provides instructions for configuring nginx [87].

WebSocket Upgrade Handshake. Figure 3 (e, i) shows that the first two packets contribute 20.48 nats to TLS Divergence for VLESS-over-WebSocket, while the first and fourth packets contribute 22.1 nats for WebTunnel. These contributions are driven by the subsequences $\langle +211, -151 \rangle$ and $\langle +152, \ldots, -158 \rangle$, respectively. In *CenKL*'s TLS model, we never observe $\langle +211, -151 \rangle$, while 152-byte client-to-server packets occur at $i = 1$ only 0.01% of the time and 158-byte server-to-client packets occur at $i = 4$ only 0.03% of the time.

Reason: We found that these patterns arise from the initial WebSocket handshake: an HTTP/1.1 `Upgrade` request followed by a `101 Switching Protocols` response. Although both the client request and server response have fixed sizes in our testbed, the client request size depends on server-specific parameters such as the hostname and WebSocket path, making the server response more practical as a distinguishing feature. VLESS-over-WebSocket's 151-byte response is a minimal WebSocket upgrade response containing only the `Upgrade`, `Connection`, and `Sec-WebSocket-Accept` headers, all of which have a fixed length. WebTunnel's response sets only the first two of these headers [79], though it may also inherit headers automatically added by the web server. For example, nginx adds `Server` and `Date` headers by default, both of which have a fixed size.

GOST SOCKS5 Handshake. Figure 3 (h) shows that the first nine packets of GOST connections contribute 62.72 nats to TLS Divergence. These packets follow a nearly deterministic pattern: $\langle +26, -24, +1448, +77, -1448, -*, +86, +*, -54 \rangle$. This pattern never occurs in our TLS dataset. Even its shortest prefixes are exceedingly rare in *CenKL*'s TLS model: 0.01% of connections begin with $\langle +26 \rangle$, and 0.0003% of connections begin with $\langle +26, -24 \rangle$.

Reason: We found that this pattern arises from SOCKS5 and TLS handshakes performed before any user traffic is tunneled. The first two packets correspond to the SOCKS5 authentication-method negotiation, in which the client advertises its supported authentication methods and the server

selects one. The next five packets correspond to a GOST extension to SOCKS that establishes an additional layer of TLS encryption [36]. This extension is enabled by default, even when SOCKS5 is already tunneled over TLS, but it can be disabled using the `notls` option. The final two packets represent the SOCKS5 `CONNECT` request and reply, in which the SOCKS5 server establishes a TCP connection to the requested destination and returns the connection status.

HTTP/2 Connection Preface. Figure 3 (f, g) shows the first six packets of VLESS-over-gRPC and NaïveProxy contribute 27.81 nats and 28.88 nats to TLS Divergence, respectively. VLESS-over-gRPC connections typically begin with $\langle +46, +31, -37, +31, *, -31 \rangle$. NaïveProxy connections typically begin with $\langle +92, *, -61, +31, -35, -31 \rangle$.

Reason: We found that despite the substantial differences in packetization, both patterns represent the HTTP/2 connection preface—consisting of the client connection preface (a 24-byte magic string), a `SETTINGS` frame from each endpoint, and a `SETTINGS` acknowledgment frame from each endpoint [88]—as well as the client request. The most notable differences concern the sizes of the first packet and the client `SETTINGS` frame. First, unlike VLESS-over-gRPC, which sends only the client connection preface in its first packet, NaïveProxy also sends the client `SETTINGS` frame (containing Chromium's four default HTTP/2 settings) and a `WINDOW_UPDATE` frame. By doing so, NaïveProxy makes its first packet substantially more representative of real-world TLS traffic: only 0.3% of connections begin with $\langle +46 \rangle$, whereas 3.18% begin with $\langle +92 \rangle$. Second, unlike NaïveProxy, VLESS-over-gRPC sends an empty client `SETTINGS` frame. In our TLS dataset, $\langle +46, +31 \rangle$ is approximately 23 times rarer than sequences consistent with a connection preface followed by a non-empty `SETTINGS` frame.

Timings. *CenKL*'s timing models revealed that several protocols exhibit spikes in TLS Divergence at $i = 3$ (*e.g.,* GOST) or $i = 4$ (*e.g.,* VLESS-over-gRPC). The spikes at these indices arise from length-2 timing sequences in which a packet falls into bin 0 (near-zero delay between same-sender packets) followed by a packet in a much later bin, typically corresponding to delays of 0.4–1 RTTs. In *CenKL*'s TLS model, however, over 63% of length-2 timing sequences at $i = 4$ place both values in the first five bins, corresponding to delays under 0.08 RTTs.

These timing differences reflect the dependency structure of inner handshakes. In normal TLS traffic, the client's initial request is typically followed by several server response packets arriving in close succession, producing short same-sender delays. Inner handshakes break this pattern by introducing request-response exchanges that increase same-sender delays. This effect is particularly prominent for GOST and WebTunnel, which both perform multiple protocol handshakes before the user's inner TLS handshake. GOST performs a SOCKS handshake followed by a TLS handshake [36], while WebTunnel performs a WebSocket handshake followed by Tor circuit establishment [87].

5. Reducing Distinguishability with *CenKL*

So far, we have used *CenKL* to *evaluate* the distinguishability of obfuscated protocols from TLS. In this section, we explore how obfuscated protocols can use *CenKL* to *reduce* that distinguishability. To this end, we develop and evaluate *ProxyShaper*, a proof-of-concept traffic shaper for VMess and VLESS. An overview of *ProxyShaper* is shown in Figure 4. At a high level, *ProxyShaper* uses *CenKL*'s TLS model as a generative model of *traffic schedules*, which specify the sender and size of the ith packet of a flow. At the beginning of each new proxy connection, *ProxyShaper* reshapes proxy traffic using a generated traffic schedule.

5.1. *ProxyShaper* Design Overview

Like other circumvention tools, *ProxyShaper* operates in user space and therefore cannot directly control TCP packetization performed by the kernel. Instead, *ProxyShaper* targets the packet sizes and directions specified by its traffic schedule by controlling TLS record sizes and which endpoint transmits each record. Because we observe that most divergence occurs early in a flow (Figure 3), our current implementation shapes only the first 10 records following the proxy's TLS handshake. This far exceeds the observation window required by state-of-the-art deployed censors, which can still block circumvention traffic using no more than the first data-carrying packet [1], [111]. *ProxyShaper* focuses on packet sizes and directions and does not attempt to shape timing, which is difficult to do reliably because network and endpoint conditions outside the proxy's control can alter the timing observed by a censor. Building a practical traffic shaper that can defend against censors that use timing remains an important direction for future work.

