The road not taken: Secure asymmetry and deployability for decoy routing systems

Cecylia Bocovich Ian Goldberg
Cheriton School of Computer Science University of Waterloo

ABSTRACT

Censorship circumvention is often characterized as an arms race between a nation-state censor and the developers of censorship resistance systems. Decoy routing systems offer a solution to censorship resistance that has the potential to tilt this arms race in the favour of the censorship resistor. Decoy routing uses real connections to unblocked, overt sites to deliver censored content to users. It aims to make connections to censored content indistinguishable from connections to uncensored sites. This is achieved by employing the help of Internet Service Providers or Autonomous Systems (ASes) that own routers in the middle of the network. However, the deployment of decoy routers has yet to reach fruition. Obstacles to deployment such as the heavy requirements on routers that deploy decoy router relay stations, and possible effects on the quality of service for existing customers that pass through these routers have deterred potential participants from deploying existing systems. Furthermore, connections from clients to overt sites often follow different paths in the upstream and downstream direction, hampering most existing designs. Although decoy routing systems that lessen the burden on participating routers and accommodate asymmetric flows have been proposed, these arguably more deployable systems suffer from security vulnerabilities that put their users at risk of discovery. In this paper, we propose two different techniques for supporting route asymmetry in previously symmetric decoy routing systems. The resulting asymmetric solutions are more secure than previous asymmetric proposals and provide an option for tiered deployment, allowing more cautious ASes to deploy a lightweight, non-blocking relay station that aids in defending against routing-capable adversaries. We also provide an experimental evaluation of relay station performance on off-the-shelf hardware and additional security improvements to recently proposed systems.

CCS CONCEPTS
• Security and privacy → Network security; • Networks → Session protocols;

KEYWORDS

censorship resistance, decoy routing, deployability

1 INTRODUCTION

In recent years, Internet censorship has become an increasing worldwide concern. A 2016 Freedom House report declared that Internet freedom has now been in a steady decline for six consecutive years [23]. They reported that in 2016, roughly two-thirds of Internet users deal with government censorship. This censorship aims to cut off access from websites that support political opposition, marginalized communities, and images that criticize or satirize those in power. Furthermore, journalists and users of social media that disseminate, or merely read, content that a censoring nation deems contrary have faced personal dangers such as arrest or increased scrutiny. This makes hiding the use of censorship resistance systems paramount in developing new circumvention technologies.

Tools for censorship circumvention range from simple proxies that hide the IP addresses of sites that users access, to systems that disguise traffic patterns by padding packets or mimicking allowed protocols. These systems have evolved as a result of a cat-and-mouse game between nation-state censors and censorship resisters [32]. As new techniques for evading censorship arise, censors tweak their filtering systems to identify the weaknesses in existing tools that signal their usage. This makes hiding the fact that the user is using a specific tool (given that the censor knows the tool exists and the details of the system) critical to both the user’s safety and the success of the censorship resistance system.

Decoy routing (also known as end-to-middle (E2M) proxying) [3, 10, 16, 22, 37, 38] is a technique for censorship resistance that has the potential to skew the arms race in the favour of the censorship resistor. The key way decoy routing hides its usage is by using real, uncensored (“overt”) traffic to provide access to covert information instead of mimicking allowed traffic. Mimicry is a common technique employed by censorship circumvention systems [5, 8, 26, 34, 36], but by its nature deviates from real traffic in ways that a sufficiently advanced censor could detect [15]. Although such techniques have yet to be documented for use by nation-states in an effort to detect the usage of censorship resistance systems, they still pose a threat to individual users who may, now or in the future, face dire consequences for deviating from their jurisdiction’s strict controls on Internet usage.

Although decoy routing provides strong security properties against both active and passive attacks, there are numerous obstacles to deployment. The deployment of a decoy routing system relies on the participation of autonomous systems (ASes) that own routers in the middle of the network. Previous work on the optimal placement of decoy routers aims to maximize the number of unblocked, overt sites available and minimize the required amount of deployed stations [4, 18, 28]. However, researchers have yet to convince even small ASes with low bandwidth requirements to deploy decoy routing in a production setting. Concerns such as the hardware required to block, modify, or drop traffic at the router, the effect checking for steganographic tags would have on regular traffic, and the logistics involved in setting up and maintaining a relay station are all deterrents for both large and small ASes.

Furthermore, connections to overt sites are often asymmetric. While they may cross a router with a deployed decoy routing relay station on the path to an overt site, the path taken back from the overt site to the user may not cross the same router. This makes
tor has been adopted by many for its usefulness in circumvent- vulnerabilities that could put users already under the scrutiny of a nation-state censor at risk.

In this paper, we address the main challenges to deployability that current decoy routing systems face. Our contributions are as follows:

- We propose a new solution for routing asymmetry that is applicable to all previously symmetric systems. Our solution has stronger security properties than second-generation asymmetric decoy routing systems.
- We show how existing traffic-shaping hardware can be used to further combat the asymmetry problem by forcing flow symmetry for flows that are asymmetric at the router level, but traverse the same autonomous system.
- We provide measurements on the impact relay station deployment would have on regular traffic through a participant ISP. We hope our results will convince an AS interested in deploying a relay station that their quality of service will not be largely affected by even the most complex of the recently proposed decoy routing systems.
- We describe a possible vulnerability in decoy routing systems that modify and re-encrypt TLS application data, and our modification that defends against an adversary capable of seeing traffic on both sides of the relay station. This adversary falls outside the decoy routing threat model for the censor, but a non-censoring adversary could exploit the vulnerability to decrypt or modify covert traffic.

In the next section, we discuss existing work in censorship circumvention. In Section 3 we propose our solution for handling asymmetric routes, followed by experimental results on relay station efficiency in Section 4 and security improvements to existing schemes in Section 5. We end with a conclusion and a discussion of future steps towards deployment in Section 6.

2 CENSORSHIP CIRCUMVENTION

Nation-state censors filter Internet traffic before it leaves their area of influence. Past studies on Internet filtering have revealed a variety of techniques such as blocking access to specific IP addresses [2, 27], filtering DNS requests by the URL or keyword [27], or performing more sophisticated deep-packet inspection techniques to determine the usage of censorship resistance tools [32, 35]. Many Internet filtering techniques employed by censors have evolved in response to the development of the censorship circumvention systems. A notable example of this censorship arms race is the interaction between Internet filtering in China and advances in the censorship resistance aspects of Tor [7].

