Conjure: Summoning Proxies from Unused Address Space

Sergey Frolov
University of Colorado Boulder

Jack Wampler
University of Colorado Boulder

Sze Chuen Tan
UIUC

J. Alex Halderman
University of Michigan

Nikita Borisov
UIUC

Eric Wustrow
University of Colorado Boulder

ABSTRACT

Refraction Networking (formerly known as "Decoy Routing") has emerged as a promising next-generation approach for circumventing Internet censorship. Rather than trying to hide individual circumvention proxy servers from censors, proxy functionality is implemented in the core of the network, at cooperating ISPs in friendly countries. Any connection that traverses these ISPs could be a conduit for the free flow of information, so censors cannot easily block access without also blocking many legitimate sites. While one Refraction scheme, TapDance, has recently been deployed at ISP-scale, it suffers from several problems: a limited number of "decoy" sites in realistic deployments, high technical complexity, and undesirable tradeoffs between performance and observability by the censor. These challenges may impede broader deployment and ultimately allow censors to block such techniques.

We present Conjure, an improved Refraction Networking approach that overcomes these limitations by leveraging unused address space at deploying ISPs. Instead of using real websites as the decoy destinations for proxy connections, our scheme connects to IP addresses where no web server exists leveraging proxy functionality from the core of the network. These phantom hosts are difficult for a censor to distinguish from real ones, but can be used by clients as proxies. We define the Conjure protocol, analyze its security, and evaluate a prototype using an ISP testbed. Our results suggest that Conjure can be harder to block than TapDance, is simpler to maintain and deploy, and offers substantially better network performance.

CCS CONCEPTS

• **Networks** → *Network security*; • **Social and professional topics** → *Censorship.*

KEYWORDS

anticensorship, proxies, network security

ACM Reference Format:
Sergey Frolov, Jack Wampler, Sze Chuen Tan, J. Alex Halderman, Nikita Borisov, and Eric Wustrow. 2019. Conjure: Summoning Proxies from Unused Address Space. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3319535.3363218

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11.
https://doi.org/10.1145/3319535.3363218

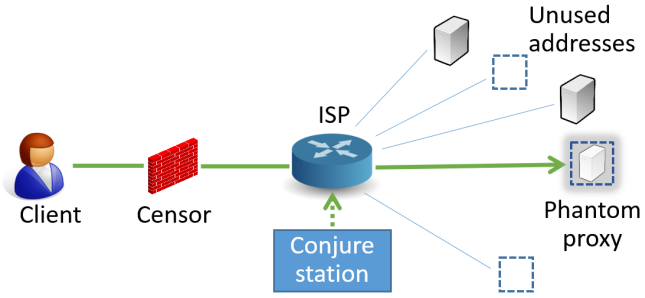


Figure 1: Conjure Overview — An ISP deploys a Conjure station, which sees a passive tap of passing traffic. Following a steganographic registration process, a client can connect to an unused IP address in the ISP's AS, and the station will inject packets to communicate with the client as if there were a proxy server at that address.

1 INTRODUCTION

Over half of Internet users globally now live in countries that block political, social, or religious content online [19]. Meanwhile, many popular tools and techniques for circumventing such censorship become ineffective, because censors have evolved new ways to block them [14, 35] or infrastructure they rely on has become unavailable.

For example, domain fronting [17] was a popular circumvention strategy used by meek (in Tor), as well as by the Signal secure messaging app to get around censorship in countries where it was blocked [36, 43]. But in May 2018, Google and Amazon made both technical and policy changes to their cloud infrastructures that removed support for domain fronting [33]. While meek continues to use other cloud providers that (for now) continue to allow domain fronting, Signal abandoned the strategy altogether [37]. There is an urgent need for new, more robust approaches to circumvention.

A family of techniques called Refraction Networking [3, 13, 26, 29, 40, 59, 60], formerly known as Decoy Routing, has made promising steps towards that goal. These techniques operate in the network's core, at cooperating Internet Service Providers (ISPs) outside censoring countries [45]. Clients access circumvention services by connecting to a "decoy site"—any uncensored website for which the connection travels over a participating ISP. Upon recognizing a steganographic signal from the client, the ISP modifies the connection response to return censored content requested by the user. Censors cannot easily block access without also blocking legitimate connections to the decoy sites—collateral damage that may be prohibitive for censors if Refraction Networking is widely deployed [48].

However, deploying such schemes is more difficult than with most edge-based circumvention techniques, since ISPs must be convinced to operate the systems in their production networks.

To date, only TapDance [59], one of six Refraction Networking proposals, has been deployed at ISP scale [20].

TapDance was designed for ease of deployment. Instead of in-line network devices required by earlier schemes, it calls for only a passive tap. This "on-the-side" approach, though much friendlier from an ISP's perspective, leads to major challenges when interposing in an ongoing client-to-decoy connection:

- Implementation is complex and error-prone, requiring kernel patches or a custom TCP stack.
- To avoid detection, the system must carefully mimic subtle features of each decoy's TCP and TLS behavior.
- The architecture cannot resist active probing attacks, where the censor sends specially crafted packets to determine whether a suspected connection is using TapDance.
- Interactions with the decoy's network stack limit the length and duration of each connection, forcing TapDance to multiplex long-lived proxy connections over many shorter decoy connections. This adds overhead and creates a traffic pattern that is challenging to conceal.

Conjure In this paper, we present Conjure, a Refraction Networking protocol that overcomes these challenges while retaining TapDance's ISP-friendly deployment requirements Our key innovation is an architecture that avoids having to actively participate in client-to-decoy connections.

In our scheme (Figure 1), clients register their intentions to connect to phantom hosts in the "dark" or unused address space of the deploying ISP. Once registered, clients can connect to these phantom hosts IP addresses as if they were real proxy servers. The Conjure station (deployed at an ISP) acts as the other end of these connections, and responds as if it were a legitimate site or service. To the censor, these phantom hosts appear as legitimate sites or services, and even active probes will not reveal information that would allow the censor to block them.

Phantom hosts are cheap to connect to, and greatly expand the number of viable proxy endpoints that a censor must consider. This increases the cost for censors to block, as they must detect and block in real time. Meanwhile, even a censor that could theoretically detect 90% of phantom hosts with confidence does not significantly reduce the effectiveness of a circumvention system, giving Conjure an advantage in the censor/circumventor cat-and-mouse game.

Conjure supports both IPv4 and IPv6, though we note that the technique is especially powerful in IPv6, where censors cannot exhaustively scan the address space ahead of time to identify addresses that change behavior. Because we fully control the proxy transport, connections can live as long as needed, without the complexity faced by TapDance.

We introduce the Conjure protocol (Section 4) and analyze its security, finding that it resists a broader range of detection attacks than TapDance. We have implemented Conjure (Section 5) and deployed it on a 20 Gbps ISP testbed similar to the TapDance deployment [20]. Compared to TapDance, we find that Conjure has reduced complexity and substantially improved performance (Section 6): on average, Conjure has 20% lower latency, 14% faster download bandwidth, and over 1400 times faster upload bandwidth. In addition, Conjure is significantly more flexible than existing Refraction Networking protocols, allowing maintainers to respond to future censor techniques with greater agility. We believe that these advantages will make Conjure a strong choice for future Refraction Networking deployments.

We have released the open source implementation of the Conjure client at https://github.com/refraction-networking/gotapdance/tree/dark-decoy.

2 BACKGROUND

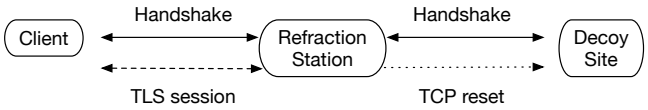
Refraction Networking operates by injecting covert communication inside a client's HTTPS connection with a reachable site, also known as a decoy site. In a regular HTTPS session, a client establishes a TCP connection, performs a TLS handshake with a destination site, sends an encrypted web request, and receives an encrypted response. In Refraction Networking, at least one direction of this exchange is observed by a *refraction station*, deployed at some Internet service provider (ISP). The station watches for a covert signal from the client that this connection is to be used for censorship circumvention. Upon seeing the signal, the station will take over the HTTPS session, and establish a proxy session with the client that can then be used for covert communication.

One of the key challenges for Refraction Networking is in taking over a session. The station must start responding to the client's traffic as if it were the decoy destination, and at the same time prevent the destination from sending its own responses back to the client. A simple approach is to have the refraction station act as an inline transparent proxy (Figure 2a) that forwards the traffic between the client and the decoy site. After a TLS handshake has been completed, the station terminates the connection with the decoy site by sending a TCP reset and takes over the session with the client.

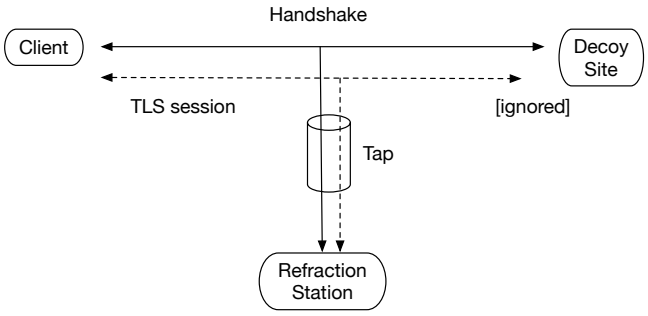
An inline element, however, can significantly affect the reliability and performance of the regular, non-refraction traffic of an ISP. Cirripede [26] and Telex [60] attempted to mitigate this by dynamically adding router rules to forward only a subset of traffic from a registered client or an active session through the element, but this nevertheless presented a deployability challenge.

TapDance [59] offered an alternative design that did not require the blocking or redirection of traffic, but used a mirror port instead (Figure 2b). In TapDance a client sends an incomplete HTTP request, which causes the decoy site to pause waiting for more data while the station takes over the connection in its place. After a client would receive a packet initiated by the station, its TCP sequence numbers would become desynchronized with the decoy site, causing the decoy to ignore the packets sent by the client.

