# Cirripede: Circumvention Infrastructure using Router Redirection with Plausible Deniability

Amir Houmansadr† Giang T. K. Nguyen‡ Matthew Caesar‡ Nikita Borisov†
†Department of Electrical and Computer Engineering
‡Department of Computer Science
University of Illinois at Urbana-Champaign
{ahouman2,nguyen59,caesar,nikita}@illinois.edu

## ABSTRACT

Many users face surveillance of their Internet communications and a significant fraction suffer from outright blocking of certain destinations. Anonymous communication systems allow users to conceal the destinations they communicate with, but do not hide the fact that the users are using them. The mere use of such systems may invite suspicion, or access to them may be blocked.

We therefore propose Cirripede, a system that can be used for *unobservable* communication with Internet destinations. Cirripede is designed to be deployed by ISPs; it intercepts connections from clients to innocent-looking destinations and redirects them to the true destination requested by the client. The communication is encoded in a way that is indistinguishable from normal communications to anyone without the master secret key, while public-key cryptography is used to eliminate the need for any secret information that must be shared with Cirripede users.

Cirripede is designed to work scalably with routers that handle large volumes of traffic while imposing minimal overhead on ISPs and not disrupting existing traffic. This allows Cirripede proxies to be strategically deployed at central locations, making access to Cirripede very difficult to block. We built a proof-of-concept implementation of Cirripede and performed a testbed evaluation of its performance properties.

## Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; E.3 [**Data**]: Data Encryption; K.4.1 [**Computers and Society**]: Privacy

## General Terms

Algorithms, Design, Security

## Keywords

Censorship-resistance, unobservability

## 1. INTRODUCTION

The Internet has become a serious threat to repressive regimes as it plays an increasingly powerful role in the free circulation of speech, information, and ideas. To counter this, countries that restrict political freedom and corporations that aim to silence whistleblowers limits open access to the Internet [61], thereby disrupting the free flow of information and ideas as well as preventing citizens and employees from sharing their own information with the world. To do this, repressive regimes leverage IP-address blocking, DNS hijacking, and TCP content filtering to block access to certain destinations or to prevent certain forms of content from being transmitted [26, 33]. To ensure compliance and to detect undercover political/social activists, repressive regimes also monitor usage by leveraging Deep Packet Inspection (DPI) and other technologies [33]. Consequences of non-compliance can be severe, ranging from identification and termination of employment to life-threatening prosecutions under repressive governments. Manipulating access to the Internet seems increasingly important in silencing corporate misconduct and for repressive governments to retain their power. Recent events in Tunisia, Libya, and the rest of the Middle East give strong indications that oppressive regimes can even be overthrown by the power of people mobilized to fight by organizing, communicating, and raising awareness through use of the Internet.

To help Internet users retain openness of communication, a number of systems have been developed, including *anti-censorship* and *anonymous communication* systems [2, 8, 12, 16, 30]. These systems, composed of computer and networking technologies, enable evasion of monitoring, blocking, and tracing of the information over the Internet. HTTP proxies [1, 8, 12] are the early circumvention tools that perform simple proxying of the HTTP requests in order to evade the IP-blocking and DNS hijacking techniques used by the early censors. The use of advanced content filtering by the censors [14, 26] led to the advent of more sophisticated circumvention tools such as Ultrasurf [2] and Psiphon [30]. These systems evade blocking techniques by avoiding firewalls and content-filtering systems. To evade monitoring, anonymous communication systems, such as Tor [16], were developed. These systems aim to hide the destinations a user visits from outside observers. Different designs have been proposed for anonymous communication, including the onion routing mechanisms [53] and the mix networks [11], each having different points of strengths and weaknesses.

While these systems are of great benefit, they share one common shortcoming: *they do not hide the fact that the user* *is using these technologies*. For example, a repressive regime in control of its nation's networks may not be able to detect which remote sites a Tor user is visiting, but it will be able to detect that the user is using Tor. This presents several problems. It calls attention to certain users of these services, which may bring about further investigation or detainment. In addition, if use of these services can be detected, access to them can also be blocked. For example, relays that run the Tor service have been blacklisted by several countries [3].

To address these issues, we need technologies that provide *unobservable* communication that circumvent monitoring and censorship technologies. As a first step in that direction, we describe *Cirripede*, a platform for unobservable communication with Internet destinations. From the perspective of a monitoring entity, a user of Cirripede appears to be making regular network connections, while the user is actually getting connected to destinations that are forbidden by that monitoring entity. To do this, Cirripede requires in-network support, through configuration changes and deployment of overlay nodes outside of the repressive regime's network. We envision these in-network changes may be offered as a service by some participating ISPs, or mandated/supported by non-repressive governments (or NGOs) to encourage the freedom of speech abroad.

In order to use Cirripede for the communication, a user needs to be registered in the system. This is performed by using covert channels inside the TCP headers of *legitimate traffic*. We design a cryptographic protocol that ensures that a client's registration messages can only be recognized by the Cirripede registration servers, whereas anyone else cannot distinguish them from regular traffic. The protocol uses public-key technology to eliminate the need of a (long-term) shared key between the Cirripede service and its clients.

The list of the registered clients is provided to border routers of participating ISPs, which *deflect* traffic from the clients to a Cirripede proxy. The proxy tunnels communication between the client and its true destination by replacing the encrypted payload of an HTTPS connection seemingly destined to a benign site. This provides a high-bandwidth channel that can be used to access a wide range of Internet services, including blocked websites and anonymous communication systems such as Tor. By strategically placing deflecting routers in the Internet core, it is possible to make the Cirripede service available to a large user population—we show that using *only two* tier-1 ASes can deliver Cirripede service to *all* Internet hosts. In this case, censors cannot block access to Cirripede without blocking a significant fraction of all Internet sites. Cirripede can operate on the existing router infrastructure using off-the-shelf networking tools, with low-cost commodity servers providing the registration and proxy services.

The rest of this paper is organized as follows; in Section 2, we define the problem targeted in this paper and describe the system model and assumptions. We describe the Cirripede design in Section 3. In Section 4, we address some technical and security issues of Cirripede in practice. Section 5 describes our prototype implementation on commodity hardware and opensource software. Section 6 describes a laboratory evaluation of our prototype implementation as well as studies Cirripede's usability under different deployment scenarios. In Section 7 we discuss the related work. Finally, the paper is concluded in Section 8.

## 2. PROBLEM STATEMENT AND THREAT MODEL

We consider the following problem: a client is being monitored by its host ISP, which we will call the warden. (Note that we use the term ISP loosely here; in the case of China, for example, the warden is comprised of all of the Chinese ISPs and monitors all traffic leaving China.) The client aims to communicate *unobservably* with a *covert destination*, without this communication being detected by the warden either directly or indirectly (by, e.g., observing that the client is using some sort of anonymous communication system). We assume that the warden can inspect the entirety of the traffic between the client and the outside world. Additionally, the warden can *block* traffic with certain destinations (including the covert destination), based on simple IP filtering and/or deep packet inspection. We assume, however, that the warden is only willing to perform *selective* blocking and is unwilling to block Internet access entirely, use a small "white list" of allowed sites, or block all HTTPS (port 443) traffic. As evidenced by the shutdown of Internet access in Egypt in January of 2011, there are some wardens who might take the above actions; we feel, however, that in the majority of cases, the wardens will be reluctant to take such drastic steps. (Indeed, the Internet shutdown in Egypt lasted only a week.) On the other hand, we assume that the warden *is* willing to block certain web sites and services (even popular ones) if it learns that they are being used to circumvent blocking.

Finally, we assume that the client is not privy to any secret information that is unavailable to the warden. In particular, we assume that the warden is aware of all the details of the system design, while the client relies on public information only. This is in contrast to previous proxy-based designs that assume, at a minimum, that the warden cannot learn the addresses of all proxies, yet must make the same proxy addresses available to a large population of users. In our case, the client only needs to obtain a public key of the system, which we assume is also available to the warden; we also assume that the warden can identify where the relevant proxies are located but is unable to prevent access to them without blocking a significant fraction of all Internet sites.

We assume that the warden does not actively tamper with the traffic sent by the client; we leave unobservability with respect to more active wardens to future work.

## 3. Cirripede ARCHITECTURE

The architecture of Cirripede is shown in Figure 1. The main components are:

- Client ($C$): the Internet user who aims to establish an unobservable connection with a covert destination ($CD$).
- Warden ISP: the network provider hosting client $C$.
- Covert destination ($CD$): a website, access to which is blocked by the warden ISP.
- Overt destination ($OD$): a website, access to which is allowed by the warden ISP. $C$ communicates with this destination overtly to provide a carrier channel for covert communication with $CD$.
- Participating ISP: an ISP who participates in Cirripede by deflecting network traffic to Cirripede servers. This ISP must be on the (forward) network path from $C$ to $OD$.

