REDACT: Refraction Networking from the Data Center

Arjun Devraj
Princeton University
adevraj@alumni.princeton.edu

Liang Wang
Princeton University
lw19@princeton.edu

Jennifer Rexford
Princeton University
jrex@cs.princeton.edu

ABSTRACT

Refraction networking is a promising censorship circumvention technique in which a participating router along the path to an innocuous destination deflects traffic to a covert site that is otherwise blocked by the censor. However, refraction networking faces major practical challenges due to performance issues and various attacks (e.g., routing-around-the-decoy and fingerprinting). Given that many sites are now hosted in the cloud, data centers offer an advantageous setting to implement refraction networking due to the physical proximity and similarity of hosted sites. We propose REDACT, a novel class of refraction networking solutions where the decoy router is a border router of a multi-tenant data center and the decoy and covert sites are tenants within the same data center. We highlight one specific example REDACT protocol, which leverages TLS session resumption to address the performance and implementation challenges in prior refraction networking protocols. REDACT also offers scope for other designs with different realistic use cases and assumptions.

CCS CONCEPTS

• **Networks** → **Network privacy and anonymity**; • **Security and privacy** → **Privacy-preserving protocols**; • **Social and professional topics** → **Censorship**;

KEYWORDS

Refraction networking, Decoy routing, Censorship circumvention

1 INTRODUCTION

With the growth of the Internet, online censorship has become a serious impediment to the freedom of speech and access to information [23]. Censors employ firewalls and other censorship tools to prevent users from accessing restricted websites [23]. Fortunately, researchers have developed many technologies, such as proxy servers, the Tor network [5], and virtual private networks (VPNs), that make use of "relay servers," which covertly redirect traffic, in order to help individuals evade Internet censorship [14, 18]. Nonetheless, powerful adversaries increasingly block censorship circumvention services by detecting relay servers with active probing attacks or traffic analysis, and subsequently blocking them [7, 26].

In order to prevent censors from simply blocking relay servers, researchers developed a more sophisticated approach to censorship circumvention: *refraction networking*, also known as *decoy routing*. In refraction networking, the functionality of the relay server is implemented by a router, referred to as the *decoy router*. The decoy router redirects seemingly innocuous traffic, which appears to be destined to a *decoy site* (a site that is permitted by the censor), to a *covert site* (the blocked site that the client would like to access) instead [14]. Optimal placement of the decoy router forces the censor to face significant collateral damage from filtering the router to prevent use of the refraction networking service because filtering a well-placed router would not only hinder access to the covert site, but also disrupt traffic destined to any permissible sites reached via the decoy router [14].

Existing refraction networking protocols. Researchers have developed several generations of refraction networking protocols. First-generation systems, such as Decoy Routing [14], Telex [28], and Cirripede [13], established fundamental principles for refraction networking and often opted for steganographic tagging schemes for clients to register for the service. Later generations addressed various problems with earlier systems. TapDance [27] eliminates the need for inline blocking, a problem that hindered deployment by ISPs. Rebound [6] and Slitheen [2] offer greater protection against connection probing attacks and some fingerprinting attacks by maintaining the client's connection with the decoy site but require a complex decoy router. Multiflow [15] and SiegeBreaker [22] are more recent systems that solve existing challenges (focused on supporting multiple connections and authenticating clients in Multiflow and overcoming implementation and deployment concerns in SiegeBreaker) but rely on fundamentally different frameworks, whether that involves asynchronous content downloads in Multiflow or the SDN architecture in SiegeBreaker.

Unfortunately, all of these systems are vulnerable to *routing-around-the-decoy attacks* [21]: as long as the censor is able to direct traffic along an alternative path that does not include the decoy router, evading censorship becomes an impossible exercise for the client. While Waterfall [17] uses a mechanism called *downstream-only decoy routing* to mitigate routing around the decoy router, it faces similar real-world deployment challenges as Rebound and Slitheen. Conjure [10] uses phantom IPv6 addresses in the participating ISP's unused address space to avoid any dependence on real decoy sites and hence mitigates routing-around-the-decoy attacks since the ISP-controlled decoy router would intercept all traffic destined to any such decoy "sites" (phantom IP addresses). However, the implementation of Conjure [10] with mask sites involves a complex decoy router and requires the client to use tunneling to protect their privacy, as the decoy router must maintain TLS session state and routinely decrypt and re-encrypt packets between separate TLS sessions. Furthermore, sophisticated fingerprinting attacks may be able to detect the use of a refraction networking service due to clear differences (in operating systems, locations, delays, etc.) between the decoy site and covert site and increased latency resulting from high computational overhead imposed on the decoy router (i.e., complex cryptographic operations).

