OpenVPN is Open to VPN Fingerprinting

Diwen Xue* Reethika Ramesh* Arham Jain* Michalis Kallitsis†
J. Alex Halderman* Jedidiah R. Crandall‡ Roya Ensafi*
*University of Michigan †Merit Network, Inc.
‡ Arizona State University/Breakpointing Bad

Abstract

VPN adoption has seen steady growth over the past decade due to increased public awareness of privacy and surveillance threats. In response, certain governments are attempting to restrict VPN access by identifying connections using "dual use" DPI technology. To investigate the potential for VPN blocking, we develop mechanisms for accurately fingerprinting connections using OpenVPN, the most popular protocol for commercial VPN services. We identify three fingerprints based on protocol features such as byte pattern, packet size, and server response. Playing the role of an attacker who controls the network, we design a two-phase framework that performs passive fingerprinting and active probing in sequence. We evaluate our framework in partnership with a million-user ISP and find that we identify over 85% of OpenVPN flows with only negligible false positives, suggesting that OpenVPN-based services can be effectively blocked with little collateral damage. Although some commercial VPNs implement countermeasures to avoid detection, our framework successfully identified connections to 34 out of 41 "obfuscated" VPN configurations. We discuss the implications of the VPN fingerprintability for different threat models and propose short-term defenses. In the longer term, we urge commercial VPN providers to be more transparent about their obfuscation approaches and to adopt more principled detection countermeasures, such as those developed in censorship circumvention research.

1 Introduction

ISPs, advertisers, and national governments are increasingly disrupting, manipulating, and monitoring Internet traffic [16, 22, 27, 47, 69]. As a result, virtual private network (VPN) adoption has been growing rapidly, not only among activists and journalists with heightened threat models but also among average users, who employ VPNs for reasons ranging from protecting their privacy on untrusted networks to circumventing censorship. As a recent example, with the passage of Hong Kong's new national security law, popular VPN providers observed a 120-fold surge in downloads due to fears of escalating surveillance and censorship [62].

In response to the growing popularity of VPNs, numerous ISPs and governments are now seeking to track or block VPN traffic in order to maintain visibility and control over the traffic within their jurisdictions. Binxing Fang, the designer of the Great Firewall of China (GFW) said there is an "eternal war" between the Firewall and VPNs, and the country has ordered ISPs to report and block personal VPN usage [60,61]. More recently, Russia and India have proposed to block VPN services in their countries, both labeling VPNs a national cybersecurity threat [44, 59]. Commercial ISPs are also motivated to track VPN connections. For example, in early 2021, a large ISP in South Africa, Rain, Ltd., started throttling VPN connections by over 90 percent in order to enforce quality-of-service restrictions in their data plans [64].

ISPs and censors are known to employ a variety of simple anti-VPN techniques, such as tracking connections based on IP reputation, blocking VPN provider (provider from hereon) websites, and enacting laws or terms of service forbidding VPN usage [46,53,60]. Yet, these methods are not robust; motivated users find ways to access VPN services in spite of them. However, even less-powerful ISPs and censors now have access to technologies such as carrier-grade deep packet inspection (DPI) with which they can implement more sophisticated modes of detection based on protocol semantics [43, 48].

In this paper, we explore the implications of DPI for VPN detection and blocking by studying the fingerprintability of OpenVPN (the most popular protocol for commercial VPN services [6]) from the perspective of an adversarial ISP. We seek to answer two research questions: *(1) can ISPs and governments identify traffic flows as OpenVPN connections in real time*? and *(2) can they do so at-scale without incurring significant collateral damage from false positives?* Answering these questions requires more than just identifying fingerprinting vulnerabilities; although challenging, we need to demonstrate practical exploits under the constraints of how ISPs and nation-state censors operate in the real world.

We build a detection framework that is inspired by the architecture of the Great Firewall [1, 11, 71], consisting of *Filter* and *Prober* components. A *Filter* performs passive filtering over passing network traffic in real time, exploiting protocol quirks we identified in OpenVPN's handshake stage. After a flow is flagged by a *Filter*, the destination address is passed

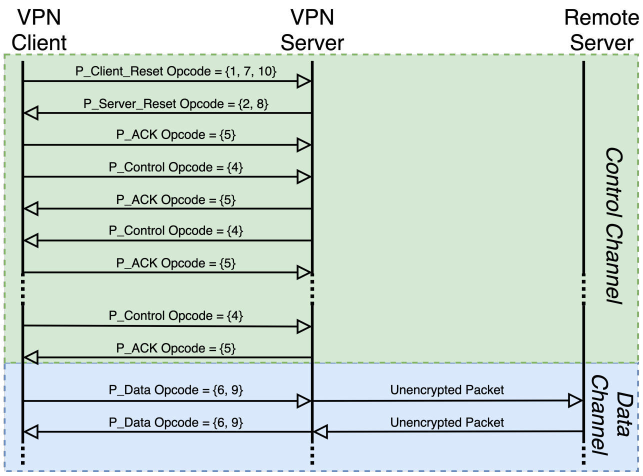


Figure 1: **OpenVPN Session Establishment (TLS mode).**

to a *Prober* that performs active probing as confirmation. By sending probes carefully designed to elicit protocol-specific behaviors, the *Prober* is able to identify an OpenVPN server using side channels even if the server enables OpenVPN's optional defense against active probing. Our two-phase framework is capable of processing ISP-scale traffic at line-speed with an extremely low false positive rate.

In addition to core or "vanilla" OpenVPN, we also include commercial "obfuscated" VPN services in this study. In response to increasing interference from ISPs and censors, obfuscated VPN services have started to gain traction, especially from users in countries with heavy censorship or laws against the personal usage of VPNs. Obfuscated VPN services, whose operators often tout them as "invisible" and "unblockable" [5, 49, 54], typically use OpenVPN with an additional obfuscation layer to avoid detection [2, 66].

Partnering with *Merit* (a mid-size regional ISP that serves a population of 1 million users), we deploy our framework at a monitor server that observes 20 Gbps of ingress and egress traffic mirrored from a major *Merit* point-of-presence. (Refer to § 5 for ethical considerations.) We use PF_RING [38] in zero-copy mode for fast packet processing by parallelized *Filters*. In our tests, we are able to identify 1718 out of 2000 flows originating from a control client machine residing within the network, corresponding to 39 out of 40 unique "vanilla" OpenVPN configurations.

More strikingly, we also successfully identify over two-thirds of obfuscated OpenVPN flows. Eight out of the top 10 providers offer obfuscated services, yet *all* of them are flagged by our *Filter*. Despite providers' lofty unobservability claims (such as "… even your Internet provider can't tell that you're using a VPN" [49]), we find most implementations of obfuscated services resemble OpenVPN masked with the simple XOR-Patch [36], which is easily fingerprintable. Lack of random padding at the obfuscation layer and co-location with vanilla OpenVPN servers also make the obfuscated services more vulnerable to detection.

In a typical day, our single-server setup analyzes 15 TB of traffic and 2 billion flows. Over an eight-day evaluation, our framework flagged 3,638 flows as OpenVPN connections. Among these, we are able to find evidence that supports our detection results for 3,245 flows, suggesting an upper-bound false-positive rate three orders of magnitude lower than previous ML-based approaches [3, 14, 26].

We conclude that tracking and blocking the use of OpenVPN, even with most current obfuscation methods, is straightforward and within the reach of any ISP or network operator, as well as nation-state adversaries. Unlike circumvention tools such as Tor or Refraction Networking [8, 74], which employ sophisticated strategies to avoid detection, robust obfuscation techniques have been conspicuously absent from OpenVPN and the broader VPN ecosystem. For average users, this means that they may face blocking or throttling from ISPs, but for high-profile, sensitive users, this fingerprintability may lead to follow-up attacks that aim to compromise the security of OpenVPN tunnels [40, 51]. We warn users with heightened threat models *not* to expect that their VPN usage will be unobservable, even when connected to obfuscated services. While we propose several short-term defenses for the fingerprinting exploits described in this paper, we fear that, in the long term, a cat-and-mouse game similar to the one between the Great Firewall and Tor is imminent in the VPN ecosystem as well. We implore VPN developers and providers to develop, standardize, and adopt robust, well-validated obfuscation strategies and to adapt them as the threats posed by adversaries continue to evolve.

2 Background & Related Work

VPN tools create private networks across the public Internet through encrypted tunneling. Although many VPN protocols are being used, such as IPSec and WireGuard, OpenVPN remains the most supported and trusted protocol among commercial VPN providers [6]. Due to its versatility and open-source nature, OpenVPN has been used as the underlying protocol in numerous VPN products, which often advertise the protocol for its proven security [66]. In addition, OpenVPN's popularity continues to rise with the trend of users choosing to self-host open-source VPN tools [65].