- Cirripede registration server ($RS$): a server that is part of the Cirripede infrastructure, used for registering clients who want to use the Cirripede service.
- Cirripede service proxy ($SP$): another server, also part of the Cirripede infrastructure, that connects with $CD$ and proxies communications with it over the overt communication stream between $C$ and $OD$.
- Deflecting router ($DR$): an Internet router, owned by a participating ISP, that deflects traffic from a registered client $C$ to the service proxy $SP$.

We next provide an overview of the operation of Cirripede.

1. *Client registration:* A client $C$ who wishes to use the Cirripede service first must register with Cirripede. Since its traffic is observed by the warden, $C$ uses a covert channel in TCP headers to signal its intent to register and to establish a secret key, shared with Cirripede, to be used in later communication. The deflecting router $DR$ mirrors part of all TCP traffic it sees to the registration server $RS$, which detects the covert registration message and instructs $DR$ to redirect HTTPS traffic from $C$ to the service proxy $SP$.
2. *Cover connection*: After registration, $C$ makes a TCP connection with the overt destination $OD$, which is deflected by $DR$ to $SP$.
3. *Covert communication*: $C$ performs a TLS handshake with $OD$, which $SP$ interposes upon. After the handshake, $SP$ disconnects from $OD$ and takes over the TLS connection, which is used to tunnel communication between $C$ and $CD$.

We detail these steps in the following sections.

## 3.1 Client registration

The first step in using Cirripede is registration. The client must send a covert message to the registration server while avoiding detection by the warden ISP. We chose to use a covert channel embedded in TCP initial sequence numbers (ISNs) [41]. Each endpoint of a TCP connection selects a new initial sequence number for each connection, to prevent potential confusion between different TCP sessions that use the same ports [44]. These sequence numbers, however, should also be difficult to predict [5]. Modern operating systems, therefore, include a random component in the ISN generation that can be used to carry a covert message. Murdoch and Lewis [41] describe a covert channel that can embed 24 bits of covert information inside an ISN while faithfully mimicking the ISN generation algorithms of either Linux or OpenBSD. We adopt this channel for the purposes of our registration protocol. One particular advantage of using ISNs is that they are observed in the very first packet of the TCP session (the “SYN” packet); thus, to look for registrations sent over this channel, the registration server needs only examine a small fraction of the total packets seen by the deflection router. (See Section 7 for more details.) Each registration is valid for a registration time interval $T$ to minimize the effect of failures, as described later.

### 3.1.1 *Registration Protocol*

The client uses the TCP ISN covert channel to announce its intention to use Cirripede. A simple approach would be to embed a special tag inside the ISN that will be recognized by the registration server. This, however, only obscures the behavior of the client from a warden who is completely unaware of Cirripede; if Cirripede became widely used, the warden could start looking for the embedded tag directly. To provide stronger protection from detection, we use a cryptographic registration protocol based on Diffie-Hellman key exchange [52].

The registration server creates a secret key $k_{RS}$ and a Diffie-Hellman public key $K_{RS} = g^{k_{RS}}$, using some group $G = \langle g \rangle$ where the computational Diffie-Hellman assumption (CDH) is believed to hold. The client generates its own secret key $k_C$ and public key $K_C = g^{k_C}$ and sends it to the registration server over the covert channel. This allows the client and the registration server to establish a shared key, $k_{C,RS} = g^{k_{RS} k_C}$, which is then used to generate a registration tag. In particular, the shared key is used to seed a cryptographically-strong pseudo-random number generator (PRNG), which is then used to produce an $m$-bit tag $\tau$. Note that a secure PRNG will ensure that anyone not in possession of the secret key cannot distinguish it from random; we chose to use the AES-128 cipher in counter mode as the PRNG. Altogether, the client sends the following message over the covert channel:

$$K_C || \text{first } m \text{ bits of PRNG}(k_{C,RS})$$

To implement the Diffie-Hellman protocol, we chose to use Curve25519 [7], a state-of-the-art elliptic-curve Diffie-Hellman design. Curve25519 provides a high level of security and has a very fast implementation due to Bernstein[1]. It also uses a bit string encoding of group elements that makes it difficult for the warden to distinguish $K_C$ from random to detect the use of Cirripede. The underlying group is an elliptic curve over $F_p$, where $p = 2^{255} - 19$, thus a uniformly random element of $F_p$, represented using 255 bits, will be distributed nearly identically to a uniformly random 255-bit string. Furthermore, although elliptic curve points are of the form $(x, y) \in F_p \times F_p$, for the purposes of Diffie-Hellman, it suffices to send the $x$ coordinate only, as Bernstein's implementation does. This allows us to send $K_C$, along with a 33-bit tag $\tau$, using a total of 36 bytes, carried over the ISN covert channel of 12 TCP connections.

The Curve25519 implementation in its recommended usage, however, produces only a fraction of all elements in $F_p$ as public keys. First, the recommended domain for the secret key $k_C$ uses only integers that are 0 mod 8, because active subgroup confinement attacks [34] can be used to determine $k_C$ mod 8. Second, the recommended base point $g = 9$ in fact generates a subgroup of the elliptic curve of prime order $p_1$, whereas the curve itself has size $8p_1$. These choices can easily be adjusted in our scenario; learning $k_C$ mod 8 is not helpful to an attacker, especially since $k_C$ is only used once. Likewise, switching a generator of the full group allows an attacker to determine a few bits of information about $k_C$ but does not invalidate the CDH assumption.

A more significant issue is that only approximately half of the elements of $F_p$ form a valid $x$ coordinate for the elliptic curve, and it is easy for the warden to check whether the client is sending a value that falls into this category over the ISN covert channel. If, over time, all of the ISNs from the client, decoded as elements of $F_p$, are valid $x$ coordinates, the warden can be reasonably certain that the client is in fact using Cirripede. To address this, we follow the approach

[1]http://cr.yp.to/ecdh.html

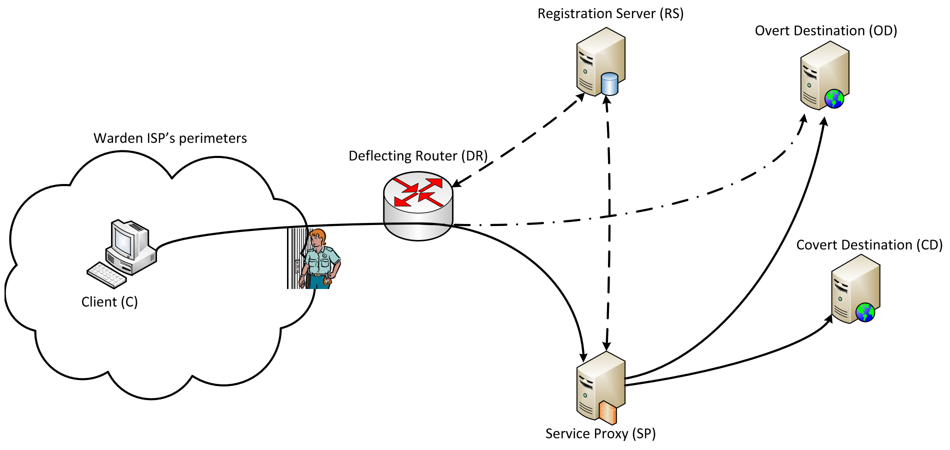


**Figure 1: Cirripede architecture.**

used in Telex [57] and use the *twist* of the elliptic curve. The $x$ coordinates of points on the twist are precisely those points that are invalid $x$ coordinates for the original curve. To avoid certain active attacks, Curve25519 was designed to have a twist group with order $4p_2$, where $p_2$ is a large prime, and thus the CDH is also believed to hold for the twist. The implementation can be made to use the twist group $G'$ simply by choosing a generator $g'$ of $G'$. The registration server has two public keys $K_{RS} = g^{k_{RS}}$ and $K'_{RS} = (g')^{k_{RS'}}$. The client randomly chooses to use the twist or the original curve, each with probability 1/2. The registration server can easily check whether $K_C$ sent by the client belongs to the curve or the twist and use the corresponding secret key to complete the Diffie-Hellman protocol.

## 3.2 Traffic deflection

As mentioned in the previous subsection the $RS$ needs to observe the contents of TCP SYN traffic, in order to perform the registration. To do this, the $DR$s must forward this information to the $RS$. Since the $DR$ may be an existing router within the ISP network, to simplify deployment, we aim to minimize the required changes to $DR$ hardware or vendor software. One way to do this is for the $DR$ to simply forward all traffic to the $RS$ in a similar manner to how scrubbing systems are deployed at some ISP networks (e.g., by using the ERSPAN primitive in Cisco IOS ). Alternatively, the router can be configured to only forward TCP SYN traffic to the $RS$ (e.g., by using the `match field tcp control bits eq 2 mask 0x3D` command in Cisco IOS). The $RS$ in turn can write rules ("deflecting tables") into the $DR$ to forward certain flows to the $SP$, in a similar fashion.