New design choices and opportunities. With the proliferation of cloud computing, data centers offer a novel paradigm for thinking about refraction networking. Critically, multi-tenant data centers host many different sites in a single physical location. As a result, all traffic directed to and from any tenant in the data center must

traverse a border router of the data center. By selecting a decoy site and covert site in the same data center and using that data center's border router(s) as the decoy router(s), we can eliminate the routing-around-the-decoy attack. Moreover, as tenants in the same data center, the decoy site and covert site likely run on similar hardware and could even run on similar operating systems by using a common VM image extended by the cloud provider [16], so some fingerprinting attacks may become less effective. Additionally, the physical proximity of the decoy site and covert site ensures similar packet round-trip times (RTTs) to the client, further reducing the risk of detection from latency-based analysis attacks. As the trends of colocating multiple sites in the same data center [12] and decoupling IP addresses from physical servers [8] continue to grow, we strongly believe that a refraction networking system that utilizes the many advantages conferred by the multi-tenant data center context will be more robust to the most critical attacks that compromise existing protocols.

Previously-proposed refraction networking protocols usually assume that decoy sites are third-party sites that do not participate in the protocol. Conjure [10] relaxes this assumption and assumes that decoy sites could be set up by the refraction network provider. Moreover, we posit that the decoy site does not necessarily need to be controlled by the refraction network provider—it can be set up by the covert site or volunteers. Such scenarios are realistic: Signal has asked users to set up proxy servers for it because its traffic has been blocked by Iran [1]. In fact, even using a website that is willing to offer limited help as the decoy site, as demonstrated in Section 3, can greatly simplify the design and improve the performance of a refraction networking system, by offloading expensive computations from the decoy router to the decoy site.

Our solution: REDACT. Inspired by these observations, we propose REDACT (Refraction Networking from the Data Center), an efficient and secure refraction networking system that leverages a cooperative data center and decoy site. Compared to existing refraction networking systems, REDACT is able to

(1) Allow an end-to-end native TLS connection between the client and covert site without using any tunneling.
(2) Offload expensive computations to the control plane and other participants, so the decoy router data plane can be as simple as a NAT (thereby improving privacy for the client, as the decoy router cannot decrypt client communications).
(3) Maximize the system similarity and minimize the geographic distance between the decoy site and covert site to reduce the risk of latency analysis and fingerprinting attacks.

We will present the details of an example protocol in Section 3.

One key enabler of our example REDACT protocol is *TLS session resumption*. In our example protocol, a client uses TLS session resumption in order to "migrate" the TLS session endpoint from the decoy site to the covert site. Traditional refraction networking protocols usually only maintain one TCP connection and one TLS session between the client and decoy router, so the TLS session would appear uninterrupted and normal to the censor when the client switches from overt to covert communication. Under this constraint, refraction networking protocols commonly use the decoy router as a TLS MITM proxy, and the client must use another encrypted protocol inside the TLS session to carry the real data (i.e., tunneling) to prevent the decoy router from intercepting its covert communications. However, the one-connection (session) design is not the only way to make the client's behavior appear normal. In TLS session resumption, a session resumption request in close temporal proximity to the original TLS handshake would appear ordinary to the censor, as session resumption could be used for repeated connections to high-traffic sites, which are often hosted in large data centers. It also offers several additional advantages, e.g., improving performance by reducing the handshake overhead and enhancing privacy by preventing the need for an accurate server name indication (SNI) and any server certificate for authentication. Leveraging client-side TLS session resumption, we not only simplify the refraction networking protocol, but also reduce the performance overheads in various parts of the protocol. Particularly, the client no longer needs to use any tunneling techniques during communication with the covert site, unlike domain fronting [9] and other refraction networking protocols. Although TLS session resumption has been used in prior censorship circumvention techniques [15, 20], the sharing of session credentials between two hosts in the same data center to bootstrap the TLS state and enable direct, in-connection covert communication is novel.