Originally developed to provide anonymity for web browsing, Tor has been adopted by many for its usefulness in circumventing government censorship. By disguising which website a user is browsing, Tor prevents a censor from learning whether or not a user is accessing a blocked website. As such, many countries that censor web traffic began to block all access to Tor. Tshantz et al. [32] document the arms race between Tor and China’s Great Firewall (GFW) with extensive empirical evidence taken from bug reports, correspondence with The Tor Project, and changes in the Tor protocol. In response to the blocking of publicly listed Tor relays, unlisted Tor relays called bridges began to be circulated privately, enabling their use for a short period of time before their discovery by censors [6]. When bridges are discovered by censors and subsequently blocked, new bridges are cycled into use. The GFW responded to the introduction of bridges by using more sophisticated deep-packet inspection techniques and exploiting unique patterns in the Tor protocol to differentiate Tor traffic from regular web browsing. This led to the development pluggable transports [5, 8, 11, 26, 34, 36], designed to encapsulate and disguise the defining characteristics of Tor traffic.

The majority of pluggable transports take one of three different approaches to disguising Tor traffic: obfuscation, mimicry, or appropriation. The first approach aims to mask the defining characteristics of Tor traffic by making the connection look as random as possible [5, 36]. The success of this technique is grounded in the assumption that censors are unwilling to block traffic that they are unable to definitively classify as censorship resistance or contrary to their governance, as that would possibly lead to an increase in public unrest [9]. However, past precedent indicates that in critical times censors may be willing to take the risk; Aryan et al. recorded the blocking of undefined Internet protocols by the government of Iran during the 2013 presidential elections [2].

Mimicry aims to make connections indistinguishable from popular unblocked content or services, forcing censors to make a difficult decision: to either continue to expand their list of blocked sites to include popular services (thereby risking public unrest), or surrender their position in the arms race. Many pluggable transports shape traffic or encapsulate it in messages that closely resemble protocols such as HTTP [8], Skype [26], or HTML [34]. The ultimatum presented to the censor rests entirely on the ability of these systems to mimic allowed sites and services more closely than the censor’s ability to exploit minor differences. Houmansadr et al. [15] argue that the maintenance of near-perfect mimicry is extremely difficult; as advances in computing allow censors to classify large amounts of traffic more accurately, censorship resisters will see themselves on the losing side of this reactive battle.

While the cycling of bridges and use of pluggable transports has proven effective in many regions for providing access to Tor, there is a danger that after their discovery, censoring nations will start to punish users that have connected to IP addresses revealed to be entry points to the Tor network. Meek is a pluggable transport that appropriates connections to allowed sites and services. It disguises the IP address of bridges by hiding them behind popular services such as Google, Amazon Web Services, or Microsoft Azure using a technique called domain fronting [11]. A user makes a real connection to one of these large domains and accesses a proxy running inside their systems. Not only does this protect the user by making it impossible for a censor to link them to a specific IP address used to access Tor, it also leverages a powerful incentive for governments that do not control equivalent services not to block access to these powerful sites. For nations that do posses equivalence, the efficacy of this method diminishes. Other circumvention systems appropriate allowed protocols and tunnel censorship resistance traffic through them [17, 19, 25]. By using existing implementations of
Decoy routing was originally proposed by three independent research groups in 2011 as a means to move censorship resistance systems from easily blocked endpoints to the middle of the network [16, 22, 38]. These first-generation systems were followed by second-generation systems that aimed to improve the deployability and security of their predecessors [3, 10, 37]. The motivation of decoy routing was to level the playing field by fighting powerful nation-state censors with powerful nation-state defenses. The technique requires the cooperation and active participation of Internet Service Providers (ISPs) and Autonomous Systems (ASes) that own routers outside of regions that censor Internet traffic. Non-censoring jurisdictions would place decoy routers, or relay stations, at strategic points on the path between users in censoring regions and popular, unblocked sites. Users suffering from censorship could then make a connection to these unblocked overt sites and send a steganographic tag, recognizable by the deployed relay stations—and only by them—as a request to access censored content, appropriating the connection. From the censor’s point of view, this tag and other features of the user’s connection are identical to any other access to the overt site. We give an overview of a generic decoy routing system architecture in Figure 1.

The details of the steganographic tagging procedure vary across decoy routing systems. Similar across systems, however, is the usage of a random-looking channel as the mechanism to communicate a user’s circumvention intent to a deployed relay station. Telex [37], Curveball [22], Rebound [10], and Slitheen [3] place a tag in the random nonce of the ClientHello message in the TLS handshake with the overt site. This tag is recognizable only by a targeted deployed relay station and gives the relay station the information necessary to compute the TLS master secret for the session and man-in-the-middle the connection between the user and the overt site. Cirripede [16] uses the initial sequence numbers of TCP SYN packets to register a user with the system, while TapDance [37] uses the ciphertext of an established TLS session to send a public key and the encrypted TLS master secret to the relay station, enabling a man-in-the-middle of the connection.

After the tagging/registration phase is complete, the decoy routing system begins to relay covert information to the client. The details of this process also vary, and this phase is the aspect of decoy routing subject to the identifying characteristics of appropriation. It is at this point that the relay stations for most decoy routing systems abandon or sever the connection to the overt site. Telex and Curveball spoof a TCP RST packet from the client to the overt site and assume the overt site’s responsibilities by decrypting upstream traffic from the client, establishing a connection to the censored covert site specified by the client, and proxying information back and forth between the client and this covert site. Cirripede directly intercepts client traffic after the registration phase for a fixed period of time. All connections from the client’s IP address that pass through a Cirripede router are redirected to a proxy server. This server will set up connections to covert sites and relay information to and from the client. TapDance leaves the initial connection to the overt site open, failing to end the initial HTTP GET request. The client sends their upstream covert data as a part of this GET request, while receiving downstream covert content from the relay station. While the registration phase of these systems is provably indistinguishable from regular, non-decoy, connections to the overt site, traffic in the proxy phase takes the shape and characteristics of the covert site. This makes these systems vulnerable to website fingerprinting [14, 33] or latency analysis attacks [30].

Rebound and Slitheen do not sever or abandon the connection to the overt site. After the TLS handshake, Rebound actively involves the overt site by sending HTTP requests to the overt server. These requests are invalid, prompting an HTTP error that returns the name of the missing or invalid resource. This enables a relay station to transport encrypted covert data to the user by encoding it in the name of the requested resource. In this manner, they effectively “bounce” information off of the overt site. However, this unusual technique makes Rebound trivially identifiable by a passively observing censor. To keep a consistent TCP state (matching sequence and acknowledgement numbers), a client has to send data in an amount directly proportional to the data they receive. This traffic pattern is highly unusual for normal web browsing, and the bounce technique is actively harmful to many overt sites, constituting an HTTP flood attack.