Sampling Traffic Schedules. To generate a schedule, *ProxyShaper* samples packets sequentially from *CenKL*'s TLS model, conditioned on previously sampled packets. Formally, for an m-order model:

$$x_i \sim Q_i(X_i \mid x_{i-m}, \ldots, x_{i-1})$$

Because *CenKL* quantizes packet features, each bin must be mapped back to a concrete value. Direction bins correspond to a single direction. Size bins, however, are variable-width and may represent multiple sizes rather than a single value. Rather than assuming all sizes within a bin are equally likely, *ProxyShaper* samples from the empirical distribution of sizes observed within that bin.

Shaping Proxy Traffic. Before shaping begins, both the proxy server and the client must first agree on the traffic schedule to avoid a mismatch between the expected packet order. However, sending an initial packet containing a traffic schedule at the beginning of a new connection may itself introduce a fingerprint that can be detected by censors. *ProxyShaper* instead relies on the already established TLS session to independently derive the same schedule on both endpoints. At both endpoints, *ProxyShaper* computes Exported Keying Material (EKM) [70] from the TLS session

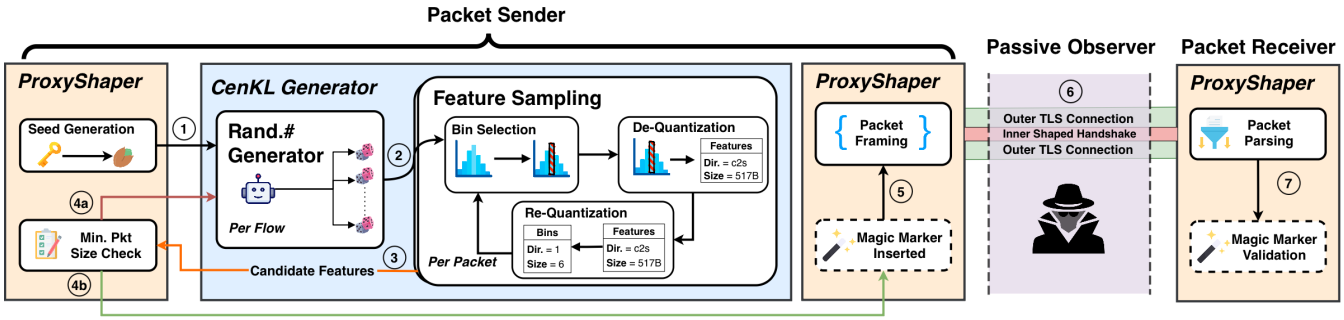


Figure 4: ***ProxyShaper* Shaping Process.** *ProxyShaper* samples candidate traffic schedules from *CenKL* using a shared seed, dequantizes them, and checks whether the resulting schedule can accommodate protocol overhead. If not, additional candidates are sampled. Otherwise, *ProxyShaper* shapes the proxy traffic according to the selected schedule. On the receiver side, the shaped packets are parsed and validated using a magic marker embedded in the first packet.

secret to derive the same 8-byte value. *ProxyShaper* uses this value as the seed for generation, ensuring that the sampled traffic schedule is identical at both endpoints.

ProxyShaper then shapes outgoing traffic according to the sizes and directions specified by the generated schedule. For each scheduled packet, *ProxyShaper* determines the corresponding TLS `Write` size by subtracting the outer TLS overhead (record header and authentication tag) from the target TCP payload size. Each shaped record includes a 2-byte length field that allows the receiver to separate application data from padding, and the first shaped record additionally includes a fixed 4-byte encrypted marker that allows the receiver to verify that both endpoints are using *ProxyShaper*. If the available application data does not fill the remaining space, *ProxyShaper* pads the record; if it exceeds the remaining space, the excess is carried in subsequent records. When no application data is available, *ProxyShaper* sends a padding-only dummy record. To maintain the intended packet sizes as traffic moves down the network stack, *ProxyShaper* disables dynamic TLS record sizing, preventing Go's TLS implementation from producing the 1208-byte packet-size fingerprint identified in Section 4. Further, Go enables `TCP_NODELAY` by default [84], sending data as soon as possible and minimizing coalescing of consecutive records into a single packet.

Not every sampled packet size is large enough to accommodate the required TLS and *ProxyShaper* overhead. *ProxyShaper* therefore samples a batch of candidate schedules from the shared seed, validates them in order, and selects the first feasible schedule. If no candidate is feasible, both endpoints increment the seed and repeat the process. Because schedule generation and validation are deterministic, the client and server independently arrive at the same schedule without negotiation.

5.2. Distinguishability & Detection

We evaluate *ProxyShaper* along two dimensions: its effect on TLS Divergence, our classifier-agnostic measure of distinguishability, and its effect on the detection accuracy of real-world traffic classifiers, ranging from a TLS fingerprinting attack [112] to machine learning classifiers of varying complexity [38], [55].

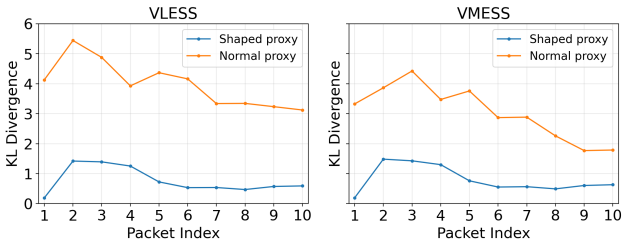


Figure 5: **TLS Divergence Before and After Shaping.** *ProxyShaper* significantly reduces TLS Divergence for both VMess and VLESS. The post-shaping TLS Divergence for both protocols is nearly identical as *ProxyShaper* samples traffic schedules from the same distribution.

Dataset. We collected 137,997 shaped VLESS flows and 118,743 shaped VMess flows using the testbed described in Section 3.2. Approximately half of the flows use TLS 1.2 for the outer tunnel and the other half use TLS 1.3. We sampled traffic schedules from *CenKL*'s size-and-direction TLS model with Markov order $m = 1$.