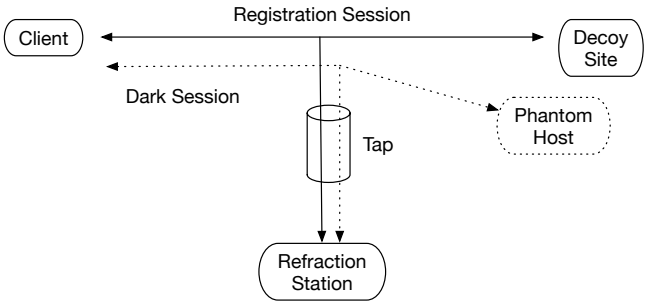
This approach reduced the barriers to deployment and TapDance was used in production during a pilot study, serving upwards of 50 000 real-world users [20]. The tap-based approach, however, has some disadvantages. A decoy site will only ignore packets as long as the sequence numbers stay within its TCP window, and will terminate the connection after a timeout. Frolov et al. report that in their pilot, they eliminated roughly a third of potential decoy sites due to their measured window or timeout values being too small [20]. Even so, sessions that try to upload non-trivial data amounts (in excess of about 15 KB) or last longer than the timeout value (ranging from 20–120 s) require the user to create new refraction connections, adding overhead, complexity, and opportunities



(a) First generation systems for Refraction Networking, such as Telex and Cirripede, operated as inline network elements, with the ability to observe traffic and block specific flows. ISPs worried that if the inline element failed, it could bring down the network.



(b) TapDance is a second-generation Refraction Network scheme that operates without flow blocking, needing only to passively observe traffic and inject packets. TapDance has recently been deployed at a mid-size ISP, but the techniques used to silence the decoy site and participate in the client–decoy TCP connection mid-stream add significant complexity, performance bottlenecks, and detection risk.



(c) Conjure, our third-generation Refraction Networking design, overcomes these limitations. It uses two sessions. First, the client connects to a decoy site and embeds a steganographic registration message, which the station receives using only a passive tap. Second, the client connects to a "phantom host" where there is no running server, and the station proxies the connection in its entirety.

Figure 2: Evolution of Refraction Networking

for errors. Additionally, keeping the connections to the decoy site open for tens of seconds uses up the site's resources; Frolov et al. found that a default configuration of the Apache web server would only keep 150 simultaneous connections open, while the pilot deployment would often result in dozens of connections to the same decoy site, creating a scaling concern.

Conjure is able to avoid these problems by creating proxies at unused IP addresses, allowing the station full control over a host it has created, rather than forcing it to mimic an already existing decoy (Figure 2c). This design obviates the need for taking over a session already in progress, which both simplifies the implementation and eliminates certain attacks, as we will discuss in Section 7.

Registration Signal In all implementations of Refraction Networking, a client must send a covert signal to the station to initiate communication. This covert signal is embedded inside communication fields that must be indistinguishable from random by a censor without access to a secret/private key available to the station. Past implementations have used TCP initial sequence numbers [26], the ClientRandom field inside a TLS handshake [29, 60], and the encrypted body of an HTTPS request [59]. In principle Conjure can use any of these mechanisms for registration, but in our prototype we used the HTTPS request body as it offers the greatest flexibility for the amount of data that can be sent with the registration.

3 THREAT MODEL

Our deployment model is identical to that of TapDance: we only require a passive tap at the deploying ISP, and the ability to inject (spoofed) traffic from phantom hosts. Furthermore, we assume asymmetric routing (i.e. that the tap might only see packets from but not to the client). However, we assume a stronger threat model for the adversary than TapDance, as our design resists active attacks.

We assume the censor can block arbitrary IP addresses and networks, but faces a cost in doing so if it blocks potentially useful resources. In particular, we assume it is difficult for the censor to have complete knowledge of legitimate addresses used, and so instead resorts to a blacklist approach to blocking proxies and objectionable content. Whitelists are expensive for censors to maintain and can stifle innovation, and are rarely employed by country-level censors.

We assume that the censor can know what network the Conjure station(s) are deployed in and the prefixes phantom hosts are selected from, but that blocking those networks outright brings a collateral damage the censor is unwilling to suffer. Instead, the censor aims to identify the addresses that are phantom hosts, and block only those. We note this assumption supposes that the censor does not mount effective routing decoy attacks [27, 49]; we discuss these attacks further in Section 8.2.

We allow the censor access to the client to register and use its own phantom hosts, so the system should ensure that these will not reveal the phantom hosts of other users. The censor can also actively probe addresses that it sees users accessing, and can employ tools such as ZMap [11] to scan large network blocks, excepting large IPv6 prefixes (e.g. a /32 IPv6 prefix contains 2^{96} addresses).

Finally, we assume the censor can replay or preplay any connections that it suspects involve phantom hosts (or their registration) in an attempt to confirm. However, the censor wishes to avoid disrupting any connections before it knows for certain they are from Conjure clients, lest they disrupt legitimate connections. This means that injecting false data or corrupting TLS sessions is outside the scope of the censor, but that the censor can send non-disruptive probes (such as stale TCP acknowledgments) that would normally be ignored. We emphasize that TapDance is observable by censors that can send TCP packets in suspected connections, but that our protocol is robust against this class of censor.

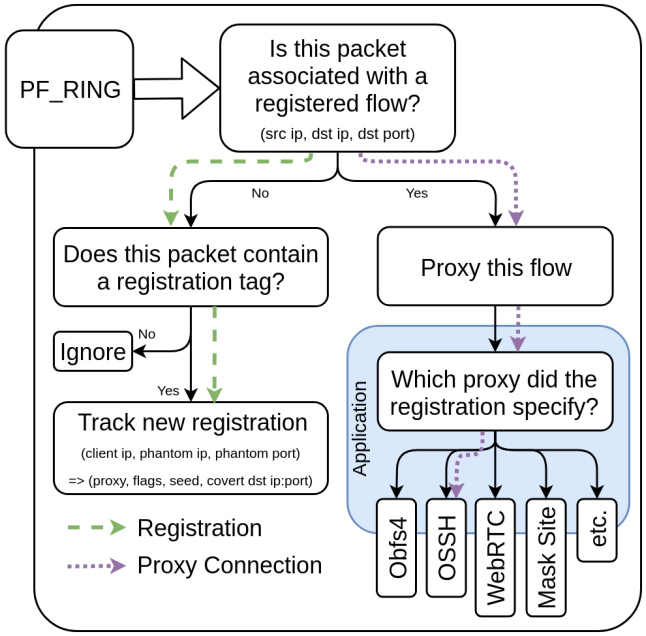


Figure 3: Conjure Operations — A Conjure session is constructed in two pieces. First a client registers by sending a TLS connection to the decoy host (see Figure 2c) with a steganographically tagged payload. This registration contains the true "covert" address that the client would like to connect to and a seed which the Conjure station uses to derive the Phantom IP address for the session (green dashed flow). Second, the client derives the same Phantom IP address and connects directly to it using the proxy protocol (e.g. OSSH) specified in the registration (purple dotted flow). The station identifies registered packets by source IP, destination IP, and destination port (client IP, phantom IP and port respectively). The traffic is passed to a proxy server handling the specific proxy protocol, which ultimately proxies traffic to the covert address. Secrets for the individual application proxy service are shared out of band or derived from the secret seed shared between client and station during registration, preventing an adversary from successfully connecting to the application in follow up probes.

4 ARCHITECTURE

Conjure involves two high-level steps. First, clients **register** with a Conjure station deployed at an ISP, and derive a phantom host IP address from a seed shared in the registration. Then, clients **connect** to the agreed upon phantom address, and the station listening on the tap relays the client's traffic to a local application. The application brokers traffic to a proxy application specified by the client in their registration providing a probe resistant tunnel. Figure 3 describes a high-level overview of the Conjure registration and connection behavior.

Similar to TapDance [59], our design does not require expensive in-line flow blocking and is accomplished with only a passive tap at the ISP, imparting little to no burden on their infrastructure and service reliability. Our architecture is also modular, in that the registration and connection steps operate independently, allowing a wide range of flexibility to evade censors. We describe each of these components, and then describe our implementation and deployment in Section 5.

4.1 Registration

We use a form of Refraction Networking similar to TapDance to register directly with the station, though Conjure registration is significantly simpler and more difficult to detect than vanilla TapDance. This is because registration flows are unidirectional; a client communicates their intent to register to the station without any response from the station itself. This makes registration flows very difficult to detect as they can be sent to any site hosted behind the ISP, and display no evidence of proxy behavior.

While this model of registration gives the client no definitive indication that their registration was successful, the client can attempt to register multiple times concurrently with the same information, and expect that one gets through. Alternatively clients can register intent to use the proxy in other covert ways. For instance, they could use email [28] or any other existing intermittently available proxies to register.

A registration connection is sent to a **registration decoy**: any site that supports TLS behind the ISP relative to the client. The client completes the handshake, and sends a normal HTTPS request that embeds a ciphertext tag in the payload. Conjure leverages the same steganographic technique as TapDance to encode the ciphertext [59, §3], however we send a complete request allowing the registration decoy to respond or close the connection. The tag is encrypted such that it is only visible to the station. The Conjure station passively observes tagged flows obviating the need to mimic the decoy. In addition, more potential decoys can be used in comparison to TapDance, as there is no need to exclude decoys that have timeouts or TCP windows unfavorable for keeping connections open. In our deployment, this results in a modest increase of 25% more decoys that could be used than in TapDance.

The tag contains a public key (encoded to be indistinguishable from random using Elligator [2]), and a message encrypted under the shared secret derived from a Diffie-Hellman key exchange with the station's long-term public key hard-coded in the client. The station uses its private key to compute the same shared secret from the (decoded) client public key, and decrypts the message in the tag. The censor, without knowledge of either the station or client's private key, cannot derive the shared secret, preventing it from being able to decrypt the message, or even learn of its existence.