OpenVPN Protocol. OpenVPN was first released in 2002 with the aim of creating a tunneling protocol focusing on security, while also being free and fast over the standard TCP and UDP [34]. When the OpenVPN tunnel is active, raw IP packets being sent to or from the tunnel to the final destination are encapsulated inside OpenVPN packets. To achieve secure communication, OpenVPN leverages the OpenSSL library as its cryptographic layer. Two methods for authentication and key exchange are provided to establish trust with peers: either pre-shared static key(s) or TLS-based negotiations. The latter has been adopted by the majority of commercial VPN services. Two separate channels are used for key exchange and data transfer, both sharing a single multiplexed TCP/UDP stream. In the control channel, the client and server engage in a TLS-style exchange of key materials. As TLS is designed to operate over a reliable transport, OpenVPN provides its control channel with a sequential, reliable layer based on an explicit acknowledgement and re-transmission mechanism. The negotiated key from the control channel will be used to encrypt packets transferred in the data channel, which does not provide any reliability guarantee. Figure 1 presents a typical initialization sequence of OpenVPN packets leading to a fully encrypted data channel.

Tor, Proxy, and VPN Detection. The ongoing arms-race between the GFW and Tor has been extensively studied and is most representative of the conflict between censorship & surveillance and circumvention tools [9, 11, 12, 55, 56, 71]. Censors started by blocking Tor's website and public relays, which Tor responded to by deploying website mirrors and private, unpublished bridges. Next, censors moved to blocking with DPI by fingerprinting Tor's TLS handshake, e.g. cipher suites. Tor used Pluggable Transports (PT) obfuscators, such as Obfsproxy and meek [39], to mask the handshake. In response, censors deployed active probing to complement DPI-based fingerprinting to detect Tor and certain obfuscators.

There is limited previous work focusing on VPN traffic detection. Hoogstraaten [19] explored server-side VPN detection methods, ranging from using existing information databases (e.g. WHOIS, rDNS) to fingerprinting TCP options (e.g. advertised MSS). Webb et al. [70] proposed detecting proxies and VPNs based on traffic timing and latency. Their approach relied on the hypothesis that when a service is accessed through a proxy, the RTT measurement will be different from the RTT of a direct connection. Another class of previous work uses computational and machine learning models to passively detect VPN traffic [3,14,15,17,24,26,68], leveraging flow-level statistics such as connection duration and packet interval. Most of this work uses the same synthetic `ISCXVPN2016` dataset [17]—which contains a balanced mixture of VPN and non-VPN traffic—to train and test a variety of machine learning and neural network classifiers in an offline, lab-setting. In contrast, our work primarily focuses on whether ISP-level adversaries can identify OpenVPN flows in near real time, and whether they can do so at scale, under practical constraints, and with minimal collateral damage. For this reason, we omit a full analysis of ML-based work, and only compare them with our approach in terms of false positives (falsely blocking legitimate traffic).

Obfuscated (Open)VPN. Various traffic obfuscation techniques have been examined in previous work. Wang et al. examined the detectability of Obfsproxy, FTE, and meek [67]. Using attacks based on protocol semantics, packet entropy, and timing-related features, they concluded that a determined censor could detect all three obfuscators reliably. Houmansadr et al. demonstrated that popular mimicry-based obfuscation tools failed to achieve unobservability because seamlessly simulating another protocol is extremely challenging [20]. Previous studies have suggested censors can use active probing to detect proxies that obfuscate traffic [1,11,71]. In response, "probe-resistant" proxies were developed, which remain silent when being probed by an unauthenticated adversary. However, researchers have demonstrated that carefully designed probes could still identify these proxies [13].

There is a marked demand for an emerging class of services called "stealth" or "obfuscated" VPN, especially from users in countries with heavy censorship or laws against personal VPN usage [60, 63]. Most obfuscated VPN services use OpenVPN as the underlying protocol for security and routing, with an obfuscation layer overlaid to avoid detection [2,66] [1]. OpenVPN's core developers prefer that obfuscation remains a separate project operating alongside the vanilla/core protocol, as they "do not want to play the cat-and-mouse game [as Tor]" [35]. The absence of a standardized obfuscation solution has led to a plethora of obfuscators implemented by different VPN providers, who often claim that their obfuscated services can remain undetected by ISPs and censors alike. For example, TorGuard introduces their obfuscated VPN service as "Engineered from the ground up to be impossible to detect" [54]. BolehVPN claims that their VPN obfuscation "…keeps you out of trouble, even in China" [5]. Common obfuscation strategies adopted by commercial VPNs include employing XOR-based scramblers, wrapping OpenVPN inside encrypted tunnels, or using proprietary protocols.

OpenVPN XOR Patch: Originally developed by Clayface as a patch for vanilla OpenVPN, the XOR patch scrambles a packet by either xor-ing bytes with a pre-shared key, reversing the order of the bytes, xor-ing each byte with its position, or a combination of these steps [36]. Notably, OpenVPN developers discourage its use due to the lack of code audit [57].

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

Proprietary Protocols: A few VPN providers have developed proprietary obfuscated protocols, some of which are built on top of OpenVPN with a proprietary obfuscation layer added, such as VyprVPN or Astrill [2, 66].

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

[1] There are discussions on obfuscating WireGuard [72, 73], but to the best of our knowledge, they have yet to be deployed by any commercial VPNs

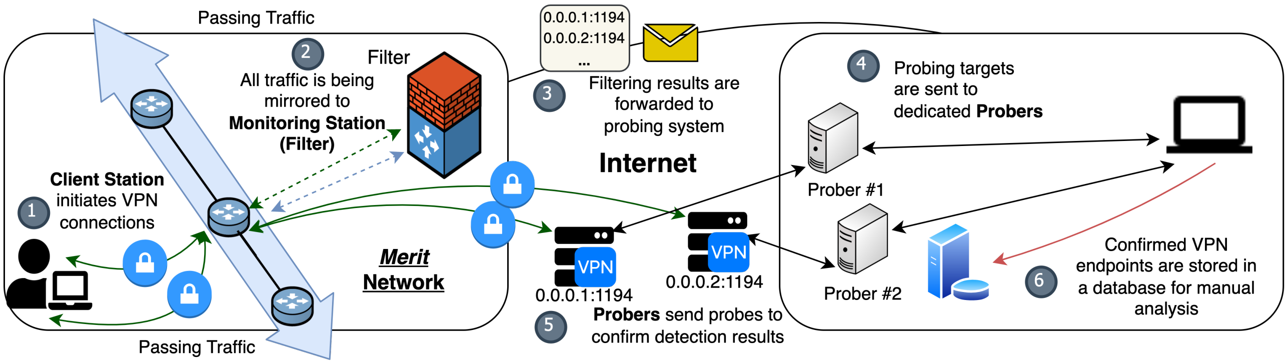

Passing Traffic
2
Filter
All traffic is being mirrored to
Monitoring Station (Filter)
0.0.0.1:1194
0.0.0.2:1194
...
3
Filtering results are forwarded to probing system
Internet
4
Probing targets are sent to dedicated Probers
Prober #1
Prober #2
Client Station initiates VPN connections
1
Merit Network
VPN
0.0.0.1:1194
VPN
0.0.0.2:1194
5
Probers send probes to confirm detection results
Confirmed VPN endpoints are stored in a database for manual analysis
6
Passing Traffic

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

3 Challenges in Real-world VPN Detection

Effective investigation of fingerprintability requires incorporating perspectives of how ISPs and censors operate in practice. It is not enough to simply identify fingerprinting vulnerabilities, we need to demonstrate realistic exploits to illustrate the practicality of exploiting the vulnerability, while taking into consideration the ISP and censors' capabilities and constraints [56]. For instance, previous academic works considered using flow-level features to train ML classifiers for VPN detection [3, 14, 17, 24, 26, 68]. Yet, it remains unclear how practical these detection approaches are for ISPs and censors, and we know of no rigorous studies that examine real-world deployment of an ML-based censorship system [56]. Furthermore, previous works test on the `ISCXVPN2016` dataset [17] with balanced OpenVPN and non-VPN traffic. However, we note that due to the low base rate of VPN traffic in the wild, even the best-performing ML system has false positive rates that can be economically impractical for real-world censors sensitive to collateral damage [67].