Effect on TLS Divergence. Since *ProxyShaper* is designed to mitigate the early-connection artifacts that *CenKL* identified as the primary sources of divergence from TLS, successful shaping should move proxy traffic closer to *CenKL*'s TLS model. In Figure 5, we compare the size-and-direction models of shaped and unshaped VLESS and VMess traffic; for each protocol, both models contain the same number of flows. Shaping substantially reduces TLS Divergence for both protocols but does not eliminate it, particularly in the early packets. This residual divergence is largely due to TCP configuration differences between our testbed and the diverse endpoints represented in our ISP baseline. Linux enables TCP timestamps by default [45], reducing the effective MSS in our testbed to 1448 bytes, whereas our baseline contains 1460-byte TCP segments.

Protocol	TLS	Unshaped			Shaped		
		Prec.	Recall	FPR	Prec.	Recall	FPR
VLESS	1.2	0.997	0.712	0.002	0.991	0.611	0.005
	1.3	0.996	0.866	0.003	0.988	0.466	0.006
VMess	1.2	0.994	0.716	0.004	0.989	0.622	0.007
	1.3	0.997	0.828	0.002	0.981	0.483	0.009

TABLE 2: **TLS Fingerprinting Attack Detection Results.** While precision remains high and FPR remains low after shaping, recall decreases substantially.

Protocol	Model	Unshaped			Shaped		
		Prec.	Recall	FPR	Prec.	Recall	FPR
VLESS	LR	0.785	0.621	0.170	0.574	0.824	0.611
	MLP	0.972	0.981	0.029	0.768	0.723	0.218
	k-NN	0.981	0.987	0.019	0.793	0.781	0.204
	GBDT	0.977	0.986	0.023	0.830	0.727	0.149
VMESS	LR	0.812	0.649	0.150	0.574	0.816	0.607
	MLP	0.952	0.976	0.049	0.783	0.682	0.189
	k-NN	0.962	0.973	0.038	0.790	0.775	0.206
	GBDT	0.960	0.983	0.041	0.823	0.731	0.157

TABLE 3: **ML Classifier Detection Results.** Shaping substantially decreases precision for all classifiers and recall for all except LR, while increasing the FPR by at least $3\times$ for all classifiers.

Consequently, a 1460-byte TCP payload targeted by *ProxyShaper* is split into 1448- and 12-byte packets, desynchronizing the actual packet sequence from the traffic schedule. After shaping, the previously distinct divergence profiles of VLESS and VMess become nearly identical, suggesting that protocol-specific patterns have largely been removed.

Effect on Detection. Next, we evaluate *ProxyShaper* against a protocol-agnostic TLS fingerprinting attack, as well as four machine learning classifiers trained on the sizes and directions of up to the first 10 post-handshake packets. The machine learning classifiers use an 80/20 train-test split, while the TLS fingerprinting attack uses a 70/15/15 train-validation-test split. In both cases, we sample an equal number of TLS flows from the ISP dataset as negative examples, matching their packet-count distribution to the corresponding proxy dataset. Without this matching, a classifier could easily distinguish proxy from TLS traffic based on flow length alone, rather than learning meaningful differences in traffic shape. We use the following five detection methods:

Fingerprinting Encapsulated TLS Handshakes. Xue et al. showed that proxy traffic can be identified by exploiting artifacts of TLS handshakes encapsulated within outer proxy tunnels [112]. We re-implement the classifier, but restrict its input to the first 10 post-handshake packets that are reshaped by *ProxyShaper*.

k-Nearest Neighbors (k-NN). k-NN is an instance-based method that classifies a sample according to the labels of its nearest neighbors in the feature space.

Logistic Regression (LR). LR estimates the probability of class membership using linear boundaries. Because network traffic is unlikely to be linearly separable, we augment the data with *Nyström feature maps*, which allow efficient linear algorithms to solve complex, non-linear problems.

Multi-Layer Perceptron (MLP). MLP is a flexible classifier with multiple layers of non-linear transformations. The middle ("hidden") layers dictate model complexity. We use a compact architecture with two hidden layers to reflect the computational constraints of line-rate deployment.

Gradient Boosted Decision Tree (GBDT). GBDT is an ensemble method that sequentially combines decision trees, each correcting errors made by the preceding ensemble. It is considered a state-of-the-art model for structured tabular data, often outperforming deep neural networks [38], [55].

Performance Metrics. Prior work has argued that accuracy is insufficient for evaluating censorship classifiers, as it obscures the operational cost of collateral damage [78], [98], [99], [112]. Because proxy traffic is rare, even a small false positive rate (FPR) can correspond to a large number of legitimate connections being disrupted. We therefore report precision, recall, and FPR. Precision captures how often a connection flagged as a proxy is actually a proxy, while recall measures how much proxy traffic a censor can successfully identify. FPR measures the fraction of legitimate TLS traffic incorrectly classified as proxy traffic and therefore directly reflects the collateral damage incurred by the censor. In order to be *effective*, a classifier must have a high precision and recall. However, to be deployable in practice, FPR must be exceedingly low, with prior work putting estimates under 0.6% [111], [112].

Results. Table 2 compares the results of encrypted TLS handshake detection for both unshaped and shaped proxy traffic. Shaping has a small effect on precision and FPR but significantly reduces recall, especially for TLS 1.3, which now accounts for the majority of web traffic. This suggests the fingerprinting attack is highly conservative, allowing obfuscated flows rather than blocking legitimate TLS.

Table 3 shows that the effect is more significant for the machine learning classifiers. Unshaped traffic is readily distinguishable from TLS, with the best-performing classifiers achieving precision and recall above 95% and a false positive rate of less than 5%. Shaping substantially reduces precision for all classifiers and recall for all except LR, while increasing false positive rates by severalfold (up to $10\times$ for k-NN). For LR, recall increases, but precision drops by over 20 percentage points and the false positive rate exceeds 60% for both VLESS and VMess, making this operating point impractical for a real-world censor.

Takeaways. These results demonstrate the practical value of *CenKL*'s TLS model as a target for traffic shaping. While *ProxyShaper* does not make VLESS or VMess fully indistinguishable from TLS, it consistently reduces the effectiveness of several classifiers and raises their FPRs, making them less suitable for blocking at scale.

6. Limitations

CenKL is a diagnostic framework that gives information-theoretic visibility into the traffic patterns of censorship circumvention tools. As discussed in Section 3, the value of the diagnosis depends on the fidelity of the TLS model, which in turn depends on the amount and representativeness of the data collected. *CenKL*'s models can be improved by collecting more data, *e.g.,* as TLS traffic evolves or as further research illuminates what is representative of "normal" TLS traffic from a censor's perspective. Moreover, the goal of *CenKL* is to raise the cost of censorship by requiring censors to maintain more state, observe flows for longer, or tolerate more collateral damage. This means that *CenKL* does not claim to solve indistinguishability—absolute security against a theoretically optimal censor is unattainable in practice—but rather empowers circumvention tool developers to proactively address fingerprints in their traffic patterns. Finally, *CenKL* only applies to circumvention tools that use TLS over TCP. While this covers many popular obfuscated proxies, it does not cover those that use DTLS, WebRTC, or QUIC. *CenKL* can be extended to these protocols using the same general collection and modeling methodology.