Inside the message, the client communicates a random seed, the covert address they would like to connect to, the transport protocol they will use, and other configuration-specific information, such as flags to signify version, and feature support. The client and server hash the seed to determine which specific IP address from the set of shared CIDR prefixes will be used and registered as a phantom host. It may seem intuitive to instead have the client send the specific IP address to register, but allowing the client arbitrary choice also allows the censor to register suspected phantom hosts and block them if they can be used as proxies. By using a hash of a seed, the censor would have to pre-image the hash to obtain a seed it could use to register for a desired IP address. We discuss the intricacies of phantom IP address selection in Section 6.2.

Once the phantom host IP address has been selected, the station watches for packets with the source address of the original client and the phantom as the destination, and forwards them on to the a local application handling transports. The station only forwards

packets that originate from the IP address of the registering client, making the phantom host appear firewalled off to everyone but the client. We note that censors have been observed taking over client IP addresses for follow-up probing [14]. This would allow censors to hijack registrations if they can connect within the small window between client registration and connection. However this only allows the censor to communicate to the local application that handles transports, it does not connect them to the covert address that the client indicated in their registration. Filtering by client IP and phantom IP also prevents censors from enumerating the address space before hand, as they would have to do so from every potential client IP address. Simply scanning the prefixes with ZMap [11] from a single vantage point would not reveal hosts that only respond to specific IPs (e.g., firewalled subnets).

4.2 Transports

Once the client has registered, packets sent to the phantom host IP address are detected at the station and passed to the local **application** which provides proxy access to the client. A viable Conjure transport has two main requirements: first, the protocol it uses with the client must be difficult for the censor to *passively detect* and block by traffic inspection. Second, the endpoint must resist *active probes* by the censor (who does not know some shared secret).

Any protocol that satisfies these criteria can be used as an effective transport with Conjure. In this section, we describe various existing protocols (OSSH and obfs4) as well as introduce our own (Mask sites, TLS 1.3 eSNI, and phantom WebRTC clients) that can be used in Conjure, and evaluate how each meet the necessary requirements. Table 1 compares the application protocols. Conjure uses a modular approach to transports because research into proxy detection is ongoing. Having a variety of supported transports gives clients a quick way to pivot and maintain proxy access even when new proxy protocol vulnerabilities are discovered.

4.2.1 Obfuscated SSH. Obfuscated SSH [31] (OSSH) is a protocol that attempts to mask the Secure Shell (SSH) protocol in a thin layer of encryption. This makes it difficult for censors to identify using basic packet filters, as there are no identifying headers or fields to search for. Instead, Obfuscated SSH clients first send a 16-byte random seed, which is used to derive a symmetric key that encrypts the rest of the communication. Early versions of OSSH were passively detectable by censors, who could observe the random seed and derive the key, allowing them to de-obfuscate the protocol. These versions also did not protect against active probing attacks, as a censor could easily create their own connections to confirm if a server supports the protocol.

More recent versions of OSSH, such as those used by Psiphon [44], mix a secret value into the key derivation, thwarting the naive passive detection/decryption attack. The secret is distributed out-of-band along with the proxy's address, and is unknown to a passive or active-probing censor. If a client connects and cannot demonstrate knowledge of the secret, the OSSH server does not respond, making it more difficult for censors to discover OSSH servers via active probing attacks.

4.2.2 obfs4. `obfs4` [52] is a pluggable transport used by Tor [8] designed to resist both passive detection and active probing. Traffic is obfuscated by encrypting it and sending headerless ciphertext messages. Similar to OSSH, clients can only connect to `obfs4` servers by proving knowledge of a secret. Probing censors that do not have the secret get no response from `obfs4` servers, making it difficult for censors to confirm if a host is a proxy. Server IPs and their corresponding secrets are normally distributed out-of-band through Tor's bridge distribution system.

During registration, the Conjure client and station could use the registration seed to derive the `obfs4` secrets (NODE_ID and server private/public keys) needed for the client to connect. The station could then launch an `obfs4` server instance locally for the client to connect to as a transport using the derived secrets. If a censor attempts to connect to the phantom address (even using the client's IP), it will not receive a response, as it does not know the registration seed used to derive the `obfs4` secrets.

Using `obfs4` as a Conjure application has the added benefit that servers and secrets do not need to be distributed out-of-band, eliminating one of the main ways censors currently block existing `obfs4` instances [7]. Instead, each Conjure `obfs4` instance is private to its registering client, and there is no public service that censors can use to discover them.

4.2.3 TLS. TLS is a natural protocol for Conjure applications, because it is ubiquitous on the Internet (making it difficult for censors to block), while also providing strong cryptographic protection against passive and active network adversaries. However, there are several challenges to make it robust against censors that wish to block a particular service.

One challenge is that TLS sends important server-identifying content in plaintext during the TLS handshake. This includes the Server Name Indication (SNI) in the Client Hello message that specifies the domain name, and the X.509 Certificate of the server.

To evade censors, we must send a plausible SNI value (sending no SNI is uncommon and easily blocked—only 1% of TLS connections do not send the SNI extension [21]), and we must have the server respond with a plausible (and corresponding) certificate. Even if we manage to avoid sending either in the clear (e.g. using session resumption), censors could actively probe the server in a way that would normally elicit a certificate.

Encrypted SNI TLS 1.3 [46] offers several features that may greatly simplify Conjure transport design. For instance, TLS 1.3 handshakes include encrypted certificates, removing a strong traffic classification feature. Unfortunately, TLS 1.3 currently still sends the SNI in the (plaintext) Client Hello, meaning we would have to choose a realistic domain to fool a censor.

However, there are proposals to encrypt the SNI in the Client Hello [47], though none have been implemented or deployed as of 2019. Nonetheless, if widely adopted, Encrypted SNI (ESNI) would offer a powerful solution for Conjure applications by allowing the client to use plain TLS as the transport while remaining hidden from the censor. Censors could still try to actively probe with guesses for the SNI, but servers could respond with generic "Unknown SNI" errors. If such responses were common for incorrect SNI, the censor's efforts to identify phantom hosts would be frustrated.

Mask Sites Another option to overcome active and passive probing attacks is to mimic existing TLS sites. In this application, we simply forward traffic between any connecting clients and a real

TLS site. To a censor, our phantom site will be difficult to distinguish from the actual "mask" site, making it expensive for them to block without potentially blocking the real site. TLS connections to the Conjure station will terminate exactly as connections to the mask site would, with Conjure acting as a transparent TCP-layer proxy between the client and mask site. However, this leaves the application unable to introspect on the contents of the TLS connection to the mask site, as it does not have the client-side shared secrets, and it cannot overtly man-in-the-middle the connection before knowing it is communicating with the legitimate client (and not the censor).

To covertly signal to the relaying application, the client changes the shared secret it derives with the mask site to something that the Conjure station can also derive. The client's first `Application Data` packet is thus encrypted under a different secret than the client/mask site secret. Specifically, the client uses the **seed** sent during registration to derive the pre-master secret for the connection. The new pre-master secret is hashed along with the client and server randoms of the current (mask site) TLS connection to obtain the master secret that determines encryption/decryption/authentication keys.

The Conjure station can determine if the client did this by trial decryption with the master secret derived from the known seed shared at registration. If it succeeds, the client has proved knowledge of the seed, and the application can respond as a proxy. If not, the application simply continues to forward data between the client and the mask site, in case a client's IP was taken over by a censor after registration. As the censor does not have knowledge of the **seed** used in registration, it cannot coerce the application to appear as anything besides the mask site.

Mask Site Selection Selecting which sites to masquerade as must be done carefully to avoid censors being able to detect obvious choices. For example, if a small university network has a phantom host in their network that appears to be `apple.com`, it would be easy for a censor to block as a likely non-legitimate host. Likewise, if a phantom host at an IP address pretends to be a domain that globally resolves to a single (different) IP address, the censor could also trivially identify and block the phantom host. Several approaches are possible:

Nearby sites: pick websites that are legitimately hosted in or near the network of the phantom host addresses effectively creating copies of legitimate sites. However, other signals such as DNS may reveal the true mask site.

Popular sites: choose mask sites from a list such as the Alexa top site [1] list. Although it may be wise to avoid sites that are obviously not hosted in the phantom host address range, such as large companies that run their own data centers and own their own ASN. The list could also be filtered to domains that resolve to different IP addresses from different vantage points, making it harder for a censor to know if a phantom host corresponds to a domain's IP.

Passive observation: collect sites by passively observing DNS requests, TLS SNI, or certificates that pass by at the network tap. This would allow for building a realistic set of sites that are plausibly in the vicinity of the phantom host addresses that pass by the tap.

	OSSH [31]	obfs4 [52]	Mask Sites	TLS eSNI	WebRTC
Active probe resistant	●	●	●	●	●
Randomized or Tunneling	R	R	T	T	T
Known passive attack	[15]	[53]	-	-	-
Conjure implementation	●	○	●	○	○

Table 1: Conjure Applications — "Active probe resistant" protocols are designed to look innocuous even if scanned by a censor. "Tunneling" (T) protocols use another protocol (e.g. TLS) to blend in, while "Randomized" (R) ones attempt to have no discernable protocol fingerprint or headers. For existing protocols, we list any known attacks suggested in the literature that let censors passively detect them. We also list if we have implemented the application in our prototype.

In practice, clients can often try multiple phantom hosts/mask sites over several attempts, as blocking the client outright may negatively impact other unrelated users behind the same network (e.g. in the case of NAT). Thus, even a censor that can block most but not all mask site usage (i.e. by employing website fingerprinting) only delays access, and doesn't prevent it outright.

4.2.4 Phantom WebRTC Clients. Phantom hosts could also pretend to be clients instead of servers. This may potentially give censors less to block on, as actively probing clients commonly returns few or no open ports. A censor may also be hesitant to block client-to-client communication, as it could block peer-to-peer applications as well as many video conferencing protocols. WebRTC is a natural choice for a client-to-client transport in censorship circumvention, and is already used in existing schemes like Snowflake [23]. Conjure could also use WebRTC as the transport protocol, convincing the censor that two clients are communicating.