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

4 Adversary Model and Deployment

We assume a realistic censor (ISP) capability model based on knowledge from previous measurement studies on the arms race between censors and circumventors [1, 11, 56, 71]. We outline a censor-controlled on-path filter that passively observes and examines passing network traffic. The filter is stateful, but has limited resources and can maintain a limited amount of per-connection states for a short time. The filter is also constrained by long-term data storage and computational resources. In addition to filters installed inside the monitored networks, we assume the censor also operates measurement machines that can send protocol-specific probes to further confirm the detection result. Such two-phase systems have already been adopted by real-world censors such as the GFW against Tor and Shadowsocks [1, 71]. Finally, we expect the censor is familiar with the protocol of interest and has access to the different obfuscators deployed by VPN providers (e.g., as a paid customer). We emphasize that this threat model corresponds to censor's capabilities as observed in practice *today*, rather than future capabilities.

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

5 Ethics, Privacy, and Responsible Disclosure

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

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

Our framework is fine-tuned on both real and lab-generated traffic data, and it is evaluated on live ISP traffic. For controlled fine-tuning, a small traffic snapshot (the ISP Dataset in section 7) was used to calibrate parameters, e.g., the size of observation window. The traffic snapshot, sampling 1/30 of all flows for 45 minutes on July 28, 2021, was generated and analyzed entirely on *Merit* systems, with security mechanisms limiting access to select members of the team. As with the design described in Section 6, *Filter* analyzed only the first payload byte, completely ignoring the remainder of the payload, and it recorded only the observed degree of variation. The raw snapshot was never inspected by humans and was deleted after the fine-tuning concluded.

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

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

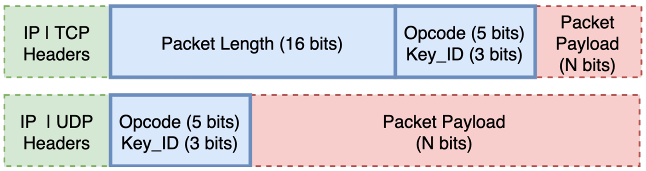


Figure 3: **OpenVPN Header in TCP and UDP modes.** (TLS only)

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

6 Identifying Fingerprintable Features

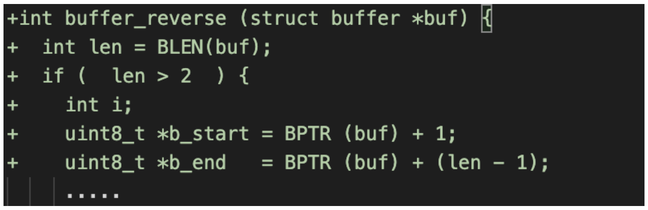
In this section, we identify three features that fingerprint OpenVPN, exploiting byte pattern, packet length, and server behaviors, respectively.

6.1 Opcode-based Fingerprinting

As shown in Figure 3, each OpenVPN packet has a header of 24 bits in TCP mode or 8 bits in UDP mode, which is not part of the encrypted payload. Each OpenVPN header starts with an opcode that specifies the message type of the current packet and a key ID that refers to a (new) TLS session. The opcode field can take over 10 defined values, corresponding to message types transmitted during different communication stages. A typical OpenVPN session starts with the client sending a `Client Reset` packet. The server then responds with a `Server Reset` packet, and a TLS handshake follows. OpenVPN packets that carry TLS ciphertexts have `P_Control` as their message type. Since OpenVPN can run over UDP but has to provide a reliable channel for TLS, each `P_Control` packet is explicitly acknowledged by `P_ACK` packets. Finally,

Algorithm 1 Opcode Fingerprinting Logic

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



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

Figure 4: **XOR-Patch that leaves first byte un-reversed**

actual payloads are transmitted as `P_Data` packets. Figure 1 illustrates this packet exchange with opcode annotations.

A packet field taking a fixed number of values can be easy to fingerprint and has been exploited before against other protocols [1]. We fingerprint OpenVPN's handshake sequence by analyzing each opcode byte for the first *N* packets of a flow (the threshold *N* is explored in Section 7.2). Algorithm 1 shows the process of opcode fingerprinting, with `Opcode` referring to the sequence of *N* opcode values found in the first *N* packets of a given flow. Briefly, the filter flags a flow if the number of different opcodes observed accords with the protocol *and* the `Client` and `Server Resets` are not seen once the handshake is completed.

Previous work and existing open-source DPIs [23, 29, 37, 75] considered statically matching opcode values and packet sizes based on the protocol specification. In contrast, we propose to dynamically capture the variation in opcode values that reflects the establishment of OpenVPN sessions. Notably, our heuristics do not require exact matching of opcode values or packet length (e.g., do not require the third byte of the first packet to be 0x38), thereby ensuring it works effectively against XOR-obfuscated flows. The XOR obfuscation masks packet payloads to ensure that the opcode bytes are altered. Notably, according to the specification [36], when it reverses the packet as one of the obfuscation steps, it excludes the first character of the buffer (where the opcode byte is located) from reversal, as shown in Figure 4. As such, the opcode byte is always XOR-ed with the same byte of the XOR key, and the same opcodes would be mapped to the same value after obfuscation. This behavior is preserved when Tunnelblick (a popular OpenVPN client on macOS) adopts the patch [57], and has been used in multiple mobile apps [76]. By considering only the number of unique opcodes seen so far, our heuristics are more flexible and target various XOR-based obfuscations of OpenVPN.

6.2 ACK-based Fingerprinting

OpenVPN engages in a TLS-style handshake with its peer over the control channel. Since TLS is designed to operate over a reliable layer, OpenVPN implements an explicit acknowledgement and re-transmission mechanism for its control channel messages [30]. Specifically, incoming `P_Control` packets are acknowledged by `P_ACK` packets, which do not carry any TLS payloads and are uniform in size (Note these ACK packets are carried over by TCP as payload and are not the same as TCP ACK flags). Moreover, these ACK packets are seen mostly only in the early stage of a flow, during the handshake phase, and are not used in the actual data transfer channel, which can run over an unreliable layer.

To our knowledge, we are the first to devise fingerprinting attempts based on the distinct protocol-layer ACKs against OpenVPN. Previously, the unique timing pattern in meek's TCP-level ACK traffic has rendered the obfuscation tool vulnerable to detection [67]. For OpenVPN, the presence of explicit ACK packets, uniform in size and only seen in some parts of a session, provides another fingerprintable feature. Specifically, we first identify a likely ACK packet of a session by locating an initial packet exchange sequence of C->S (Client-Reset), S->C (Server-Reset), C->S (ACK), C->S (Control), as illustrated in Figure 1. For vanilla OpenVPN and XOR-based obfuscation, the first ACK packet usually appears as the third (data) packet transmitted in a session. For tunnels or obfuscators that have their own handshake or key exchange process (e.g., Stunnel, SSH tunnel, or Obfsproxy), this counting is offset by the number of tunnel handshake packets. Next, we group packets into 10-packet bins, and we derive the ACK fingerprint for each flow by counting the number of packets in each bin that have the same size as the identified ACK packet. For OpenVPN flows, we expect to observe a high number of ACK packets in early bins and an absence of them in later bins. (Later in the session, Control and ACK packets can be exchanged again to transfer random key materials, but it is not expected to be observed within our observation window N.) This approach proves effective to fingerprint vanilla OpenVPN as well as obfuscated services running over encrypted tunnels that lack random padding. We quantify exact fingerprinting thresholds in Section 7.1.

6.3 Active Server Fingerprinting

We explore the feasibility of identifying an OpenVPN server through active probing. Typically, OpenVPN servers respond to a client reset with an explicit server reset, thereby giving

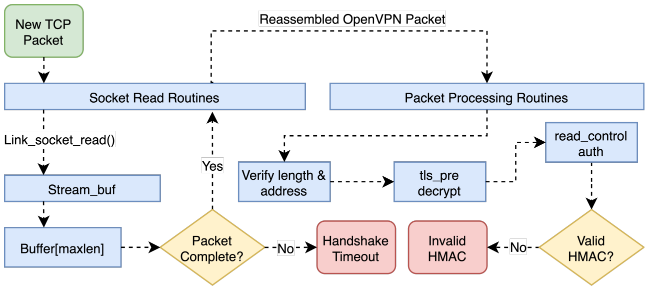


Figure 5: **OpenVPN TCP new packet processing routines**

away their identity. However, most commercial providers now have adopted *tls-auth* or *tls-crypt* options [50]. These options add an additional HMAC signature—signed by a pre-shared key—to every control channel packet for integrity verification, including the initial reset packets. With either of these options enabled, an OpenVPN server would not respond to an unauthenticated client reset with a server reset, but would instead drop such packets without further processing. The presence of such HMAC mechanism increases the complexity of doing active probing: it effectively makes OpenVPN servers "probe-resistant" [13] by remaining silent when probed by an unauthenticated client.