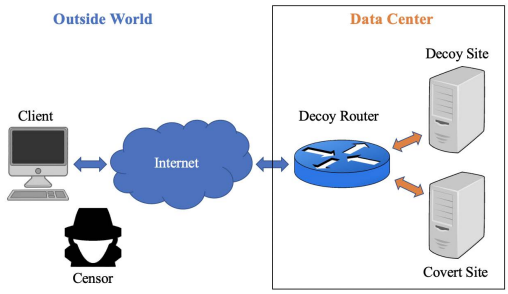


Figure 1: The client, located in a censored network, aims to communicate with the covert site, which is blocked by the censor and is located in a multi-tenant data center.

In this paper, we first discuss the motivating censorship threat model, including the properties of and assumptions about each of the participants, and the REDACT design space aiming to solve this challenge. Next, we delineate the design of an example REDACT protocol, from registering for the service to communicating surreptitiously with the covert site. We also mention a basic proof-of-concept prototype of REDACT in Mininet [11]. Finally, we conduct a security analysis of REDACT and discuss the limitations of the system and areas for future work.

2 PROBLEM STATEMENT

In this section, we first describe the censorship threat model and participants (and their capabilities) in the REDACT class of protocols, and then summarize the key characteristics and areas of variation in the REDACT design space.

2.1 Censorship Threat Model

In our threat model, a **client** in a censored network would like to communicate with a server, the **covert site**, hosted in a multi-tenant data center operated by the **cloud provider**. However, the

censor actively monitors all traffic in its network and blocks any attempts to access the covert site via either passive (e.g., traffic analysis) or active (e.g., probing and replay) attacks. Our protocol makes use of the **decoy site**, also hosted within the same data center but permitted by the censor, to help obfuscate the client's communication with the covert site.

Our assumptions about the censor are similar to those in Conjure [10]. The censor cannot observe any traffic outside of the censored network, particularly traffic exchanged within the data center. The censor may control hosts located behind the same NAT as the client or be able to spoof the client's IP address. The censor can falsely imitate a legitimate client and hence register for and ultimately use REDACT. We assume that the censor can block and permanently blacklist specific IP addresses with unlimited capacity, but that blocking certain IP addresses may induce some collateral damage. Most importantly, we assume that the censor would face substantial collateral damage by blocking IP prefixes for entire multi-tenant data centers and hence is unlikely to do so.

The covert site, decoy site, and cloud provider all participate in REDACT and are *honest* (i.e., they will correctly follow the protocol); no parties will collude with the censor. Nonetheless, the cloud provider is *honest-but-curious* and may attempt to snoop on client communications. The cloud provider deploys the decoy router at the border of the data center to facilitate REDACT. The decoy site can be an independent tenant in the data center, a server controlled by the same entity as the covert site, or even a site deployed by the cloud provider. While we implicitly focus on a decoy site that is controlled by the covert site in the example REDACT protocol outlined in this paper, it would not be unreasonable to consider decoy sites that temporarily volunteer to participate, a scenario similar to uProxy [25], or a privacy extension to HTTPS that enables decoy site participation without impacting performance or significantly modifying the server. We also assume that there are many possible decoy sites to choose from in the data center.

2.2 Design Space

REDACT is a broad design space that allows for the development of a wide range of specific protocols and systems. These protocols vary along two main axes: (1) the degree to which the decoy site participates in the protocol, and (2) the trade-off between overall protocol simplicity and the level of client involvement.

Many of these choices can also be reduced to a question of incentives, a problem that REDACT addresses directly. The design of Conjure [10] and the success of Tor [5] inspire us to consider protocol designs that also involve a decoy site that cooperates with the client to circumvent censorship. Additionally, although there are legitimate concerns about whether major cloud providers would be willing to implement decoy routers given their historical opposition to domain fronting [19], the REDACT design space also involves scenarios in which, for example, a data-center tenant might establish multiple servers and use its load-balancer as the decoy router, without explicit support from the cloud provider.

Some protocols may favor an involved decoy router that could man-in-the-middle connections, bootstrap the covert site, and be privy to cryptographic parameters for the client's connections with the decoy and covert sites. Other protocols may favor an endpoint-heavy design, with a decoy site that would need to be an active and willing participant in REDACT and "migrate" its secure session with the client to the covert site. Similarly, some protocols may allow for an unmodified client but require a system that is more complex overall—say, by altering the TLS stack—while other protocols may require a modified client but enable greater simplicity by working with existing modular components of TLS.

In this paper, we select a particular system within the design space that focuses on extensive decoy site participation and opts for overall protocol simplicity, by leaving TLS untouched, but thereby requires the client to be modified. It is important to keep in mind that the proposed system is just one of many possibilities within the REDACT design space.