Slitheen maintains a connection to the overt site, but sends valid HTTP requests for real resources. A slitheen client loads overt
websites in the exact manner that a regular user would, parsing HTTP responses and loading additional content with the help of an actual web browser, termed an overt user simulator (OUS). Covert content is delivered to the user by the relay station in the place of “leaf resources”, or resources such as images or videos that would not prompt a browser to make additional connections for more resources. Leaf resources are replaced on a per-packet basis as they pass through the relay station, making decoy routing traffic using Slitheen identical to a regular access of the overt site. The pattern of connections to overt servers, packet sizes, and page load times are indistinguishable from a non-decoy session, removing the distinguishing characteristics that arise from appropriating the connection to the overt site. The task of the censor now falls on determining whether the access pattern of the overt sites themselves is done by a user simulator or a regular user, a problem that is more likely to be error-prone for a censor than existing site fingerprinting techniques.

2.2 Known challenges to deployment

Recently, a number of research groups have proposed solutions to the decoy router placement problem (DRP) that aims to maximize the coverage of overt sites available through decoy routing stations and minimize the number of decoy routers needed to successfully inhibit a censor’s ability to evade decoy routers and block overt sites [4, 18, 28]. Sufficiently powerful censors can perform Routing Around Decoys (RAD) attacks by manipulating BGP and routing tables to send traffic to overt sites down paths that do not contain a deployed relay station [30]. With enough deployed stations, these attacks become extremely difficult and expensive [18]. However, we have yet to see any significant deployment, let alone the widespread placement of relay stations in the middle of the network.

Wustrow et al. [37] were the first to closely examine deployment challenges, and developed TapDance as the result of discussions with ISPs about their reluctance to deploy existing systems. The resource requirements of relay stations and route asymmetry were cited as the most onerous to ISPs and practical usage of existing systems. Telex and Curveball both require the relay station to perform in-line flow blocking, severing the connection between the user and the overt site after the TLS handshake. This not only requires sophisticated and potentially expensive hardware, it also violates the terms of service many ISPs have with overt sites. To our knowledge, there have yet to be experiments on the resources needed by a relay station to check steganographic tags and the impact these operations would have on the quality of service for all overt sites accessible through the deployed relay station. Tags need to be checked for every TLS connection, which now comprise over a third of all Internet traffic [1] and require the relay station to perform expensive public key operations. In Section 4, we provide an extensive analysis of the impact of checking Telex tags using specialized hardware. We chose Telex tags as they are used by multiple systems, including Telex, Slitheen, and Rebound.

Another obstacle in the deployment of decoy routing systems is the prevalence of asymmetric flows. The upstream path from a user to an overt site may pass through a relay station, but the downstream path may take a different route and miss the relay station targeted by the user’s tag. Of the six existing decoy routing systems, only Cirripede, TapDance, and Rebound support asymmetric flows. As long as the user’s traffic passes through a relay station on its upstream path to the overt site, the relay station can effectively deliver covert content to the user. However, all existing asymmetric solutions have significant flaws that could allow a passive censor to identify their usage.

For Telex, Curveball, and Slitheen, the relay station has to see both upstream and downstream traffic of a tagged session. We propose two options for allowing previously symmetric systems to recognize and use asymmetric flows for the delivery of covert content. Our first solution is a gossip protocol for deployed relay stations to share information about potential steganographic tags. Our second solution takes advantage of specialized, but off-the-shelf, hardware to allow ASes to force symmetry for routers within their control. We describe the details of our solutions and compare them with existing systems in the next section.

3 ROUTING ASYMMETRY

Traffic between a client and an overt site often takes a different route, passing through different routers or ASes, in the upstream and downstream directions. Past studies have found somewhere between 80% and 90% of routes to be asymmetric [13, 20, 31]. This asymmetry becomes more prevalent in the centre of the network.

John et al. [20] found that only about 10% of flows are symmetric in Tier-1 networks (i.e., the backbone of the Internet), while flows at the edge of the network are symmetric about 70% of the time. The ability of a decoy routing system to work in the presence of asymmetric flows enhances the system’s deployability by increasing the effectiveness of deployed stations and lowering the number of relay stations that must be deployed to defend against routing-capable adversaries. Each individual relay station can intercept traffic meant for a larger number of overt sites. Only three of the existing decoy routing systems accommodate routing asymmetry: Cirripede [16], TapDance [37], and Rebound [10] function properly if a user’s traffic passes through a deployed relay station only in the upstream direction towards the overt site, but each has significant issues, which we outline next.

Cirripede accomplishes routing asymmetry by handling client registration (i.e., recognizing that a client wishes to begin a decoy routing session) solely through the passive observation of TCP SYN packets. These packets are sent from the client to the overt site at the start of every connection. After recording the ISNs from 12 of the client’s TCP connections, they make a rule in their routing table to divert all traffic from the client’s IP address to a service proxy for a fixed period of time. During this time, as long as the client’s traffic passes through this router in the upstream direction towards any overt site, it will be redirected to a service proxy that will relay data to and from the client and a covert site. Downstream data from the covert site is sent directly from the service proxy to the client, eliminating any need for a relay station to be placed downstream.

On the usability side, a disadvantage of this approach is that all of a client’s traffic will be redirected to the service proxy during the fixed time set by the relay station. If a client wishes to browse a site normally, they must wait for the duration of the decoy routing session to end. There is also a security vulnerability due to the fact that traffic between the user and the covert site does not follow the
same downstream path it normally would in a connection to the overt site during the relay phase. If the overt site and the covert site are significantly far apart, a censor could easily notice a significant difference in latency or in where the traffic enters their network to identify decoy routing sessions.

TapDance implements asymmetry by waiting for the client and overt site to complete the TLS handshake before initiating the tagging procedure. The first upstream HTTP GET request from the client contains a tag in the ciphertext that gives the relay station the client’s public key and the encrypted TLS master secret for the session. After retrieving the TLS master secret, the relay station can decrypt upstream data from the client and establish a connection to the covert site. It then sends covert data to the client directly, encrypting it with the TLS master secret and assuming the role of the overt server. Unfortunately, the non-blocking nature of TapDance and its inability to block or modify downstream traffic leaves the system vulnerable to active attacks by an adversarial censor. Because the relay station is sending traffic to the client on behalf of the overt site, the TCP sequence numbers for downstream data will differ from the overt site’s TCP state. A censor can then replay a stale TCP packet to the overt site, prompting an acknowledgement that reveals the server’s true state, inconsistent with what the censor has witnessed. TapDance also suffers from the same passive attack as Cirripede, that stems from the difference in the locations of the relay station and the overt site.

Rebound’s asymmetric solution presents a different problem by making traffic vulnerable to attack from a passive adversary. Rebound’s upstream-only relays receive necessary handshake information from the client in an encoding method similar to TapDance. After reconstructing the TLS master secret, the relay delivers covert content to the user by encrypting it and sending it as an invalid resource name to the overt server in an HTTP GET request. To maintain a consistent TCP state between the overt server and what a passive censor sees, a client must send a GET request with a length that matches the length of the downstream data she wishes to receive. This results in a nearly equal amount of upstream traffic and downstream traffic, which is a highly atypical traffic pattern for any type of web browsing activity. Furthermore, the ethical implications of sending several bad requests to overt sites makes this technique undesirable.