5 IMPLEMENTATION

We implemented Conjure and deployed a station at a mid-sized transit ISP tapping traffic from a 20 Gbps router. We used PF_RING to consume the 20 Gbps link, and feed it to a custom **detector** written in Rust. The detector processes all packets and watches for new registrations. Once a registration is detected the local application is notified via an out-of-band ZMQ [61] connection, which provides the registering client's IP address, the seed, and other configuration information. We note that this is not along a critical timing path for proxying connections and no client packets are sent over ZMQ.

The detectors forwards all packets destined for a (registered) phantom host address to the local **application** via tun interfaces and `iptables` DNAT rules that rewrite the destination IP, allowing the local application to accept and respond to connections using the native operating system's interface. Figure 4 shows the overall architecture of our implementation, which we describe in the following subsections.

5.1 Detector

We implemented our detector in approximately 1,800 lines of Rust, compared to over 5,000 lines for TapDance (excluding from both auto-generated protobuf code).

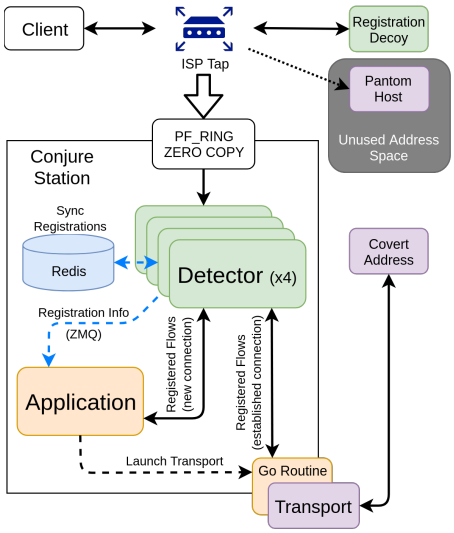


Figure 4: Station Architecture — We used PF_RING to receive packets from a 20 Gbps tap which we load balance across 4 CPU cores. The detector processes identify registrations, sending notification to all other cores via redis and to the local application via ZMQ when a new registration is received. The detector also identifies packets associated with registered flows and diverts them to the local application which handles proxying connections. The local application determines which transport the flow should use based on a parameter specified in the client's registration and initializes a goroutine to handle forwarding. A censor trying to interfere in this connection would need to take over the clients IP address in a precise window (after the registration but before the client connects to the phantom) and correctly guess the phantom host's IP address (or connect to all possible phantom IPs) before the active probe traffic gets to the local application. At this point their connection will fail in a manner specific to the transport that the client specified in the registration (e.g. connecting to OSSH without knowledge of the pre-shared secret).

To achieve performance needed to operate at 20 Gbps, we used PF_RING [42] to load balance incoming packets across 4 CPU cores, which each run a dedicated detector process. PF_RING supports load balancing packets based on either their flow (5-tuple), or their source/destination IPs, which allows connections to be processed by a single process without requiring communication across independent cores.

However, in Conjure, registration connections and phantom proxy connections could end up being load balanced to different cores. In order for an individual detector process to forward phantom proxy connections to the application, it must know about the original registration, even if that registration was observed by a different core. To address this, we used Redis [30] to allow each core's process to broadcast (publish) newly registered decoys to the other cores so they can add them to their local list of registered phantom host addresses. Broadcasting registrations across all cores ensures that each detector process sees all registrations, and can forward phantom proxy connections accordingly.

A second problem arises when considering how to *timeout* unused phantom proxies in the detector. When a phantom proxy is timed out, it no longer responds to any active probes. A naive implementation might simply have each core timeout unused phantom proxies after they go unused for a set time. However, this could leave one core (that sees active proxy use) forwarding packets to the application, while other cores (that do not see use) would timeout the proxy. A censor could probe the phantom proxy and observe this behavior: if the censor's packets are processed on a forwarding core, the censor can establish a TCP connection with the phantom application. Otherwise, if they are processed on a timed out core, the censor's packets will be ignored. Through multiple connections, the censor could use this strange behavior (of intermittent TCP response) to identify and block potential phantom proxies.

To address this issue of differing core timeouts, we implemented a new load-balancing algorithm in PF_RING to select the core based only off the destination IP address of a packet. This means that *all* packets sent to a particular phantom proxy address are processed by the same detector core, which allows the phantom proxy's timeout state to be consistent regardless of what source attempts to connect to it.

5.2 Client

We created a Conjure client and integrated it with the Psiphon [44] anticensorship client. Psiphon has millions of users in censored countries, and we are in the early stages of rolling Conjure out to real-world users.

Our Conjure client is written in Golang, and uses the Refraction Networking tagging schema (Section 4.1) for registration. We note that this protocol will be more difficult for censors to observe than normal TapDance because it consists of only a single (complete) request, and the station does not have to spoof packets as the decoy during registration, only passively observe them.

We implemented support for two of the transports in our client: Obfuscated SSH used by Psiphon (Section 4.2.1) and our TLS Mask Site protocol (Section 4.2.3). Our client signals which transport it will use in the registration tag along with a flag indicating whether the client supports IPv6, allowing IPv4-only clients to derive exclusively IPv4 phantom hosts. After registration, the client connects to the derived phantom host address, and speaks the specified transport protocol, which tunnels between the client and either the mask site transport local to the station or Psiphon's backend servers. A SOCKS connection can be initiated through this tunnel to allow for connection multiplexing.

5.3 Application and Transports

We implemented our station-side application in about 500 lines of Golang. This includes support for OSSH (via integration with Psiphon) and Mask Sites, both specifiable by the client during registration. We note that support for other transport protocols (e.g. Obfs4 or WebRTC) can be added as development continues.

5.3.1 OSSH. Our Conjure implementation includes support for OSSH through integration with Psiphon. Client traffic is forwarded to a Psiphon server by using Conjure as a transparent proxy. This symbiotic relationship provides Conjure with active and passive probe resistance, while preventing censors from being able to inexpensively block Psiphon's backend servers individually.

5.3.2 Mask Sites. We implemented a mask site mimicking proxy, that pretends to be a mask site when actively probed by the censor.

Once the station accepts accepts a connection for a registered flow, it initially acts as a transparent proxy to a mask site specified by the client during registration. The application parses the handshake, forwarding packets back and forth between client and mask site without modification, extracting the server and client randoms. The application attempts to decrypt the first application data record from the client using a key derived from the secret seed, client, and server randoms. We use the uTLS library [21, 41] on both the application and client to allow us to change the TLS secrets being used after the handshake.

If the decryption is successful, the application switches to forwarding (decrypted) data back and forth with a client-specified endpoint, such as a SOCKS proxy, which can provide multiple secure connections over the single connection to the phantom host.

6 EVALUATION

To evaluate our Conjure implementation, we compare its bandwidth and latency to that of TapDance in a realistic ISP setting. We used a 20 Gbps network tap at a mid-sized ISP and run both implementations on a 1U server with an 8-core Intel Xeon E5-2640 CPU, 64GB of RAM, and a dual-port Intel X710 10GbE SFP+ network interface. A typical week of bandwidth seen on the tap is shown in Figure 5, ranging from 2.4 Gbps to peaks above 17 Gbps.

6.1 Performance

We evaluated the performance of a client from an India-based VPS. Figures 6 and 7 show the upload and download bandwidth as measured by iperf for TapDance, our Conjure implementation (using the mask site application), and a direct connection to our iperf server in the ISP's network.

TapDance must reconnect if the amount of data sent by the client exceeds a short TCP window (typically on the order of 32 KBytes) or the connection persists until a timeout (18-120 seconds). At each reconnect, the TapDance client naively blocks until a new TLS connection to the decoy and station has been established. Thus, when uploading files, TapDance has to create a new TLS connection for every 32 KBytes of data it sends, limiting its average upload bandwidth to around 0.1 Mbps due to the high overhead. In contrast, our Conjure implementation is able to maintain the same connection during large uploads, and achieves performance inline with the direct connection, over 1400 times faster.

During download-only workloads, TapDance is able to better utilize the network, but must still reconnect before the decoy times out. In our tests, we see TapDance reconnect every 25 seconds, which can negatively impact the performance of downloads or any real-time streaming applications. Again, our Conjure implementation is able to maintain a single connection and provide the maximum download rate without interruption, 14% faster than TapDance.

We also measure the latency of repeated small requests. In both Conjure (using the OSSH protocol) and TapDance we establish a single session tunnel using our integrated Psiphon client, and make 1000 requests through each using Apache Benchmark (ab). We find that our India-based VPS throttles TLS but not OSSH, making TapDance twice as slow as Conjure. We repeated these tests on a US-based VPS which does not have such throttling, and show results in Figure 8. TapDance's frequent reconnects adds significant latency to about 10% of requests. In addition, the median latency of Conjure is about 19% faster, due to the added overhead of TLS and the complex path that TapDance data packets take through the station compared to Conjure.

6.2 Address Selection

Phantom host IP addresses must be derived from network blocks that are routed (so they pass the Conjure station) and contain other legitimate hosts (so that censors cannot block the entire network without collateral damage). Because of the large number of IPv6 addresses, even moderately-sized network prefixes have astronomical numbers of addresses: a single /32 prefix has 2^{96} possible addresses. Therefore, client-chosen seeds have negligible probability of corresponding to addresses that are already being used by legitimate hosts. This allows us to select phantom host addresses from network prefixes that contain legitimate hosts—crucial to discouraging the censor from blocking them outright—without worry that registrations could interfere with legitimate services.