7. Open-Source Availability

Given the value that *CenKL* provides to circumvention tool developers, we open-source five artifacts. *CenKL* is available at https://github.com/censoredplanet/obfuscation and *ProxyShaper* is available at https://github.com/censoredplanet/Xray-ProxyShaper.

(1) **Full *CenKL* Pipeline.** The `obfs` command-line tool parses Zeek logs into compact binary flow files, computes per-packet feature statistics, constructs quantizers, fits Markov-chain traffic models, computes TLS Divergence between two models, and generates synthetic flows.

(2) ***CenKL* Traffic Flow Analysis.** A Zeek script (`obfuscation.zeek`) extracts the per-packet features required by *CenKL*, including timestamp, direction, and TCP payload size, while enforcing strict data-quality filters such as discarding flows with sequence number gaps or missing handshake packets. The same script is used for both ISP and testbed traffic collection.

(3) **TLS Models.** Pre-built TLS models at Markov orders $m = 0$ and $m = 1$, with separate models for sizes and directions (timing masked) and for timing (size and direction masked), released as serialized `TrafficProfile` files and their corresponding quantizer definitions. Developers can use these directly without needing access to ISP traffic. Following our Ethics Considerations in the Appendix, we do not release the underlying flow-level data. As TLS traffic patterns evolve and additional data is collected, we will periodically update the model.

(4) **Containerized Testbed.** A Docker Compose-based testbed with client–server container pairs for all nine protocols evaluated in Section 4, along with Playwright-based browser automation, `netem` delay injection using our ISP-derived RTT distribution, and capture harnesses that automate `tcpdump` collection. Adding a new protocol requires only a container definition and a SOCKS port mapping. A setup flowchart is provided in Figure 6 (Appendix B).

(5) ***ProxyShaper*.** A modified Xray fork with *ProxyShaper* integrated, along with build instructions and example configurations. The fork adds a shaping hook to Xray's TCP-over-TLS and TCP-over-uTLS stream path that operates after TLS establishment and before the proxy handshake, letting ordinary VLESS and VMess configurations use *ProxyShaper* without changes to their implementations. The artifact serves as a proof-of-concept demonstrating the potential of traffic shaping using *CenKL*.

8. Discussion and Conclusion

In this paper, we present *CenKL*, a system that evaluates how distinguishable obfuscated traffic is from real-world TLS traffic and localizes the packets contributing most to distinguishability. *CenKL* addresses a fundamental asymmetry that currently gives a significant advantage to the censor: censors can observe traffic exchanged by many diverse endpoints, while circumvention tool developers have no direct visibility into the middleboxes that perform censorship or the models they use to detect circumvention traffic.

A key takeaway from *CenKL* is that each additional layer within the circumvention tunnel *increases* the difficulty of obfuscation. Inner layers such as WebSocket may provide deployment benefits, including CDN routing and compatibility with existing web infrastructure, but also narrow the effective cover population from all TLS traffic to flows with the same inner protocol stack. Furthermore, the request-response exchanges of nested handshakes can also introduce inter-packet delays which are consistently larger than those in normal TLS traffic.

Our results also show that obfuscation depends on the entire implementation, not only the protocol design. Several of the strongest fingerprints we identified originate from libraries and dependencies, including Go's dynamic record sizing and OpenSSL's `NewSessionTicket`. *CenKL* demonstrates that these implementation artifacts are information-theoretically observable and can make an otherwise well-motivated protocol distinguishable in deployment.

Finally, *ProxyShaper* illustrates the development loop that *CenKL* is designed to support, guiding the community toward more effective and principled obfuscation. *ProxyShaper* is a proof of concept rather than a complete traffic-analysis defense: it inherits *CenKL*'s modeling assumptions and does not shape packet timing because network conditions significantly complicate reliable shaping. Nevertheless, it shows how *CenKL* can localize distinguishing artifacts and that fixing them can reduce the utility of multiple classifiers. We believe this proactive cycle supports an apt and realistic goal for censorship resistance: making fingerprinting require more computational resources and incur more collateral damage than censors are willing or able to accept.

Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive feedback. This material is based upon work supported by the National Science Foundation under Grant Numbers 2237552, 2452885, and CCF-2312537, as well as by the Open Technology Fund.

References

[1] Anna Ablove, Johnnie Walker, Ben Wolin, Niklas Niere, Felix Lange, Aaron Ortwein, Armin Huremagic, Richa Priyanka, Ali Zohaib, Jade Sheffey, Nico Heitmann, J. Alex Halderman, Juraj Somorovsky, Amir Houmansadr, Roya Ensafi, Mingshi Wu, and Eric Wustrow. Technical Analysis of the Geedge Networks Firewall Source Code Leak. In *35th USENIX Security Symposium (USENIX Security 26)*, 2026.

[2] Sadia Afroz and David Fifield. Timeline of Tor censorship. https://archive.torproject.org/websites/lists.torproject.org/pipermail/ooni-dev/attachments/20150202/8552d5c2/attachment-0001.pdf.

[3] Suleman Ahmad and Peter Wu. Measuring characteristics of TCP connections at Internet scale. https://blog.cloudflare.com/measuring-network-connections-at-scale/, 2025.

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

[5] Anonymous and Anonymous. Sharing a modified Shadowsocks as well as our thoughts on the cat-and-mouse game. https://gfw.report/blog/thoughs_on_cat_and_mouse_game/en/, 2022.

[6] Jacob Appelbaum, Marsh Ray, Karl Koscher, and Ian Finder. vpwns: Virtual Pwned Networks. In *2nd USENIX Workshop on Free and Open Communications on the Internet (FOCI 12)*, 2012.

[7] Diogo Barradas, Nuno Santos, and Luís Rodrigues. DeltaShaper: Enabling Unobservable Censorship-resistant TCP Tunneling over Videoconferencing Streams. *Privacy Enhancing Technologies*, 2017.

[8] Diogo Barradas, Nuno Santos, Luís Rodrigues, and Vítor Nunes. Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communications by Parasitizing on WebRTC. In *Computer and Communications Security*, 2020.

