HTTPT: A Probe-Resistant Proxy

Sergey Frolov
University of Colorado Boulder

Eric Wustrow
University of Colorado Boulder

Abstract

Recently, censors have been observed using increasingly sophisticated active probing attacks to reliably identify and block proxies.

In this paper, we introduce HTTPT, a proxy designed to hide behind HTTPS servers to resist these active probing attacks. HTTPT leverages the ubiquity of the HTTPS protocol to effectively blend in with Internet traffic, making it more difficult for censors to block. We describe the challenges that HTTPT must overcome, and the benefits it has over previous probe-resistant designs.

1 Introduction

Internet censors strive to detect and block circumvention proxies. Many censors have adopted sophisticated proxy discovery techniques, such as *active probing* [14, 38, 39], where the censor connects to suspected proxy servers and sends probes designed to distinguish them from non-proxies. In response to this attack, developers designed and built *probe-resistant* proxies, such as ScrambleSuit [40], obfs4 [44], and Shadowsocks [2] which require clients to prove knowledge of a secret key (obtained out of band) before the proxy will respond to them.

While probe-resistant proxies are harder for censors to identify, recent work demonstrates there are still ways censors could detect probe-resistant proxies [20]. For instance, not responding to common protocols like HTTP or TLS is unusual behavior for servers. Recently, China's GFW has been observed using similar active probing techniques to detect and block Shadowsocks [3].

In this work, we propose HTTPT, an alternative proxy architecture that is designed to defend against advanced active probing. While previous probe-resistant designs avoid looking like any protocol, we argue this approach is fundamentally limited, as this behavior is unusual for servers on the Internet [20]. Instead, HTTPT uses HTTPS, a ubiquitous protocol with many diverse implementations, providing a heterogeneous set of behaviors to blend in with. HTTPT can be deployed behind already-existing Web Servers, making it impossible for censors to access and identify the proxy without knowing the secret necessary to use it.

1.1 Benefits

HTTPT has several benefits over existing designs:

Replay attack protection Existing probe-resistant proxies are vulnerable to replay attacks, where the censor re-sends observed client messages. Some proxies, such as shadowsocks-libev [43], implement a cache to prevent replays of previous connections. However, China's active probing thwarts this defense by permuting replays [3], and the behavior of not responding at all to replays may be unusual. In contrast, HTTPT is immune to replay attacks due to its reliance on TLS, which includes bidirectional nonces in the handshake.

Using existing web servers HTTPT leverages existing web servers, making it more difficult for censors to single out or identify. Hiding a proxy behind an authentic web server, such as Apache or nginx, obviates the need to mimic TLS or any other server fingerprints [21] that could allow censors to block it.

Overhead After the initial TLS handshake, HTTPT's overhead is minimal. To avoid the overhead associated with sending HTTP-compatible encodings, HTTPT instead uses WebSockets between client and server. This allows the client and server to send binary data over the web server, obviating the need for costly HTTP-safe encodings such as MIME or base64.

Using a popular protocol Existing probe-resistant proxies rely on randomized protocols that try to blend in with generic Internet traffic and make it hard for censors to identify passively. However, prior work has shown that these protocols might still be detectable using entropy analysis and machine learning [5, 37]. In HTTPT, we rely on the popular TLS protocol to tunnel data, making it more difficult for censors to block outright (because TLS is popular), and hard to perform fingerprinting attacks due to the heterogeneity of the protocol.

2 Background

The arms race between censors and circumvention tools has led to new techniques from both sides. In this section, we provide background on the active probing attacks and defenses employed recently.

Active Probing To detect and block proxies, censors often use *active probing* attacks, where they connect to suspected proxy servers and attempt to communicate using known circumvention protocols. If a server responds to these probes using the protocol (e.g. they offer proxy service to the censor), the censor learns the server is in fact a proxy, and can block it. The Great Firewall of China (GFW) have used this technique for nearly a decade to find and block Tor bridges and other proxies [14,38,39].

Censors typically find suspected proxies by traffic analysis or Internet scanning [13]. For instance, the GFW has been observed to send probes to TLS (TCP port 443) servers when a client first connects to them [38]. These probing techniques include sending random data and attempting to perform a Tor handshake. If the server responds like a proxy, for instance by responding with the Tor protocol, the endpoint is blocked by the censor.