In fact, similar HMAC mechanisms are used by more popular "probe-resistant" proxies, such as obfs4 [33]. However, unlike obfs4 which waits for a server-specific random delay before dropping an unauthenticated connection, OpenVPN always *immediately* closes the connection if a valid HMAC cannot be located. We design our probes to leverage this protocol-specific behavior, and as a result, we manage to fingerprint OpenVPN servers even if they do not respond throughout our probing cycles. The key concept is that although the application may not respond to probing, an attacker may still be able to fingerprint application-specific thresholds at the TCP level, such as timeouts or RST thresholds, as demonstrated by Frolov et al. [13].

We use two datasets in this section to help with designing probes. **ZMap Set**: to construct a realistic non-VPN endpoints dataset, we use ZMap to scan each of the 65,535 TCP ports over the entire IPv4 space, limiting results for each port to 200 endpoints (with the specific port open), resulting in over 13 million endpoints. **Censys Set**: We query the `Censys.io` [10] database for hosts with TCP port 1194/OpenVPN open. Next, we probe each endpoint with a typical OpenVPN `Client Reset` and group endpoints that respond with explicit `Server Resets`. This results in 180,858 hosts known to be OpenVPN endpoints (with "tls-auth" disabled).

6.3.1 Base Probes

We design probes exploiting a behavior associated with how OpenVPN packetizes TCP streams. When OpenVPN operates over TCP, it needs to split the continuous stream into discrete OpenVPN packets. Figure 5 presents a high-level abstraction of this process. The most relevant parts are: a buffer is allocated in memory to reassemble fragments of OpenVPN packets encapsulated in TCP streams. The length N for the next OpenVPN packet is extracted from the first two bytes of the header (see Figure 3), and the routine keeps reading N additional bytes before it returns the reassembled packet to the caller. This means that an OpenVPN packet will not be parsed and checked for syntax and encryption errors until all its parts arrive at the server. Based on this behavior, we design two sequential probes to trigger an OpenVPN server into different code paths—which result in different connection timeouts—and measure the time elapsed before the server responds or terminates the connection. As shown in Table 1, *Base Probe 1* carries a typical 16-byte OpenVPN `Client Reset`, while *Base Probe 2* has the same payload with the last byte stripped off. The assumption is since our two probes only differ in one byte, most non-OpenVPN servers will respond to our probes in a similar way. However, for an OpenVPN server with HMAC enabled, the connection sending the first probe will be dropped immediately because the OpenVPN packet is reassembled and a valid HMAC cannot be located. The second probe will not receive an immediate response, as the server will wait for an additional byte to arrive for reassembly. The connection will stay idle until a server specific handshake timeout has passed, after which the connection will be dropped. As such, the first probe will be dropped at the decryption routine, while the second probe will be dropped at the packet reassembly routine, both labeled red in Figure 5.

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

Table 1: **Summary of Probes and the expected behaviors from an OpenVPN server.**

6.3.2 Additional Probes

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

We design additional probes based on the fact that OpenVPN validates packet length and will drop connections sending invalid length without waiting for the next packet to be

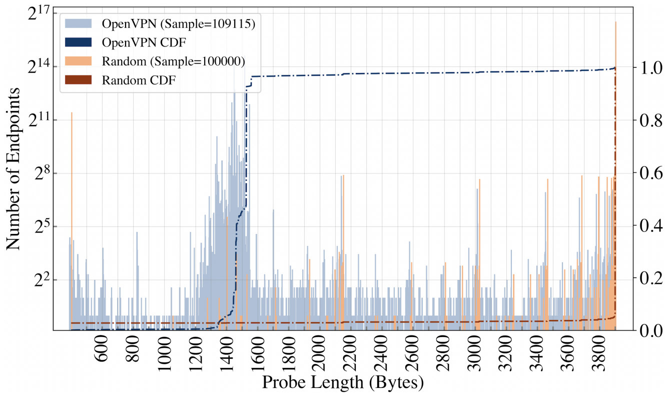


Figure 6: **RST thresholds for OpenVPN and random endpoints.**

reassembled. Here, packet length refers to the length declared by the first two bytes of an OpenVPN header (see Figure 3), rather than the TCP packet length. A "valid" length is in the range of [1, *max_len*], where *max_len* is derived from the server's MTU configurations. For instance, default TUN MTU of 1500 bytes, combined with overheads (crypto IV, packet length, *etc.*), results in a *max_len* of 1627 bytes. In this case, probes whose first two bytes have a decimal value greater than 1627 (0x06,0x5B) will be dropped immediately.

We also design probes leveraging the way a Linux server closes a TCP connection. When a TCP connection terminates, the operating systems at both ends typically complete a FIN 4-way handshake. However, previous work has found that if a connection is closed with unread bytes in buffer, Linux will send a RST packet [13]. A server's "RST Threshold" is defined as the minimum number of bytes needed to send to the server to trigger a RST. We determine the RST threshold distribution for both *ZMap Set* and *Censys Set*. As shown in Figure 6, the vast majority of OpenVPN servers have a RST threshold around 1550-1660 bytes, corresponding to buffers allocated with typical MTU configurations. In contrast, over 97% of random ZMap endpoints have a RST threshold less than 500 or greater than 4000. We therefore construct an additional probe with 2,000 random bytes, which we expect over 98% of legitimate OpenVPN servers and less than 3% of random servers to respond to with RST packets.

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

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

Table 1 lists all probes and the expected behaviors from an OpenVPN server. An evaluation process is shown in Appendix Figure 11.

6.4 Constructing *Filters* and *Probers*

Our *Filter* performs both opcode and ACK-based fingerprinting, flagging a flow if at least one fingerprint matches. This is because the opcode and ACK fingerprints are designed to be complementary: both are effective against vanilla OpenVPN and they each target a specific subset of obfuscations. The former works against XOR-based obfuscations that work like Vigenère ciphers, i.e. they always encrypt the same plaintext opcodes at the same position to the same ciphertext bytes. The latter targets tunneling-based obfuscation that lacks random padding and preserves the 1:1 correspondence between the original and obfuscated packet streams. Combining the two features maximizes our fingerprinting coverage, as we discovered that even within the same provider, obfuscating strategies can vary a lot (§ 9). Table 5 in Appendix shows the effectiveness of each feature against each commercial VPN service we tested. Following *Filter*'s result, the *Prober* performs the active probing scheme to further lower potential false positives.

We implement the *Filter* in Zeek [75], an open-source network monitoring tool. We note that the evaluation processes for opcode and ACK-based fingerprinting are quite simple: both only require several dozen integer comparisons (limited by the observation window) while maintaining a small number of per-flow states. We implement the *Prober* in Nim [31]. We believe that both components can be easily deployed by any ISP or censor.

7 Fine-tuning for Deployment

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

We use two datasets here. **ISP Dataset**: we collected a snapshot of network traffic going through a server installed within *Merit*. Over 45 minutes on July 28, 2021, we sampled 1/30 of all flows passing through the server, resulting in 461 GB of traffic that corresponds to 221,534 flows with

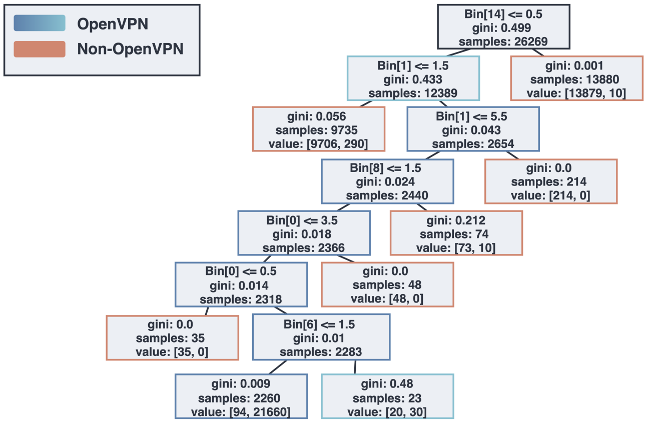


Figure 7: **ACK fingerprint DT from the ISP and VPN datasets.**

Bin Number	Threshold
1	$1 \leq Bin[1] \leq 3$
2	$2 \leq Bin[2] \leq 5$
For i in range $[3,5]$	$Bin[i] \leq 5$
For i in range $[6,N/10]$	$Bin[i] \leq 1$

Table 2: **Set of thresholds for ACK Filtering.**

full packet payloads. Refer to § 5 for details on how this data was handled to limit privacy risks. **VPN Dataset**: we collected traces from 20 commercial VPN providers as well as 2 self-hosted OpenVPN services (Streisand, OpenVPN Access Server) following the automated process described in Section 8. Note the 20 VPN providers do not overlap with the providers used in evaluation. For each provider, we repeated the trace collection process 50 times each in TCP and UDP mode, resulting in a 7.65 GB dataset comprised of 2,200 vanilla OpenVPN traces.