[9] Kevin Bock, iyouport, Anonymous, Louis-Henri Merino, David Fifield, Amir Houmansadr, and Dave Levin. Exposing and Circumventing China's Censorship of ESNI. https://gfw.report/blog/gfw_esni_blocking/en/, 2020.

[10] Cecylia Bocovich, Arlo Breault, David Fifield, Serene, and Xiaokang Wang. Snowflake, a censorship circumvention system using temporary WebRTC proxies. In *USENIX Security Symposium*. USENIX, 2024.

[11] Brave New Software Project. Lantern: Unlock the full internet. https://lantern.io/, 2026.

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

[13] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[14] Clément L. Canonne. A short note on learning discrete distributions. https://arxiv.org/abs/2002.11457, 2020.

[15] Matt Caswell. openssl/openssl/ssl/ssl_lib.c. https://github.com/openssl/openssl/commit/36ff232cf2bf5dfcaf9e60a8c492439428a243bb, 2018. Commit 36ff232, line 3038.

[16] Giovanni Cherubin. Bayes, not naïve: Security bounds on website fingerprinting defenses. *Proceedings on Privacy Enhancing Technologies*, 2017.

[17] Erik Chi, Gaukas Wang, J. Alex Halderman, Eric Wustrow, and Jack Wampler. Just add water: Webassembly-based circumvention transports, 2024.

[18] Tom Chothia, Yusuke Kawamoto, and Chris Novakovic. A tool for estimating information leakage. In *Proceedings of the 25th International Conference on Computer Aided Verification*, 2013.

[19] Catalin Cimpanu. Amazon Follows Google in Banning "Domain Fronting". https://www.bleepingcomputer.com/news/cloud/amazon-follows-google-in-banning-domain-fronting/, 2018.

[20] NaïveProxy Contributors. NaïveProxy. https://github.com/klzgrad/naiveproxy.

[21] Shadowsocks contributors. Shadowsocks — A fast tunnel proxy that helps you bypass firewalls. https://shadowsocks.org/.

[22] Luca Deri. Improving Passive Packet Capture: Beyond Device Polling. In *Proceedings of SANE*, 2004.

[23] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *13th USENIX Security Symposium (USENIX Security 04)*, 2004.

[24] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Protocol Misidentification Made Easy with Format-Transforming Encryption. In *Computer and Communications Security*, 2013.

[25] Kevin P. Dyer, Scott E. Coull, and Thomas Shrimpton. Marionette: A Programmable Network-Traffic Obfuscation System. In *USENIX Security Symposium*, 2015.

[26] David Fifield. dnstt. https://www.bamsoftware.com/software/dnstt/index.html, 2020.

[27] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. *Privacy Enhancing Technologies*, 2015.

[28] Gabriel Figueira, Diogo Barradas, and Nuno Santos. Stegozoa: Enhancing WebRTC Covert Channels with Video Steganography for Internet Censorship Circumvention. In *Asia CCS*, 2022.

[29] David A. Freedman and Persi Diaconis. On the histogram as a density estimator: L_2 theory. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 1981.

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

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

[32] Anastasia Gavrilyuk. Server Bypass: It is becoming more difficult for Roskomnadzor to block Internet resources. https://www.forbes.ru/tekhnologii/557481-obhod-na-server-roskomnadzoru-stanovitsa-tazelee-blokirovat-internet-resursy, 2026.

[33] John Geddes, Max Schuchard, and Nicholas Hopper. Cover Your ACKs: Pitfalls of Covert Channel Censorship Circumvention. In *Computer and Communications Security*, 2013.

[34] gfw-report. Large scale blocking of TLS-based censorship circumvention tools in China. https://github.com/net4people/bbs/issues/129, 2022.

[35] gost Contributors. GO Simple Tunnel. https://github.com/go-gost/gost/blob/master/README_en.md.

[36] GOST Project. SOCKSv4/v5 – GOST. https://gost.run/en/tutorials/protocols/socks/.

[37] Alonso Granados, Mohammad Sujan Miah, Anthony Ortiz, and Christopher Kiekintveld. A Realistic Approach for Network Traffic Obfuscation Using Adversarial Machine Learning. In *Decision and Game Theory for Security*, 2020.

[38] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[39] Bridger Hahn, Rishab Nithyanand, Phillipa Gill, and Rob Johnson. Games Without Frontiers: Investigating Video Games as a Covert Channel. In *European Symposium on Security & Privacy*, 2016.

[40] 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.

[41] Nguyen Phong Hoang, Michalis Polychronakis, and Phillipa Gill. Measuring the Accessibility of Domain Name Encryption and its Impact on Internet Filtering. In *Proceedings of the 23rd Conference on Passive and Active Network Measurement*, 2022.

[42] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. The parrot is dead: Observing unobservable network communications. In *Symposium on Security & Privacy*, 2013.

[43] 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*, 2013.

[44] Aaron Johnson and Rob Jansen. Modular Traffic Shaping for Censorship Evasion. In *Workshop on Privacy in the Electronic Society*, 2026.

[45] The kernel development community. IP Sysctl. https://docs.kernel.org/networking/ip-sysctl.html.

[46] Joud Khoury, Minyoung Kim, Christophe Merlin, Jose Meseguer, Zachary Ratliff, and Carolyn Talcott. Maude-HCS: Model Checking the Undetectability-Performance Tradeoffs of Hidden Communication Systems. https://arxiv.org/abs/2603.03369, 2026.

[47] klzgrad. Looking for: A good paper on how TLS-in-TLS detection works? https://github.com/net4people/bbs/issues/281#issuecomment-1721463811, 2023.

[48] Maciej Korczyński and Andrzej Duda. Markov Chain Fingerprinting to Classify Encrypted Traffic. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, 2014.

[49] 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.

[50] Shuai Li, Huajun Guo, and Nicholas Hopper. Measuring information leakage in website fingerprinting attacks and defenses. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[51] Haoyu Liu, Alec F. Diallo, and Paul Patras. Amoeba: Circumventing ML-supported Network Censorship via Adversarial Reinforcement Learning. *Proc. ACM Netw.*, 2023.

[52] Likun Liu, Haining Yu, Shilin Yu, and Xiangzhan Yu. Network Traffic Obfuscation against Traffic Classification. *Security and Communication Networks*, 2022.

[53] Daniel Luchaup, Kevin P. Dyer, Somesh Jha, Thomas Ristenpart, and Thomas Shrimpton. LibFTE: A Toolkit for Constructing Practical, Format-Abiding Encryption Schemes. In *USENIX Security Symposium*, 2014.