Probe-resistant proxies To resist active probing attacks, circumvention tools have developed **probe-resistant** proxy protocols, such as ScrambleSuit [40], obfs4 [44], Shadowsocks [2], and Lampshade [28]. These protocols require clients to prove knowledge of a shared secret before the server will respond. This shared secret is distributed among the clients through private channels, such as Tor's BridgeDB [36] or email, and without the knowledge of the secret, censors are unable to get responses to their probes. As these proxies effectively remain silent to such probes, it is difficult for censors to confirm if they are proxies.

Detecting probe-resistant proxies However, there are still ways censors can identify probe-resistant proxies. In 2020, Frolov et al. showed how not responding to any probes is uncommon behavior online: over 94% of servers responded with data to at least one popular protocol [20]. Furthermore, circumvention servers would often have unique timeouts or data limits before they closed connections, allowing censors to identify and even fingerprint probe-resistant proxies that never respond to the censor. For example, Lampshade reads 256 bytes from the client, and closes the connection immediately if it does not demonstrate knowledge of the secret (e.g. a censor sends random data). However, if less than 256 bytes is read, the server will wait for 90 seconds, and then timeout and close the connection.

This gives censors a new strategy for identifying probe-resistant proxies: send several probes for popular protocols and random data, and identify the servers whose responses are consistent with responses of a given proxy (e.g. close immediately for probes over 256 bytes or close after 90 seconds otherwise for Lampshade).

Another potential way censors can identify probe-resistant proxies is with replay attacks, where they replay a previous legitimate client's initial message to the server. Since this message proves knowledge of the secret, the server will respond, alerting the censor it is a proxy.

Recently, the GFW has been observed using replay attacks and the aforementioned timeout fingerprinting to identify and block Shadowsocks servers [3]. When a client connects to a server and sends a Shadowsocks-sized initial message, the GFW will send several follow-up probes, including replays of the client's message, and random-data probes of various sizes up to and exceeding the 50-byte data limit unique to Shadowsocks.

Probe-resistant proxies could try to prevent replay attacks using server randoms or challenges, where the client proves knowledge of the shared secret by hashing it with server-provided (random) data. However, this would require the server send data before the client has authenticated, which could be used by censors to identify those proxies.

In contrast, web servers provide a natural defense against replays due to the security properties of TLS. TLS defends against client replay attacks by including the server random in the key agreement, ensuring each connection uses unique cryptographic keys.

TLS also has a large number of popular implementations, providing a plethora of fingerprints to blend into [21]. While previous work has shown the difficulty in mimicking existing protocols for circumvention [25], many of these protocols only had a single used implementation, making it difficult for circumvention tools to copy perfectly. In contrast, our technique leverages *existing* TLS implementations and efficiently tunnels through them, avoiding the difficult task of mimicry altogether. While prior work has tunneled circumvention traffic through other protocols, including chat applications, Skype, or video streaming [4,29,31,32], these protocols are generally less popular than TLS and used in a narrow set of applications. This allows censors to use traffic analysis to find proxy use [23]. In contrast, the wide range of applications used by TLS makes it more difficult (though not impossible) to perform this kind of analysis.

3 Design

At a high level, HTTPT functions as a TLS server that also has secret proxy functionality. The proxy function can only be accessed by clients that know a secret key, distributed out of band. If a censor attempts to probe an HTTPT server, they will receive the TLS server's benign response, making it difficult to identify the server as a proxy. We first describe

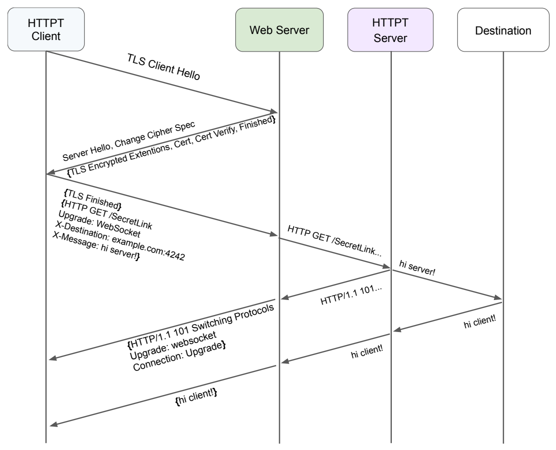

Figure 1: **HTTPT Handshake** — HTTPT minimizes Time To First Byte by passing application data together with the first HTTP request. Following initial handshake, the traffic passes through the proxy with relatively small overhead, associated with encrypting all traffic between HTTPT client and the web server using TLS. Curly brackets denote messages in encrypted TLS Records. Overhead per TLS Record amounts to only 5 bytes of the TLS header frame, and the message authentication code, which is 16 bytes for most common MACs, such as Galois MAC and Poly1305.