3 EXAMPLE REDACT SYSTEM

REDACT enables the client to secretly and efficiently communicate with the covert site by obfuscating any traffic between the client and covert site as being between the client and decoy site. This protocol introduces a new component, the storage server, which is located within the data center and stores the session credentials from the client's TLS session with the decoy site. The unique details of this example REDACT system—based on the the underlying assumptions of the data center setting—are (1) the use of the decoy site (a willing participant) for registration, (2) the typically randomized TCP initial sequence number (ISN) for identification of a legitimate client, and (3) TLS session resumption (enabled by the shared session state in the storage server) to bootstrap the client's communication with the covert site. The steps of this protocol are

(1) The client establishes a secure connection with the decoy site using TLS.
(2) The client registers for REDACT using the decoy site.
(3) The decoy site deposits the session credentials in the storage server, which the covert site reads from to update its local TLS session cache.
(4) The decoy router updates its routing table after receiving a notification from the decoy site.
(5) The client establishes a secure connection with the covert site using TLS session resumption.

A proof-of-concept prototype of this example system (with relaxed registration assumptions and decoy router capabilities) was implemented in Mininet [11] and confirmed that the client can successfully communicate with the covert site by resuming the original session with the decoy site (with a local transfer of session credentials from the decoy site to the covert site).

3.1 Initiating a TLS Session with Decoy Site

The client first performs a normal three-way TCP handshake with the decoy site in order to establish the underlying TCP connection, and then completes a normal TLS handshake with the decoy site in order to establish the encrypted TLS session. The decoy site is permitted by the censor, so the IP address and any other information that may reveal the identity of the decoy site, such as DNS traffic or the SNI in the `ClientHello` of the TLS handshake, do not have to be obfuscated at all. During this phase of the protocol, the decoy router simply forwards packets back and forth.

3.2 Registering for REDACT with Decoy Site

Now that the secure TLS session has been established with the decoy site, the client signals its intent to register for REDACT by sending a HTTPS `GET` request with *registration information*: the domain name of the covert site and a set of TCP ISNs, separated by commas, that it plans to use for later connections with the covert site. (These ISNs will be used to authenticate the client as a valid REDACT registration.) Upon receiving such a request, the decoy site responds by sending a HTTPS response with a 200 "OK" status code and some random data in the response body. This response is necessary to reduce the risk of traffic-analysis attacks that might identify client communications that do not receive any response from the server, or that receive a minimal response, as suspicious.

After receiving this HTTPS response from the decoy site, the client can ignore the random data, confirm that it has registered for REDACT with those specific ISNs, and then terminate its TCP connection with the decoy site. Since registration occurs within the established TLS session between the client and decoy site, all of the communication during registration is encrypted; the censor cannot determine that the client even signaled its intent to use REDACT, let alone discover the domain name of the covert site or the ISNs used for later connections.

3.3 Transferring the TLS Session Credentials

Immediately after receiving the HTTPS registration request from the client, the decoy site needs to transfer the TLS session data to the covert site. One solution is to use a storage server set up by the covert site. The decoy site stores the TLS session data in a session-id-indexed table hosted in the storage server in `<session_id, master_secret>` format. The storage server is only accessible from within the data center in order to protect the confidentiality of session information. Once the session information has been deposited by the decoy site, the storage server informs the covert site, which then reads the table entry and stores it in its own local TLS session cache. At this point, the covert site would be prepared to serve any session resumption requests initiated by the client in the near future. Note that this step occurs concurrently with the sending of the registration response and the subsequent connection termination (Section 3.2) and the updating of the routing table (Section 3.4) in order to ensure read/write synchronization without significant performance overheads.

3.4 Updating the Decoy Router's Routing Table

After responding to the client's registration request with a HTTPS response, the decoy site notifies the decoy router of the registered client's IP address and the client registration information (the domain name of the covert site that the client would like to access and the set of ISNs the client will use for subsequent connections with the covert site). The decoy router updates its routing table to store the mapping between the client's IP address, the set of ISNs, the decoy site's IP address and port, and the covert site's IP address. Later, the client will pick one ISN to initiate a covert site connection that appears to be destined to the same decoy site IP address/port.

For subsequent TCP SYN packets sent by the client and destined to the decoy site, the decoy router will use the ISN to determine whether to forward the packet to the decoy site or the covert site.