After a client $C$ is registered, the registration server $RS$ informs the deflecting routers to update their deflecting tables by adding an entry for $C$ with the corresponding registration information. The registration information for a client includes the client's IP address and the registration expiration. At this point, any traffic from the client $C$ being intercepted by any of the deflecting routers will be deflected to the service proxy $SP$.

## 3.3 Unobservable communication

Once the traffic from a registered client $C$ is deflected by a deflecting router, the service proxy $SP$ intercepts this traffic in order to start the unobservable communication with the client. The unobservable communication scheme is based on establishing covert communication using legitimate encrypted traffic, e.g., HTTPS traffic. Note that such unobservable communication can also be designed using the existing TCP/IP steganography techniques over a non-encrypted traffic; as mentioned in Section 7, that results in significantly lowering the capacity of the covert communication as compared to using HTTPS.

Figure 2 illustrates the sequence of the messages exchanged within the Cirripede system in order to establish an unobservable communication. Suppose that a client $C$ has successfully been registered by Cirripede; thus, $SP$ possesses the shared secret key $k_{C,RS}$. The client starts an HTTPS connection with an allowed overt destination $OD$. This traffic is deflected by a deflecting router $DR$ toward the service proxy $SP$. Consequently, the following messages are exchanged in order to have the unobservable connection established:

1. Client $C$ initiates an HTTPS connection with $OD$. This connection is deflected to and transparently proxied by the service proxy $SP$ until $C$ and $OD$ complete their TLS handshake. (packets 1 to 4)
2. $C$ sends a legitimate encrypted "application_data" TLS record (e.g., an HTTP GET request) to $OD$ (this is to prevent connection termination due to a *false-connection* failure as discussed in Section 4.1). (packet 5)
3. $SP$ terminates its TCP connection to $OD$ on behalf of $C$. (packet 6)
4. $SP$ uses the shared key $k_{C,RS}$ to derive a new shared TLS session key $k_{C,SP}$ (and initializes a new "cipher spec"[2]). Under the new cipher spec, $SP$ sends an encrypted TLS record of a 64-byte message $M_1 = [M_0||r]$ to $C$ to inform $C$ of $SP$'s presence. $M_0$ is a fixed known 32-byte message and $r$ is a 32-byte random number. (packet 7)
5. ($C$ similarly derives $k_{C,SP}$ and initializes the new cipher spec.) $C$ decrypts the TLS record received from $SP$ and verifies that the cleartext contains the message $M_0$. At this time, $C$ switches to using the new cipher spec for communication with $SP$.
6. $C$ instructs $SP$ to initiate a new connection to covert destination $CD$. (packet 8)
7. Finally, $C$ communicates with $CD$ through the existing

[2]http://tools.ietf.org/html/rfc4346#section-7

TLS connection, while appearing to the warden to be communicating with $OD$.

By having the $CD$ be an entry point to an anonymous network like Tor, the client's traffic gets anonymized from Cirripede as well. This is discussed more in Section 4.3.

## 4. Cirripede IN PRACTICE

In this section we describe and address some of the technical and security issues of the Cirripede scheme. We start by listing two potential cases where Cirripede may not be able to join a new client, and our extensions to address them (Sections 4.1 and 4.2). We then describe some security properties of Cirripede (Section 4.3).

### 4.1 False connection failure

This is the case where an oblivious client $\hat{C}$, who is not trying to use Cirripede, is mistakenly registered by Cirripede's registration server. Because this may interfere with the client's normal Internet activity, Cirripede should minimize the rate of false-connection. As discussed in Section 3.1, the registration tag $\tau$ is 33-bit long. Thus, the false-connection rate is $2^{-33}$, which is practically negligible for a given client. In the rare case of a false connection, the $SP$ informs the $RS$ to remove the mistakenly registered client $\hat{C}$ from the list of the registered clients, and the $RS$ will consequently update the deflecting table of the $DR$ routers by removing the entry corresponding to $\hat{C}$.

The following is another source that may cause a false-connection failure:

**Clients with dynamic IPs:** In many networks, IP addresses are assigned to the clients dynamically, e.g., using the DHCP protocol. In this case, it might happen that an oblivious client $\hat{C}$ gets assigned to an IP address that is currently registered with Cirripede.

Xie et al. [58] show that the inter-user duration of dynamic IP addresses, i.e., the time between two different clients using the same dynamic IP, depends on the type of the ISP and has a direct relation with the bandwidth provided by the ISP, but is typically on the order of several days. As an example, Comcast Cable has an inter-user duration of at least 10 days in more than 75% of the IP re-allocations. For the SBC DSL provider, this interval is about one day. As mentioned in Section 3.1, each client registration in Cirripede is valid for a registration time interval $T$. After the registration expires, the client needs to re-register with the Cirripede in order to use its service; this significantly reduces the rate of false-connection failures caused by dynamic IP reassignments. In fact, the $T$ value makes a tradeoff between the user's need for re-registration and the false-connections due to dynamic allocations of IP addresses.

### 4.2 Mis-connection failure

This is the case when a client $C$ who has requested to be registered by Cirripede, by sending the registration request as mentioned before, has not been registered successfully. Since no confirmation message is sent to the requesting client during the client registration step, due to some technical constraints[3], there should be sufficient mechanisms by Cirripede in order to prevent possible connection failures. In particular, when a not-successfully registered client $C$ tries to communicate with the service proxy $SP$ by sending messages encrypted with $k_{C,SP}$, these messages directly reach $OD$, which cannot interpret them and will tear down the connection. In this case, the warden ISP can infer the client's anomalous behavior by observing frequent connection terminations for the client. To prevent the mis-connection failure, the first message that deviates from the TLS protocol—the "confirmation message"—is sent by $SP$. By having $SP$ send the confirmation message, the client can reliably determine whether to continue communicating with $OD$ or switch to communicating with $SP$.

[3]This requires deflecting routers to embed steganographic messages into the TCP headers.

### 4.3 Security analysis

Table 1 summarizes different security and privacy properties provided by Cirripede, as compared to other systems. A censorship circumvention tool or an anonymous network, e.g., Tor, do not provide unobservability from the warden ISP. Using Cirripede alone provides unobservability from the warden ISP, but it does not provide destination-anonymity from the Cirripede system itself: the service proxy knows the covert destination being targeted by a client. In some cases, a client does not trust the Cirripede service itself and requires destination-anonymity from the Cirripede as well. This can be easily provided by having the client use Cirripede to reach a traditional anonymous network like Tor, which is then used to reach the intended covert destination. In other words, in this case an entry point to the Tor network is requested for the $CD$ destination. An alternative approach for providing destination-anonymity from Cirripede is to design a redundant structure for Cirripede; we leave this for future research.

The mentioned security and privacy promises of Cirripede are based on the assumption that Cirripede's private key is not compromised. Having the private key of Cirripede being compromised results in the exposure of Cirripede users as well as their covert destinations and contents of their covert communications, as is the case in other circumvention services, e.g., Tor.

As mentioned before, Cirripede relies on participating ISPs to deflect HTTPS traffic from registered clients to service proxies. In the following, we discuss the security and privacy promises of Cirripede considering different assumptions for the participating ISP.

**Malicious participating ISPs:** We consider a case where a participating ISP is malicious, i.e., tries to identify clients from its own network that use Cirripede. A participating ISP has access to the deflecting table, e.g., the list of registered clients, provided by Cirripede's registration server. However, as the communication of Cirripede is encrypted using a key shared between Cirripede and a client, a participating ISP will not be able to disclose either the covert destination address or the communicated content.

It may be possible to enforce two requirements to further reduce the consequences of having a malicious participating ISP; first, the participating ISPs are not selected from the countries under the control of the oppressive regimes. This reduces the chances of a malicious ISP to get access to the deflecting table of Cirripede. Second, for each newly registered client $C$ the registration server $RS$ only informs the $DR$s that belong to the participating ISPs other than $C$'s host ISP. Hence, for a malicious ISP participating with Cirripede the maximum information disclosed will be only the identity of the registered clients that reside in other ISPs.