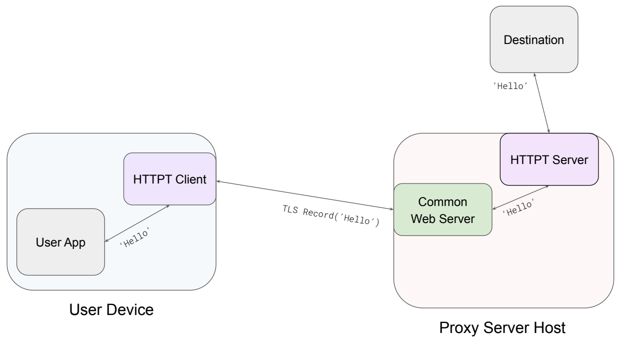

Figure 2: **HTTPT Overall Diagram** — In addition to client and server applications, present in most proxy designs, HTTPT adds an extra component: a Web Server used to deter active probing by providing plausible responses to the censors' probes.

the challenges in crafting benign responses in Section 3.1 and several solutions in Section 3.2. Finally, we describe how clients can access the proxy functionality in Section 3.3.

3.1 Benign Response Challenges

When probed by a censor, the HTTPT server must respond in a normal-seeming way that does not identify it as a proxy. There are several features of TLS servers that might be used to identify HTTPT servers.

TLS implementation TLS servers (and clients) can be *fingerprinted* based on identifying features they send in the TLS handshake. For instance, the cipher suites or extensions they support or use may vary by implementation, and those differences have been used by censors previously to identify proxies [15, 21]. Thus, in HTTPT we must be careful to avoid having identifying characteristics in the TLS layer. On the server side we address this by using existing popular TLS implementations and servers, obviating the need to mimic them. Our client implementation currently uses uTLS [21] library to mimic popular TLS fingerprints; alternatively, we may use a popular browser to establish TLS connections.

TLS certificate A related feature is the certificate the server sends. TLS certificates contain a public key, one or more domain names, a CA signature, and other extensions. To avoid detection, the HTTPT server must respond with a realistic certificate. This is challenging because real certificates often contain legitimate domain names and are usually signed by trusted certificate authorities, preventing us from forging arbitrary CA-signed certificates for domains we do not control. Self-signed certificates could be used instead, allowing us to construct arbitrary values. However, self-signed certificates may be easier for censors to block, since websites, that are popular and therefore their blocking may cause collateral damage, use CA-signed certificates.

TLS Content Once a TLS connection is established and the client (or censor) sends a request, the HTTPT server must respond with benign data, such as an HTTP response with an HTML payload. The contents of this payload must be carefully chosen so that censors cannot identify HTTPT servers based on their responses. Ideally, this content appears to (or actually does) provide real-value to legitimate users, so that censors have a harder time blocking the service. For instance, while a blank page may be common to non-HTTPT servers, it does not carry significant collateral damage that would discourage a censor from blocking it.

3.2 Benign Responses

We propose several approaches that allow HTTPT to make the TLS certificate and content appear innocuous to probing censors. If HTTPT servers are able to utilize multiple approaches, censors must find ways to identify and block *all* of these variants to prevent its use. If any of these schemes are successful at evading detection, HTTPT will still be able to provide circumvention value to users.

3.2.1 Existing servers

One option is to place HTTPT proxies at already existing HTTPS servers run by volunteers. These servers already have valid certificates and content that can be returned to a probing censor, providing legitimate camouflage that the HTTPT proxy hides behind. Because HTTPT proxies don't change the behavior of the underlying website, the censor will not be able to determine based on certificates or content if the site is a proxy or not.