[54] Luke Mather and Elisabeth Oswald. Pinpointing side-channel information leaks in web applications. *Journal of Cryptographic Engineering*, 2012.

[55] Duncan McElfresh, Sujay Khandagale, Jonathan Valverde, Vishak Prasad C., Ganesh Ramakrishnan, Micah Goldblum, and Colin White. When do neural nets outperform boosted trees on tabular data? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[56] Microsoft. Playwright: Fast and reliable end-to-end testing for modern web apps, 2026.

[57] Ayush Mishra, Archit Bhatnagar, Yixuan Zhang, Ben Leong, Ya Gao, and Raj Joshi. Managing Congestion Control Heterogeneity on the Internet with Approximate Performance Isolation. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, 2026.

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

[59] Mozilla. mozilla/cipherscan/top1m. https://github.com/mozilla/cipherscan/tree/master/top1m.

[60] Niklas Niere, Felix Lange, Nico Heitmann, and Juraj Somorovsky. Encrypted client hello (ECH) in censorship circumvention. In *Free and Open Communications on the Internet*, 2025.

[61] Outline Foundation. Outline SDK (Beta). https://github.com/OutlineFoundation/outline-sdk, 2023.

[62] Palo Alto Networks. What is TLS decryption? methods, risks & best practices. https://www.paloaltonetworks.com/cyberpedia/what-is-tls-decryption.

[63] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *7th USENIX Security Symposium*, 1998.

[64] Hugo Santos Pereira, Afonso Vilalonga, Kevin Gallagher, and Henrique Domingos. Extended Abstract: Traffic Shaping for Network Protocols: A Modular and Developer-Friendly Framework. In *Free and Open Communications on the Internet*, 2025.

[65] Vitor Pereira, Ahmed Irfan, Vinod Yegneswaran, Nick Feamster, Prateek Mittal, and Vitaly Shmatikov. Position paper: A case for machine-checked verification of circumvention systems. In *Free and Open Communications on the Internet*, 2025.

[66] Viswanath Poosala, Peter J. Haas, Yannis E. Ioannidis, and Eugene J. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. *SIGMOD Rec.*, 1996.

[67] Project X Community. Project X: Xray-core. https://xtls.github.io/en/, 2026.

[68] Project X Team. VLESS Protocol Configuration, 2026.

[69] Psiphon Inc. Psiphon: Uncensored internet access for Windows and mobile. https://psiphon.ca/, 2026.

[70] Eric Rescorla. Keying Material Exporters for Transport Layer Security (TLS). RFC 5705, March 2010.

[71] Marc B. Rosen, James Parker, and Alex J. Malozemoff. Balboa: Bobbing and Weaving around Network Censorship. In *USENIX Security Symposium*, 2021.

[72] RPRX. Trojan-Killer. https://github.com/XTLS/Trojan-killer/blob/main/README.md, 2023.

[73] Runa. Ethiopia Introduces Deep Packet Inspection. https://blog.torproject.org/ethiopia-introduces-deep-packet-inspection/, 2012.

[74] Runa Sandvik. Kazakhstan uses DPI to block Tor. https://gitlab.torproject.org/legacy/trac/-/work_items/6140, 2012.

[75] Runa Sandvik. UAE uses DPI to block Tor. https://gitlab.torproject.org/legacy/trac/-/work_items/6246, 2012.

[76] SekiBetu. Large scale blocking of TLS-based censorship circumvention tools in China. https://github.com/net4people/bbs/issues/129#issuecomment-1268274011, 2022.

[77] SekiBetu. Large scale blocking of TLS-based censorship circumvention tools in China. https://github.com/net4people/bbs/issues/129#issuecomment-1268028271, 2022.

[78] Jade Sheffey and Amir Houmansadr. CensorLab: A Testbed for Censorship Experimentation. https://arxiv.org/abs/2412.16349, 2025.

[79] Shelikhoo. webtunnel/transport/httpupgrade/httpupgrade.go. https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transports/webtunnel/-/blob/main/transport/httpupgrade/httpupgrade.go. Commit: 382e620e, lines 81–82.

[80] shelikhoo and ggus. Hiding in plain sight: Introducing WebTunnel. https://blog.torproject.org/introducing-webtunnel-evading-censorship-by-hiding-in-plain-sight.

[81] Meng Shen, Mingwei Wei, Liehuang Zhu, and Mingzhong Wang. Classification of Encrypted Traffic With Second-Order Markov Chains and Application Attribute Bigrams. *IEEE Transactions on Information Forensics and Security*, 2017.

[82] The Go Authors. Package tls — crypto/tls. Go Packages, 2026. Go 1.26.4, BSD-3-Clause.

[83] The Go Authors. src/crypto/tls/conn.go. https://github.com/golang/go/blob/8213feb42277dd335bbab97e8d97837d8634b0fc/src/crypto/tls/conn.go#L876, 2026.

[84] The Go Authors. src/net/tcpsock.go. https://github.com/golang/go/blob/8213feb42277dd335bbab97e8d97837d8634b0fc/src/net/tcpsock.go#L290, 2026.

[85] The Tor Project. meek. https://gitlab.torproject.org/legacy/trac/-/wikis/doc/meek.

[86] The Tor Project. Snowflake. https://snowflake.torproject.org/.

[87] The Tor Project. WebTunnel Bridge. https://community.torproject.org/relay/setup/webtunnel/.

[88] Martin Thomson and Cory Benfield. HTTP/2. RFC 9113, Internet Engineering Task Force, June 2022.

[89] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The Space of Transferable Adversarial Examples. https://arxiv.org/abs/1704.03453, 2017.

[90] trojan gfw. trojan. https://github.com/trojan-gfw/trojan, 2020.

[91] uProxy. https://www.uproxy.org/.

[92] V2Fly Community. VMess Protocol Specification. https://www.v2fly.org/en_US/developer/protocols/vmess.html, 2026.

[93] Benjamin VanderSloot, Allison McDonald, Will Scott, J. Alex Halderman, and Roya Ensafi. Quack: Scalable Remote Measurement of Application-Layer Censorship. In *USENIX Security Symposium*, 2018.

[94] Gunjan Verma, Ertugrul Ciftcioglu, Ryan Sheatsley, Kevin Chan, and Lisa Scott. Network Traffic Obfuscation: An Adversarial Machine Learning Approach. In *2018 IEEE Military Communications Conference*, 2018.