7.1 ACK Fingerprint Thresholds

We quantify the exact ACK fingerprint based on the *ISP* and *VPN Dataset*. We only include flows with at least 150 data packets (15 bins), which leaves us with 24,069 ISP flows and 2,200 VPN flows. A classification decision tree is constructed based on the two labeled sets with weights applied to account for the imbalanced data size. Figure 7 shows the constructed tree (depth and leaf limited, a complete graph can be found in Appendix Figure 12). The ACK fingerprint is a sequence of thresholds based on the derived decision tree, as shown in Table 2. ($Bin[i]$ refers to the number of ACK-size packets for i^{th} Bin.)

7.2 Choice of Observation Window N

Previous works attempt to identify VPN traffic only *after* the flow terminates, making use of aggregated statistics such as connection duration [17, 24, 26]. However, detecting disallowed traffic only after the flow is finished may be of limited interest to a real-world censor [4]. We therefore have two objectives for our *Filter*: to reduce probing targets by being as selective as possible, *and* to detect OpenVPN as soon as possible within a flow.

Inspired by [4, 67], we consider the windowing strategy of limiting the inspection to only the first N data packets of a flow. We tested N from [10, 20, 30, ... 200] on the *ISP* and *VPN Dataset*. As shown in Figure 8 (a), the number of ISP flows that are flagged by the *Filter* declines from over 62,000 to 322 as we increase the observation window. However, we note that a window size of 100 packets has already achieved a precision within 2% of the best performing (200 packets).

Detection Speed and Potential Impact on Blocking A smaller window size can sever a connection at an earlier stage, thereby reducing transfer of data to a censored endpoint, while a more conservative windowing strategy excels at accuracy. In our deployment, we use 100 packets as the window size to balance detection speed and accuracy. To put this choice into perspective, we note that the Great Firewall of China (GFW) was previously observed to send confirmation probes to suspected endpoints in 15-minute intervals, and it has only recently moved to near real-time operation [11]. Recent work on how it detects Shadowsocks shows the median delay between the beginning of a connection and probing is about a minute, with probes being replayed for up to 47 times for confirmation [1]. In comparison, our deployment with a window size of 100 packets gives a median time of 7.9 seconds for the filter to flag an OpenVPN connection. We believe even with this delay, our system is still useful for censors who are interested in blocking OpenVPN connections. In addition, we note that a motivated adversary can further optimize this delay and speed the detection by tuning window size and probing rate, but with some potential loss of accuracy.

7.3 Effects of Packet Loss

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

We configure Zeek to ignore events signaling new packets with different probabilities in order to simulate random packet loss. We test loss rates from 1% up to 80%. The experiment is repeated three times, and the average result is reported in Figure 8 (b). We find that packet loss starts to have noticeable effects on the *Filter*'s outputs once the loss rate surpasses 10%. Notably, when packet loss starts to affect the detection

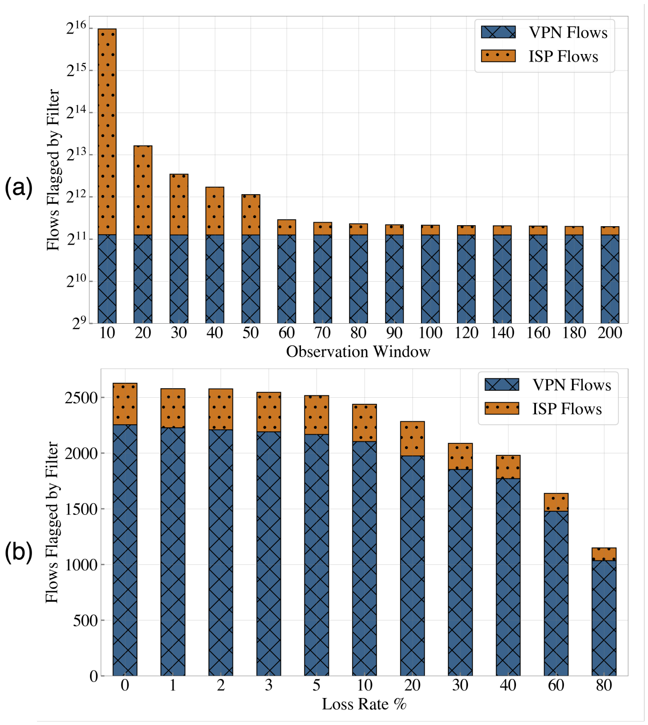


Figure 8: **Effects of (a) observation window and (b) average packet loss rate on *Filter*'s performance**

accuracy, both opcode and ACK fingerprint vulnerabilities always produce *underblocking* instead of *overblocking*, which is favoured by real-world censors [56]. Still, in order to minimize the effects of packet loss, we always configure the Monitoring Station to sample flows with a given rate (adjusted with CPU resources and traffic volume) for our online evaluation.

7.4 Server Churn for Asynchronous Probing

After the *Filter* generates a list of probing targets, the *Prober* can either send probes synchronously as soon as a target is emitted, or asynchronously, waiting for a pre-configured interval before sending probes to targets in batches. Sending probes synchronously has the advantage of obtaining the most accurate results before the server IP is churned. However, this requires the probing system to be online the whole time. In contrast, sending probes in batches is more efficient and easier to manage, but the server IP may be churned if the interval between the filtering phase and the probing phase is excessively long. We explore a probing frequency that achieves efficiency and accounts for possible server IP churn. To do this, we monitor the 180,858 known OpenVPN servers from the *Censys Set* described in 6.3. Starting from August 2nd, 18:00 EDT, we probe the servers every 3 hours for a week and record their responses.

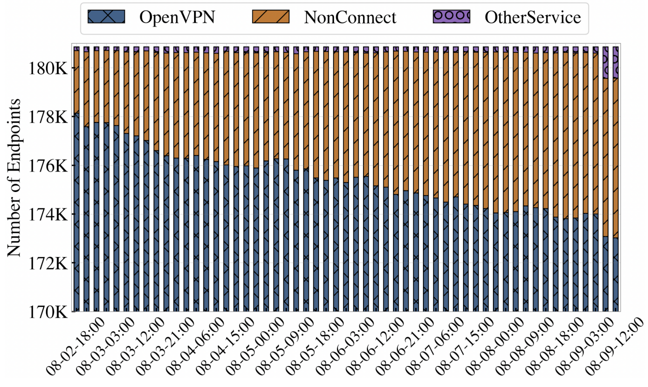


Figure 9: **OpenVPN server churns over time.**

As shown in Figure 9, even after a week, only 2.39% of OpenVPN servers either are not in the listening state or have been replaced by a different service. This suggests that the majority of OpenVPN servers are not churned frequently. In our online evaluation, we choose to probe targets in batches on a daily basis to balance between efficiency and potential IP churn. Based on the result of this test, approximately 0.9% of servers may be churned within 24 hours.

7.5 Probe UDP and Obfuscated OpenVPN Servers

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

For UDP or other obfuscated servers, our probes are no longer effective because the length field is either not present (UDP) or encrypted (tunnel-based obfuscation). However, a critical observation is that most commercial VPN providers usually offer vanilla TCP servers along with UDP and/or obfuscated variants. This is expected as commercial VPN providers attempt to optimize their VPN's performance as well as reliability, since tunnel-based obfuscation adds overhead and UDP traffic may encounter more problems than TCP in a firewalled network. Furthermore, the vanilla TCP service is often co-located with the UDP or obfuscated OpenVPN services, presumably due to lower hosting and maintenance cost. They could be on the same host by listening on different ports, or they could be located in adjacent IPs in the same VPN provider subnet. In other words, probing adjacent netblocks of a suspected UDP or obfuscated endpoint may reveal nearby vanilla TCP servers, whose existence corroborates the *Filter* results. For our *Prober* deployment on two dedicated measurement machines, we limit our probing to the /29 subnet the target IP belongs to over all TCP ports. This specific subnet size is chosen primarily due to probing resources limitation, and a more well-resourced adversary may expand the probing to larger subnets. With only two measurement machines, the parallelized /29 *Prober* is able to probe targets generated by a *Filter* monitoring a 5 Gbps network interface.

8 Real-world Deployment Setup

We set out to explore if an ISP or censor can fingerprint OpenVPN connections at scale, without significant collateral damage. Adopting the viewpoint of an adversarial ISP, we deploy our framework inside *Merit*, as shown in Figure 2. Our evaluation is two-fold: we generate *control* vanilla and obfuscated flows with commercial VPN providers and attempt to identify them as a network intermediary; we also process other traffic passing through our Monitoring Station in order to estimate the false positive rate of our framework.