Existing servers provide an ideal deployment for HTTPT, particularly if they are popular. We discuss strategies for recruiting existing sites to operate HTTPT proxies in Section 5.1.

3.2.2 Mask sites

Alternatively, HTTPT can adopt an idea from Conjure [19] and mimic existing servers not controlled by the HTTPT proxy admin. In this configuration, HTTPT will transparently proxy all traffic to a dedicated "mask site", which is an external TLS server. When a censor makes a connection to the HTTPT server, their packets are relayed to the mask site, and the responses are sent back to the censor. This allows an HTTPT proxy admin to have their server mimic any existing site online, in both certificate and content. We describe how the HTTPT proxy can detect the secret from the client in Section 3.3. Censors unable to prove knowledge of the secret will continue to have their traffic transparently relayed to the mask site, making it appear as if the HTTPT server is in fact the mask site.

3.2.3 Common error pages

For content, HTTPT servers could serve a benign error page and respond with 4xx or 5xx HTTP status code. Censys [12] scans of IPv4 HTTPS [9] servers demonstrate that these status codes are common online: as of June 2020 over 21% of the servers probed for / responded with `400 Bad Request`, 11.19% responded with `403 Forbidden`, 8.62% with `404 Not Found`, and 2.91% with `401 Unauthorized`.

3.2.4 Copying Content

Content could also be copied directly from existing websites, providing a defense similar to mask sites. Unlike mask sites, hosting the content directly avoids delays in TCP/TLS packets that would be present when transparently proxying at the packet level. However, copying content has several downsides. First, it may violate copyright law, making HTTPT servers a target from domestic legal threats as well as from censors. Second, it still requires servers to provide a plausible certificate (and potentially a domain) for this content, which may be difficult to pair with already-existing content.

3.2.5 Restricted Access

Finally, we can avoid having to send content altogether by restricting access to it. For instance, requiring an HTTP authorization, or a login page, and returning generic errors or denial pages for all attempted passwords. Such prompts are common online, causing censors to hesitate to block such sites, even without being able to see the content.

These restrictions could also be implemented at the TLS level: clients that send an incorrect server name indication (SNI) could trigger a handshake alert—behavior shared with approximately 3% of endpoints online [20]. However, censors could still replay the correct SNI value from legitimate HTTPT clients, as it is sent in the clear in the Client Hello message. TLS 1.3 offers an encrypted SNI extension [34, 35] that would address this, but it has yet to see wide use, and there have already been mixed reports of censors blocking it [10, 22].

3.3 Proving knowledge of the secret

Legitimate clients must prove knowledge of a secret shared out-of-band to use the proxy. However, they must do so in a way such that incorrect guesses do not reveal to the client that the guess was even checked. For instance, if the server responded with an "invalid secret" response, a censor could use this to identify and block HTTPT proxies via probing.

HTTPT provides two ways for clients to prove knowledge of the secret.

Secret URL First, HTTPT clients can include the shared secret in the URL of the initial HTTP request as an authentication mechanism. The web server is configured to reverse proxy all requests visiting the secret phrase to the HTTPT server. This allows the web server to respond with its normal error page (e.g. 404 Not Found) for incorrect guesses from a censor, indistinguishable from a non-proxy web server. This method works for existing sites, where the HTTPT admin can easily configure the site.

In-band TLS When using mask sites, the HTTPT server is not able to directly observe the URL that the client visits. Instead, we adopt a technique from Conjure [19] to covertly detect legitimate users. During connection establishment, the HTTPT server proxies all traffic transparently to the mask site, allowing the HTTPT client to complete a normal TLS handshake with the mask site. This means the HTTPT server appears to be sending certificates as the mask site, though does not have control over the private key and thus cannot decrypt or inject data into this connection.

After the TLS handshake completes, the HTTPT client switches to a new Master Secret, derived from the HTTPT shared secret combined with the client and server random values in the mask site TLS connection to prevent replay attacks. The HTTPT server also derives this new master secret, and if it is able to decrypt the client's application data, it knows the client is authentic (i.e. knows the shared secret). Otherwise, the HTTPT server continues forwarding packets to the mask site.