6.2.1 IPv4. While Conjure works best with IPv6, it can also support IPv4, with some careful caveats.

First, in IPv4, there are substantially fewer addresses, allowing censors to potentially **enumerate all the network prefixes** that pass by the ISP station, compose the list of innocuous sites, and block other websites, as they are being summoned by Conjure. To address this, Conjure phantom hosts are firewalled from all IPs other than the client that registered them, providing a reason why the address hasn't been seen in an enumerating scan, conducted by a censor from a single vantage point. Censors could attempt to scan the network from *all* potential client vantage points, by co-opting client IPs to perform scans—a behavior previously observed by the Great Firewall of China to scan for Tor bridges [14]. To prevent this, for IPv4 Conjure, we dynamically generate the TCP port of the phantom host (along with its IP) from the registration seed, which further makes exhaustive scans infeasible: a censor that must enumerate from the vantage point of a /10 of client IPs (4 million IPs) to a /16 (65K IPs) of potential phantom proxies on each of 65K potential ports would take nearly 50 years of scanning with ZMap at 10 Gbps. We note that while the use of non-standard ports could potentially be suspicious, several successful circumvention tools—including Psiphon [44] and `obfs4` [52]—use random ports on their obfuscated protocols. Finally, we note that censors that whitelist either standard ports or discovered hosts from enumerations scans would over-block new services that came online after their scans.

A second problem in IPv4 Conjure is that the limited range of IPs (and ports) makes it possible for a censor to **pre-image the hash** used to derive the phantom address from the seed. Even with the /16 of IPs and all 65K ports, in order to find a seed for any desired address a censor needs to only test an expected 2^{32} possible seeds. The censor could then register a suspected address, and see if it provides proxy access. If it does, the censor learns there is no legitimate service there, and can block it. To combat this, we allow only a single client to register for a particular phantom address at a time. A censor could attempt to register all addresses in an attempt to deny proxy service to legitimate users, but this would be easily observed at the registration system, where rate limits via client puzzles or account-based fees could be enforced.

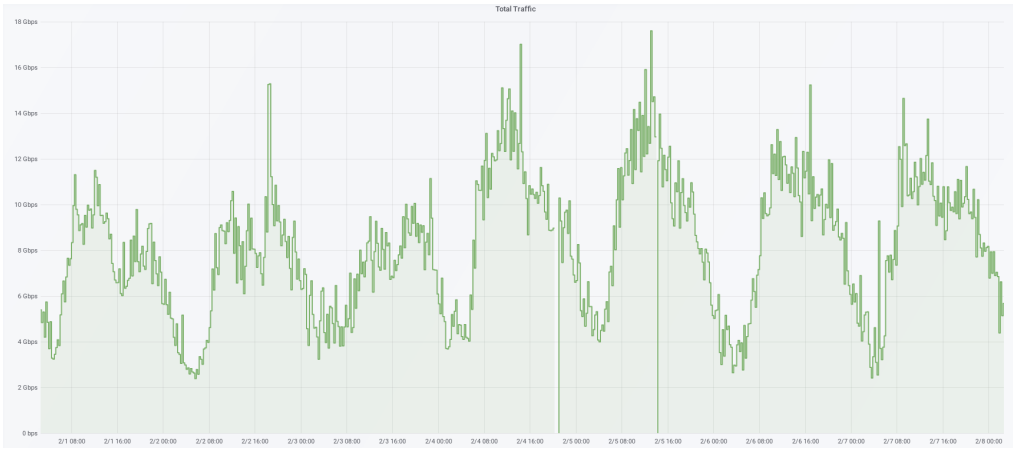


Figure 5: Tap Bandwidth — We deployed Conjure in a realistic ISP testbed on a tap of a 20 Gbps router. In a typical week, traffic ranges from 2–17 Gbps.

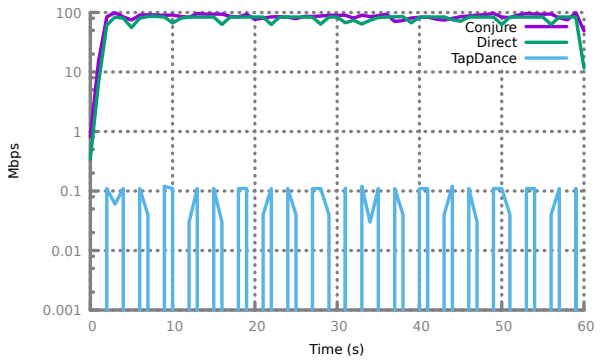


Figure 6: Upload Performance — We used iperf to measure the upload bandwidth for a direct connection, TapDance, and Conjure. As expected, TapDance's upload performance is several orders of magnitude lower than the link capacity, due to the overhead of making frequent reconnects to the decoy site.

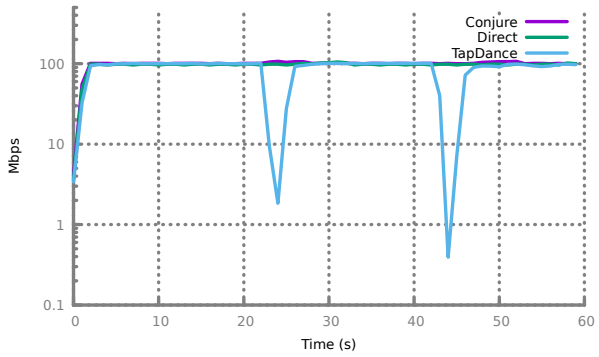


Figure 7: Download Performance — Using iperf, we compare the download performance of a direct connection, TapDance, and Conjure. While TapDance can achieve link-capacity download, it still has to occasionally reconnect to the decoy, as seen by the periodic dips. These reconnects are more common the more data the client sends (e.g. requests or uploads).

Finally, legitimate **IPv4 addresses density** is much higher than IPv6, increasing the potential for users to accidentally register seeds that derive phantom addresses corresponding to live hosts. To address this, the station sends probes to potential IPv4 phantom hosts during registration, and ignores the registration if a real host responds. Censors that try to register specific phantom proxies will be unable to distinguish if another user has registered it or a legitimate host is there, as in both cases we ignore the censor's registration. This also serves to prevent abuse by attackers that attempt to use the Conjure station to interfere with innocuous services.

6.2.2 IPv6. In IPv6 Conjure address enumeration and pre-image attacks are infeasible due to a large amount of potential IP addresses. Our ISP routes a /32 IPv6 prefix, which provides 2^{96} potential phantom host IP addresses. However, IPv6 addresses often have long runs of 0 bits in them, and rarely use all 128-bits equally. For instance, google.com has the address `2607:f8b0:400f:806::2004`, which only has 24 bits set to 1. Censors might try to use this observation and block high entropy IPv6 addresses.

To measure and quantify this problem, we collected and analyzed 16 hours of netflow data at our ISP tap. We extracted 4013 IPv6 addresses observed in the /32 IPv6 prefix routed by the ISP (out of 32,817 total observed). To confirm the hypothesis that 0 bits are more common in IPv6 addresses, we counted the number of bits set in each address. Figure 9 shows a histogram of the number of bits set and compares it to the histogram of a uniformly random set of addresses. (The random distribution's center is skewed from 64 due to the number of set bits in our fixed /32 network prefix.) Although these distributions are distinguishable, they do have significant overlap. Given enough samples, a censor could trivially tell if the addresses were chosen randomly or were legitimate hosts. However, a censor's job is significantly harder, and they must tell from a single sample which distribution it comes from. The presence of random-looking addresses makes it difficult for censors to block such hosts outright.

Prior work by Foremski et al. [18] has developed models to generate likely IPv6 addresses from a set of known addresses, useful for discovering new hosts to scan given known ones. We use their

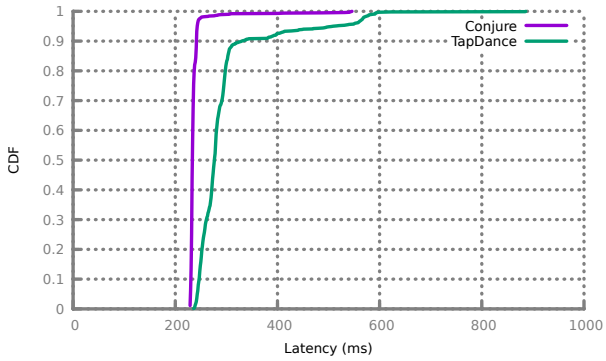


Figure 8: Latency — We compare the CDF of latency for requests between Conjure and TapDance. In each case, we have a long-lived session over which requests are being made using Apache Benchmark (ab). At the median, Conjure has 44 ms (19%) faster latency than TapDance. In addition, TapDance requests are frequently slowed by its intermittent reconnections to the decoy, as shown in the longer tail of TapDance's latency CDF. Conjure has no such reconnects, and thus has more uniform latency. At the 99th percentile, Conjure is 281 ms (92%) faster than TapDance.

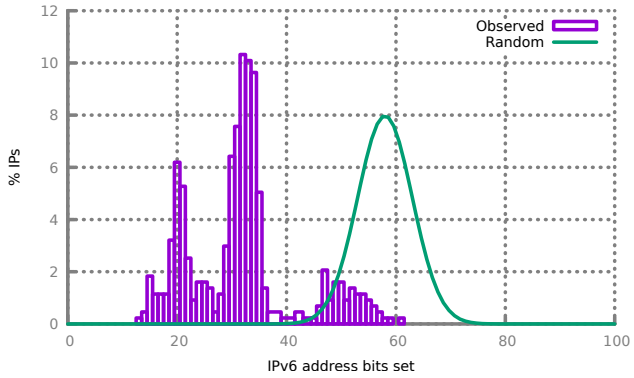


Figure 9: IPv6 Bits Set — We measure the number of bits set to 1 in IPv6 addresses in our ISP's /32 and observed by our tap, and compare it to the Binomial distribution (Random) we would expect to see if the 96 non-network bits of each address were chosen randomly. In practice, we observe much fewer bits set. Nonetheless, the significant overlap of these distributions would make it difficult for censors to block any individual phantom hosts without collateral risk of blocking legitimate services.