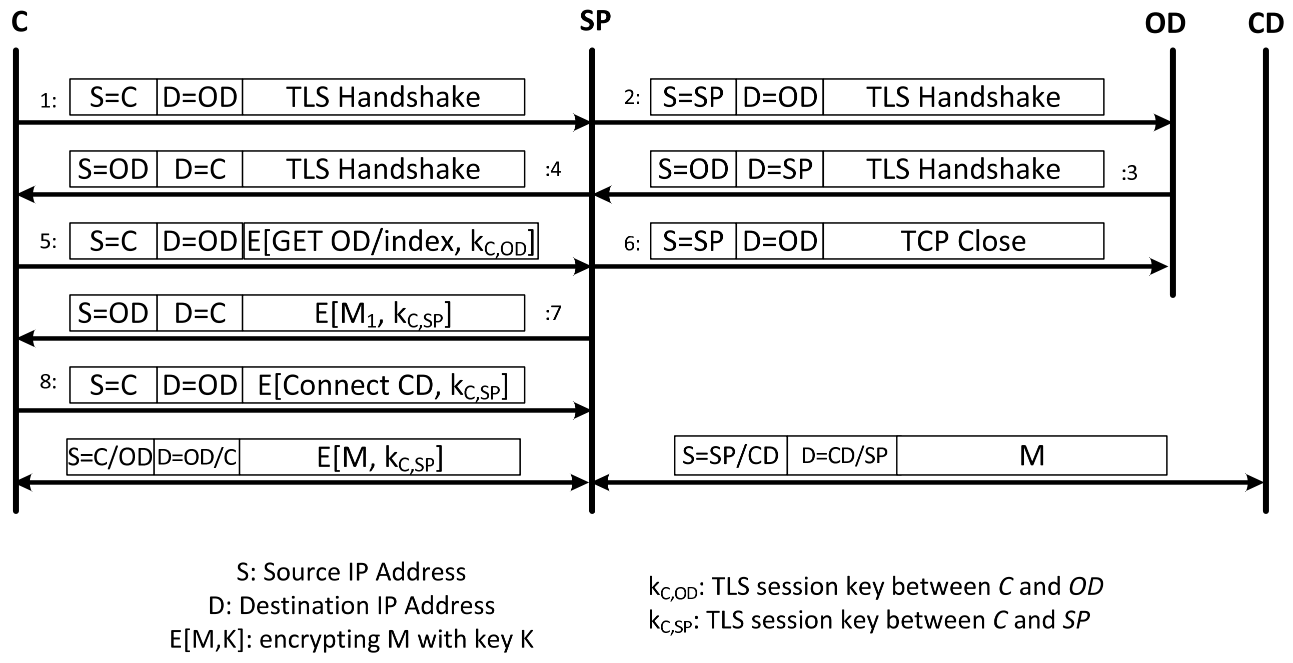

C
SP
OD
CD
1: S=C D=OD TLS Handshake
2: S=SP D=OD TLS Handshake
S=OD D=SP TLS Handshake :3
S=OD D=C TLS Handshake :4
5: S=C D=OD E[GET OD/index, $k_{C,OD}$]
6: S=SP D=OD TCP Close
S=OD D=C E[$M_1$, $k_{C,SP}$] :7
8: S=C D=OD E[Connect CD, $k_{C,SP}$]
S=C/OD D=OD/C E[M, $k_{C,SP}$]
S=SP/CD D=CD/SP M
S: Source IP Address
D: Destination IP Address
E[M,K]: encrypting M with key K
$k_{C,OD}$: TLS session key between *C* and *OD*
$k_{C,SP}$: TLS session key between *C* and *SP*

**Figure 2: The connection sequence to perform the unobservable communication.**

**Table 1: Features provided by using different circumvention services.**

| Circumvention service being used | Source-Anonymity from destination | Destination-Anonymity from warden ISP | Destination-anonymity from circumvention service | Unobservability from warden ISP |
|---|---|---|---|---|
| None | No | No | N/A | No |
| Anonymous Network (Tor) | Yes | Yes | Yes | No |
| Cirripede only | Yes | Yes | No | Yes |
| Cirripede + Tor | Yes | Yes | Yes | Yes |

**Honest but curious participating ISPs:** Similar to the case of a malicious participating ISP, an honest-but-curious ISP can only get information about the identity of registered clients that reside in other ISPs. However, no information about either of the covert destinations, nor the communication contents of the clients is disclosed to the participating ISPs.

## 5. PROTOTYPE IMPLEMENTATION

We have implemented a prototype of Cirripede on Linux.

### 5.1 Deflecting router

Without a real commercial router, we take advantage of the routing capabilities of Linux. The *DR* forwards copies of all TCP SYN packets to the *RS* by using the "TEE" target of `iptables`[4]. In order to deflect packets from registered sources to the *SP*, the *DR* uses policy-based routing. First, we create a new routing table `deflect`, which contains a single default route to the *SP*. (We assume the *SP* can be multiple hops away from the *DR*; thus, to ensure correct routing of deflected packets, which have *OD* as the destination IP address, the *DR* sends those packets to *SP* in an IP-in-IP tunnel.) Second, the *DR* uses the `ip rule` command to match the source IP addresses of registered clients and route them according to the `deflect` table.

[4]http://www.netfilter.org/projects/iptables/index.html

### 5.2 Registration server

The *RS* uses `libpcap`[5] to capture the TCP SYN packets forwarded to it by *DR*. From each captured packet, *RS* extracts the source IP address and the (24 LSB of the) ISN. The *RS* needs to accumulate 12 SYN packets from the client before it can attempt to validate the client as described in Section 3.1. (If the *RS* does not see a new SYN packet from a client after a *validation interval* of $T_v$ seconds, then it removes the state for that client to reclaim memory.) If the client is validated, *RS* (1) notifies *SP* of the client IP address and the key $k_{C,RS}$, and (2) notifies *DR* to start deflecting the client's traffic.

### 5.3 Service proxy

At a high level, *SP* acts as a transparent proxy (i.e., client's packets have the *OD* for destination IP address), so the standard configurations necessary for transparent proxying apply. For the proxy software itself, we use `squid`[6] version 3.1.9, a popular HTTP proxy written in C++. We select `squid` only because it is mature and can act as a transparent proxy. We modify `squid` to receive notification packets from *RS* and maintain a mapping from registered client IP addresses to the corresponding $k_{C,RS}$ keys. For the TLS protocol, `squid` uses the OpenSSL 0.9.8q library[7].

Effectively, *SP* transparently intercepts and tunnels the client's TLS handshake with *OD*. When *SP* detects the

[5]http://www.tcpdump.org/
[6]http://www.squid-cache.org/
[7]http://www.openssl.org/

TLS handshake between the client and *OD* has completed, it "changes the CipherSpec" of the TLS connection with the client to use the stream cipher `RC4` (though a real implementation should use the same cipher as agreed upon by the client and *OD*), and the "CipherSpec" (the cipher key and MAC secrets) is derived from $k_{C,RS}$. *SP* also clears the read and write TLS sequence numbers to 0. The *SP* then (optionally) closes its TCP connection with *OD* and immediately creates a new TCP connection to a local SOCKS proxy, for which we use `3proxy`[8] version 0.6.1. (Alternatively, one could add the SOCKS protocol support into `squid` itself and not have to connect to a separate SOCKS proxy.) Afterwards, `squid` simply tunnels traffic between the client and the SOCKS proxy.

## 5.4 Client-side proxy

It is undesirable to require modifications to existing applications to use Cirripede. Thus we employ at the client host a local proxy, similar to the Tor proxy. This local proxy—henceforth referred to as *client proxy* or *CP*—exposes an apparent (see below) SOCKS interface. The *CP* is configured with the hostnames/IP addresses of two servers to whom packets from the client host will pass through a *DR*, though these two servers can be the same. The first one is used by the registration phase, and the second one is the *OD*. Upon starting, the *CP* will generate TCP traffic towards the *DR* to register itself with Cirripede. The generated TCP traffic needs to contain the special ISNs, so either kernel support or a userspace TCP stack is necessary. Our prototype *CP* simply generates SYN packets using raw sockets, without using a full application connection.

Then, applications at the client host can use *CP* as a regular SOCKS proxy. However, *CP* does not interpret the SOCKS requests. Instead, upon receiving the TCP connection of the request, it initiates an HTTPS connection to the *OD* and proceeds to complete the TLS handshake with *OD*. Then it "changes the CipherSpec" and clears the TLS sequence numbers similar to the *SP*, and then it expects the very next TLS record (of type "application_data") to contain the confirmation message. If that is the case, meaning *CP* is in fact connected to the *SP*, then *CP* proceeds to simply tunnel traffic between the application and *SP* over the TLS channel, without interpreting the SOCKS protocol. Otherwise, it rejects the SOCKS request.

## 6. EVALUATION

We evaluate the registration component and the throughput provided by Cirripede with experiments on the University of Utah's Emulab testbed [55]. Also, we use simulations to study the effect of *DR* deployment on the ability of clients to register with Cirripede.

## 6.1 Registration performance

### 6.1.1 Metrics

We are interested in the ability of the RS to handle real traffic load. The two main metrics are: (1) the fraction of registration signals that the RS can detect, and (2) the load on the RS (in particular, CPU and memory utilization).