4 Evaluation

4.1 Implementation

We implemented a prototype HTTPT client and server in Golang. Our prototype is relatively simple: our client and server are written in 163 and 158 lines respectively. We confirmed our prototype is compatible with several popular web servers, including Apache 2.4.43, nginx 1.16.1, and Caddy 2.1.0. We currently support existing sites configured in any of these servers, which allows us to support all of the schemes in Section 3.2 except mask sites.

Figure 2 gives an overview of the components in the HTTPT system. Our implementation focuses on minimizing *framing overhead* between the web server and the HTTPT server. HTTPT clients make a normal TLS connection with the web server, and send an HTTP request with the secret URL and a WebSocket upgrade header. The web server (e.g. Caddy) is configured to reverse proxy this request to the HTTPT server WebSocket application based on the URL and the HTTPT server confirms the WebSocket upgrade. At this point, the web server simply forwards all application traffic transparently between the client and the HTTPT server. While normally WebSockets contain a small framing overhead, we found that this was not necessary for the web servers we tested, allowing us to have no framing between HTTPT client and server after the initial WebSocket upgrade.

This means any bytes the client sends will be encapsulated in TLS, decrypted by the web server, and forwarded directly on to the HTTPT server. Over this transport, HTTPT could support any proxying protocol, such as SOCKS. We choose to implement HTTP proxying, allowing clients to connect to and transparently communicate with an arbitrary endpoint over the HTTPT tunnel.

4.2 Performance

We evaluate the performance of HTTPT by comparing it with Shadowsocks. Shadowsocks is designed to be lightweight: it does not add padding and eliminates round trips in the initial connection, which provides exceptional performance, but with no forward secrecy.

We launched shadowsocks-libev 3.1.3 and HTTPT clients in a VM in Karnataka, India, a Shadowsocks and HTTPT proxy server in Oregon, USA, and had the client connect to a "covert" destination web server in Virginia, USA via its respective proxy. The round-trip latency between the proxy client

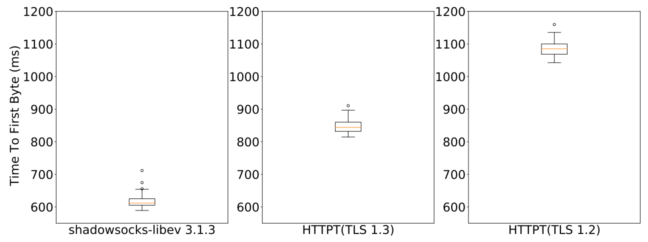

Figure 3: **Time To First Byte** — Having performed 100 measurements, we found that the median time to the first byte for Shadowsocks was 612ms, 844ms for HTTPT (TLS 1.3), and 1085 ms for HTTPT (TLS 1.2).

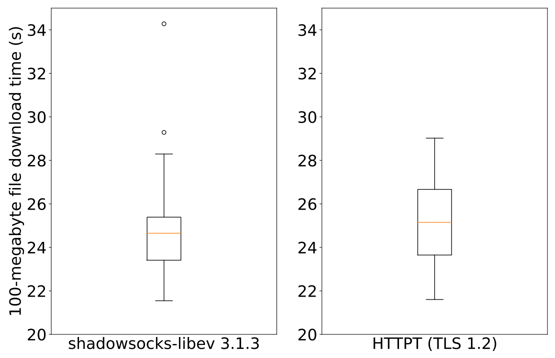

Figure 4: **Bandwidth Test** — For a bandwidth test, we download a 100-megabyte file 25 times. The resulting median time for shadowsocks is 24.65 seconds, which is comparable to the HTTPT median time of 25.15 seconds.

and server VMs is around 230ms, and the latency between the proxy server and the destination server is 80 ms.

We perform 100 measurements of Time To First Byte for shadowsocks, and for two configurations of HTTPT, one using TLS 1.2, and another TLS 1.3, and show results in Figure 3. Shadowsocks provides faster latency by having fewer round trips in its handshake: compared to Shadowsocks, TLS 1.2 requires 2 additional RTT, and TLS 1.3 requires 1 additional RTT. We note that it is possible to take advantage of TLS 1.3 Zero Round Trip Resumption to match Shadowsocks' TTFB, however, this feature sacrifices forward secrecy, and may be used by censors as a distinguisher due to its infrequency on the Internet, or allow them to execute replay attacks.