We set up our framework on a 16-core server (Monitoring Station) inside *Merit* with two mirroring interfaces that have an aggregated 20 Gbps bandwidth. Due to the large traffic volume, we optimize our deployment with PF_RING [38] in order to improve the packet processing speed. We employ PF_RING in zero-copy mode and spread the traffic load across a Zeek cluster of 15 workers. Nonetheless, due to limited CPU resources, we only sample 12.5% of all TCP and UDP flows arriving at the network interfaces in order to minimize the effect of packet loss. The sampling is based on IP pairs so that all bi-directional traffic of a flow will be selected/dropped together. With these settings, we are able to operate with an end-to-end packet loss rate under 3%. Even though we process only a fraction of all traffic, our *Filter* still handles over 15 Terabytes of traffic from over 2 billion flows on an average day on a single server. In addition, processing all traffic without sampling is feasible through parallelism or using faster CPUs.

Next, we set up *Probers* on two dedicated measurement machines, each provisioned with 10 IPv4 and 1 IPv6 addresses. By the end of each day during the evaluation, the *Probers* fetch filtering logs from the Monitoring Station. For each target, we run a Masscan [25] to the /29 subnet the IP belongs to over all TCP ports (1-65535). We follow up each discovered open port by running our probing scheme, and endpoints confirmed through probing are recorded for manual analysis.

To select VPN services for evaluation, we first generate a list of "top" VPN services ranked by popularity. We combine 80 providers, most of which are paid premium VPN services, from top VPN recommendation sites based on previous work [42], listed in Appendix Table 4. Next, we visit the websites of these VPN providers searching for "Obfuscation", "Stealth", or "Camouflage Mode" *etc.*, and include providers that offer at least one obfuscated VPN configuration. In total, we find 24 providers offering obfuscated services. We test all obfuscation configurations if more than one is offered as well as vanilla OpenVPN for each provider. If TCP and UDP modes are both available, we test them separately. In total, we have 81 configurations, 41 of which are obfuscated ones.

We configure the Client Station inside *Merit* to act as a VPN client. Both upstream and downstream traffic of the Client Station go through the router that mirrors traffic to the Monitoring Station. In addition, we exclude this server from our random sampling so that *all* traffic to/from this server will be analyzed. On the client, we run an automated script to generate *control* traffic for our evaluation. For each iteration, we start the VPN client application and connect to the "default / recommended" server using Pywinauto [41]. After a random wait of 20 to 180 seconds, we confirm that the VPN tunnel is active and generate random browsing traffic with Selenium [45] by sending requests to a random website from the Alexa top 500. Finally, we disconnect from the VPN server and wait for 180 seconds before proceeding to the next iteration. For each VPN configuration, we repeat the process 50 times and collect packet captures for reference.

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

Table 3: **High-level evaluation statistics on *Merit*.**

9 Evaluation & Findings

We started the evaluation on August 13, 2021, and kept the monitor running for a week until August 20. Table 3 contains high-level statistics of the evaluation. A more detailed result that breaks down statistics by each *control* VPN configuration can be found in Appendix Table 5.

9.1 Results for *control* VPN flows

Overall, we are able to identify 1,718 out of 2,000 vanilla flows, corresponding to 39 out of 40 unique configurations. This suggests the majority of OpenVPN traffic and servers are vulnerable to passive filtering and active probing, respectively. The few exceptions correspond to VPN providers that only offer UDP-based services or hide their servers behind IDS [7], which thwarts our probing attempts. Surprisingly, we also identify over two-thirds of all obfuscated flows, corresponding to 34 out of 41 obfuscated configurations. This result is mostly due to obfuscated services using OpenVPN as their backbone protocol and insufficient obfuscation failing to mask OpenVPN's fingerprints. Alarmingly, out of the "top 10" VPN providers ranked by *top10vpn.com* [52], eight provide obfuscation services of some sort, suggesting that being undetectable is within the providers' threat model for their clients. Yet, *all* of them are flagged as suspect flows due to either insufficient encryption (Opcode) or insufficient obfuscation over packet length (ACK). Considering that these obfuscated

VPN services usually claim to be "undetectable" or claim that the obfuscation "keeps you out of trouble" [5, 54], this result is alarming as users who use these services may have a false sense of privacy and "unobservability".

4 out of the "top 5" VPN providers use XOR-based obfuscation, which is easily fingerprintable. We find that among the "top 5" VPN providers [52], four offer obfuscated services, all of which nonetheless are flagged as OpenVPN flows by our *Filter* over 90% of the time. A closer look at the raw packet capture suggests that *all* of them employ obfuscations that are almost identical to the unofficial XOR patch, thereby making them vulnerable to fingerprinting. Alarmingly, the XOR-based obfuscation—despite being rejected by OpenVPN developers [57]—appears to be a major obfuscation strategy adopted by the majority of VPN providers we test, who often praise the patch for its simplicity and low overhead. Although the patch can bypass some of the most basic filters adopted by existing open-source DPI tools, we have demonstrated that even a slightly more sophisticated filter will be able to reliably and accurately detect them.

Wrapping OpenVPN inside encrypted tunnels is a popular obfuscation strategy, yet some flows are still recognizable due to a lack of random padding. Another popular class of obfuscation strategies is tunnel-based, which wraps OpenVPN traffic inside an encrypted tunnel to frustrate any analysis over packet payloads. Examples include Stunnel, SSH tunnel, Shadowsocks, obfs{2/3/4}, and V2Ray(VMess). Overall, we find 20 obfuscated configurations deployed by 14 VPN providers that are tunnel-based. However, most of these tunnels do not add random padding to the payload being tunneled, with the only exceptions being obfs4 and VMess which can draw packet sizes from certain distributions. Among the 20 tunnel-based obfuscated services, only three of them deploy obfs4 and only one deploys VMess, leaving the remaining 16 vulnerable to ACK fingerprinting. We note that this does not mean these tunneling tools do not work, but rather that protection against traffic analysis is not among the design goals. For example, the threat model of obfs3, which is deployed by Perfect Privacy VPN, states that the obfuscator "does not try to protect against non-content protocol fingerprints, like the packet size or timing" [32]. Yet, we have demonstrated that for applications with distinct signature over packet length, such as OpenVPN, even the simplest threshold-based detection can identify them with reasonable accuracy.

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

On the positive side, some deployed services successfully evade our detection. Some providers deploy randomizers such as obfs4, v2ray, or proprietary protocols with random padding, which stopped us at the filtering stage (e.g. Tunnelbear). In addition, some providers deploy their obfuscated servers behind a firewall or IDS, which would respond with SYN-ACKs to every arriving SYN packet on almost all TCP ports [13,21]. Since we limit probe targets to 2,000 open ports per IP for practical considerations, this leads to false negatives when none of the probes hit an OpenVPN-listening port. Moreover, some VPN providers do not host vanilla OpenVPN TCP servers at all, such as VyprVPN, which currently only supports UDP as transport. For these providers, even though our *Filter* flags their flows as suspected OpenVPN, we were not able to confirm with subsequent probing. Finally, some providers host UDP or obfuscated servers outside the /29 probing range of the vanilla TCP ones, and we miss them due to probing resource constraints.

9.2 Results for all flows

Figure 10 shows an hour-level breakdown of the evaluation statistics, excluding *control* flows. Overall, both the *Filter* and *Prober* are able to reduce the number of suspected flows by several orders of magnitude, which when combined flagged 3,638 flows as OpenVPN connections. We manually analyze these flows to confirm our detection results.