[8] http://www.3proxy.ru/

### 6.1.2 Experiment setup and topology

For this experiment, we take an existing packet trace and embed registration signals into the existing packets, without introducing new packets. We use a one-hour trace [32] captured in March, 2011 at CAIDA's equinix-sanjose monitor, and filter it to keep only TCP SYN packets. We assume that all clients in the trace want to register; however, each client registers only once, at the earliest opportunity. Because we do not inject new packets, a client needs at least 12 SYN packets to register. Out of over 94 million SYN packets from over 6.4 million unique client IP addresses, we can embed only 1,069,318 complete registrations.

The experiment consists of two machines: the *DR* and the *RS*. Both are 2.4 GHz 64-bit Quad Core Xeon E5530 machines, with 12 GB of RAM, running Ubuntu 10.04 64-bit. The machines are connected via a 1 Gbps Ethernet link with zero latency. The *DR* simply uses `tcpreplay`[9] to replay the processed packet trace (which contains only TCP SYN packets) against the *RS*. The replay speed is about 41,000 packets/second, resulting in a replay duration of 2300 seconds, which puts more pressure on the *RS* than a live capture would have. The *RS* uses four threads, each handling a different set of clients, partitioned by the hash of the client IP address. The *validation interval* is one hour, effectively meaning the *RS* does not timeout any client during the experiment. The *RS* uses `sar`[10] to collect CPU and memory utilizations at two second intervals.

### 6.1.3 Results

The *RS* is able to receive 100% of packets that the *DR* successfully sends. It detects 1,038,689 registrations, which is a 97% success rate. We manually inspect a few of the missed registrations and find the cause to be out-of-order packets. Investigating further, we find via `tcpdump` that the network is at fault: the *DR* sends packets in the correct order, but they arrive at the *RS* re-ordered. In terms of the load on the *RS*, for the duration of the experiment, the average CPU utilization is 56% and the max 73%. The average memory utilization is 1.1 GB and the max 1.6 GB. The memory utilization increases through the experiment, but this is as expected because as noted above the *RS* does not timeout any client. From these results, we believe the registration component of Cirripede scales well.

## 6.2 Throughput performance

### 6.2.1 Metrics

For these experiments, we only measure the performance of downloading data from a web server (thus all clients are pre-registered with Cirripede before each experiment). The first metric we are interested in is the time to download the first byte. This is a measure of the perceived responsiveness of loading a website, especially for fetching small amounts of data such as a static web page. The second metric is the full page download time (though we only download a single file in our experiments, instead of a full web page containing multiple objects, possibly from different servers).

### 6.2.2 Experiment setup and topology

All hosts in the experiments, including the routers, are 2.4 GHz 64-bit Quad Core Xeon E5530 machines with 12 GB

[9] http://tcpreplay.synfin.net/
[10] http://sebastien.godard.pagesperso-orange.fr/

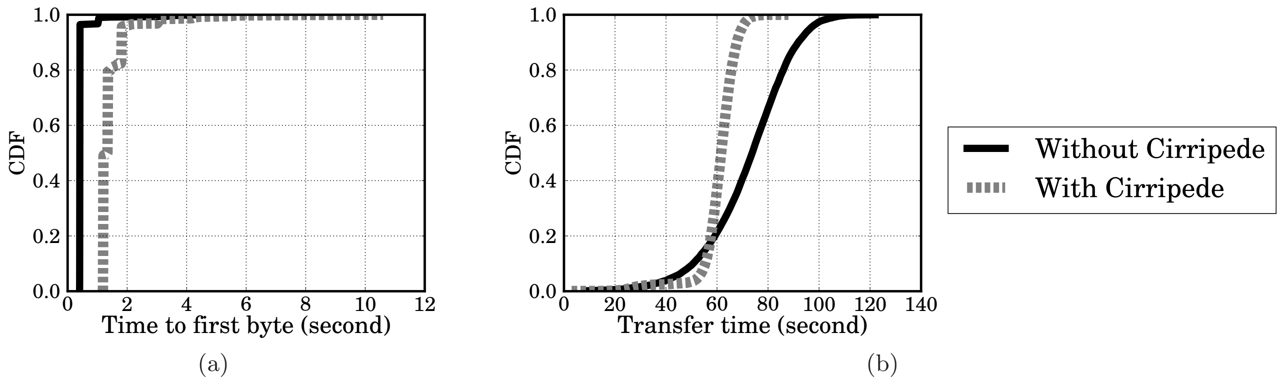


**Figure 3: CDF of the time to receive the first byte and the total time to download a 10 MB file, with and without using Cirripede, for 100 simultaneous clients, each downloading the file 100 times.**

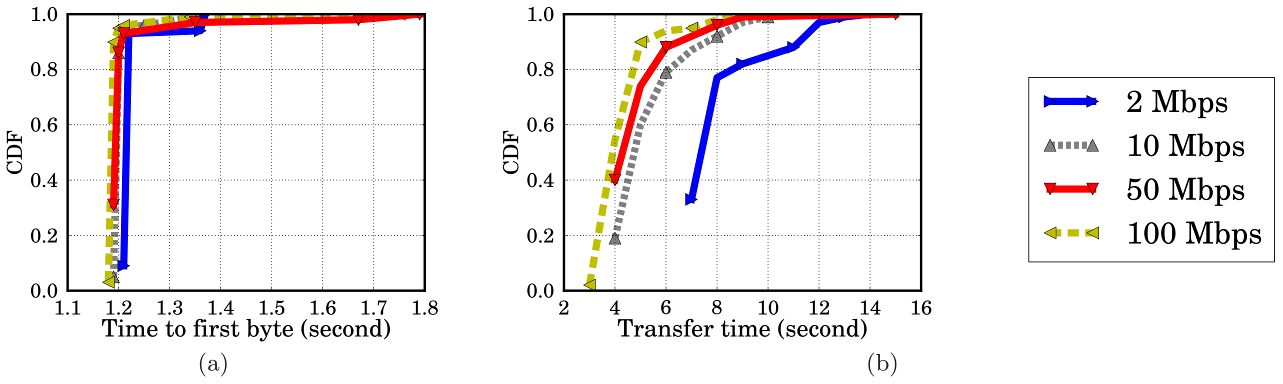


**Figure 4: CDF of the time to receive the first byte and the total time to download a 1 MB file for four simultaneous clients, each downloading the file 100 times using Cirripede.**

of memory, running CentOS 5.5 64-bit. Unless otherwise specified, the links are 1 Gbps. Five servers run the Apache web server[11] version 2.2.3 with SSL support enabled. Five client hosts use `curl`[12] version 7.15.5 as the application to fetch files from the servers, using HTTP. Due to NIC limits, the clients are connected to the *DR* via two intermediate routers; however, in all experiments, the link bandwidths at the client hosts are the bottleneck. The *DR* is on the path between all client-server pairs. The *SP* is directly connected to the *DR*, and both the *SP* and *DR* have RTT of 150 ms to all clients and 50 ms to all servers. Thus, the effective RTT between all client-server pairs is 200 ms, approximating a client in Asia accessing a server in the US.

In the first set of experiments, all five client hosts have link bandwidths of 100 Mbps. On each client host, we launch 20 simultaneous "client" instances, each using `curl` to download a 10 MB file from a particular server 100 times over HTTP. Across all five client hosts, we have 100 "client" instances. We will compare results from using Cirripede and without using Cirripede.

In the second set of experiments, we use four client hosts, with different link bandwidths to the network: 2 Mbps, 10 Mbps, 50 Mbps, and 100 Mbps. Each client host runs only one client instance, using `curl` to download a 1 MB file from a server 100 times. All clients use the Cirripede service.

[11]http://www.apache.org/
[12]http://curl.haxx.se/

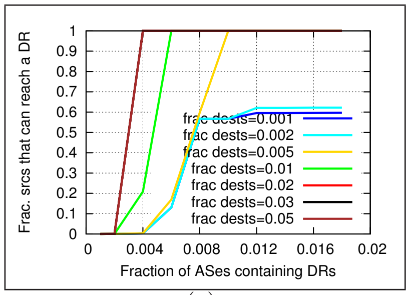
#### *6.2.3 Results*

For the first set of experiments, Figure 3(a) shows the results for the time to the first byte. We see that Cirripede adds a delay of no more than a few seconds, most of which is due to the two extra round-trips of the TLS handshake and the SOCKS request-response. Figure 3(b) compares the total download times. The main take-away point is that Cirripede provides comparable performance to the baseline of not using Cirripede. For this particular setup, Cirripede can also result in faster download time. This is because high latencies negatively affect TCP throughput. So, when we use the *SP*, the original TCP connection is split into two separate TCP connections, each with a lower RTT, so each of these two connections has a higher throughput than the original connection would have. Thus the effective end-to-end throughput is improved. (We performed separate experiments using a standard, non-Cirripede SOCKS proxy to confirm this behavior.)