To evaluate bandwidth, we perform 25 measurements of the time it takes to fetch a 100-megabyte file from the covert destination through each proxy. As shown in Figure 4, the median time to fetch the file for Shadowsocks was 24.65 seconds, and the median time for HTTPT was 25.15 seconds (2% overhead), a difference explained by the extra round trips involved for HTTPT.

Figures 3 and 4 demonstrate that HTTPT performance is comparable to that of Shadowsocks, and we argue that a single extra round trip of connection establishment time is an acceptable price to pay for forward secrecy and increased probe resistance.

5 Discussion

5.1 Deployment Incentives

In order for HTTPT proxies to be effective, they must be deployed to a wide range of heterogeneous servers. If the servers all share some fingerprint (e.g. follow a template or have certain features in their domain names), censors may be able to easily identify and block these proxies. We envision two likely modes of deployment of HTTPT.

First, circumvention tool developers could spin up their own proxies. This would likely involve using a combination of error-page or restricted access-type servers, as well as leveraging mask sites. Deployers in this model should be careful to deploy to multiple cloud providers to avoid identifying clusters of proxies based on IP or AS.

Second, volunteers could be encouraged to run HTTPT proxies on their existing sites. Websites that wish to support censored users could enable these proxies. We note it may be difficult to recruit volunteers, as ideally, they do not advertise they are supporting this effort. Nonetheless, sites that wish to support these efforts could deploy HTTPT proxies and have a direct impact. Additionally, if a large enough cohort of high-profile websites supports HTTPT proxies, then advertising support could still be possible, assuming censors are unwilling to collectively block these sites.

This second type of deployment would require coordination from a central party, such as a circumvention tool, that keeps track of what sites and secret URLs can be used as HTTPT proxies, and provides easy-to-install configurations to volunteers. Websites could also choose whether they are comfortable exiting proxy traffic directly from the web server (which increases user privacy), or if they want to relay such traffic to the central party (which limits abuse and liability concerns for the website).

5.2 Future Work

In the future, we plan to implement other benign response strategies (Section 3.2) such as mask sites, which will allow HTTPT to be used in an even wider range of scenarios.

In addition, there are several implementation features that we believe would benefit HTTPT. These include HTTP/2 [6], which could allow multiplexing of multiple covert connections over a single stream, or fine-grained padding control.

We also plan to adopt TurboTunnel [16], an inner reliability layer for circumvention protocols. This layer makes it easier to multiplex sessions over multiple overt connections in the event they are disconnected or disrupted.

5.3 Limitations

HTTPT is not designed to defend the proxy distribution system against enumeration attacks, where censors could discover IPs and secrets of proxies in bulk. Solutions such as Salmon [11] or Hyphae [30] could be used to defend against such attacks.

HTTPT also does not protect against website fingerprinting attacks. While prior research has shown machine learning classification may allow future censors to passively distinguish proxy use from other website access [24,33], researchers have not yet observed these techniques used by censors in practice. This is likely due to the base-rate of non-proxy web traffic making it difficult for censors to justify using methods with even small (but non-zero) false positive detection rates [37]. Therefore, we believe that website fingerprinting is not an immediately likely attack against HTTPT, though it may pose one in the future. In any case, it should be possible to adapt existing fingerprinting defenses to our design [7,8].

5.4 Related Work

The HTTPS protocol has been previously proposed in censorship circumvention technologies, including Domain Fronting [17] and Refraction Networking [18,19,26,27,41,42]. However, these technologies are intended to protect open proxies against enumeration attacks, and have deployment requirements that constrain deployment. In contrast, HTTPT can be deployed at any web server and does not require ISP or CDN cooperation.

TLS has also been used in VPN proxies, such as OpenVPN [1]. However, these proxies do not address any of the benign response challenges that HTTPT solves, and are thus easily actively probed by censors.

6 Conclusion

Censors have recently adapted new methods to detect probe-resistant proxies [3,20]. In this paper, we present HTTPT, which provides a defense against recent such active probe attacks employed by censors. By leveraging the existing popular TLS protocol, HTTPT avoids the bulk of attacks that existing probe-resistant proxies face. We address several new challenges that HTTPT introduces, and evaluate our prototype, finding that it is comparable to existing popular proxies used today. We plan to release our code as open source[1].