[95] Afonso Vilalonga, Joao S. Resende, and Henrique Domingos. TorKameleon: Improving Tor's Censorship Resistance with K-anonymization and Media-based Covert Channels. In *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2023.

[96] Paul Vines and Tadayoshi Kohno. Rook: Using Video Games as a Low-Bandwidth Censorship Resistant Communication Platform. In *Workshop on Privacy in the Electronic Society*, 2015.

[97] Ryan Wails, Rob Jansen, Aaron Johnson, and Micah Sherr. Proteus: Programmable Protocols for Censorship Circumvention. In *Free and Open Communications on the Internet*, 2023.

[98] Ryan Wails, Rob Jansen, Aaron Johnson, and Micah Sherr. Censorship Evasion with Unidentified Protocol Generation. In *34th USENIX Security Symposium (USENIX Security 25)*, 2025.

[99] Liang Wang, Kevin P. Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. Seeing through network-protocol obfuscation. In *Computer and Communications Security*, 2015.

[100] Qiyan Wang, Xun Gong, Giang T. K. Nguyen, Amir Houmansadr, and Nikita Borisov. CensorSpoofer: Asymmetric communication using IP spoofing for censorship-resistant web browsing. In *Computer and Communications Security*, 2012.

[101] 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*, 2012.

[102] Bas Westerbaan. Cloudflare targets 2029 for full post-quantum security. https://blog.cloudflare.com/post-quantum-roadmap/, 2026.

[103] Nimesha Wickramasinghe, Arash Shaghaghi, Gene Tsudik, and Sanjay Jha. SoK: Decoding the Enigma of Encrypted Network Traffic Classifiers. In *2025 IEEE Symposium on Security and Privacy*, 2025.

[104] Brandon Wiley. Dust: A Blocking-Resistant Internet Transport Protocol. Technical report, University of Texas at Austin, 2011.

[105] Sarah Wilson, Stella Tian, and Sina Kamali. Extended Abstract: Shaperd: Easily Adoptable Real-Time Traffic Shaper for Fully Encrypted Protocols. In *Free and Open Communications on the Internet*, 2025.

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

[107] Philipp Winter, Tobias Pulls, and Juergen Fuss. ScrambleSuit: A Polymorphic Network Protocol to Circumvent Censorship. In *Workshop on Privacy in the Electronic Society*, 2013.

[108] wkrp. Looking for: A good paper on how TLS-in-TLS detection works? https://github.com/net4people/bbs/issues/281#issuecomment-1724755111, 2023.

[109] wkrp. Blocking of Cloudflare ECH in Russia, 2024-11-05. https://github.com/net4people/bbs/issues/417, 2024.

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

[111] 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 *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[112] Diwen Xue, Michalis Kallitsis, Amir Houmansadr, and Roya Ensafi. Fingerprinting obfuscated proxy traffic with encapsulated TLS handshakes. In *USENIX Security Symposium*. USENIX, 2024.

[113] 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*, 2025.

[114] Yawning Angel. obfs4. https://gitlab.com/yawning/obfs4, 2019.

[115] yuhan6665. XTLS Vision, fixes TLS in TLS, to the star and beyond. https://github.com/XTLS/Xray-core/discussions/1295, 2022.

[116] Wenxuan Zhou, Amir Houmansadr, Matthew Caesar, and Nikita Borisov. SWEET: Serving the Web by Exploiting Email Tunnels. In *Hot Topics in Privacy Enhancing Technologies*, 2013.

Appendix A. Ethics Considerations

Observing real users' network traffic raises important privacy considerations. We therefore sought approval from our IRB for our data collection plan. Although our work was determined to be "Not Regulated," we took several steps to preserve user privacy. First, we engaged with the ISP that oversees the network tap and has extensive experience helping with such projects to ensure that we followed their internal privacy and ethics guidelines. Second, we restricted inspection to TLS traffic by considering only traffic on TCP port 443. Instead of recording raw traffic, we limited logging to the minimum information required by our TLS model. For each packet, we logged only its timestamp, direction, and size. For each flow, we logged the timestamps of each packet

Protocol	Model	Unshaped			Shaped		
		Prec.	Recall	FPR	Prec.	Recall	FPR
VLESS	LR	0.837	0.812	0.160	0.594	0.829	0.571
	MLP	0.966	0.976	0.035	0.857	0.806	0.136
	k-NN	0.979	0.587	0.013	0.855	0.179	0.031
	GBDT	0.978	0.983	0.022	0.923	0.892	0.075
VMESS	LR	0.850	0.746	0.131	0.585	0.824	0.584
	MLP	0.954	0.975	0.047	0.781	0.810	0.227
	k-NN	0.959	0.517	0.022	0.840	0.157	0.030
	GBDT	0.962	0.984	0.039	0.905	0.908	0.096

TABLE 4: **ML Classifier Detection Results with Timing Features.** Results compare normal and shaped proxy traffic against matched TLS flows using size, direction, and RTT-normalized timing features.

in the TCP three-way handshake, the TLS version, and the index of the last packet in the TLS handshake. Third, all collected data was processed directly on the network tap and was not removed from the ISP's network except in aggregated form as part of our TLS model. Finally, we disclosed the sources of distinguishability detailed in Section 4 to the corresponding circumvention tool developers.

Appendix B. Supplementary Material

B.1. *ProxyShaper* and Timing

Section 5 considers only classifiers trained on packet size and direction. *ProxyShaper* does not shape timing, as packet timings are strongly affected by network conditions and are therefore difficult to control reliably. However, as shown in Section 4, timing can itself provide distinguishing features that a censor could exploit. We therefore also evaluate *ProxyShaper* against our four ML classifiers trained on packet size, direction, *and* timing. For *ProxyShaper* traffic, the classifiers use shaped packet sizes and directions with unshaped timings. The results are shown in Table 4.

Timing provides an additional source of discriminative information, reducing but not eliminating the benefits of shaping. Across the classifiers, we generally observe a reduction in precision and increase in FPR, although the degree to which classifiers benefit from timing varies. GBDT and MLP benefit the most from the additional timing information, while LR benefits only modestly from timing and retains a high FPR. k-NN, on the other hand, becomes overly conservative because adding continuous timing features increases the dimensionality and differentiates flows that are otherwise similar in size and direction, altering its distance-based neighborhoods and reducing both FPR and recall.