Entropy/IP tool to analyze the addresses we collected. Figure 10 shows the normalized entropy of each address 4-bit nibble and the total entropy (18.8 out of 32). Nibbles that were constant across all addresses (such as the /32 network prefix nibbles) have zero entropy, while those that equally span the range of values have maximum entropy (normalized to 1). In our addresses, we observe an entropy of over 75 bits, with more entropy in the later segments of the address. While not quite the full 96 bits that uniformly random would produce, this is still a significant amount for phantom hosts to hide in.

For a specific deployment, operators should be careful to observe the distribution of addresses in the subnets they use, and possibly limit to randomizing "realistic" bits (e.g. the upper and/or last 32 bits within the given /32). As an improvement, we could also use the Entropy/IP tool [18] to generate the random IPv6 phantom hosts from the registration seed based on the Bayesian Network model created by the tool.

7 ATTACKS AND DEFENSES

In this section, we discuss several attacks a censor might attempt to either block phantom hosts from being registered or used.

7.1 Probing phantom hosts

Censors may attempt to actively probe suspected phantom hosts to determine if they are proxies or real hosts. China has been observed using exactly this technique to discover existing proxies and Tor bridge nodes [7, 14, 57, 58]. In response to China's active probing, Tor and other circumvention tool developers have developed *probe-resistant* protocols, such as `obfs4` [52], Obfuscated SSH (OSSH) [31], and Shadowsocks [50]. Each of these protocols require the client to know a secret distributed alongside the original proxy address. Without knowledge of this secret, active-probing censors will not receive any response from these hosts, making it difficult for censors to tell if a server is a proxy or a non-responsive legitimate host.

Obfuscated SSH Modern OSSH protocols are intended to be probe-resistant. Censors that attempt to probe suspected OSSH servers without knowledge of the secret receive no data response, making it difficult to distinguish them from other non-responsive hosts. To confirm this, we use ZMap [11] to scan over 1 billion random IP/port combinations, and sent 25 random bytes (corresponding to the OSSH handshake) to the approximately 800,000 servers that responded. We expect very few of these to actually be OSSH servers as they are simply random servers on random TCP ports. However, over 99.4% of them did not respond with any data, behavior mirrored by OSSH servers. Furthermore, 7.42% of servers closed the connection with a TCP RST after we send our random data, a response we also see with OSSH. While there may be other ways to differentiate OSSH servers from others online, our tests suggest censors could face steep false-positive rates in identifying OSSH servers with active probing.

`obfs4` Unlike previous versions of obfsproxy, the `obfs4` protocol is designed to be resistant to active probing attacks, requiring the client prove knowledge of a secret before the server will respond. We verified that naive active probing attacks (where we attempt to connect to an `obfs4` server without knowledge of the secret and see if it provides proxy access) do not work against `obfs4`. In addition, although China is effective at blocking `obfs3`, the more recent probe-resistant `obfs4` remains a viable proxy in the country.[1]

Mask sites The censor could also attempt to fingerprint a masked site and compare it to a suspected Conjure application. For instance, if a phantom host IP responds to a censors' probes pretending to be `example.com`, the censor could probe real instances of `example.com` on different ports and see how it responds. Then, the censor can probe the phantom host IP, and see if it responds similarly (e.g. with the same set of open ports and payloads for certain kinds of probes). To defend against this, we forward *all* traffic destined to the phantom host to the masked site, including ports that

[1] E.g., https://metrics.torproject.org/userstats-bridge-combined.html?start=2019-06-27&end=2019-09-25&country=cn shows `obfs4` clients from China successfully using Tor.

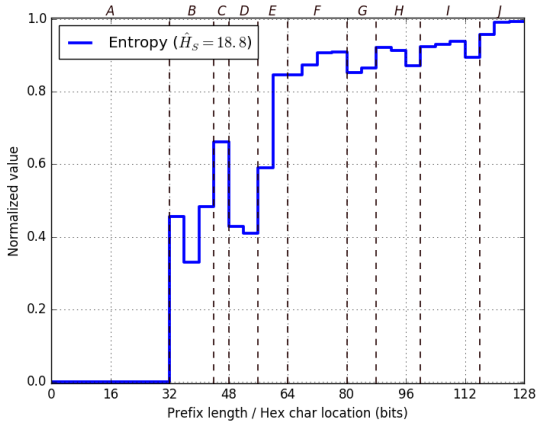


Figure 10: IPv6 Entropy — Censors may be able to distinguish random-looking phantom host IPv6 addresses from legitimately used ones based on the address's entropy. We used the Entropy/IP tool [18] to analyze the entropy of 4013 IPv6 addresses observed by our tap. This plot from the tool shows the normalized entropy of each nibble in the addresses, which is fairly high for most of the non-network prefix nibbles. In total, these addresses have about 75 bits of entropy (out of 96 expected), and the relatively high entropy present in each nibble would make it difficult for a censor to block without significant false positives / negatives. While distinguishable from random given enough samples, we can also use the Entropy/IP tool to generate addresses from the Bayesian Network model it produces.

are not relevant to the proxy application (e.g. non 443). This ensures that above the TCP layer, we appear to be the mask site. However, there may be differences in TCP/IP implementations, for instance, how IP IDs are incremented, or how TCP timestamps are incremented (or supported at all) that may be different from the mask site. To combat this, we can filter mask sites by those that have identical TCP/IP stacks to ours, as we use a common Linux implementation. We also note that this attack only applies when we use the mask site as an application, and that Conjure can support other applications (e.g. obfsproxy, WebRTC, etc) that do not have this issue.

We acknowledge that perfect mimicry of the real site is likely infeasible: Houmansadr et al. [25] demonstrated the difficulty in fully mimicking known applications, showing that staying fully-feature compatible is an expensive and potentially intractable problem for circumvention systems. However, their study focuses on *application-specific* mimicry, used by circumvention tools like CensorSpoofer, StegoTorus, and Skypemorph. These tools generally attempt to mimic a specific protocol used by a complex application such as Skype, where only one or a handful of official popular clients are used. Houmansadr et al. [25] show that such mimicry is difficult in practice due to the complexity and opaqueness of specifications, side protocols that applications may use, and intra- and inter-dependence on connections and other protocols.

In contrast, mask sites have the advantage that we can mimic any sites from a large set of heterogeneous real websites, making it harder for a censor to fully fingerprint and characterize *all* potential mask sites. As websites may themselves use distributed or heterogeneous server implementations, it could be more difficult for a censor to block all mask site application use. Nonetheless, it is still possible for censors to use active probing attacks to detect either protocol feature, payload size, or timing differences between mask sites and the real sites they attempt to mimic.

7.2 Passive identification

Censors might also attempt to use passive methods to distinguish and block communication with registration decoys or phantom hosts. For example, Website Fingerprinting [24, 51, 56] (WF) uses the patterns of encrypted traffic to identify which website a client is connected to. WF uses a classifier to label the traffic as belonging to one of several known websites, or (in some variants), an unknown or background class. Though WF is commonly studied in the context of anonymous web browsing, a censor could also use WF to detect Conjure applications by distinguishing known traffic patterns of legitimate website use from traffic generated by Conjure (unknown class). Alternatively, the censor could monitor traffic for patterns consistent with tunneled access to a set of blocked websites.

Wang et al. [53] demonstrated several effective classifiers that can passively distinguish normal traffic from obfuscated circumvention protocols including `obfs3`, `obfs4`, Format-transforming encryption [12], and meek [17]. However, despite the low false positive rate of these classifiers, Wang et al. [53] caution that the base rate of normal traffic compared to obfuscated traffic would force censors that employ such attacks to suffer significant collateral damage in degrading performance and reliability of legitimate traffic. Indeed, to date, censors have not been observed using WF or other passive techniques, possibly due to the inaccuracy of these identification techniques in practice. Even small false positive rates means blocking mostly legitimate connections, and false negatives could allow clients to retry until they gain access.

More importantly, Conjure applications offer great flexibility in deploying traffic analysis defenses; for example, traffic shaping strategies such as implemented in Slitheen [3] could be easily employed in Conjure. Conjure clients could also choose from a large set of potential applications, forcing censors to have to block access to all of them to block use.

7.3 Blocking registration

To make registration more difficult, censors could block TLS connections to all of the limited decoys available in a deployment. Using TapDance's current deployment, this would involve blocking over 1500 sites. We note that such an attack would completely disable all existing Refraction Networking schemes, as none work without being able to access legitimate decoys past the station. In Conjure, this would only block new registrations, and would not impact users that previously registered. Furthermore, registrations could also occur over email, or over lower-bandwidth covert channels, such as port (or even IP) knocking past the station, that would be more difficult for the censor to block.

7.4 ICMP

Censors can use ping or traceroute utilities (via ICMP) to probe potential phantom hosts. Because there is (usually) no host at the phantom host address, these probes will timeout and produce no response. They might also produce "Destination Unreachable" responses from routers depending on how they are configured. We

performed a scan of 10 million IPv6 addresses in a routable /32 prefix to see if it is common to respond with such tell-tale ICMP messages for unused messages. We found only 0.016% of addresses responded with any ICMP messages (mainly "Time Exceeded" and "Destination Unreachable").

Many legitimate hosts and routers do not respond to or forward ICMP packets, and it is common for firewalls to block traceroutes from penetrating inside networks. Thus, simply ignoring ICMP messages (or low TTL packets that might be used by traceroute) may be a viable strategy. Alternatively, we could spoof responses to convince an adversary that a phantom host is part of a particular network. However, this strategy requires careful consideration of what network makes sense for a mask site to be in. Also, the censor may try to probe for addresses around the phantom host (but still likely to be in the same network), which must also be responded to.

8 RELATED WORK

We first compare Conjure with other Refraction Networking schemes and then discuss other related work.

8.1 Prior Refraction Networking Schemes

Since 2011, there have been several proposed Refraction Networking schemes. Telex [60], Cirripede [26] and Decoy Routing (aka Curveball) [29] are "first generation" protocols with nearly identical features. These designs require inline flow blocking at the ISP to allow the station to intercept flows with detected tags and act as the decoy host for them. However, inline blocking is difficult for ISPs to deploy, as it requires special-purpose hardware to be placed inline with production traffic, introducing risk of failures and outages that may be expensive for the ISP and potentially violate their contractual obligations (SLAs).