In this section, we describe two solutions to achieve asymmetry for previously symmetric decoy routing systems such as Telex, Curveball, or Slitheen, that maintain the security properties of these systems. The first solution allows relay stations positioned in the upstream half of a connection from a user to an overt site to gossip ClientHello random nonces to possible downstream relay stations that may be able to recognize a tag. As this random nonce is the only upstream part of the TLS handshake a relay station needs to compute the TLS master secret, a downstream station only needs this small amount of gossiped information—and not necessarily in real time—to successfully use that and subsequent flows for decoy routing. Our second solution to routing asymmetry targets flows that cross through different routers of the same AS in both directions. In this case, we describe the use of off-the-shelf hardware to force flow symmetry within an AS.

---

**Figure 2:** Modified TLS handshake for tagged flows. We define the context string \( \chi \) as server_ip || p, where \( p = \text{ClientHello random}[0..3] \). For a full comparison of the original TLSv1.2 protocol to these modifications, see Appendix A.

### 3.1 Asymmetric Gossip Protocol

Our first solution for asymmetric decoy routing takes a slightly different approach from existing solutions. First, we require the existence of a relay station in both the upstream half of the flow (on the path from the client to the overt site), and the downstream half (on the path from the overt site to the client). These relay stations, however, do not need to be placed on the same router, or even in the same AS. Second, the majority of the work is done by the relay station in the downstream half of the flow, contrary to previous schemes that target relay stations that are upstream from the client.

The basis of our focus on the downstream relay station comes from the fact that a relay station only needs to observe one upstream handshake message to compute the TLS master secret: the ClientHello message that contains the steganographic tag in its random nonce. However, the relay station needs to see multiple downstream handshake messages: the ServerHello, ServerKeyExchange, and (in the case of Slitheen), the downstream Finished message. To minimize the communication between two relay stations either side of the flow, the upstream relay station “gossips” received ClientHello messages to other known relay stations, in an attempt to reach a relay station on the downstream path.

This approach spans multiple flows between a client and an overt site and therefore requires route stability, in which although each flow is routed asymmetrically, the routers traversed in each direction do not vary significantly between the same two endpoints. There is evidence that routes are highly stable; a 2009 study by Schwartz et al. [31] compared the routes taken between over 10,000 sets of endpoints with an average of about 100 measurements for each pair. They found that most pairs of endpoints had a dominant route, or one in which over half of the traffic between these routes traversed. 25% of pairs had absolute stability where all traffic (although possibly asymmetric) always crossed the same routers. Furthermore, the number of distinct routes for endpoints that did experience variance was usually small, only about 20% of all endpoint pairs had over 20 distinct routes, and very few had over 60. This leads us to believe that there is a high probability that subsequent flows between the same client and overt site will cross
the same downstream relay station, particularly in the short term due to load-balancing practices.

We first describe the tagging phase of our asymmetric solution using the modified TLS handshake used by Slitheen for tagging flows. We follow this with a discussion of asymmetry in the relay, or proxying, phase of the decoy routing session.

3.1.1 Asymmetric Tagging. We use the Slitheen tagging protocol as it is a slightly modified, updated version of Telex’s tagging procedure, differing only in the handling of TLS Finished messages, as shown in Figure 2. This method varies only slightly from that used by Curveball, in which the client and the relay station share a symmetric secret that was exchanged out of band.

In Slitheen, a steganographic tag is placed in the last 28 bytes of the 32-byte random nonce of the TLS ClientHello handshake message. The first 4 bytes are reserved, and depending on the TLS implementation, these bytes may be randomly generated (as in Firefox’s NSS), or generated from a timestamp (as in OpenSSL). Our solution uses an implementation that randomly computes these bytes, and we will refer to this random value as \(\rho = \text{ClientHello random}[0..3]\) for ease of reference in the rest of the paper.

To enable the downstream relay station to compute the TLS master secret from the previous ClientHello random nonce and the server’s TLS handshake messages, we make one further small modification to the tag. In Telex and Slitheen, the tag and the client’s TLS key exchange parameters are computed from the client-relay shared secret as well as a context string \(\chi\), where

\[
\chi = \text{server_ip} || \rho \parallel \text{TLS_session_id}
\]

In our asymmetric setup, the downstream relay station is responsible for intercepting the flow and may not have access to the TLS session id (as this may or may not be reflected in the ServerHello message, depending on the session’s resumption status). Therefore, we use a different context string \(\chi = \text{server_ip} || \rho\) that depends only on the server’s IP address and the first 4 bytes of the ClientHello random nonce. Removing the TLS session ID from the context string will not affect the security of the scheme as an adversarial censor is still unable to perform a tag replay attack. In this attack, a censor would observe a suspect flow and then initiate a TLS connection to the same overt site, reusing the suspect ClientHello random nonce in the hopes of observing their own decoy routing session. Such an attack would require the censor to generate the correct TLS session key matching the tag without the client or the relay private secret, which violates the security of public-key cryptography.

We give an overview of our asymmetric solution in Figure 3. A client begins an asymmetric decoy routing session by generating a random secret \(s\) and composing the steganographic tag \(g^s || H_1(g^s || \chi)\) for the ClientHello random nonce using the public key, \(g^s\), of a relay station in the downstream half of the flow. The client computes their secret exponent in the key exchange part of the TLS handshake from the client-relay shared secret, \(g^{r_s}\) by seeding a secure pseudo-random number generator with \(H_2(g^{r_s} || \chi)\).

When a relay station in the upstream half of the flow receives a ClientHello message for which it does not recognize the tag, and for which it has not seen the SYN/ACK packet for the flow (indicating routing asymmetry), it gossipes the ClientHello random nonce along with the flow’s identifying information (i.e., the IP addresses and ports for both the client and server) to nearby relay stations that could possibly be on the downstream path of the flow. If a relay station receives this information, recognizes the tag, and is able to view the downstream path of the flow defined by the IP addresses

Figure 3: Gossip protocol for symmetric flow tagging. The user tags a connection to the overt site for a relay station positioned in the downstream half of the flow. When a relay station sees a ClientHello message in an upstream flow for which it has not seen the downstream SYN/ACK packet, and does not recognize a tag in the random nonce, they gossip this nonce, along with identifying flow information, to nearby relay stations (1). Each of these relay stations checks the gossiped nonce for a tag with their own private keys. If it was tagged for them, and they observe the downstream half of the flow, they save the client and server IP addresses and wait for a future TLS session between the client and the overt site to begin (2). When the user next makes a connection to the same overt site (3), they will generate the new client exponent as a hash of the previous client-relay shared secret \(s' = H_3(g^{r_s} || \text{server_ip} || \rho)\). The downstream relay can reconstruct the client exponent and the new ClientHello random nonce themselves. After seeing the server’s handshake messages, the relay can compute the TLS master secret (4) and replace downstream content. After the tagging phase, the upstream relay station will continue to send copies of the upstream data to the downstream server, however, this communication is not time critical.