While HTTPT is not the final word in the censorship arms race, we believe it presents a unique new challenge for censors to effectively detect at scale, making it well-suited to protecting circumvention tools from discovery.

[1] `https://github.com/sergeyfrolov/httpt`

References

[1] OpenVPN: VPN Software Solutions & Services For Business. https://openvpn.net.

[2] Shadowsocks: A secure socks5 proxy. https://shadowsocks.org/assets/whitepaper.pdf.

[3] Anonymous, Fifield, D., and Houmansadr, A. How China Detects and Blocks Shadowsocks. https://gfw.report/blog/gfw_shadowsocks.

[4] Barradas, D., Santos, N., and Rodrigues, L. DeltaShaper: Enabling unobservable censorship-resistant TCP tunneling over videoconferencing streams. *Proceedings on Privacy Enhancing Technologies 2017*, 4 (2017), 5–22.

[5] Barradas, D., Santos, N., and Rodrigues, L. Effective detection of multimedia protocol tunneling using machine learning. In *27th USENIX Security Symposium* (2018), USENIX Association.

[6] Belshe, M., Thomson, M., and Peon, R. Hypertext transfer protocol version 2 (http/2).

[7] Bocovich, C., and Goldberg, I. Slitheen: Perfectly imitated decoy routing through traffic replacement. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 1702–1714.

[8] Cai, X., Nithyanand, R., and Johnson, R. CS-BuFLO: A congestion sensitive website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society* (2014), ACM, pp. 121–130.

[9] Censys. Most common Status Lines responses of an internet-wide scan of IPv4 HTTPS servers on port 443. https://censys.io/ipv4/report?field=443.https.get.status_line.raw, June 2020.

[10] Chai, Z., Ghafari, A., and Houmansadr, A. On the importance of encrypted-sni ({ESNI}) to censorship circumvention. In *9th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 19)* (2019).

[11] Douglas, F., Pan, W., Caesar, M., et al. Salmon: Robust proxy distribution for censorship circumvention. *Proceedings on Privacy Enhancing Technologies 2016*, 4 (2016), 4–20.

[12] Durumeric, Z., Adrian, D., Mirian, A., Bailey, M., and Halderman, J. A. A search engine backed by Internet-wide scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 542–553.

[13] Durumeric, Z., Wustrow, E., and Halderman, J. A. ZMap: Fast internet-wide scanning and its security applications. In *22nd USENIX Security Symposium (USENIX Security '13)* (2013), pp. 605–620.

[14] Ensafi, R., Fifield, D., Winter, P., Feamster, N., Weaver, N., and Paxson, V. Examining how the great firewall discovers hidden circumvention servers. In *Proceedings of the 2015 Internet Measurement Conference* (2015), ACM, pp. 445–458.

[15] Fifield, D. Cyberoam firewall blocks meek by TLS signature. https://groups.google.com/forum/#!topic/traffic-obf/BpFSCVgi5rs/, 2016.

[16] Fifield, D. Turbo tunnel. designing circumvention protocols for speed, flexibility, and robustness. https://www.bamsoftware.com/sec/turbotunnel.html, 2019.

[17] Fifield, D., Lan, C., Hynes, R., Wegmann, P., and Paxson, V. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies 2015*, 2 (2015), 46–64.

[18] Frolov, S., Douglas, F., Scott, W., McDonald, A., VanderSloot, B., Hynes, R., Kruger, A., Kallitsis, M., Robinson, D. G., Schultze, S., et al. An ISP-scale deployment of TapDance. In *7th USENIX Workshop on Free and Open Communications on the Internet (FOCI '17)* (2017).

[19] Frolov, S., Wampler, J., Tan, S. C., Halderman, J. A., Borisov, N., and Wustrow, E. Conjure: Summoning proxies from unused address space. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (2019), pp. 2215–2229.

[20] Frolov, S., Wampler, J., and Wustrow, E. Detecting Probe-resistant Proxies. In *Proc. Network and Distributed System Security Symposium (NDSS)* (2020).

[21] Frolov, S., and Wustrow, E. The use of TLS in censorship circumvention. In *Proc. Network and Distributed System Security Symposium (NDSS)* (2019).