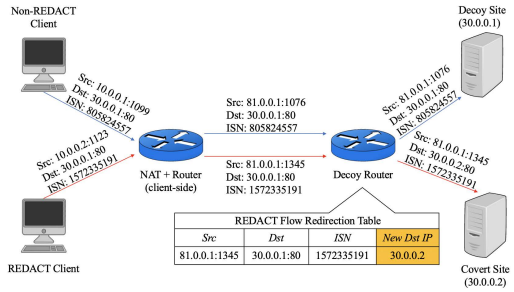


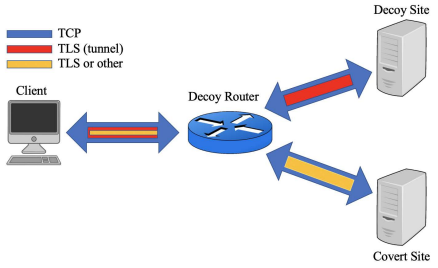
Figure 2: An example of the ISN verification and flow redirection process when the client is located behind a NAT. The arrows denote the SYN packet of a new TCP connection.

If the ISN of a SYN packet matches an ISN dictated during registration (as well as the client's source IP address and the decoy site's destination IP address/port), the decoy router will forward this SYN packet and subsequent packets associated with the TCP connection to the *covert* site (and return traffic will be adjusted accordingly as well); otherwise, the decoy router will forward all packets associated with the TCP connection to the *decoy* site as usual. Once the decoy router has verified the client based on the ISN, it will install a new packet forwarding rule for the client-to-covert-site TCP connection and simply NAT traffic back and forth between the client and covert site (hence avoiding the need to decrypt any secure communications). A similar ISN-based client authentication approach has been used in Waterfall [17]. In the case that the REDACT client is behind a NAT, the use of the ISN reduces the risk of affecting non-REDACT traffic from other users behind the same NAT who may simply want to communicate with the decoy site normally, since the client's IP address will not be specific enough for accurate identification and the client's port number is ultimately controlled by the NAT box and could differ across connections. This process is shown in Figure 2. We assume the client's IP address will not be changed by the NAT for a certain time period.

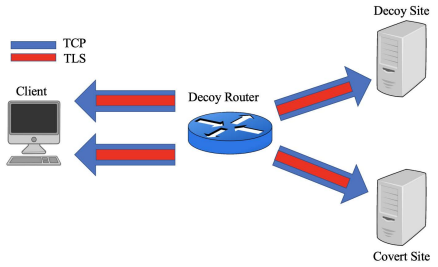
3.5 Resuming the TLS Session with Covert Site

The client initiates its connection with the covert site using TLS session resumption after the first TCP connection (between the client and decoy site) has been terminated. The client begins by establishing a TCP connection with packets destined to the decoy site's IP address; however, as mentioned earlier, the SYN packet must use the ISN specified during registration so that the decoy router knows to establish the TCP connection (by appropriately forwarding handshake packets) with the covert site instead.

Next, the client uses TLS session resumption, with the `session_id` and `master_secret` from the registration session, to establish a TLS session over the TCP connection. The decoy router will forward all packets in this TCP connection to the covert site. The covert site extracts the `session_id` from the `ClientHello` and locates the corresponding `master_secret` in its local TLS session cache. The covert site proceeds as in normal TLS session resumption but instead by using the `master_secret` found in its session cache (as originally obtained from the storage server) for generating session keys in the TLS session. After the handshake, the client and covert



(a) Typical Refraction Networking Systems



(b) REDACT

Figure 3: While existing systems only require one TCP connection from the client to the decoy router, REDACT requires two end-to-end TCP connections, one from the client to the decoy site and another from the client to the covert site. However, existing systems usually require the use of tunneling to prevent the decoy router from intercepting covert communications, while REDACT only requires one end-to-end TLS session, with a simplified decoy router that cannot decrypt client communications.

site communicate seamlessly, with the decoy router simply translating IP addresses in both directions of traffic so that the censor believes that the client is communicating with the decoy site.

In TLS session resumption, the client and server are able to avoid a full handshake overhead by storing the previously used `master_secret` and re-deriving session keys by exchanging solely the `ClientHello` and `ServerHello` (and the `ChangeCipherSpec` and `Finished` messages). In fact, TLS session resumption halves the TLS handshake performance overhead from two RTTs to one RTT. In addition to performance benefits, there are significant privacy and security reasons for using session resumption in our protocol: because session resumption does not involve the sending of the server certificate back to the client, it reduces the risk of covert site detection based on identifiable information in the certificate. This also means the SNI, which reveals the identity of the server, is not an important component of the `ClientHello` for resumption and can be harmlessly spoofed by the client to prevent detection of the covert site. While session resumption makes use of the old `master_secret` generated during the original handshake, new session keys are still computed based on more recently exchanged random values in order to maintain forward secrecy [4].