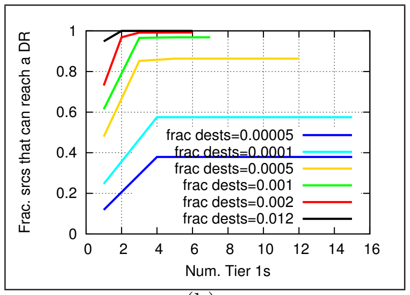
Figure 4 shows the results of the second set of experiments. We can see in Figure 4(a) that as expected, higher bandwidths improve performance. However, because of the high RTTs between clients and servers, increasing the bandwidth yields diminishing returns for a standard TCP.

### 6.3 DR deployment simulation

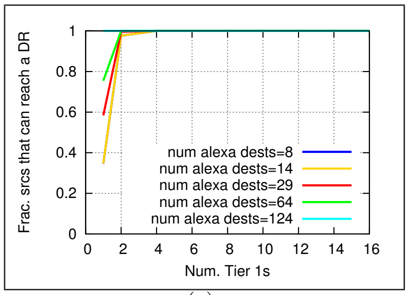
In order to use Cirripede, a client needs to discover a path to a website that traverses a DR. To ensure this happens commonly, the provider of the Cirripede service needs to



(a)

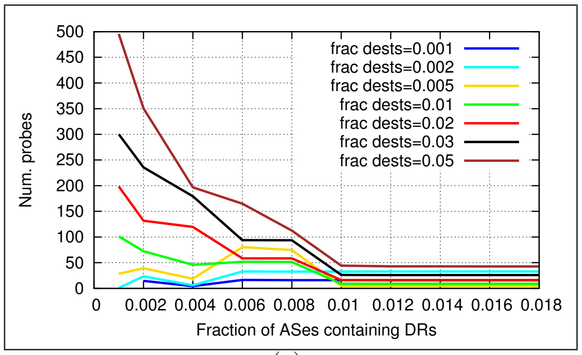


(b)

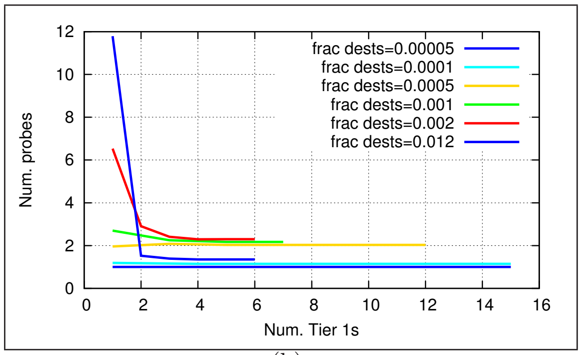


(c)

**Figure 5: Fraction of sources that can utilize the system (i.e., that can discover a path that traverses a *DR*), when *DR*s are randomly placed (a) across all ASes (b) only tier-1 ASes, and (c) when the warden ISP attempts to block a randomly-selected fraction of most-popular web sites by Alexa Internet rank.**



(a)



(b)

**Figure 6: Average number of probes source needs to send before discovering a *DR*, when *DR*s are randomly placed (a) across all ASes (b) only tier-1 ASes.**

make sure that sufficient *DR*s are deployed to "cover" most clients. However, for economic, business, and security considerations, the provider would like the number of *DR*s to be small, and deployed at trust-worthy ISPs or in locations under the provider's control.

To explore deployment considerations in practice, we performed a simulation study. We constructed a simple path-level simulator that reads in the Internet AS-level graph, allows us to place the system components of the Cirripede design at various ASes, and computes shortest policy-safe paths. We used the most recent CAIDA AS-level topology (sampled yearly, most recent sample on January 11, 2011). To compute shortest paths, we applied Gao-Rexford policies to the graph based on CAIDA's inferred AS-relationships. We performed two key experiments:

**Fraction of clients that can utilize the network (Figure 5):** First, we studied how many clients (source hosts) in the Internet would be able to join Cirripede, while varying the extensiveness of *DR* deployment. Figure 5(a) shows this result under the assumption that a random subset of ASes in the Internet act as *DR*s. We also vary the fraction of overt destinations the client is allowed to probe (in practice, the client may wish to limit the set of overt destinations probed, to reduce potential suspicion). We find that even if only 5% of overt destinations are allowed to be probed, then 0.4% of ASes deploying *DR*s is sufficient to enable all hosts in the Internet to join the Cirripede network. In practice, fewer ASes may suffice, if cooperation from ASes located near the repressive country/company are available. In addition, if *DR* functionality is located near the Internet core, fewer ASes suffice, as larger numbers of paths traverse these ASes. To study this, we evaluated performance under the assumption that only tier-1 ASes act as *DR*s, as shown in Figure 5(b). Here we find that even if only one tier-1 AS deploys Cirripede, 97% of clients can use Cirripede, and this number increases to 100% if two tier-1 ASes participate. Since hosts that communicate often via Cirripede may appear suspicious to the warden ISP by contacting randomly-placed destinations, we repeated our study placing destinations according to the Alexa Internet Top Global Sites ranking (Figure 5(c)), and observed similar results (for example, if two tier-1 ISPs run Cirripede, and hosts are allowed to probe among the top-30 most popular sites, 100% of hosts are able to join Cirripede). Here, the client selects among the top-$k$ most popular Alexa-ranked sites (each line in Figure 5(c) corresponds to a different value of $k$). Finally, Figure 5(c) shows the fraction of Alexa-ranked top web sites the warden ISP would have to block to prevent a host from joining Cirripede. We find that if hosts are allowed to probe the top 1000 Alexa-ranked sites, the warden would have to block more than 95% of these sites to be able to block hosts. We also note that if hosts are aware of which sites are blocked (e.g., if their join fails), they could simply continue to probe more sites.

**Overhead required to join the network (Figure 6):** To join the network, the client must discover a path to an overt destination that traverses a *DR*. However, the set of ASes containing *DR*s may wish to keep the fact that they are participating in Cirripede a secret. Hence, the client may have to "probe" multiple potential overt destinations before it can discover a path that traverses a $DR$. To speed joining, we would like the number of probes required to join the network to be low. Figure 6(a) shows the number of probes required under the assumption that a random subset of ASes in the Internet act as $DR$s. Here, we find that even if only 5% of overt destinations are allowed to be probed, the source only requires ten probes before discovering a path contained a $DR$. Under the assumption that $DR$s are only deployed at tier-1 ASes (Figure 6(b)), this value reduces to less than two. This happens because most Internet paths traverse tier-1 ASes, meaning the client usually finds a path through the selected tier-1 on the first probe.

## 7. RELATED WORK

### 7.1 Covert Channels

Covert channels provide a means for unobservable communication. These channels can be classified into covert timing channels and covert storage channels. Timing channels send information by embedding it inside the timing of a stream of packets [9, 49]. The use of timing channels can sometimes be detected using statistical tests by noticing the modifications they make to the ordinary distribution of packet timings [6, 10, 22]. To address this, newer channels use various models to generate timings with a target distribution [23, 29, 35, 48, 60]. However, it may nevertheless be possible to detect the use of such channels if the models do not capture all of the statistical properties of regular traffic; for example, Zander et al. [60] detect the use of covert channels that use i.i.d. traffic models [23, 48] by exploiting inter-packet timing correlations present in regular traffic.

Covert storage channels use some packet fields that are either unused or pseudo-random [24, 27, 41, 46]. Murdoch and Lewis [41] analyze a number of covert storage channels in TCP/IP; they show that a number of channels can be detected because the packet fields differ significantly from those generated by a regular TCP/IP stack. They also propose new covert channels that use TCP initial sequence numbers in a manner that is indistinguishable from either a Linux or an OpenBSD implementation. We use their channel in our registration protocol, see Section 3.1. Other research suggests the use of the IP header TTL field [59], IP addresses [25], TCP timestamp [24], and the order of the packets [21]. Lucena et al. identified a number of IPv6 header field that can be used for covert communication [36], and application-layer protocols, such as HTTP or DNS, can also be used for covert communications [4, 50].

The aforementioned channels typically have limited capacities: storage channels typically send a small number of bits per packet and timing channels likewise are subject to capacity bounds [42, 48]. Fortunately, the growing prevalence of end-to-end encryption creates a possibility for higher-bandwidth covert communication, since an observer not in possession of a secret key cannot learn the contents of an encrypted session. In particular, Cirripede captures HTTPS [45] connections to innocent-looking destinations and uses them for tunneling covert communication. Note that an observer may be able to detect differences in traffic patterns, such as packet sizes and timings, in tunneled traffic, so to ensure full unobservability, techniques to morph the traffic patterns should be used [47, 56]; we leave a full investigation of the application of these techniques to Cirripede to future work.