Among the 3,638 flows, the destination servers for 469 of them respond to our *Base Probe #1* with an explicit server reset, indicating the presence of a legitimate OpenVPN server not configured with HMAC protection. For the remaining 3,169 flows, we first noticed that 2,580 of them are between a single IP pair. Based on our log, the client initiates a connection every 4 minutes to the server on port 1194 (assigned to OpenVPN). Reverse DNS lookup associates the client IP with the "lib-locker" subdomain under a private university in the US. Furthermore, the server runs a TLS service listening on port 443, which sends a certificate belonging to a smart locker company with subject and issuer CN as "vpn._COMPANY_-.com". Based on these evidence, we believe the captured flows correspond to the secure communications between a deployed smart locker and the infrastructure that controls it. This also suggests that the fingerprintability of OpenVPN may

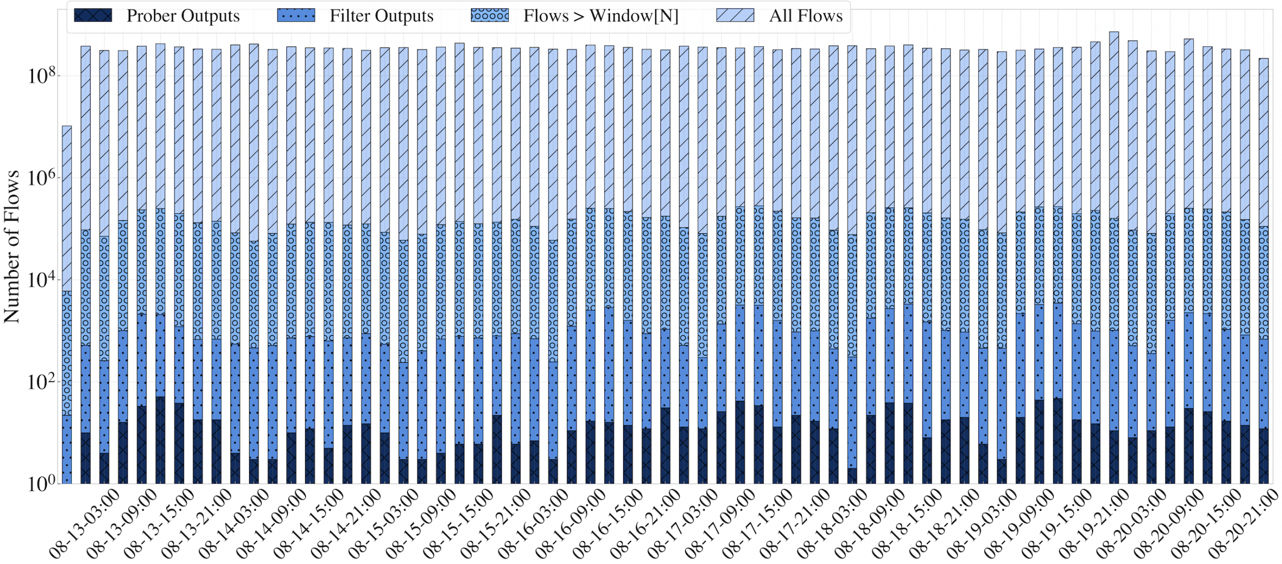


Figure 10: ***Merit* evaluation results over days, excluding Control VPN flows.**

not only be a problem concerning censorship circumvention, but it may also be used for reconnaissance to identify and target IoT devices that communicate to their servers over an OpenVPN channel. Finally, we attempt to further characterize the remaining 589 flows based on circumstantial evidence about the destination endpoint.

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

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

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

Our 7-day evaluation flagged 3,638 flows that are identified as "OpenVPN" from over 10 million flows that exceed our observation window. Among these, we are able find evidence that supports our detection result for 3,245 flows. The majority of the remaining 393 flows have server IPs belonging to cloud hosting services, and we are not able to further classify them. Conservatively, we can upper bound the false positive rate to 0.0039%, which is three orders of magnitude lower than previous ML-based approaches (1.4%-5.5%) [3, 14, 26]

10 Discussion and Mitigations

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

Putting the human danger aside, the ease of fingerprinting makes OpenVPN more susceptible to throttling or blocking from ISPs and governments. Previous research suggests that some censors already use two-stage pipelines, which are highly similar to our deployment, to detect other protocols such as Tor or Shadowsocks [1, 71]. These adversaries can quickly adapt such infrastructure to detect OpenVPN traffic by simply adding protocol-specific fingerprints and probes. Furthermore, while we focus on OpenVPN due to its overwhelming popularity among commercial VPNs, it is possible to extend our two-stage framework to other VPN protocols (e.g., WireGuard and StrongSwan) by analyzing their communication patterns and server behaviors. Governments can also quickly adopt these fingerprints to track and block VPN usage during sensitive times, like political upheavals, when VPN connections are most vital to the free flow of information.

Short-Term Mitigation There are several defensive strategies to achieve near-term protection from the fingerprinting attack we describe. First, VPN providers offering both vanilla and obfuscated OpenVPN services should avoid co-locating them. Ideally, obfuscation servers should be well separated from OpenVPN instances in the network address space and operate as "bridge servers" that forward client traffic to VPN servers elsewhere. For example, Mullvad VPN offers a Shadowsocks-based obfuscator service as a dedicated bridge, separating the VPN servers from the obfuscation [28].

Second, VPN providers should switch from static to random padding for their obfuscated services. As we have shown, for protocols with a stable and distinctive handshake phase, even the most basic threshold-based detector is able to fingerprint them by packet sizes. Ideally, the obfuscation layer should be able to send zero-length packets (packets whose payloads are all padding) to break the one-to-one correspondence between the unobfuscated and obfuscated packet streams [72]. Yet it is worth noting that previous work has shown that even fully randomized obfuscators (e.g., obfs4) can themselves be vulnerable to entropy-based fingerprinting attacks [67].

Third, we suggest that the OpenVPN developers follow recommendations from previous work with regard to how servers respond to failed handshake attempts. Servers closing failed connections immediately or in a predictable manner has enabled active probing attacks against a variety of other protocols [1, 13, 58]. In response, these protocols have implemented either unlimited timeouts (reading from the buffer indefinitely) or diversified close behaviors (in which each server instance closes failed connections in a different manner).

Long-Term Defenses In the long term, we fear that the cat-and-mouse game between censors and circumvention tools, such as the Great Firewall and Tor, will occur in the VPN ecosystem as well, and developers and providers will have to adapt their obfuscation strategies to the evolving adversaries. We urge commercial VPN providers to adopt more standardized obfuscation solutions, such as Pluggable Transports [39], and to be more transparent about the techniques used by their obfuscated services. This transparency will help foster development of stronger obfuscation methods and encourage developers to design better techniques to overcome the progress of information control technologies. Additional future work is needed to characterize the performance costs of different approaches to VPN obfuscation and to help users with varying threat models make appropriate trade-offs between performance and resilient unobservability.

11 Conclusion

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

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

12 Acknowledgement

The authors are grateful to Matthew Wright for shepherding the paper, and to the anonymous reviewers for their constructive feedback. This material is based upon work supported by the National Science Foundation under Grant No.1518888, 1823192, 2007741, 2042795, 2120400.

References

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

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

[3] S. Bagui, X. Fang, E. Kalaimannan, S. C. Bagui, and J. Sheehan. Comparison of machine-learning algorithms for classification of vpn network traffic flow using time-related features. In *Journal of Cyber Security Technology*, 2017.

[4] L. Bernaille, R. Teixeira, I. Akodjenou, A. Soule, and K. Salamatian. Traffic classification on the fly. In *Computer Communication Review, Association for Computing Machinery*, 2006.

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

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

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

[8] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *13th USENIX Security Symposium (USENIX Security 04)*.

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

[10] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. A search engine backed by Internet-wide scanning. In *22nd ACM Conference on Computer and Communications Security*, 2015.

[11] R. Ensafi, D. Fifield, P. Winter, N. Feamster, N. Weaver, and V. Paxson. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *Proceedings of the 2015 Internet Measurement Conference*.

[12] R. Ensafi, P. Winter, A. Mueen, and J. Crandall. Analyzing the great firewall of china over space and time. *Proceedings on Privacy Enhancing Technologies*, 2015.

[13] S. Frolov, J. Wampler, and E. Wustrow. Detecting Probe-resistant Proxies. In *Network and Distributed System Security*, 2020.

[14] P. Fu, C. Liu, Q. Yang, Z. Li, G. Gou, G. Xiong, and Z. Li. A NetFlow Sequence Attention Network for Virtual Private Network Traffic Detection. In *International Conference on Web Information Systems Engineering*.

[15] P. Gao, G. Li, Y. Shi, and Y. Wang. VPN Traffic Classification Based on Payload Length Sequence. In *2020 International Conference on Networking and Network Applications (NaNA)*.

[16] T. Garrett, L. E. Setenareski, L. M. Peres, L. C. Bona, and E. P. Duarte. Monitoring network neutrality: A survey on traffic differentiation detection. *IEEE Communications Surveys & Tutorials*, 2018.

[17] G. D. Gil, A. H. Lashkari, M. Mamun, and A. A. Ghorbani. Characterization of Encrypted and VPN Traffic Using Time-Related Features. In *the 2nd International Conference on Information Systems Security and Privacy(ICISSP)*, 2016.

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

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

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

[21] L. Izhikevich, R. Teixeira, and Z. Durumeric. LZR: Identifying unexpected internet services. In *30th USENIX Security Symposium (USENIX Security 21)*.

[22] F. Li, A. A. Niaki, D. Choffnes, P. Gill, and A. Mislove. A large-scale analysis of deployed traffic differentiation practices. In *Proceedings of the ACM Special Interest Group on Data Communication*. SIGCOMM, 2019.