3.6 Discussion

Decoy router implementation. In our example protocol, the decoy router does not need to perform complicated cryptographic operations. It is relatively easy to implement the decoy router using a programmable switch [3], with the control plane for receiving data from the decoy site and updating forwarding tables, and the data plane for forwarding packets and translating IP addresses.

Honest-but-curious decoy site and decoy router. Our protocol is applicable even for an honest-but-curious decoy site in the data center setting: the decoy site cannot access the client-to-covert-site traffic unless it compromises the data center's networking infrastructure. An honest-but-curious decoy router cannot decrypt the end-to-end encrypted client traffic without the session keys.

Volunteer decoy site. Our protocol only requires minimal modifications to the decoy site, so any sites that are interested in expanding Internet freedom can participate in REDACT with little effort. One may imagine adding a privacy extension to the HTTPS protocol that triggers session data transfer for special requests.

Recovery from failure. The state of the REDACT system can only be compromised at the critical juncture point when the connection "migration" is triggered because this is the only time when the decoy router has to change its packet forwarding rules. Hence, recovery from failure is much simpler in REDACT than in prior systems, as we only need to store a backup of the system before this juncture point in order to avoid a complete restart upon detecting failure.

Session cache and routing table maintenance. The routing table and session cache must be pruned periodically to avoid memory overflow. While we lack data on traffic patterns to impose a strict pruning rule, we discuss the general trade-offs: frequent pruning reduces security threats by disposing of session credentials and redirection information quickly but also forces individual connections with the covert site to be more short-lived. A static pruning rule only requires a single TTL for all cache and table entries and is scalable, as there is no additional per-entry compute or storage.

4 SECURITY ANALYSIS

4.1 Active Attacks

Routing attacks. A powerful censor can route client traffic through network paths that do not contain decoy routers to bypass refraction networks (i.e., routing-around-the-decoy attacks). In REDACT, such attacks are no longer possible because the decoy site is located within the same data center network as the decoy router, which also serves as the border router of the network. Asymmetric routing similarly does not affect REDACT because all traffic sent to/from the decoy site must pass through the border router. REDACT also reduces the complexity of refraction networking, as researchers do not need to evaluate the optimal placement of decoy routers to achieve robust anti-censorship guarantees.

Decoy site detection. The censor can follow the REDACT protocol as a normal client to interact with "suspicious" sites to infer if they are participating in REDACT, and block the discovered decoy sites. In fact, all refraction networking and proxy systems are vulnerable to such attacks in which the censor successfully registers for the

service. However, this type of attack is generally considered to be heavyweight and is not very cost-efficient. Assuming that the censor is unwilling to block the entire data center network due to high collateral damage, the blocked decoy sites can frequently change their IP addresses within the same data center to bypass IP blocking, or use domain fluxing to defeat DNS-based blocking.

Replay attacks. While some protocols may be vulnerable to replay attacks from an incriminating response to a steganographic tag, REDACT conducts registration over an end-to-end TLS session. The censor cannot spoof the client after registration because it does not know the ISNs the client will use for communication (under the assumption that ISN selection is relatively random with a low collision probability); any SYN packet spoofing the client 5-tuple with the incorrect ISN will still be forwarded to the decoy site.

However, consider an attack in which the censor replays the client's second SYN packet (using the registered ISN and secretly intended for the covert site) with a *new* `ClientHello` that *lacks* a `session_id`. In this case, the SYN would be sent to the covert site, allowing the censor to establish a TCP connection with the covert site. Because the `ClientHello` does not contain a `session_id`, the covert site would require a full TLS handshake, resulting in it sending its `Certificate` back to the censor-spoofed client. Thus, sending a blank `ClientHello` after replaying the registered client's SYN will inform the censor that the client is accessing the covert site. Therefore, we added the requirement that in REDACT, the decoy router must forward any replayed SYNs for a registered client to the decoy site; this way, the blank `ClientHello` can never be forwarded to the decoy site and used to incriminate the client.