### 7.2 Blocking Circumvention

Censorship systems, such as the Great Firewall of China, use IP address blocking as their first line of defense [33]; a natural circumvention strategy is, therefore, to connect to forbidden destinations via proxies. This approach is used by the UltraSurf [2] and Psiphon [30] services; likewise, the Tor anonymous communication network [16] uses bridges that act as proxies for clients that cannot connection to Tor directly [15]. A key problem with this approach is how to distribute the IP addresses of proxies to users without their falling into the hands of the censors [19,38,39,51]; over time, it is expected that the censors can enumerate all proxy IP addresses [40], allowing them to block new users as well as identify past users in traces.

Infranet [18] attempts to disguise communication between the client and the circumvention proxy as a normal web browsing session: a client uses a covert channel based on the sequence of HTTP requests to communicate what true destination it wants to reach; the proxy then fetches the data and uses steganography to embed it inside images that it serves back to the client. However, as with other proxy approaches, it relies on the censor not knowing the proxy address. Burnett et al. propose Collage [18], which avoids this problem by using sites that share user-generated content, such as images on flickr.com for carrying forbidden content. Client requests are, likewise, sent via such images, improving unobservability, but significantly reducing the interactive performance of the client. A second problem is that a censor might discover which sites are used for such communication and block them entirely; for example, Burnett et al. consider sending user generated content via Twitter, which has already been blocked by several countries.

A key difference between Cirripede and the above proxy-based approaches is that, in our case, the proxy is embedded within the network itself, and blocking individual web sites is ineffective against Cirripede. Concurrently with our work, Wustrow et al. [57] and Karlin et al. [31] proposed circumvention systems—Telex and Decoy routing—that share this high-level strategy, though make slightly different design choices. Unlike Cirripede, Telex and Decoy routing do not use a registration protocol; instead, clients insert a signal directly into a TLS handshake. This strategy requires the service to monitor all port 443 traffic, including the payload, and reconstruct the underlying TCP sessions. In Cirripede, on the other hand, the registration server monitors only packet headers of TCP SYN packets, and the service proxy monitors only traffic from registered clients. This results in a significant reduction in traffic volumes: using two CAIDA traces gathered in March 2011 [32], we found that the volume (in IP datagram length in bytes) of SYN packets for all ports was only 4–7% that of port 443 traffic. On the other hand, adding signals to individual flows sidesteps issues around packet loss, dynamic IP addresses, and network address translation.

A second consequence of this strategy is that Telex and Decoy routing must hijack a TCP session already in place, whereas Cirripede (transparently) proxies the entire TCP session of registered users. Such hijacking is complicated by the possibility of asymmetric paths, which are common in the Internet [17], since only half of the connection might be seen by the server. Telex is designed to work only for symmetric communication paths, whereas Decoy routing handles this case by having the client use a covert channel to send information about the part of the connection unseen by the router. Hijacking must also be performed in real time, whereas the Cirripede registration process is more tolerant of processing delays at the service.

Telex uses a signaling protocol that, like Cirripede, uses Diffie-Hellman over an elliptic curve; it uses a different curve and has additional security features that protect against potential replay attacks by the warden. Decoy routing opts to use symmetric-key cryptography for signaling in order to achieve better scalability; it therefore requires each client to negotiate a shared key with the service using some out-of-band mechanism. Finally, Telex has some support for mimicking the TCP stack of the overt destination to avoid fingerprinting attacks [37] that could be used by the warden to detect the presence of a proxy. We leave a full analysis of the practical implications of the above design tradeoffs to future work.

### 7.3 Unobservable Communication

Vasserman et al. [54] create a membership concealing overlay network (MCON) for unobservable communication; they aim to make it difficult for either an insider or outsider adversary to learn the set of participating members. All communication in MCON takes place over links between individual members who trust each other. This is similar to previous "darknet" designs, such as WASTE [20], Turtle [43], and recent versions of Freenet [12, 13]; Vasserman et al. identify security flaws in those designs and propose improvements. To be effective, MCON requires a large number of participants to create a well-connected network, so that new clients can easily find points to join, whereas Cirripede requires a small number of servers, albeit well-placed in the network. The MCON evaluation further focuses on communication between members, rather than connection to external destinations. Another system for unobservable communication, NoneSuch [28], uses image steganography, similar to Collage. It uses the Usenet network for exchanging images and does not aim to address blocking by censors.

## 8. CONCLUSIONS

In this paper, we presented Cirripede, a system for unobservable communication with Internet destinations to circumvent monitoring and censoring technologies. Our design leverages in-network support to intercept client requests and redirect them to a covert destination, in a manner difficult to observe by adversaries who may control the client's access network. Through simulations on the Internet topology, we find that a small number of tier-1 ASes deploying our design can provide access to most end hosts in the Internet. Through an emulation-based study of an implementation of our design, we find that our design can process packets with low overheads.

## Acknowledgments

We would like to thank Ian Goldberg, J. Alex Halderman, Josh Karlin, and Vern Paxson for helpful discussions about our design; we in particular are grateful for J. Alex Halderman for providing a detailed comparison of Telex and Cirripede. We also thank anonymous referees for their comments. This work was supported in part by NSF CNS 09-53655 and the Boeing Trusted Software Center at the Information Trust Institute, University of Illinois. Support for CAIDA's Internet Traces is provided by the National Science Foundation, the US Department of Homeland Security, and CAIDA Members.

## 9. REFERENCES

[1] DynaWeb. http://www.dongtaiwang.com/home_en.php.

[2] Ultrasurf. http://www.ultrareach.com.

[3] Tor partially blocked in China, September 2007. https://blog.torproject.org/blog/tor-partially-blocked-china.

[4] M. Bauer. New covert channels in HTTP: adding unwitting Web browsers to anonymity sets. In P. Samarati and P. Syverson, editors, *ACM Workshop on Privacy in the Electronic Society*, pages 72–78. ACM, Oct. 2003.

[5] S. M. Bellovin. Security problems in the TCP/IP protocol suite. *ACM SIGCOMM Computer Communication Review*, 19(2):32–48, Apr. 1989.

[6] V. Berk, A. Giani, and G. Cybenko. Detection of covert channel encoding in network packet delays. Technical Report TR2005-536, Dartmouth College, Computer Science, Aug. 2005.

[7] D. Bernstein. Curve25519: New Diffie-Hellman speed records. In M. Yung, editor, *9th International Conference on the Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, Apr. 2006.

[8] J. Boyan. The Anonymizer: Protecting user privacy on the web. *Computer-Mediated Communication Magazine*, 4(9), 1997.

[9] S. Cabuk. *Network covert channels: Design, analysis, detection, and elimination*. Ph.D. Thesis, Purdue University, Jan. 2006.

[10] S. Cabuk, C. E. Brodley, and C. Shields. IP covert timing channels: Design and detection. In B. Pfitzmann, editor, *11th ACM Conference on Computer and Communications Security (CCS'04)*, pages 178–187. ACM, Oct. 2004.

[11] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[12] I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, and B. Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[13] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In H. Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes in Computer Science*, pages 301–320. Springer, 2000.

[14] R. Clayton, S. J. Murdoch, and R. N. M. Watson. Ignoring the great firewall of China. In G. Danezis and P. Golle, editors, *6th International Workshop on Privacy Enhancing Technologies*, volume 4258 of *Lecture Notes in Computer Science*, pages 20–35. Springer, June 2006.

[15] R. Dingledine and N. Mathewson. Design of a blocking-resistant anonymity system. Technical report, The Tor Project, Nov. 2006. https://svn.

torproject.org/svn/projects/design-paper/blocking.html.

[16] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In M. Blaze, editor, *13th USENIX Security Symposium*, pages 303–320. USENIX Association, Aug. 2004.

[17] M. Faloutsos, S. Krishnamurthy, and B. Huffaker. On routing asymmetry in the Internet. In S. E. Watkins, editor, *IEEE Global Telecommunications Conference (GLOBECOM'05)*. IEEE, Nov. 2005.

[18] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger. Infranet: Circumventing web censorship and surveillance. In D. Boneh, editor, *11th USENIX Security Symposium*, pages 247–262. USENIX Association, Aug. 2002.

[19] N. Feamster, M. Balazinska, W. Wang, H. Balakrishnan, and D. Karger. Thwarting web censorship with untrusted messenger discovery. In R. Dingledine, editor, *3rd International Workshop on Privacy Enhancing Technologies*, volume 2760 of *Lecture Notes in Computer Science*, pages 125–140, March 2003.

[20] J. Frankel. WASTE. http://waste.sourceforge.net.

[21] A. Galatenko, A. Grusho, A. Kniazev, and E. Timonina. Statistical covert channels through PROXY server. In V. Gorodetsky, I. V. Kotenko, and V. A. Skormin, editors, *3rd International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security (MMM-ACNS'05)*, volume 3685 of *Lecture Notes in Computer Science*, pages 424–429. Springer, Sept. 2005.

[22] S. Gianvecchio and H. Wang. Detecting covert timing channels: an entropy-based approach. In S. De Capitani di Vemercati and P. Syverson, editors, *ACM Conference on Computer and Communications Security (CCS'07)*, pages 307–316. ACM, 2007.