In our asymmetric setup, we will refer to this random value as the TLS key exchange parameters are computed from the client-relay data to the downstream server, however, this communication is not time critical.
and ports received from the upstream station, it saves the client and server IP addresses in a table along with the client-relay shared secret $g^s$, and waits for future connections from the client to the same overt site. We emphasize that this gossipped message does not need to be received by the downstream station before the overt site responds to the ClientHello message. If it is late, it simply acts as a registration step, allowing the downstream station to successfully use the next asymmetric connection between the client and the same overt site.

To compute the TLS master secret for a decoy routing session, the relay station needs three values: 1) the premaster secret, computed from the tag in the ClientHello random nonce and the server’s public key in the ServerKeyExchange message, 2) the ClientHello random nonce, and 3) the ServerHello random nonce. In the event that the downstream relay station receives the tag and flow information before the overt site has sent the ServerHello message of the TLS handshake, it can proceed to compute the TLS master secret for the current session. If the downstream station has missed the ServerHello message by the time the gossip protocol completes, it waits for the next connection from the client to the same overt site.

The next time a client makes a connection to the same overt site, they compute the new secret exponent used to construct the steganographic tag as the hash of the previous client-relay shared secret and the IP address of the overt site:

$$s' = H_2(g^s||server_ip||\rho)$$

where the first 4 bytes of the ClientHello random nonce, $\rho$, are generated from the previous shared secret $g^{s'}$. They then place their tag, $g^s||H_1(g^{s'}||x)$, in the ClientHello random nonce of the new TLS session along with the deterministically generated first 4 bytes. When a downstream relay station receives the server handshake messages, they extract the ServerHello random nonce, ServerKeyExchange parameters, and compute the client’s secret exponent and the ClientHello random nonce from the saved client-relay shared secret, $g^s$, and the server IP address.

After computing the TLS master secret for the session, the relay station attempts to decrypt the downstream TLS Finished message. If the decryption is successful, it replaces the hash of the Finished message, $\text{finished\_hash}$, with $MAC_{H_1(g^{s'}||x)}(\text{finished\_hash})$. When the client receives the Finished message, they will compute the keyed MAC of the unmodified TLS Finished message and compare the result with the received value. If they received an unmodified Finished message, the flow was not successfully intercepted by a relay station. If they received the keyed MAC, they know the flow has been intercepted and a decoy routing session has begun.

### 3.1.2 Asymmetric Proxying

After the TLS handshake, the downstream relay station begins to proxy information between the client and a covert site. All three symmetric systems rely on upstream data from the client in order to establish a connection to a covert site and relay upstream data from the client to the covert site. We note that in this stage, the amount of upstream data from the client to the covert site is typically far less than the downstream covert data. To retrieve covert data from an upstream relay station, the downstream relay station will respond to the gossip protocol, signaling that they wish to receive TLS application data from the upstream half of the flow. The upstream station will proceed to funnel upstream TLS records (over a point-to-point encrypted and authenticated connection) to the downstream station, which then decrypts the TLS records and proceeds in the usual manner. The sending of these upstream TLS records has no time constraints; they can arrive at the downstream station asynchronously with downstream data from the covert site or (in the case of Slitheen) the overt site. Any delay in the receipt of this data will not affect the security or correctness of the system, but only the latency experienced by the client in their browsing of covert content. The downstream station will make a connection to the covert site specified by the client and send the client’s upstream covert data through this connection. Telex and Curveball will then deliver downstream covert data directly to the client, while Slitheen will insert it into downstream leaf resources.

Although our approach to asymmetry requires a relay station to be present on both the upstream and downstream paths, these two stations do not need to be the same relay station or be owned by the same AS. By using two relay stations instead of one, we maintain the strong security properties of symmetric systems, providing a more secure asymmetric solution for routing than TapDance and Rebound. We also note that both the tagging and proxying phases of our asymmetric system require no in-line blocking from the upstream relay station. Upstream relay stations in our approach do not need to perform intense operations, and can be implemented without a reliance on specialized hardware. This enables the introduction of tiered participation in decoy routing. An AS without the resources necessary to deploy a full decoy routing relay station can deploy a simple “gossip station” to cover the upstream routes from a censoring nation-state.

By deploying a large number of gossip stations and a fewer number of full downstream stations, our proposal has the potential to further reduce a censor’s ability to perform Routing Around Decoy (RAD) attacks. A routing-capable adversary can more easily and less expensively affect the path a flow takes upstream to the overt site, than the path it takes downstream back to the user. In an upstream RAD attack a censor can decide how to route packets on a flow-by-flow basis, targeting only suspect traffic or enabling route-flapping that could disrupt the processing of a flow in the middle of a TLS handshake. To affect the downstream path of a flow a censor would need to advertise incorrect BGP information for all routes into their area of influence, making it time consuming and costly, and losing the ability for fine-grained control.

### 3.2 Hardware solutions to routing asymmetry

In the event that an asymmetric route traverses two different relay stations owned by the same autonomous system, a more efficient and easier solution to routing asymmetry exists. By taking advantage of recent advances in traffic shaping hardware, ASes can force flow symmetry for two or more routers within their control. We propose the usage of off-the-shelf hardware with built-in procedures to easily accomplish this.

The Sandvine Policy Traffic Switch (PTS) is equipped with the ability to be clustered together with other PTSs such that traffic that normally goes through one PTS in the upstream direction will always traverse the same PTS in the downstream direction, even

---

1The OUS should therefore be a browser such as Firefox that uses random data instead of a timestamp in that field.
Wustrow et al. [37] found the main obstacle in convincing ISPs and ASes to deploy decoy routing systems to be the resource requirements of existing systems in checking tags and performing in-line blocking. By checking every TLS session for steganographic tags, the deployment of decoy routing systems also has the potential to affect the quality of service for all customers whose traffic traverses a relay station.