Connection probing attacks. In many earlier refraction networking protocols, censors could probe the decoy site by sending an in-window TCP packet to determine its true connection state: an error response would indicate that the client is no longer connected to the decoy site and likely using a refraction networking service [6]. However, this attack is only successful if the decoy router does not observe all traffic sent to the decoy site, allowing the probe to bypass the decoy router and reach the decoy site. In REDACT, the decoy router, the data center's border router, observes all traffic to and from the decoy site. An in-window TCP probe would pass through the decoy router and be considered a part of the registered client's flow, thereby diverted to the covert site; the probe would elicit a response, as the client is connected to the covert site at this point, so the censor would not be able to incriminate the client. (To the censor, it would appear like the client is still connected to the decoy site, due to the response received and the NATting of the source and destination IP addresses by the decoy router.) Probes sent during the registration phase of the protocol—when the client is actually connected to the decoy site—would obviously be trivial. Active probing attacks, however, can still compromise REDACT without assumptions about the number of decoy sites, as mentioned earlier (**Decoy site detection**).

4.2 Passive Attacks

Traffic analysis. In existing refraction networking systems, the routing paths taken to the decoy and covert sites can differ substantially, and decoy routers also impose significant per-packet delays due to cryptographic operations. The censor therefore may detect participating clients through latency analysis (e.g., checking the distribution of packet RTTs against the expected distribution for the decoy site). REDACT does not involve any additional per-packet latency since the decoy router simply forwards packets according to rules for the TCP flow. Most importantly, REDACT is resistant to latency analysis because the decoy and covert sites are located in the *same* data center. By geographic fact, it is likely that RTTs for client communications with the decoy and covert sites are extremely similar and indistinguishable to the censor. REDACT currently does not defend against more sophisticated traffic-analysis attacks (e.g., website fingerprinting), but it is relatively easy to incorporate existing client-side countermeasures (e.g., padding) into REDACT.

Performing TLS session resumption shortly after connection establishment seems to be a suspicious behavior. However, the findings in [24] indicate that most servers only allow a session resumption lifetime of less than ten minutes, with the mode being just five minutes, suggesting that initiating session resumption in close temporal proximity to the original TLS handshake could appear relatively "normal." We leave inspecting the typical gap between connection establishment and TLS session resumption as future work.

System fingerprinting. The censor could obtain fingerprints of the TCP/IP stacks of the decoy and covert sites, which could indicate differences in operating systems and alert the censor of the client's use of REDACT. In REDACT, the decoy and covert sites can choose to run on the same operating system, through a common VM image extended by the cloud provider [16], thereby eliminating differences in their TCP/IP stack fingerprints. Similarly, TLS handshake fingerprinting, by analyzing the list of available ciphersuites in the `ClientHello`, is not a problem in REDACT because it is the same client—the REDACT user—who sends the `ClientHello` in both TLS handshakes. It is unlikely that the censor would be able to obtain a fingerprint from the `ServerHello`, which only indicates the ciphersuite that was ultimately selected for the session.

5 CONCLUSION

We propose a novel design space for refraction networking, REDACT, in which the decoy router is a border router of a multi-tenant data center and the decoy and covert sites are tenants in this data center. REDACT defends against latency analysis and routing attacks while also reducing the complexity of and workload imposed on the decoy router under the more stringent assumption that the decoy site willingly participates in the REDACT protocol, and that the decoy and covert sites share similar properties from being colocated in the same data center. Leveraging TLS session resumption, our example REDACT protocol enables an end-to-end native TLS session between the client and covert site, and reduces the performance overhead. We believe that REDACT offers a unique approach to censorship circumvention as a middle ground between the simplicity of traditional proxy servers and the security of refraction networking. Areas for future work include researching typical TLS session resumption traffic patterns, designing pruning rules for the routing table and storage server in our example protocol, and extending REDACT to other transport-layer protocols such as UDP and QUIC.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their detailed feedback. We also thank Jack Wampler and Eric Wustrow for early discussions about Conjure.

REFERENCES

[1] Jeff Benson. 2021. Blocked by Iran, Signal App Moves to Decentralize Servers. (February 2021). https://decrypt.co/56665/blocked-by-iran-signal-app-moves-to-decentralize-servers Accessed on: May 28, 2021.

[2] Cecylia Bocovich and Ian Goldberg. 2016. Slitheen: Perfectly Imitated Decoy Routing through Traffic Replacement. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 1702–1714.

[3] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.