B.2. Using *CenKL* to Test a New Protocol

In Figure 6, we show a detailed flowchart outlining the end-to-end testbed setup and execution. The developer first deploys their protocol in the testbed and runs a capture

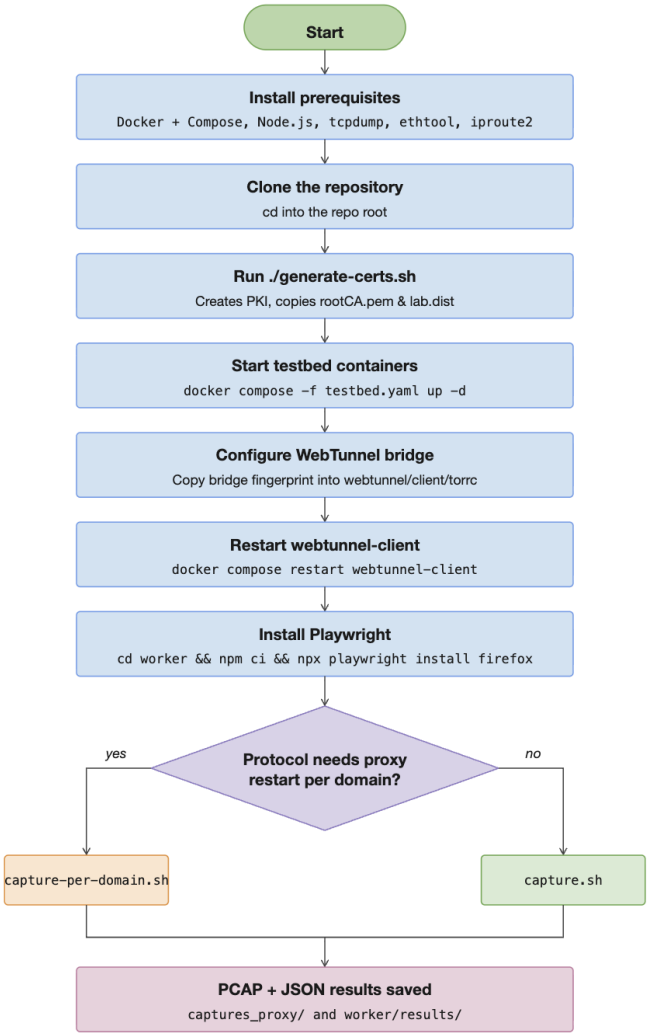


Figure 6: **Testbed Flowchart**

harness to collect traffic, which is processed by Zeek using the provided script and converted to binary flows via `obfs zeek2flows`. Next, the developer builds a traffic model from the collected flows using `obfs pipeline histograms` with the quantizer shipped with our pre-built TLS model, ensuring the two models are directly comparable. Finally, `obfs histograms divergence` computes TLS Divergence between the proxy and TLS models. The result is a per-packet-index decomposition that the developer can drill into: `obfs divergence delta` identifies which packet positions contribute most to distinguishability, and `obfs divergence terms` localizes the divergence to individual packet sequences, revealing the precise artifacts to address.

Appendix C. Meta-Review

The following meta-review was prepared by the program committee for the 2027 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

C.1. Summary

This paper proposes *CenKL*, a system for measuring how easy it is for a censor to detect obfuscated proxy traffic hidden inside TLS connections. The authors build statistical models of real TLS traffic (collected from an ISP) and of each obfuscated protocol (collected in a controlled testbed), then compare the two models using an approximation of KL divergence, which they call TLS Divergence. This gives a measure of how distinguishable a protocol is, and it can also pinpoint exactly which packets and features cause the difference.

C.2. Scientific Contributions

- Creates a New Tool to Enable Future Science
- Addresses a Long-Known Issue
- Identifies an Impactful Vulnerability
- Provides a Valuable Step Forward in an Established Field

C.3. Reasons for Acceptance

1) The paper creates a new tool to enable future science: *CenKL* is released as a complete, reusable open-source pipeline, with a documented workflow for testing new protocols. This gives developers and researchers a concrete instrument to reuse.
2) The paper addresses a long-known issue, namely how to shape the traffic. In previous literature, there has been a lot of work on how to shape payloads, but far less on how to shape sizes and timings. This paper fills this gap.
3) The paper identifies several impactful vulnerabilities: It applies *CenKL* to nine widely used protocols, revealing concrete, previously undocumented weaknesses in real deployed software, each traced to a specific, fixable cause.
4) The paper provides a step forward: Prior work mostly measured distinguishability by training a specific classifier, which only guards against classifiers researchers happen to think of. *CenKL* instead gives a classifier-agnostic, information-theoretic measure.

C.4. Noteworthy Concerns

1) *ProxyShaper* reproduces recorded TCP packet sizes but does not reproduce recorded TLS record sizes if the recorded traces contain anything other than a single record per packet (*i.e.*, if there is any TLS record fragmentation or coalescing across TCP packets). The tested classifiers only look at packet sizes and thus do not pick up on this discrepancy.
2) The *ProxyShaper* defense may not work well enough to defeat a censor. For both shaped VLESS and VMESS, the GBDT attack with timing has recall about 0.9 and FPR below 0.1 (without timing, recall was about 0.7 and FPR about 0.15); see Table 4 in Appendix B. This attack is not tuned for low FPR, and so some recall may be possible to be traded for even lower FPR, and the tradeoff may be improved by using state-of-the-art classifiers, such as those using deep learning.
3) The paper's TLS model applies Laplace smoothing, while the obfuscated-protocol models it is compared against do not (Section 3.2). This asymmetry is a deliberate design choice, explained by the one-way nature of the comparison, but the paper does not discuss how it affects the fidelity of TLS Divergence as an approximation of KL divergence. Namely, whether and in which direction the asymmetric smoothing might bias the reported divergence values.
4) The divergence numbers reported in Section 4 and Figure 3 are not paired with any measure of statistical uncertainty. This is probably not a significant problem for the TLS model, which is built from a very large dataset (Section 3.1), but some obfuscated-protocol datasets are considerably smaller (Section 3.2), and some of the discussed patterns occur very rarely, thus increasing the risk that reported divergence contributions reflect sampling noise rather than a genuine, stable distinguishing feature.
5) The paper selects the number of quantization bins based on a sample-complexity bound that guarantees, for chosen values of ϵ and δ, that the total variation distance between each estimated distribution and its true discretized counterpart is at most ϵ with probability at least $1 - \delta$; the paper reports using $\epsilon = \delta = 0.05$ (Section 4). However, the paper does not verify empirically, *e.g.*, on held-out data, that the resulting fitted models actually achieve this accuracy guarantee in practice.