We performed several experiments to determine the impact a deployed decoy routing station would have on existing traffic in a real-world scenario. Our first set of tests aims to measure the effect of flow asymmetry if the flow would normally pass through another machine [29]. However, the efficacy of this solution depends on flow asymmetry only at the router level, within a single AS, and for the upstream and downstream routes to traverse routers in similar geographic locations. Multiple studies have found that flows that are asymmetric at the router level are often symmetric at the AS level [13, 31]. That is, flows often traverse different routers in each direction, but cross the same set of autonomous systems. Schwartz et al. found that asymmetry was much more prevalent at the router level than at the city level [31]. This suggests that although traffic crosses two different routers in different directions, these routers may be located in the same physical space, allowing the use of hardware solutions to force flow symmetry.

### 3.3 Comparison to previous systems

Our asymmetric solutions in this section provide more secure alternatives to previous proposals for the asymmetric deployment of decoy routing systems. We provide asymmetry for previously symmetric systems while maintaining their stronger security properties. We leverage route stability in our first solution by introducing a gossip protocol that allows a relay station upstream from the client to send (low-volume, not-time-critical) tag information and upstream data from the client to a downstream relay. By tagging flows for a relay station on the downstream path from the overt site to the client, we minimize the data transfer required for the gossip protocol and further mitigate RAD attacks. Our tiered deployment presents a cost-effective way for hesitant ISPs to participate in censorship resistance without the need for hardware that can perform in-line blocking or traffic replacement. The extra participation of upstream gossip stations reduces the effectiveness of RAD attacks and aids those that have fully deployed relay stations in their networks.

Our second solution takes advantage of different granularities of asymmetry. Because some asymmetry occurs at the router level but not at the city or AS level, we can address some forms of routing asymmetry through the use of off-the-shelf hardware and allow the usage of more secure, symmetric decoy routing systems. As we show in the next section, this hardware can also be used to efficiently perform relay station tasks such as in-line blocking and traffic replacement and does not have a large impact on the quality of service experienced by customers of the participant ISP.

### 4 RELAY STATION EXPERIMENTS

Wustrow et al. [37] found the main obstacle in convincing ISPs and ASes to deploy decoy routing systems to be the resource requirements of existing systems in checking tags and performing in-line blocking. By checking every TLS session for steganographic tags, the deployment of decoy routing systems also has the potential to affect the quality of service for all customers whose traffic traverses a relay station.

We performed several experiments to determine the impact a deployed decoy routing station would have on existing traffic in a real-world scenario. Our first set of tests aims to measure the effect of flow asymmetry if the flow would normally pass through another machine [29]. However, the efficacy of this solution depends on flow asymmetry only at the router level, within a single AS, and for the upstream and downstream routes to traverse routers in similar geographic locations. Multiple studies have found that flows that are asymmetric at the router level are often symmetric at the AS level [13, 31]. That is, flows often traverse different routers in each direction, but cross the same set of autonomous systems. Schwartz et al. found that asymmetry was much more prevalent at the router level than at the city level [31]. This suggests that although traffic crosses two different routers in different directions, these routers may be located in the same physical space, allowing the use of hardware solutions to force flow symmetry.

handshake requires the most effort from a deployed relay station and the Slitheen source code is freely available.\(^2\)

For our tests, we used specialized (but off-the-shelf) hardware capable of performing in-line blocking and efficient deep-packet inspection. Our reasons for doing so were that 1) only TapDance does not require in-line blocking, and this feature also introduces several vulnerabilities that an active attacker can exploit to easily differentiate decoy routing sessions, and 2) by showing the capabilities of existing hardware to efficiently perform relay routing tasks we can target existing users of this hardware as the first to deploy a relay routing system. The relay station itself consists of two parts: a Sandvine\(^3\) Policy Traffic Switch (PTS) 22600 capable of performing deep-packet inspection and flow diversion, and a relay station server with two 10 Gb/s connections to the PTS. If a tagged flow is detected by the PTS, it is diverted to the relay station server. The relay station server and client machine each used 8 cores and 2 GB of RAM. The PTS is responsible for routing all traffic and checking the tags of TLS flows. If a flow is tagged for the relay station’s public key, the PTS then diverts the flow through the relay station server, which performs the rest of the tagging procedure during the TLS handshake and handles the proxy phase of the relay routing session. We provide an overview of our experimental setup in Figure 4.

For our tests, we gathered distribution statistics for Internet traffic from the Center for Applied Internet Data Analysis (CAIDA). We used the anonymized passive trace statistics through an OC48 link belonging to a large ISP in Chicago with a maximum load of 10 Gb/s. We calculated the average flows/s and average Mbit/s for 5 major types of flows: HTTPS, HTTP, DNS, generic TCP, and generic UDP, gathered over the course of an hour on April 6th, 2016 [1]. For each test, we sent a CAIDA-representative amount of traffic through the deployed relay station using four client machines and four server machines, on opposite sides of the PTS. To test the

\(^2\)https://crysp.uwaterloo.ca/software/slitheen/
\(^3\)https://www.sandvine.com/
We tested two conditions: one where the PTS checked ClientHello when the client sent the ClientHello message to the time it took TLS ClientHello packets with tag checking on and off. To measure the latency introduced by checking for tags in ClientHello messages, we conducted tests to see whether the deployment of a relay station had an impact on the latency of ClientHello messages. We performed a similar test to the one mentioned above, this time making 1000 HTTP connections to remote sites for each condition. For each connection, we calculated the average RTT of all TCP packets in the flow. The results are given as CDFs in Figure 6. The additional latency of deploying a relay station was 0.4 ms, which is very low, and falls within a standard deviation of each condition (10 ms). We note that at this time, the Slitheen tag checking and relay station code has not been optimized for quality of service. With further improvements, the results in this section for both TLS and non-TLS flows will likely show an even lower impact on the customers of participant ISPs.

Our results show that while the deployment of a relay station adds additional latency to flows due to checking for tags in ClientHello random nonces, the latency introduced is quite small.

4.2 Defenses against latency analysis attacks

The security properties of Slitheen rely on the inability of a censor to detect additional latency added by the relay station in checking tags or replacing content from the overt site. We conducted tests to see whether the divert functionality of the PTS and the implementation of the relay station added enough latency to tagged flows to allow a censor to reliably classify them as decoy routing sessions. To measure this, we simulated an attack in which the censor compiled a database of expected latencies for both decoy sessions and regular browsing sessions for each overt destination by making 100 connections to the top 5 Alexa sites for each condition. We then calculated the precision and recall an adversary could achieve in classifying flows as decoy routing or regular sessions.

We measured two different types of latency for each flow: the time it took to perform a full TLS handshake, and the average TCP acknowledgement time, or round-trip time (RTT) for application data. A censor will attempt to select a cut-off latency for each measurement type to identify decoy routing sessions. All flows with a higher latency than the cut-off value are classified as decoy routing sessions, while all flows with a lower latency are classified as a regular access to the overt site. We computed the CDFs of each type of latency for decoy routing sessions and regular accesses.