TapDance [59] solves the issue of inline-blocking by coercing the decoy into staying silent, and allowing the station to respond instead. However, as previously described, this trick comes at a cost: the decoy only stays silent for a short timeout (typically 30–120 s), and limits the amount of data the client can send before it responds. TapDance clients must keep connections short and repeatedly reconnect to decoys, increasing overhead and potentially alerting censors with this pattern. Conjure addresses this issue and allows clients to maintain long-lived connections to the phantom host.

Rebound [13] and Waterfall [40] both focus on routing asymmetries and routing attacks by the censor. Rebound modifies the client's packets on the way to the decoy, and uses error pages on the decoy site to reflect data back to the client. Waterfall only observes and modifies the decoy-to-client traffic, similarly using error pages on the decoy to reflect communication from the client to the station. These schemes also provide some resistance to traffic analysis, as they use the real decoy to reflect data to the user. Thus, the TCP/TLS behavior seen by the censor more closely matches that of a legitimate decoy connection. However, latency and other packet-timing characteristics may be observable, and both schemes require some form of inline flow blocking.

Slitheen [3] focuses on addressing observability by replacing data in packets sent by the legitimate decoy. Thus, even the packet timings and sizes of a Slitheen connection match that of a legitimate decoy connection. However, Slitheen also requires inline-blocking, and introduces a large overhead as it has to wait for the subset of data-carrying packets from the decoy that Slitheen can safely replace. We note that the Slitheen model of mimicry is compatible with Conjure, as we could use Slitheen as the application protocol. Despite using a passive tap, our scheme is effectively inline to the phantom host (which won't otherwise respond).

Bocovich and Goldberg propose an asymmetric gossip scheme [4] that combines a passive monitor on the forward path from the client to the decoy with an inline blocking element on the return path. These elements work in concert to allow schemes such as Telex and Slitheen to work on asymmetric connections. This approach, however, still requires inline blocking on one direction, and further complicates deployment by requiring the installation of more components and potentially complex coordination between them. MultiFlow [34] uses refraction networking only as a forward mechanism to communicate a web request to the station, and then uses a bulletin board or email to deliver the response back. It does not require inline flow blocking as it does not modify users' traffic at all, but it fundamentally relies on a separate data delivery mechanism, similar to other cloud- or email-based circumvention tools [5, 28].

Conjure allows a large amount of flexibility compared to previous schemes. Because we have significant degrees of freedom in choosing the specific application the phantom host will mimic or talk, our scheme can combine the best of existing Refraction Networking protocols to achieve high performance, be easy to deploy, and also be resistant to active attacks such as replaying or probing by the censor. Table 2 lists the existing Refraction Networking schemes and their features, as compared to Conjure.

8.2 Decoy Placement and Routing Attacks

Houmansadr et al. [26] found that placing refraction proxies in a handful of Tier 1 networks would be sufficient for them to be usable by the majority of the Internet population. Cesareo et al. [6] developed an algorithm for optimizing the placement of proxies based on AS-level Internet topology data. Schuchard et al. [49] suggested that a censor may actively change its routes to ensure traffic leaving its country avoids the proxies, but Houmansadr et al. [27] suggested that real-world constraints on routing make this attack difficult to carry out in practice. Nevertheless, Nasr et al. [39] propose a game-theoretic framework to optimize proxy placement in an adversarial setting, and the design of Waterfall [40] is in part motivated by resilience to routing attacks, as it is more difficult for the censor to control the return path from a decoy site, rather than the forward path.

In practice, deployment of refraction networking has so far been at access, rather than transit ISPs [20]. This may be in part because a transit ISP has a large number of routers and points-of-presence, significantly raising the costs of deployment [22].[2] Likewise, we expect Conjure to use address space announced by the ISP, rather than addresses relayed by it, which mitigates routing-based attacks. Depending on the size of the ISP, however, a censor may decide to block the entirety of its address space, which would incur smaller collateral damage than blocking all addresses seen by a transit ISP.

[2]We note that Gosain et al. [22] use an estimate of $885,000/proxy, while Frolov et al. [20] report line-rate TapDance deployment using commodity hardware that costs only several thousand dollars.

	Telex [60]	Cirripede [26]	Decoy Routing [29]	TapDance [59]	Rebound [13]	Slitheen [3]	Waterfall [40]	**Conjure**
No inline blocking	○	○	○	●	○	○	○	●
Handles asym. routing	○	●	○	●	●	○	●	●
Replay attack resistant	●	●	●	○	●	●	●	●
Traffic analysis resistant	○	○	○	○	◐	●	◐	○
Unlimited Session Length	●	●	●	○	○	○	○	●

Table 2: Comparing Refraction Networking Schemes — "No inline blocking" corresponds to schemes that can operate as a passive tap on the side without needing an inline element in the ISP network. "Handles asymmetric routes" refers to schemes that work when only one direction (either client to decoy or decoy to server) is seen by the station. "Replay attacks" refers to censors who may replay/preplay previous messages or actively probe the protocol. "Traffic analysis" includes latency, inter-packet timing, and website fingerprinting. "Unlimited Sessions" shows schemes that do not need to repeatedly reconnect to download or upload arbitrarily large content.

8.3 Avoiding Destination Blocking

Traditionally, proxies deployed for censorship are eventually identified and blocked by the censor. Several proposals have been made to carefully control the distribution of proxy addresses, using social connections and reputation [9, 38, 55]. Nevertheless, keeping this information secret is challenging; additionally, censors often employ active scanning techniques to discover proxies [10]. Refraction networking generally assumes that clients have no secret information, and instead relies on the collateral damage that would result from blocking all the potential decoy destinations. Conjure furthers this goal by creating a large number of destinations out of the dark space. A similar approach was conceptualized in DEFIANCE [32], where censored Tor clients connect to pools of addresses that are volunteered to run Tor bridge nodes. DEFIANCE also requires volunteer web servers to run specialized servers to distribute information. Unlike Conjure, DEFIANCE was not designed to run at an ISP, and involves many moving parts that present single points of failure if blocked by a censor. In contrast, Conjure has a relatively simple yet flexible design, allowing it to easily respond to censors. Another similar approach was taken by CensorSpoofer [54], which spoofed traffic from a large set of dummy destinations. CensorSpoofer, however, could only send information in one direction—to the client—and had to rely on a separate out-of-band channel for client-to-proxy communication. As an alternative approach, FlashProxy [16] and Snowflake [23] allow users to run Flash- or WebRTC-based proxies within their browser to allow censored users to connect to the Tor network with the potential to greatly increase the number. In practice, these proxies served a very small number of users, as compared with other Tor bridge transports.[3]

9 CONCLUSION AND FUTURE WORK

Conjure provides a much larger degree of engineering flexibility than previous Refraction Networking schemes. Due to its modular design, different registration protocols and proxy transports can be used interchangeably by the client. The flexibility of proxy transports and simplicity of registration allows Conjure to incorporate state of the art censorship circumvention tools and resist nation-state censors.

One obvious future direction is to study new options for registration and proxy transport. For instance, while Conjure currently uses a TapDance- style covert channel for registration, we could potentially cut down on the overhead of one-time registration by using port-knocking or using a Telex-style [60] tag (in the ClientHello rather than Application Data).

Client-side applications Conjure provides an interesting opportunity to explore *client mimicking* phantom hosts. Rather than pretend to be a server (e.g., a mask site), our transport itself could connect to a newly registered client from the phantom host address. Possible protocols could include WebRTC, mentioned in Section 4.2.4, or other peer-to-peer protocols such as BitTorrent, Skype, or Bitcoin.

Traffic analysis Conjure could also support applications that tradeoff performance for observability. While Slitheen offers ideal mimicry of decoys, it comes at a high cost of overhead. Conjure transports such as mask site could implement Slitheen in order to perfectly mimic the decoy site's latency, packet timings, and payload sizes. In addition, careful choice of mask sites may allow for higher performance, as sites with more replaceable content can carry more covert data.

Long-term deployment Ultimately, the goal for Refraction Networking protocols is to be useful in circumventing censorship. While it has taken many years for research protocols to mature, we are excited to see schemes like TapDance deployed in practice [20]. We believe Conjure can be even easier to deploy at scale, and we hope to leverage the existing success of TapDance to place Conjure stations at real ISPs.

ACKNOWLEDGMENTS

The authors thank the incredible partner organizations that have made deployment of Refraction Networking a reality, especially Merit Network and Psiphon. We also thank the University of Colorado IT Security and Network Operations staff. This material is based in part upon work supported by the U.S. National Science Foundation under Awards CNS-1518888 and OAC-1925476.

[3] https://metrics.torproject.org/userstats-bridge-transport.html?start=2017-01-01&end=2019-02-15&transport=!<OR>&transport=websocket&transport=snowflake

REFERENCES

[1] Alexa Internet, Inc. 2019. Alexa Top 500 Global Sites. https://www.alexa.com/topsites.

[2] Daniel J Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. 2013. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *20th ACM Conference on Computer and Communications Security (CCS)*. 967–980.

[3] Cecylia Bocovich and Ian Goldberg. 2016. Slitheen: Perfectly imitated decoy routing through traffic replacement. In *23rd ACM Conference on Computer and Communications Security (CCS)*. 1702–1714.

[4] Cecylia Bocovich and Ian Goldberg. 2018. Secure asymmetry and deployability for decoy routing systems. *Proceedings on Privacy Enhancing Technologies* 2018, 3 (2018), 43–62.

[5] Chad Brubaker, Amir Houmansadr, and Vitaly Shmatikov. 2014. CloudTransport: Using Cloud Storage for Censorship-Resistant Networking. In *The 14^{th} Privacy Enhancing Technologies Symposium (PETS)*.

[6] Jacopo Cesareo, Josh Karlin, Michael Shapira, and Jennifer Rexford. 2012. Optimizing the Placement of Implicit Proxies.