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

[24] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian. Deep packet: a novel approach for encrypted traffic classification using deep learning. In *Soft Comput 24*, 2019.

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

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

[27] A. Molavi Kakhki, A. Razaghpanah, A. Li, H. Koo, R. Golani, D. Choffnes, P. Gill, and A. Mislove. Identifying traffic differentiation in mobile networks. In *IMC'15*.

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

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

[30] OpenVPN Reliability Layer module. http://build.openvpn.net/doxygen/group__reliable.html#details.

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

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

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

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

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

[36] OpenVPN_XORPatch. https://github.com/clayface/openvpn_xorpatch.

[37] Y. Pang, S. Jin, S. Li, J. Li, and H. Ren. OpenVPN Traffic Identification Using Traffic Fingerprints and Statistical Characteristics. In *Internation Conference on Trustworthy Computing and Services*, 2012.

[38] PF_RING ZC (Zero Copy). https://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/.

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

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

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

[42] R. Ramesh, L. Evdokimov, D. Xue, and R. Ensafi. VPNalyzer: Systematic Investigation of the VPN Ecosystem. In *Network and Distributed System Security*, 2022.

[43] R. Ramesh, R. S. Raman, M. Bernhard, V. Ongkowijaya, L. Evdokimov, A. Edmundson, S. Sprecher, M. Ikram, and R. Ensafi. Decentralized Control: A Case Study of Russia. In *Network and Distributed System Security*, 2020.

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

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

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

[47] R. Sundara Raman, L. Evdokimov, E. Wurstrow, J. A. Halderman, and R. Ensafi. Investigating Large Scale HTTPS Interception in Kazakhstan. In *Proceedings of the 2020 ACM Internet Measurement Conference*.

[48] R. Sundara Raman, A. Stoll, J. Dalek, R. Ramesh, W. Scott, and R. Ensafi. Measuring the Deployment of Network Censorship Filters at Global Scale. In *Network and Distributed System Security*, 2020.

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

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

[51] W. J. Tolley, B. Kujath, M. T. Khan, N. Vallina-Rodriguez, and J. R. Crandall. Blind In/On-Path Attacks and Applications to VPNs. In *30th USENIX Security Symposium (USENIX Security 21)*.

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

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

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

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

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

[57] Tunnelblick and openvpn_xorpatch. https://tunnelblick.net/cOpenvpn_xorpatch.html.

[58] Summary on Recently Discovered V2Ray Weaknesses. https://gfw.report/blog/v2ray_weaknesses/en/.

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

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

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

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

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

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

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

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

[67] L. Wang, K. Dyer, A. Akella, T. Ristenpart, and T. E. Shrimpton. Seeing through network-protocol obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[68] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*.

[69] N. Weaver, C. Kreibich, and V. Paxson. Redirecting DNS for Ads and Profit. In *USENIX Workshop on Free and Open Communicationson the Internet*, 2011.

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

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

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

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

[74] E. Wustrow, C. M. Swanson, and J. A. Halderman. Tapdance: End-to-middle anticensorship without

flow blocking. In *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association.

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

[76] Q. Zhang, J. Li, Y. Zhang, H. Wang, and D. Gu. Oh-Pwn-VPN! Security Analysis of OpenVPN-Based Android Apps. In *CANS*, 2017.

A Appendix

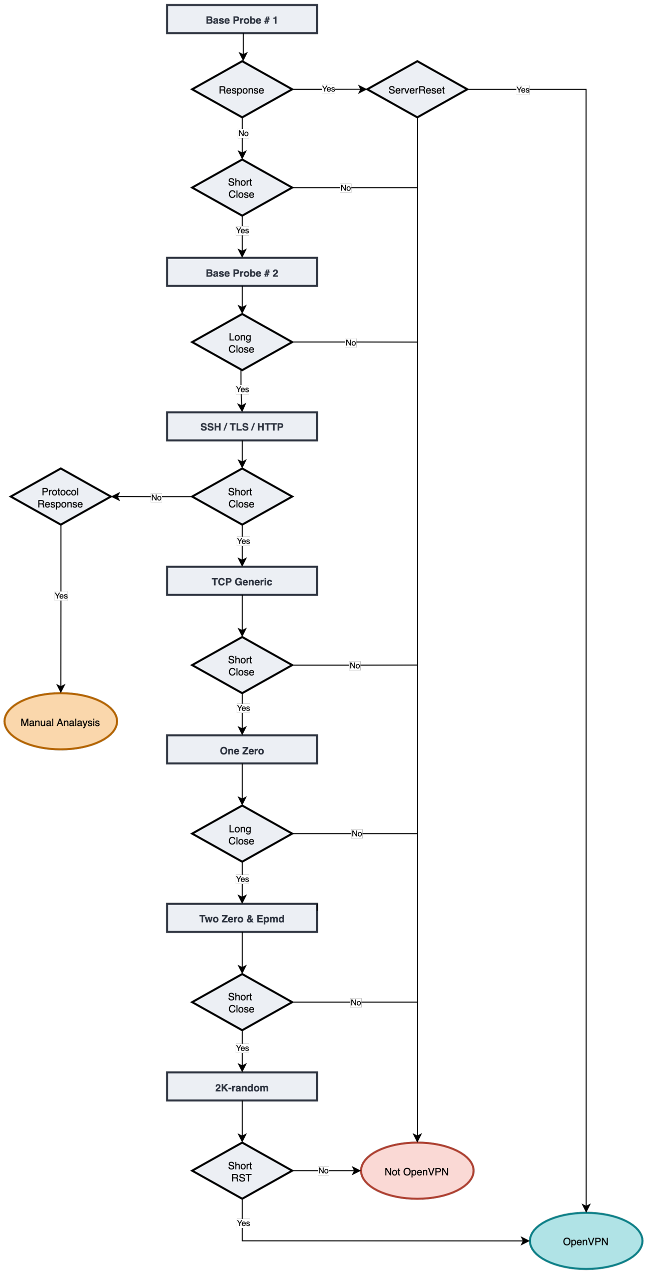


Figure 11: **Evaluation Process for Active Server Fingerprinting.**

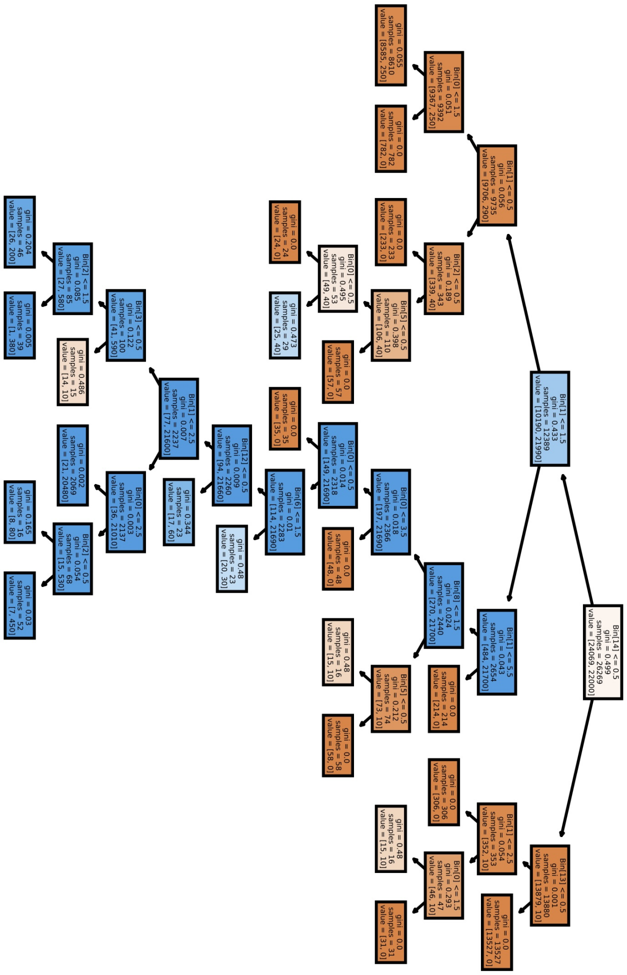


Figure 12: **Decision tree derived from ISP and VPN datasets.**

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

Table 4: **Recommendation Websites Used**

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

Table 5: **Evaluation results on *Merit*, breakdown by configuration.** Highlighted rows are "obfuscated" configurations. Variants marked with stars mean that the VPN provider does not disclose which obfuscation technique is used and we can only infer the variant type based on packet captures. Note Hide.me claims the *tls-crypt* option alone is enough to "obfuscate entire traffic" [18]. However, this option only encrypts control channel payloads but not the OpenVPN packer headers.