### Table 1: Distribution statistics (CAIDA / our experiments)

<table>
<thead>
<tr>
<th>Flow type</th>
<th>Average flows/s</th>
<th>Average Mb/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPS</td>
<td>4.43k / 4,330 (±90)</td>
<td>848.81 / 840 (±20)</td>
</tr>
<tr>
<td>HTTP</td>
<td>4.07k / 4,040 (±80)</td>
<td>814.51 / 800 (±20)</td>
</tr>
<tr>
<td>DNS</td>
<td>2.26k / 2,200 (±100)</td>
<td>N/A / N/A</td>
</tr>
<tr>
<td>TCP</td>
<td>792.74 / 790 (±20)</td>
<td>246.52 / 250 (±10)</td>
</tr>
<tr>
<td>UDP</td>
<td>527.53 / 530 (±20)</td>
<td>125.87 / 128 (±7)</td>
</tr>
<tr>
<td>Total</td>
<td>12.08k / 12,000 (±100)</td>
<td>2035.71 / 2010 (±40)</td>
</tr>
</tbody>
</table>

Figure 5: CDFs comparing the TCP round trip time (RTT) for TLS ClientHello packets with tag checking on and off.

Figure 6: CDFs comparing the average TCP round trip time (RTT) for non-TLS flows with tag checking on and off.
Alexa top 5 sites. Precision is dependent on the base rate of the incidence of decoy routing sessions. A censor can achieve a precision of 1, indicating that they do not incorrectly classify any regular accesses to the overt site as decoy routing sessions. However, often this means the censor can only identify a very small portion of decoy routing sessions while the majority will continue undetected. They can also achieve a recall of 1 by simply classifying all connections as decoy routing sessions and blocking them. For most censors, both measures are important so we define a censor’s accuracy in terms of its F-score, the harmonic mean of the precision and recall values. Precision and recall can be weighted differently according to the individual goals of the censor, but we consider the default equal weighting in this paper. For each value of \( \beta \), the adversary will select a cut-off value that maximizes their F-score, given the latency distributions of each overt site. We plot the maximum F-score values based on the latency distributions for both the TLS and handshake time and the TCP RTT for five of the Alexa top sites in Figure 7.

The maximum accuracy a censor can achieve in identifying decoy routing sessions is very low for both types of latency. For the majority of sites, this value drops to almost 0 with a base rate of occurrence of decoy routing sessions of less than \( 10^{-4} \), and meaning that if no more than one in every 10,000 connections to popular sites are decoy routing sessions, a censor is unable to reliably determine whether or not any given flow is carrying censorship resistance traffic. Even with a higher occurrence of decoy routing, the maximum F-score stays below 0.5 for most sites, making a reasonable censor that is unwilling to upset their population extremely wary of classifying and blocking potential decoy routing sessions. We note that some sites exhibit anomalous behaviour (e.g., google.com and youtube.com in their TCP RTTs and TLS handshake times, respectively). Such behaviour can be measured by the client, and those sites not selected as overt sites.

### 5 SECURITY ANALYSIS AND IMPROVEMENTS

Our proposal to add asymmetry to previously symmetric decoy routing systems has two main advantages: it has better security properties than previously proposed asymmetric systems, and it provides a path for tiered deployment, creating a less expensive defense against routing-capable adversaries. We provide a comparison of the deployability features and security properties of existing systems in Table 2. The previously symmetric systems Telex, Curveball, and Slitheen are analyzed with our asymmetric improvements.

As the table shows, each of the previously symmetric systems has advantages over systems originally designed for routing asymmetry. TapDance remains the only system capable of performing decoy routing without requiring a relay station to block or modify traffic. However, this feature comes at the cost of security. We believe a better route to deployment is by providing ISPs and ASes with experimental evidence of the impact a deployed relay station would have on customer traffic using existing hardware capable of performing tag checks efficiently and blocking or modifying tagged flows. By targeting ASes that already own this hardware or showing them a clear path to deployment, we are providing more evidence that decoy routing is an attainable option and moving...
Table 2: A comparison of the deployability features and security properties of existing systems. We indicate that a system has the property or feature listed on the left of the table with a filled circle ●. Systems that lack a feature or property are marked with an empty circle ○.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Telex [38]</th>
<th>Telex + this work</th>
<th>Telex + this work</th>
<th>Telex + this work</th>
<th>Telex + this work</th>
<th>Telex + this work</th>
<th>Telex + this work</th>
<th>Telex + this work</th>
<th>Telex + this work</th>
<th>Telex + this work</th>
<th>Telex + this work</th>
<th>Telex + this work</th>
</tr>
</thead>
<tbody>
<tr>
<td>No in-line blocking</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Asymmetric</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Defends against TCP replay attacks</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Defends against latency analysis</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Defends against website fingerprinting</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>RAD-resistant</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

towards real-world deployment. Furthermore, our asymmetric solution does not require in-line blocking for upstream relays, enabling more cautious potential participants to provide a stronger defense against RAD attacks.

5.1 Superencryption of application data

Severing or abandoning the connection with the overt site in the relay phase of the decoy routing session introduces a vulnerability in which the server’s state of the connection does not match the traffic that a censor sees. The censor can exploit this in most systems using a RAD attack or a regular TCP replay attack. Slitheen [3] and Rebound [10] avoid this vulnerability by interacting with the overt site throughout the relay phase, modifying the contents of the TLS encrypted data to give covert data to the client. However, this process of modification and re-encryption may introduce a new vulnerability to both systems. In this section, we describe the vulnerability and our solution. The adversary in this attack is outside the usual threat model for decoy routing, in which we assume that the censor is unable to compare traffic on both sides of the relay station. However, an independent adversary could use it to decrypt or modify the data between the user and the covert site.

Both Slitheen and Rebound require the modification and re-encryption of TLS application data that passes through the relay station. By necessity, this data must be re-encrypted with the same TLS master secret. Some TLS modes of operation, such as AES-GCM (see Figure 8), rely on a public nonce in addition to the secret key. It is important to the security of AES-GCM that two different messages are never encrypted with the same nonce and the same key. However, many implementations of AES-GCM mode for TLS use sequential nonces for each message, and so when Slitheen or Rebound replaces message contents with covert data, it must reuse the nonce to avoid flagging to the observing censor that the censorship-resistance system is in use. However, this presents a security problem as a third party capable of observing the ciphertext on both sides of the relay station can exploit patterns in the underlying plaintext to decrypt both ciphertext messages and modify the underlying plaintext without detection.