[23] S. Gianvecchio, H. Wang, D. Wijesekera, and S. Jajodia. Model-based covert timing channels: Automated modeling and evasion. In R. Lippmann, E. Kirda, and A. Trachtenberg, editors, *Recent Advances in Intrusion Detection*, volume 5230 of *Lecture Notes in Computer Science*, pages 211–230. Springer, 2008.

[24] J. Giffin, R. Greenstadt, P. Litwack, and R. Tibbetts. Covert messaging through TCP timestamps. In R. Dingledine and P. Syverson, editors, *International Workshop on Privacy Enhancing Technologies*, volume 2482 of *Lecture Notes in Computer Science*, pages 194–208. Springer, 2002.

[25] C. G. Girling. Covert channels in LAN's. *IEEE Transactions on Software Engineering*, SE-13(2):292–296, Feb. 1987.

[26] Global Internet Freedom Consortium (GIFC). Defeat internet censorship: Overview of advanced technologies and products. http://www.internetfreedom.org/archive/Defeat_Internet_Censorship_White_Paper.pdf, Nov. 2007.

[27] T. G. Handel and M. T. Sandford. Hiding data in the OSI network model. In R. J. Anderson, editor, *1st International Workshop of Information Hiding*, volume 1174 of *Lecture Notes in Computer Science*, pages 23–38. Springer, 1996.

[28] T. S. Heydt-Benjamin, A. Serjantov, and B. Defend. Nonesuch: a mix network with sender unobservability. In R. Dingledine and T. Yu, editors, *ACM Workshop on Privacy in Electronic Society (WPES'06)*, pages 1–8. ACM, Oct. 2006.

[29] A. Houmansadr and N. Borisov. CoCo: coding-based covert timing channels for network flows. In A. Ker, S. Craver, and T. Filler, editors, *13th Information Hiding Conference (IH'11)*. Springer, May 2011.

[30] J. Jia and P. Smith. Psiphon: Analysis and estimation, 2004. http://www.cdf.toronto.edu/~csc494h/reports/2004-fall/psiphon_ae.html.

[31] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. P. Mankins, and W. T. Strayer. Decoy routing: Toward unblockable Internet communication. In N. Feamster, editor, *1st USENIX Workshop on Free and Open Communications on the Internet (FOCI'11)*. USENIX Association, Aug. 2011.

[32] kc claffy, D. Andersen, and P. Hick. The CAIDA anonymized 2011 Internet traces, Aug. 2011. http://www.caida.org/data/passive/passive_2011_dataset.xml.

[33] C. S. Leberknight, M. Chiang, H. V. Poor, and F. Wong. A taxonomy of Internet censorship and anti-censorship, 2010. http://www.princeton.edu/~chiangm/anticensorship.pdf.

[34] C. Lim and P. Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In B. Kaliski, editor, *Advances in Cryptology (CRYPTO'97)*, volume 1294 of *Lecture Notes in Computer Science*, pages 249–263. Springer, Aug. 1997.

[35] Y. Liu, D. Ghosal, F. Armknecht, A.-R. Sadeghi, S. Schulz, and S. Katzenbeisser. Hide and seek in time—robust covert timing channels. In M. Backes and P. Ning, editors, *14th European Symposium on Research in Computer Security (ESORICS'09)*, volume 5789 of *Lecture Notes in Computer Science*, pages 120–135. Springer, Sept. 2009.

[36] N. B. Lucena, G. Lewandowski, and S. J. Chapin. Covert channels in IPv6. In G. Danezis and D. Martin, editors, *5th International Workshop on Privacy Enhancing Technologies*, volume 3856 of *Lecture Notes in Computer Science*, pages 147–166. Springer, May 2005.

[37] G. F. Lyon. *Nmap Network Scanning*. Nmap project, 1999.

[38] M. Mahdian. Fighting censorship with algorithms. In P. Boldi and L. Gargano, editors, *Fun with Algorithms*, volume 6099 of *Lecture Notes in Computer Science*, pages 296–306. Springer, 2010.

[39] D. McCoy, J. A. Morales, and K. Levchenko. Proximax: A measurement based system for proxies dissemination. In G. Danezis, editor, *Financial Cryptography and Data Security*, 2011.

[40] J. McLachlan and N. Hopper. On the risks of serving whenever you surf: vulnerabilities in Tor's blocking resistance design. In S. Paraboschi, editor, *8th ACM Workshop on Privacy in the Electronic Society*, pages 31–40. ACM, Nov. 2009.

[41] S. J. Murdoch and S. Lewis. Embedding covert channels into TCP/IP. In M. Barni and

S. Katzenbeisser, editors, *7th International Workshop of Information Hiding (IH'05)*, volume 3727 of *Lecture Notes in Computer Science*, pages 247–261. Springer, June 2005.

[42] J. A. O'Sullivan, P. Moulin, and J. M. Ettinger. Information theoretic analysis of steganography. In *IEEE International Symposium on Information Theory*, page 297, 1998.

[43] B. Popescu, B. Crispo, and A. S. Tanenbaum. Safe and private data sharing with Turtle: Friends team-up and beat the system. In B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, editors, *12th Cambridge International Workshop on Security Protocols*, volume 3957 of *Lecture Notes in Computer Science*. Springer, Apr. 2004.

[44] J. Postel. Transmission control protocol. RFC 793, Sept. 1980.

[45] E. Rescorla. HTTP over TLS. RFC 2818, May 2000.

[46] C. H. Rowland. Covert channels in the TCP/IP protocol suite. *First Monday*, 2(5), 1997.

[47] N. Schear. *Preventing Encrypted Traffic Analysis*. Ph.D. Thesis, University of Illinois at Urbana-Champaign, 2011.

[48] S. H. Sellke, C.-C. Wang, S. Bagchi, and N. B. Shroff. TCP/IP timing channels: Theory to implementation. In E. de Souza e Silva, T. Hou, and D. Towsley, editors, *28th IEEE International Conference on Computer Communications (INFOCOM'09)*, pages 2204–2212. IEEE, Apr. 2009.

[49] G. Shah, A. Molina, and M. Blaze. Keyboards and covert channels. In A. D. Keromytis, editor, *15th USENIX Security Symposium*, pages 59–75. USENIX Association, Aug. 2006.

[50] M. Smeets and M. Koot. Research report: Covert channels. `http://gray-world.net/papers/rr_cc_univAmsterdam.pdf`, Feb. 2006.

[51] Y. Sovran, A. Libonati, and J. Li. Pass it on: Social networks stymie censors. In A. Iamnitchi and S. Saroiu, editors, *7th International Conference on Peer-to-peer Systems*, Feb. 2008.

[52] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman key distribution extended to groups. In L. Gong and J. Stern, editors, *3rd ACM Conference on Computer and Communications Security*, pages 31–37. ACM, Mar. 1996.

[53] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an analysis of onion routing security. In H. Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes in Computer Science*, pages 96–114. Springer, July 2000.

[54] E. Y. Vasserman, R. Jansen, J. Tyra, N. Hopper, and Y. Kim. Membership-concealing overlay networks. In A. D. Keromytis and S. Jha, editors, *16th ACM Conference on Computer and Communications Security (CCS'09)*, pages 390–399. ACM, Nov. 2009.

[55] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In D. Culler and P. Druschel, editors, *5th Symposium on Operating Systems Design and Implementation*, pages 255–270. USENIX Association, Dec. 2002.

[56] C. V. Wright, S. E. Coull, and F. Monrose. Traffic Morphing: An efficient defense against statistical traffic analysis. In G. Vigna, editor, *Network and Distributed System Security Symposium (NDSS'09)*. The Internet Society, Feb. 2009.

[57] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman. Telex: Anticensorship in the network infrastructure. In D. Wagner, editor, *20th USENIX Security Symposium*. USENIX Association, Aug. 2011.

[58] Y. Xie, F. Yu, K. Achan, E. Gillum, M. Goldszmidt, and T. Wobber. How dynamic are IP addresses? In N. Taft and A. Feldmann, editors, *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 301–312. ACM, Aug. 2007.

[59] S. Zander, G. J. Armitage, and P. Branch. An empirical evaluation of IP Time To Live covert channels. In *15th IEEE International Conference on Networks (ICON'07)*, pages 42–47, Nov. 2007.

[60] S. Zander, G. J. Armitage, and P. Branch. Stealthier inter-packet timing covert channels. In J. Domingo-Pascual, P. Manzoni, S. Palazzo, A. Pont, and C. Scoglio, editors, *10th International IFIP TC Networking Conference (NETWORKING'11)*, volume 6640 of *Lecture Notes in Computer Science*, pages 458–470. Springer, May 2011.

[61] J. Zittrain and B. Edelman. Internet filtering in China. *IEEE Internet Computing*, 7(2):70–77, 2003.