[7] Roger Dingledine. 2011. Research problems: Ten ways to discover Tor bridges. https://blog.torproject.org/research-problems-ten-ways-discover-tor-bridges

[8] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *13th USENIX Security Symposium*. 303–320.

[9] Frederick Douglas, Rorshach, Weiyang Pan, and Matthew Caesar. 2016. Salmon: Robust Proxy Distribution for Censorship Circumvention. *PoPETs* 2016, 4 (2016), 4–20.

[10] Arun Dunna, Ciarán O'Brien, and Phillipa Gill. 2018. Analyzing China's Blocking of Unpublished Tor Bridges. In *Free and Open Communications on the Internet*. 8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18).

[11] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. 2013. ZMap: Fast Internet-Wide Scanning and its Security Applications. In *22nd USENIX Security Symposium*.

[12] K. P. Dyer, S. E. Coull, T. Ristenpart, and T Shrimpton. 2013. Protocol misidentification made easy with format-transforming encryption. In *20th ACM Conference on Computer and Communications Security (CCS)*. 61–72.

[13] Daniel Ellard, Alden Jackson, Christine Jones, Victoria Manfredi, W. Timothy Strayer, Bishal Thapa, and Megan Van Welie. 2015. Rebound: Decoy routing on asymmetric routes via error messages. In *40th IEEE Conference on Local Computer Networks (LCN)*. 91–99.

[14] R. Ensafi, D. Fifield, P. Winter, N. Feamster, N. Weaver, and V. Paxson. 2015. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *15th ACM Internet Measurement Conference (IMC)*. 445–458.

[15] David Fifield. 2017. *Threat modeling and circumvention of Internet censorship*. Ph.D. Dissertation. University of California, Berkeley.

[16] David Fifield, Nate Hardison, Jonathan Ellithorpe, Emily Stark, Roger Dingledine, Phil Porras, and Dan Boneh. 2012. Evading Censorship with Browser-Based Proxies. In *12th Privacy Enhancing Technologies Symposium (PETS)*. 239–258.

[17] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. 2015. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies* 2015, 2 (2015), 46–64.

[18] Pawel Foremski, David Plonka, and Arthur Berger. 2016. Entropy/ip: Uncovering structure in ipv6 addresses. In *Proceedings of the 2016 Internet Measurement Conference*. ACM, 167–181.

[19] Freedom House. 2018. Freedom on the Net 2018: The Rise of Digital Authoritarianism. https://freedomhouse.org/sites/default/files/FOTN_2018_FinalBooklet_11_1_2018.pdf.

[20] Sergey Frolov, Fred Douglas, Will Scott, Allison McDonald, Benjamin VanderSloot, Rod Hynes, Adam Kruger, Michalis Kallitsis, David Robinson, Nikita Borisov, J. Alex Halderman, and Eric Wustrow. 2017. An ISP-scale deployment of TapDance. *Free and Open Communications on the Internet (FOCI)* (2017), 49.

[21] Sergey Frolov and Eric Wustrow. 2019. The use of TLS in Censorship Circumvention. In *2019 Network and Distributed System Security Symposium (NDSS)*.

[22] Devashish Gosain, Anshika Agarwal, Sambuddho Chakravarty, and H. B. Acharya. 2017. The Devil's in The Details: Placing Decoy Routers in the Internet. In *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC 2017)*. ACM, 577–589.

[23] Serene Han. 2017. Snowflake. https://trac.torproject.org/projects/tor/wiki/doc/Snowflake.

[24] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium*. 1187–1203.

[25] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. 2013. The Parrot is Dead: Observing Unobservable Network Communications. In *The 34th IEEE Symposium on Security and Privacy*.

[26] Amir Houmansadr, Giang T. K. Nguyen, Matthew Caesar, and Nikita Borisov. 2011. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *18th ACM Conference on Computer and Communications Security (CCS)*. 187–200.

[27] Amir Houmansadr, Edmund L. Wong, and Vitaly Shmatikov. 2014. No Direction Home: The True Cost of Routing Around Decoys. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014*. The Internet Society.

[28] Amir Houmansadr, Wenxuan Zhou, Matthew Caesar, and Nikita Borisov. 2017. SWEET: Serving the Web by Exploiting Email Tunnels. *IEEE/ACM Transactions on Networking* 25, 3 (Jan 2017).

[29] Josh Karlin, Daniel Ellard, Alden W. Jackson, Christine E. Jones, Greg Lauer, David P. Mankins, and W. Timothy Strayer. 2011. Decoy Routing: Toward Unblockable Internet Communication. In *1st USENIX Workshop on Free and Open Communications on the Internet (FOCI)*.

[30] Redis Labs. 2019. Redis is open source, in-memory data structure store. https://redis.io/.

[31] Bruce Leidl. 2009. Obfuscated SSH. https://github.com/brl/obfuscated-openssh

[32] Patrick Lincoln, Ian Mason, Phillip A Porras, Vinod Yegneswaran, Zachary Weinberg, Jeroen Massar, William Allen Simpson, Paul Vixie, and Dan Boneh. 2012. Bootstrapping Communications into an Anti-Censorship System. In *2nd USENIX Workshop on Free and Open Communications on the Internet*. USENIX.

[33] Colm MacCarthaigh. 2018. Enhanced Domain Protections for Amazon CloudFront Requests. https://aws.amazon.com/blogs/security/enhanced-domain-protections-for-amazon-cloudfront-requests/.

[34] Victoria Manfredi and Pi Songkuntham. 2018. MultiFlow: Cross-Connection Decoy Routing using TLS 1.3 Session Resumption. In *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)*. USENIX Association.

[35] Bill Marczak, Nicholas Weaver, Jakub Dalek, Roya Ensafi, David Fifield, Sarah McKune, Arn Rey, John Scott-Railton, Ron Deibert, and Vern Paxson. 2015. An analysis of China's "Great Cannon". *FOCI. USENIX* (2015), 37.

[36] Moxie Marlinspike. 2016. Doodles, stickers, and censorship circumvention for Signal Android. https://signal.org/blog/doodles-stickers-censorship/.

[37] Moxie Marlinspike. 2018. Amazon threatens to suspend Signal's AWS account over censorship circumvention. https://signal.org/blog/looking-back-on-the-front/.

[38] Damon McCoy, Jose Andre Morales, and Kirill Levchenko. 2012. Proximax: Measurement-driven Proxy Dissemination (Short Paper). In *Proceedings of the 15th International Conference on Financial Cryptography and Data Security (FC'11)*. Springer-Verlag, Berlin, Heidelberg, 260–267.

[39] Milad Nasr and Amir Houmansadr. 2016. GAME OF DECOYS: Optimal Decoy Routing Through Game Theory. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. ACM, 1727–1738.

[40] Milad Nasr, Hadi Zolfaghari, and Amir Houmansadr. 2017. The waterfall of liberty: Decoy routing circumvention that resists routing attacks. In *24th ACM Conference on Computer and Communications Security (CCS)*. 2037–2052.

[41] Refraction Networking. 2019. uTLS—fork of the Go standard TLS library, providing low-level access to the ClientHello for mimicry purposes. https://github.com/refraction-networking/utls/.

[42] Ntop . PF_RING. http://www.ntop.org/products/pf_ring.

[43] Open Whisper Systems . Signal Private Messenger. https://signal.org/.

[44] Psiphon . Psiphon. https://psiphon.ca.

[45] Refraction Routing Site [n.d.]. Refraction Networking: Internet freedom in the network's core. https://refraction.network/.

[46] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446.

[47] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. 2018. *Encrypted Server Name Indication for TLS 1.3*. Internet-Draft draft-ietf-tls-esni-02. Internet Engineering Task Force. Work in Progress.

[48] David Robinson, Harlan Yu, and Anne An. 2013. Collateral freedom: A snapshot of chinese Internet users circumventing censorship. *Open Internet Tools Project Report* (2013).

[49] Max Schuchard, John Geddes, Christopher Thompson, and Nicholas Hopper. 2012. Routing around decoys. In *19th ACM Conference on Computer and Communications Security (CCS)*. 85–96.

[50] Shadowsocks. 2019. Shadowsocks: A secure SOCKS5 proxy.

[51] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1928–1943.

[52] The Tor Project . obfs4 Specification. https://gitweb.torproject.org/pluggable-transports/obfs4.git/tree/doc/obfs4-spec.txt.

[53] Liang Wang, Kevin P Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. 2015. Seeing through network-protocol obfuscation. In *22nd ACM Conference on Computer and Communications Security (CCS)*. ACM, 57–69.

[54] Qiyan Wang, Xun Gong, Giang T. K. Nguyen, Amir Houmansadr, and Nikita Borisov. 2012. CensorSpoofer: Asymmetric Communication using IP Spoofing for Censorship-Resistant Web Browsing. In *Computer and Communications Security*. ACM.

[55] Qiyan Wang, Zi Lin, Nikita Borisov, and Nicholas Hopper. 2013. rBridge: User Reputation based Tor Bridge Distribution with Privacy Preservation. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013*. The

Internet Society.

[56] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective attacks and provable defenses for website fingerprinting. In *23rd USENIX Security Symposium*. 143–157.

[57] Tim Wilde. Jan. 7, 2012. Knock knock knockin' on bridges' doors. Tor Blog. https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors.

[58] Philipp Winter and Stefan Lindskog. 2012. How the Great Firewall of China is Blocking Tor. In *2nd USENIX Workshop on Free and Open Communications on the Internet*. USENIX.

[59] Eric Wustrow, Colleen M. Swanson, and J. Alex Halderman. 2014. TapDance: End-to-Middle Anticensorship without Flow Blocking. In *23rd USENIX Security Symposium*. 159–174.

[60] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. 2011. Telex: Anticensorship in the Network Infrastructure. In *20th USENIX Security Symposium*.

[61] ZeroMQ . ZeroMQ Distributed Messaging. http://zeromq.org/.