The original plaintext, $P_1$, will contain part of the HTTP response body of the original overt image, while the modified plaintext, $P_2$, will contain covert data to the user. The corresponding ciphertexts (limited to 1 block each for simplicity), seen by an observer, are computed as:

\[
C_1 = E_k(n || 0^{31} || 1) \oplus P_1 \\
C_2 = E_k(n || 0^{31} || 1) \oplus P_2
\]

where $E$ is AES encryption, and $n$ is the nonce. The observer can then compute $C_1 \oplus C_2 = P_1 \oplus P_2$, and then exploit patterns in the underlying plaintexts to recover both $P_1$ and $P_2$. If the client was using Slitheen to browse a plaintext covert site, this two-time pad attack is trivial. In addition to breaking the client’s confidentiality, an attacker can also modify the plaintext and compute the correct authentication tag [21]. Given the ciphertexts $C_1$ and $C_2$, as shown above, and the corresponding authentication tags (where $A$ is one
would be detected in the TLS records sent between the client and
where
This attack is exceptionally damaging when the user of Slitheen is
in a similar style to TCP to prevent the loss of covert data chunks
and client, respectively, in a different order than it was written, as
from the covert site has the potential to arrive at the relay station
counter allows each party to process the covert data in
in a targeted denial of service attack against decoy routing users.
However, an adversary could use this to perform
consequences of both of these attacks are mitigated. An adversary
for an arbitrary ciphertext
known:

\[ T_1 = ((A \cdot E_k(0) \oplus C_1) \cdot E_k(0) \oplus L) \cdot E_k(0) \oplus E_k(n)0^{32}) \]

\[ T_2 = ((A \cdot E_k(0) \oplus C_2) \cdot E_k(0) \oplus L) \cdot E_k(0) \oplus E_k(n)0^{32}) \]

where \( L = \lfloor \text{len}(A) \rfloor \lfloor \text{len}(C) \rfloor \) and multiplications are performed in
GF(2^{128}).

The adversary can compute:

\[ E_k(0) = \sqrt{\frac{T_1 \times T_2}{C_1 \times C_2}} \]

and from that, since the additional authentication data \( A \) is known:

\[ E_k(n)0^{32}) = ((A_3 \cdot E_k(0) \oplus C_3) \cdot E_k(0) \oplus L) \cdot E_k(0) \oplus E_k(n)0^{32}) \]

In the event that a user is browsing a covert site with TLS, the
the counter is as expected and the padding at the end of the header
exists. The client can then decrypt the covert data and send it to
the client’s browser.

6 CONCLUSION AND FUTURE WORK
As the Internet becomes more centralized and the capabilities of
censors grow, so will their ability to filter Internet traffic with
increasingly sophisticated methods. It is possible in the future that
as censorship becomes more prevalent, so will the dangers of re-
isting government controls. There is a dire need for a censorship
circumvention system that provides users with blocked content
as well as hides their usage of the system. Censorship resistance
traffic that appears to be traffic to allowed sites or services is the
latest battleground in the arms race between censors and resistors.
While current methods for mimicry have proven effective against
many nation-state adversaries, the risk of future discovery and
the application of advanced traffic classification techniques will
pose a threat to users of existing systems. Decoy routing provides
a potential solution to the censorship arms race. Its strong security
properties, and trend of realistically appropriating real, uncensored
connections in the place of mimicry have the potential to end the
arms race in favour of the resistor.

Before decoy routing can be deployed and used by people in
censored regions of the world, we must first address the obstacles
to deployment that have prevented the owners of network infras-
tructure from participating in existing systems. In this paper, we
proposed a new approach to routing asymmetry that provides bet-
ter security than previous asymmetric systems and a path to tiered
deployment that allows for several lightweight, limited systems
to surround a powerful censor, limiting the censor’s ability to per-
form routing-based attacks. We also investigated the use of existing
hardware in the implementation of decoy routing systems and as
a possible second solution to routing asymmetry. We experi-
mentally tested the impact decoy router deployment would have on
the quality of service for traffic flowing through a participant router,
and more carefully examine the latency of decoy routing sessions
processed by the relay station. Finally, we identified a possible secu-
rety vulnerability in existing systems and propose a cryptographic
solution that doubles as a means to more accurately deliver covert
content to the user.

This work presents the next steps towards the deployment of
decoy routing systems, however there is still much work to be
done. With more efficient implementations of the relay station,
the possible impact of deployment may be even less than what we
measured with our limited improvements. We look at our results as
a positive indication that decoy routing may prove to be practical in the future and may convince the owners of Internet routers to consider participating in censorship circumvention.

ACKNOWLEDGEMENTS

The work benefitted from the use of the CrySP RIPPLE Facility at the University of Waterloo. We especially thank Lori Paniak (University of Waterloo) and Dave Dolson (Sandvine) for their technical expertise. The authors thank the Ontario Graduate Scholarships Program for funding Bocovich, and Sandvine and NSERC for grant STPGP-463324.

REFERENCES


A MODIFICATIONS TO TLS

Most decoy routing systems require modifications to the TLSv1.2 handshake. In this section, we describe the modifications used by Sliteen [3] described in Section 3, and directly compare them to the original TLSv1.2 handshake. These are very similar to the modifications used in Telex [38] and Curveball [22].

Figure 9 gives an overview of a TLS handshake, with modifications shown in red. The modifications do not change the number or the size of the messages sent between the client and the server, only the contents of the messages. This is done in a way to avoid detection by the censor: only a party in possession of the client
secret, the relay secret, or the TLS master secret can detect that modifications have been made.

The first modification happens in the generation of the ClientHello random nonce. This nonce is usually randomly generated and is used in the computation of the TLS master secret. In Slithereen, the last 28 bytes of this 32 byte nonce are replaced with a steganographic tag, $g^s || H_1(g^s || \chi)$, where $s$ is a secret generated by the client, $g^s$ is the public key of a relay station, and $\chi = server_ip || ClientHello$ random[0..3] is a context string that consists of the server’s IP address and the first 4 bytes of the ClientHello random nonce (in e.g., OpenSSL, this is a timestamp, while in NSS it is generated randomly). The tag is recognizable only to the relay station with the private key $r$, and appears indistinguishable from random to any other observer.

The next modification is in the computation of values in the ClientKeyExchange message. Instead of randomly generating her private key exchange parameter, the client generates it from the previously generated tag. The private key is the result of feeding the client-relay shared secret, $H_2(g^s || \chi)$ into a pseudo-random number generator. She then computes her public parameters in the ClientKeyExchange message from this private key. The relay station also has the ability to compute the private key, allowing it to later man-in-the-middle the TLS connection.

The last modification to the handshake is in the downstream Finished message, sent from the server to the client. This message usually contains a hash of all previously seen handshake messages, finished_hash. The relay station intercepts this message and replaces it with a MAC that depends on the original Finished message and the client-relay shared secret,

$$MAC_{H_4}(g^s || \chi)(finished\_hash)$$

The purpose of this modification is to alert the client that the session can safely be used for decoy routing.