[4] Tim Dierks and Eric Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246. (August 2008). https://rfc-editor.org/rfc/rfc5246.txt

[5] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*.

[6] D. Ellard, C. Jones, V. Manfredi, W. T. Strayer, B. Thapa, M. Van Welie, and A. Jackson. 2015. Rebound: Decoy Routing on Asymmetric Routes via Error Messages. In *Annual IEEE Conference on Local Computer Networks (LCN)*. 91–99.

[7] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. 2015. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *Internet Measurement Conference*. 445–458.

[8] Marwan Fayed, Lorenz Bauer, Vasileios Giotsas, Sami Kerola, Marek Majkowski, Pavel Odintsov, Jakub Sitnicki, Taejoong Chung, Dave Levin, Alan Mislove, Christopher A. Wood, and Nick Sullivan. 2021. The ties that un-bind: decoupling IP from web services and sockets for robust addressing agility at CDN-scale. In *ACM SIGCOMM Conference*. 433–446.

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

[10] Sergey Frolov, Jack Wampler, Sze Chuen Tan, J. Alex Halderman, Nikita Borisov, and Eric Wustrow. 2019. Conjure: Summoning Proxies from Unused Address Space. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2215–2229.

[11] Nikhil Handigol, Brandon Heller, Vimal Jeyakumar, Bob Lantz, and Nick McKeown. 2012. Reproducible Network Experiments Using Container-Based Emulation. In *CoNEXT*.

[12] Nguyen Phong Hoang, Arian Akhavan Niaki, Michalis Polychronakis, and Phillipa Gill. 2020. The web is still small after more than a decade. *ACM SIGCOMM Computer Communication Review (CCR)* 50, 2 (April 2020), 24–31.

[13] Amir Houmansadr, Giang Nguyen, Matthew Caesar, and Nikita Borisov. 2011. Cirripede: Circumvention Infrastructure Using Router Redirection with Plausible Deniability. In *ACM Conference on Computer and Communications Security (CCS)*. 187–200.

[14] 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 *USENIX Workshop on Free and Open Communications on the Internet (FOCI)*.

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

[16] Hodan M. Musse and Lama A. Alamro. 2016. Cloud Computing: Architecture and Operating System. In *Global Summit on Computer Information Technology (GSCIT)*. 3–8.

[17] Milad Nasr, Hadi Zolfaghari, and Amir Houmansadr. 2017. The Waterfall of Liberty: Decoy Routing Circumvention That Resists Routing Attacks. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2037–2052.

[18] Hal Roberts, Ethan Zuckerman, Jillian York, Robert Faris, and John Palfrey. 2010. *2010 Circumvention Tool Usage Report*. Technical Report. The Berkman Center for Internet & Society.

[19] James Sanders. 2018. As Google and AWS kill domain fronting, users must find a new way to fight censorship. (May 2018). https://www.techrepublic.com/article/as-google-and-aws-kill-domain-fronting-users-must-find-a-new-way-to-fight-censorship/ Accessed on: October 8, 2021.

[20] Sambhav Satija and Rahul Chatterjee. 2021. BlindTLS: Circumventing TLS-based HTTPS censorship. In *ACM SIGCOMM Workshop on Free and Open Communications on the Internet (FOCI)*. 43–49.

[21] Max Schuchard, John Geddes, Christopher Thompson, and Nicholas Hopper. 2012. Routing Around Decoys. In *ACM Conference on Computer and Communications Security (CCS)*.

[22] Piyush Kumar Sharma, Devashish Gosain, Himanshu Sagar, Chaitanya Kumar, Aneesh Dogra, Vinayak Naik, H B Acharya, and Sambuddho Chakravarty. 2020. SiegeBreaker: An SDN Based Practical Decoy Routing System. In *Proceedings on Privacy Enhancing Technologies*. 243–263.

[23] Ramesh Subramanian. 2011. The Growth of Global Internet Censorship and Circumvention: A Survey. *Communications of the International Information Management Association (CIIMA)* 11, 2 (October 2011), 33–42.

[24] Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2018. Tracking Users across the Web via TLS Session Resumption. In *Annual Computer Security Applications Conference (ACSAC)*. 289–299.

[25] uProxy. 2017. uProxy: Your private access to the open internet. (2017). https://www.uproxy.org/

[26] Philipp Winter and Stefan Lindskog. 2012. How the Great Firewall of China is Blocking Tor. In *USENIX Workshop on Free and Open Communications on the Internet (FOCI)*.

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

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