[22] Gatlan, S. South korea is censoring the internet by snooping on SNI traffic. https://www.bleepingcomputer.com/news/security/south-korea-is-censoring-the-internet-by-snooping-on-sni-traffic/, 2019.

[23] Geddes, J., Schuchard, M., and Hopper, N. Cover your ACKs: Pitfalls of covert channel censorship circumvention. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 361–372.

[24] Hayes, J., and Danezis, G. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, Aug. 2016), USENIX Association, pp. 1187–1203.

[25] Houmansadr, A., Brubaker, C., and Shmatikov, V. The parrot is dead: Observing unobservable network communications. In *Security and Privacy (SP), 2013 IEEE Symposium on* (2013), IEEE, pp. 65–79.

[26] Houmansadr, A., Nguyen, G. T., Caesar, M., and Borisov, N. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *Proceedings of the 18th ACM conference on Computer and communications security* (2011), ACM, pp. 187–200.

[27] Karlin, J., Ellard, D., Jackson, A. W., Jones, C. E., Lauer, G., Mankins, D., and Strayer, W. T. Decoy routing: Toward unblockable internet communication.

[28] Lantern Project. Lampshade: a transport between Lantern clients and proxies. https://godoc.org/github.com/getlantern/lampshade.

[29] Li, S., Schliep, M., and Hopper, N. Facet: Streaming over videoconferencing for censorship circumvention. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society* (2014), ACM, pp. 163–172.

[30] Lovecruft, I. A., and de Valence, H. HYPHAE: Social secret sharing. Tech. rep., Tech. rep. 2017-04-21. Apr. 2017.

[31] McPherson, R., Houmansadr, A., and Shmatikov, V. CovertCast: Using live streaming to evade internet censorship. *Proceedings on Privacy Enhancing Technologies 2016*, 3 (2016), 212–225.

[32] Mohajeri Moghaddam, H., Li, B., Derakhshani, M., and Goldberg, I. SkypeMorph: Protocol obfuscation for Tor bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 97–108.

[33] Panchenko, A., Lanze, F., Pennekamp, J., Engel, T., Zinnen, A., Henze, M., and Wehrle, K. Website fingerprinting at internet scale. In *NDSS* (2016).

[34] Rescorla, E. The transport layer security (TLS) protocol version 1.3. https://tlswg.github.io/tls13-spec/, May 2016.

[35] Rescorla, E., Oku, K., Sullivan, N., and Wood, C. A. TLS Encrypted Client Hello. Internet-Draft draft-ietf-tls-esni-07, Internet Engineering Task Force, June 2020. Work in Progress.

[36] Tor Project. Bridgedb. https://bridges.torproject.org/.

[37] Wang, L., Dyer, K. P., Akella, A., Ristenpart, T., and Shrimpton, T. Seeing through network-protocol obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 57–69.

[38] WILDE, T. Great firewall Tor probing circa 09 DEC 2011. https://gist.github.com/da3c7a9af01d74cd7de7, 2011.

[39] WINTER, P., AND LINDSKOG, S. How the great firewall of china is blocking tor. In *Presented as part of the 2nd USENIX Workshop on Free and Open Communications on the Internet* (Bellevue, WA, 2012), USENIX.

[40] WINTER, P., PULLS, T., AND FUSS, J. ScrambleSuit: A polymorphic network protocol to circumvent censorship. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society* (2013), ACM, pp. 213–224.

[41] WUSTROW, E., SWANSON, C., AND HALDERMAN, J. A. TapDance: End-to-middle anticensorship without flow blocking. In *23rd USENIX Security Symposium* (Aug. 2014).

[42] WUSTROW, E., WOLCHOK, S., GOLDBERG, I., AND HALDERMAN, J. A. Telex: anticensorship in the network infrastructure. In *Proceedings of the 20th USENIX conference on Security* (2011), USENIX Association.

[43] YANG, L., LV, M., AND WINDY, C. Shadowsocks-libev – libev port of shadowsocks. https://github.com/shadowsocks/shadowsocks-libev, 2014.

[44] YAWNING ANGEL. obfs4 (The obfourscator) specification. https://gitlab.com/yawning/obfs4/blob/master/doc/obfs4-spec.txt.